MySQL 5.6 Reference Manual

Including MySQL NDB Cluster 7.3-7.4 Reference Guide

Abstract

This is the MySQL™ Reference Manual. It documents MySQL 5.6 through 5.6.51, as well as NDB Cluster releases based on versions 7.3 and 7.4 of NDB through 5.6.51-ndb-7.3.32 and 5.6.51-ndb-7.4.31, respectively. It may include documentation of features of MySQL versions that have not yet been released. For information about which versions have been released, see the MySQL 5.6 Release Notes.

MySQL 5.6 features. This manual describes features that are not included in every edition of MySQL 5.6; such features may not be included in the edition of MySQL 5.6 licensed to you. If you have any questions about the features included in your edition of MySQL 5.6, refer to your MySQL 5.6 license agreement or contact your Oracle sales representative.

For notes detailing the changes in each release, see the MySQL 5.6 Release Notes.

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

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

Document generated on: 2021-01-04 (revision: 68412)

Table of Contents

Preface and Legal Notices xxiii
1 General Information 1
1.1 About This Manual 2
1.2 Overview of the MySQL Database Management System 4
1.2.1 What is MySQL? 4
1.2.2 The Main Features of MySQL 5
1.2.3 History of MySQL 8
1.3 What Is New in MySQL 5.6 8
1.4 Server and Status Variables and Options Added, Deprecated, or Removed in MySQL 5.6 20
1.5 MySQL Information Sources 34
1.6 How to Report Bugs or Problems 35
1.7 MySQL Standards Compliance 39
1.7.1 MySQL Extensions to Standard SQL 41
1.7.2 MySQL Differences from Standard SQL 43
1.7.3 How MySQL Deals with Constraints 45
1.8 Credits 48
1.8.1 Contributors to MySQL 48
1.8.2 Documenters and translators 53
1.8.3 Packages that support MySQL 54
1.8.4 Tools that were used to create MySQL 55
1.8.5 Supporters of MySQL 55
2 Installing and Upgrading MySQL 57
2.1 General Installation Guidance 59
2.1.1 Supported Platforms 60
2.1.2 Which MySQL Version and Distribution to Install 60
2.1.3 How to Get MySQL 61
2.1.4 Verifying Package Integrity Using MD5 Checksums or GnuPG 61
2.1.5 Installation Layouts 74
2.1.6 Compiler-Specific Build Characteristics 74
2.2 Installing MySQL on Unix/Linux Using Generic Binaries 74
2.3 Installing MySQL on Microsoft Windows 77
2.3.1 MySQL Installation Layout on Microsoft Windows 79
2.3.2 Choosing an Installation Package 80
2.3.3 MySQL Installer for Windows 81
2.3.4 Installing MySQL on Microsoft Windows Using a `noinstall` ZIP Archive 104
2.3.5 Troubleshooting a Microsoft Windows MySQL Server Installation 112
2.3.6 Windows Postinstallation Procedures 113
2.3.7 Windows Platform Restrictions 115
2.4 Installing MySQL on OS X 116
2.4.1 General Notes on Installing MySQL on OS X 117
2.4.2 Installing MySQL on OS X Using Native Packages 118
2.4.3 Installing a MySQL Launch Daemon 123
2.4.4 Installing and Using the MySQL Preference Pane 125
2.5 Installing MySQL on Linux 130
2.5.1 Installing MySQL on Linux Using the MySQL Yum Repository 131
2.5.2 Replacing a Third-Party Distribution of MySQL Using the MySQL Yum Repository 134
2.5.3 Installing MySQL on Linux Using the MySQL APT Repository 137
2.5.4 Installing MySQL on Linux Using the MySQL SLES Repository 137
2.5.5 Installing MySQL on Linux Using RPM Packages from Oracle 137
2.5.6 Installing MySQL on Linux Using Debian Packages from Oracle 141
2.5.7 Installing MySQL on Linux from the Native Software Repositories 142
2.5.8 Deploying MySQL on Linux with Docker 146
2.5.9 Installing MySQL on Linux with Juju 153

2.6 Installing MySQL Using Unbreakable Linux Network (ULN) 153
2.7 Installing MySQL on Solaris 153
2.7.1 Installing MySQL on Solaris Using a Solaris PKG 154
2.8 Installing MySQL on FreeBSD 155
2.9 Installing MySQL from Source 156
2.9.1 Source Installation Methods 156
2.9.2 Source Installation Prerequisites 157
2.9.3 MySQL Layout for Source Installation 158
2.9.4 Installing MySQL Using a Standard Source Distribution 158
2.9.5 Installing MySQL Using a Development Source Tree 162
2.9.6 Configuring SSL Library Support 164
2.9.7 MySQL Source-Configuration Options 165
2.9.8 Dealing with Problems Compiling MySQL 179
2.9.9 MySQL Configuration and Third-Party Tools 180
2.10 Postinstallation Setup and Testing 181
2.10.1 Initializing the Data Directory 181
2.10.2 Starting the Server 184
2.10.3 Testing the Server 187
2.10.4 Securing the Initial MySQL Accounts 189
2.10.5 Starting and Stopping MySQL Automatically 193
2.11 Upgrading MySQL 194
2.11.1 Before You Begin 194
2.11.2 Upgrade Paths 195
2.11.3 Changes in MySQL 5.6 195
2.11.4 Upgrading MySQL Binary or Package-based Installations on Unix/Linux 202
2.11.5 Upgrading MySQL with the MySQL Yum Repository 204
2.11.6 Upgrading MySQL with the MySQL APT Repository 206
2.11.7 Upgrading MySQL with the MySQL SLES Repository 206
2.11.8 Upgrading MySQL on Windows 206
2.11.9 Upgrade Troubleshooting 208
2.11.10 Rebuilding or Repairing Tables or Indexes 208
2.11.11 Copying MySQL Databases to Another Machine 210
2.12 Downgrading MySQL 211
2.12.1 Before You Begin 211
2.12.2 Downgrade Paths 211
2.12.3 Downgrade Notes 212
2.12.4 Downgrading Binary and Package-based Installations on Unix/Linux 213
2.12.5 Downgrade Troubleshooting 215
2.13 Perl Installation Notes 215
2.13.1 Installing Perl on Unix 216
2.13.2 Installing ActiveState Perl on Windows 216
2.13.3 Problems Using the Perl DBI/DBD Interface 217
3 Tutorial 219
3.1 Connecting to and Disconnecting from the Server 219
3.2 Entering Queries 220
3.3 Creating and Using a Database 223
3.3.1 Creating and Selecting a Database 224
3.3.2 Creating a Table 225
3.3.3 Loading Data into a Table 226
3.3.4 Retrieving Information from a Table 227
3.4 Getting Information About Databases and Tables 240
3.5 Using mysql in Batch Mode 241
3.6 Examples of Common Queries 242
3.6.1 The Maximum Value for a Column 242
3.6.2 The Row Holding the Maximum of a Certain Column 243
3.6.3 Maximum of Column per Group 243
3.6.4 The Rows Holding the Group-wise Maximum of a Certain Column 243
3.6.5 Using User-Defined Variables 244

3.6.6 Using Foreign Keys ... 244
3.6.7 Searching on Two Keys ... 246
3.6.8 Calculating Visits Per Day ... 246
3.6.9 Using AUTO_INCREMENT ... 247
3.7 Using MySQL with Apache ... 249
4 MySQL Programs ... 251
4.1 Overview of MySQL Programs ... 252
4.2 Using MySQL Programs ... 256
4.2.1 Invoking MySQL Programs ... 256
4.2.2 Specifying Program Options ... 257
4.2.3 Command Options for Connecting to the Server ... 270
4.2.4 Connecting to the MySQL Server Using Command Options ... 276
4.2.5 Connection Transport Protocols ... 278
4.2.6 Connection Compression Control ... 280
4.2.7 Setting Environment Variables ... 280
4.3 Server and Server-Startup Programs ... 281
4.3.1 mysqld — The MySQL Server ... 281
4.3.2 mysqld_safe — MySQL Server Startup Script ... 282
4.3.3 mysql.server — MySQL Server Startup Script ... 287
4.3.4 mysqld_multi — Manage Multiple MySQL Servers ... 289
4.4 Installation-Related Programs ... 292
4.4.1 comp_err — Compile MySQL Error Message File ... 292
4.4.2 mysqlbug — Generate Bug Report ... 293
4.4.3 mysql_install_db — Initialize MySQL Data Directory ... 293
4.4.4 mysql_plugin — Configure MySQL Server Plugins ... 297
4.4.5 mysql_secure_installation — Improve MySQL Installation Security ... 299
4.4.6 mysql_tzinfo_to_sql — Load the Time Zone Tables ... 300
4.4.7 mysql_upgrade — Check and Upgrade MySQL Tables ... 300
4.5 Client Programs ... 306
4.5.1 mysql — The MySQL Command-Line Client ... 307
4.5.2 mysqladmin — A MySQL Server Administration Program ... 331
4.5.3 mysqlcheck — A Table Maintenance Program ... 340
4.5.4 mysqldump — A Database Backup Program ... 348
4.5.5 mysqlimport — A Data Import Program ... 368
4.5.6 mysqlshow — Display Database, Table, and Column Information ... 375
4.5.7 mysqlslap — A Load Emulation Client ... 380
4.6 Administrative and Utility Programs ... 389
4.6.1 innochecksum — Offline InnoDB File Checksum Utility ... 389
4.6.2 myisam_ftdump — Display Full-Text Index information ... 390
4.6.3 myisamchk — MyISAM Table-Maintenance Utility ... 391
4.6.4 myisamlog — Display MyISAM Log File Contents ... 407
4.6.5 myisampack — Generate Compressed, Read-Only MyISAM Tables ... 408
4.6.6 mysql_config_editor — MySQL Configuration Utility ... 414
4.6.7 mysqlaccess — Client for Checking Access Privileges ... 420
4.6.8 mysqlbinlog — Utility for Processing Binary Log Files ... 423
4.6.9 mysqldumpslow — Summarize Slow Query Log Files ... 442
4.6.10 mysqlhotcopy — A Database Backup Program ... 444
4.6.11 mysql_convert_table_format — Convert Tables to Use a Given Storage Engine ... 447
4.6.12 mysql_find_rows — Extract SQL Statements from Files ... 448
4.6.13 mysql_fix_extensions — Normalize Table File Name Extensions ... 448
4.6.14 mysql_setpermission — Interactively Set Permissions in Grant Tables ... 449
4.6.15 mysql_waitpid — Kill Process and Wait for Its Termination ... 450
4.6.16 mysql_zap — Kill Processes That Match a Pattern ... 450
4.7 Program Development Utilities ... 451
4.7.1 msql2mysql — Convert mSQL Programs for Use with MySQL ... 451
4.7.2 mysql_config — Display Options for Compiling Clients ... 451
4.7.3 my_print_defaults — Display Options from Option Files ... 453

4.7.4 `resolve_stack_dump` — Resolve Numeric Stack Trace Dump to Symbols 454
4.8 Miscellaneous Programs 455
4.8.1 `perror` — Display MySQL Error Message Information 455
4.8.2 `replace` — A String-Replacement Utility 455
4.8.3 `resolveip` — Resolve Host name to IP Address or Vice Versa 456
4.9 Environment Variables 457
4.10 Unix Signal Handling in MySQL 458
5 MySQL Server Administration 461
5.1 The MySQL Server 462
5.1.1 Configuring the Server 462
5.1.2 Server Configuration Defaults 464
5.1.3 Server Option, System Variable, and Status Variable Reference 466
5.1.4 Server System Variable Reference 496
5.1.5 Server Status Variable Reference 510
5.1.6 Server Command Options 522
5.1.7 Server System Variables 545
5.1.8 Using System Variables 637
5.1.9 Server Status Variables 650
5.1.10 Server SQL Modes 668
5.1.11 Connection Management 676
5.1.12 IPv6 Support 681
5.1.13 MySQL Server Time Zone Support 685
5.1.14 Server-Side Help Support 690
5.1.15 The Server Shutdown Process 690
5.2 The MySQL Data Directory 692
5.3 The mysql System Database 692
5.4 MySQL Server Logs 694
5.4.1 Selecting General Query Log and Slow Query Log Output Destinations 695
5.4.2 The Error Log 697
5.4.3 The General Query Log 699
5.4.4 The Binary Log 701
5.4.5 The Slow Query Log 712
5.4.6 The DDL Log 714
5.4.7 Server Log Maintenance 715
5.5 MySQL Server Plugins 716
5.5.1 Installing and Uninstalling Plugins 717
5.5.2 Obtaining Server Plugin Information 721
5.5.3 MySQL Enterprise Thread Pool 722
5.5.4 MySQL Plugin Services 728
5.6 MySQL Server User-Defined Functions 728
5.6.1 Installing and Uninstalling User-Defined Functions 728
5.6.2 Obtaining User-Defined Function Information 729
5.7 Running Multiple MySQL Instances on One Machine 729
5.7.1 Setting Up Multiple Data Directories 731
5.7.2 Running Multiple MySQL Instances on Windows 732
5.7.3 Running Multiple MySQL Instances on Unix 735
5.7.4 Using Client Programs in a Multiple-Server Environment 736
5.8 Debugging MySQL 736
5.8.1 Debugging a MySQL Server 736
5.8.2 Debugging a MySQL Client 743
5.8.3 The DBUG Package 743
5.8.4 Tracing mysqld Using DTrace 746
6 Security 765
6.1 General Security Issues 766
6.1.1 Security Guidelines 766
6.1.2 Keeping Passwords Secure 767
6.1.3 Making MySQL Secure Against Attackers 775
6.1.4 Security-Related mysqld Options and Variables 777

6.1.5 How to Run MySQL as a Normal User 778
6.1.6 Security Considerations for LOAD DATA LOCAL 778
6.1.7 Client Programming Security Guidelines 780
6.2 Access Control and Account Management 781
6.2.1 Account User Names and Passwords 782
6.2.2 Privileges Provided by MySQL 784
6.2.3 Grant Tables 790
6.2.4 Specifying Account Names 796
6.2.5 Access Control, Stage 1: Connection Verification 798
6.2.6 Access Control, Stage 2: Request Verification 801
6.2.7 Adding Accounts, Assigning Privileges, and Dropping Accounts 803
6.2.8 When Privilege Changes Take Effect 806
6.2.9 Assigning Account Passwords 806
6.2.10 Server Handling of Expired Passwords 808
6.2.11 Pluggable Authentication 810
6.2.12 Proxy Users 814
6.2.13 Setting Account Resource Limits 820
6.2.14 Troubleshooting Problems Connecting to MySQL 822
6.2.15 SQL-Based Account Activity Auditing 826
6.3 Using Encrypted Connections 828
6.3.1 Configuring MySQL to Use Encrypted Connections 829
6.3.2 Encrypted Connection TLS Protocols and Ciphers 831
6.3.3 Creating SSL and RSA Certificates and Keys 835
6.3.4 SSL Library-Dependent Capabilities 840
6.3.5 Connecting to MySQL Remotely from Windows with SSH 841
6.4 Security Plugins 842
6.4.1 Authentication Plugins 842
6.4.2 The Connection-Control Plugins 871
6.4.3 The Password Validation Plugin 877
6.4.4 MySQL Enterprise Audit 883
6.4.5 MySQL Enterprise Firewall 908
6.5 MySQL Enterprise Encryption 923
6.5.1 MySQL Enterprise Encryption Installation 923
6.5.2 MySQL Enterprise Encryption Usage and Examples 924
6.5.3 MySQL Enterprise Encryption User-Defined Function Reference 926
6.5.4 MySQL Enterprise Encryption User-Defined Function Descriptions 926
6.6 SELinux 930
6.6.1 Check if SELinux is Enabled 931
6.6.2 Changing the SELinux Mode 931
6.6.3 MySQL Server SELinux Policies 932
6.6.4 SELinux File Context 932
6.6.5 SELinux TCP Port Context 933
6.6.6 Troubleshooting SELinux 934
7 Backup and Recovery 937
7.1 Backup and Recovery Types 938
7.2 Database Backup Methods 941
7.3 Example Backup and Recovery Strategy 943
7.3.1 Establishing a Backup Policy 943
7.3.2 Using Backups for Recovery 945
7.3.3 Backup Strategy Summary 946
7.4 Using mysqldump for Backups 946
7.4.1 Dumping Data in SQL Format with mysqldump 946
7.4.2 Reloading SQL-Format Backups 947
7.4.3 Dumping Data in Delimited-Text Format with mysqldump 948
7.4.4 Reloading Delimited-Text Format Backups 949
7.4.5 mysqldump Tips 950
7.5 Point-in-Time (Incremental) Recovery Using the Binary Log 951
7.5.1 Point-in-Time Recovery Using Event Times 953

7.5.2 Point-in-Time Recovery Using Event Positions 953
7.6 MyISAM Table Maintenance and Crash Recovery 954
7.6.1 Using myisamchk for Crash Recovery 955
7.6.2 How to Check MyISAM Tables for Errors 955
7.6.3 How to Repair MyISAM Tables 956
7.6.4 MyISAM Table Optimization 958
7.6.5 Setting Up a MyISAM Table Maintenance Schedule 959
8 Optimization 961
8.1 Optimization Overview 962
8.2 Optimizing SQL Statements 964
8.2.1 Optimizing SELECT Statements 964
8.2.2 Optimizing Subqueries and Derived Tables 1001
8.2.3 Optimizing INFORMATION_SCHEMA Queries 1010
8.2.4 Optimizing Data Change Statements 1015
8.2.5 Optimizing Database Privileges 1016
8.2.6 Other Optimization Tips 1016
8.3 Optimization and Indexes 1017
8.3.1 How MySQL Uses Indexes 1017
8.3.2 Primary Key Optimization 1018
8.3.3 Foreign Key Optimization 1019
8.3.4 Column Indexes 1019
8.3.5 Multiple-Column Indexes 1020
8.3.6 Verifying Index Usage 1022
8.3.7 InnoDB and MyISAM Index Statistics Collection 1022
8.3.8 Comparison of B-Tree and Hash Indexes 1023
8.3.9 Use of Index Extensions 1024
8.3.10 Indexed Lookups from TIMESTAMP Columns 1027
8.4 Optimizing Database Structure 1029
8.4.1 Optimizing Data Size 1029
8.4.2 Optimizing MySQL Data Types 1031
8.4.3 Optimizing for Many Tables 1032
8.4.4 Internal Temporary Table Use in MySQL 1034
8.4.5 Limits on Number of Databases and Tables 1035
8.4.6 Limits on Table Size 1035
8.4.7 Limits on Table Column Count and Row Size 1036
8.5 Optimizing for InnoDB Tables 1039
8.5.1 Optimizing Storage Layout for InnoDB Tables 1039
8.5.2 Optimizing InnoDB Transaction Management 1040
8.5.3 Optimizing InnoDB Read-Only Transactions 1041
8.5.4 Optimizing InnoDB Redo Logging 1042
8.5.5 Bulk Data Loading for InnoDB Tables 1042
8.5.6 Optimizing InnoDB Queries 1043
8.5.7 Optimizing InnoDB DDL Operations 1044
8.5.8 Optimizing InnoDB Disk I/O 1044
8.5.9 Optimizing InnoDB Configuration Variables 1047
8.5.10 Optimizing InnoDB for Systems with Many Tables 1048
8.6 Optimizing for MyISAM Tables 1048
8.6.1 Optimizing MyISAM Queries 1049
8.6.2 Bulk Data Loading for MyISAM Tables 1050
8.6.3 Optimizing REPAIR TABLE Statements 1051
8.7 Optimizing for MEMORY Tables 1053
8.8 Understanding the Query Execution Plan 1053
8.8.1 Optimizing Queries with EXPLAIN 1053
8.8.2 EXPLAIN Output Format 1054
8.8.3 Extended EXPLAIN Output Format 1066
8.8.4 Estimating Query Performance 1068
8.9 Controlling the Query Optimizer 1069
8.9.1 Controlling Query Plan Evaluation 1069

8.9.2 Switchable Optimizations 1069
8.9.3 Index Hints 1072
8.10 Buffering and Caching 1074
8.10.1 InnoDB Buffer Pool Optimization 1075
8.10.2 The MyISAM Key Cache 1075
8.10.3 The MySQL Query Cache 1079
8.10.4 Caching of Prepared Statements and Stored Programs 1086
8.11 Optimizing Locking Operations 1087
8.11.1 Internal Locking Methods 1087
8.11.2 Table Locking Issues 1090
8.11.3 Concurrent Inserts 1091
8.11.4 Metadata Locking 1092
8.11.5 External Locking 1095
8.12 Optimizing the MySQL Server 1096
8.12.1 System Factors 1096
8.12.2 Optimizing Disk I/O 1096
8.12.3 Using Symbolic Links 1098
8.12.4 Optimizing Memory Use 1101
8.13 Measuring Performance (Benchmarking) 1104
8.13.1 Measuring the Speed of Expressions and Functions 1105
8.13.2 The MySQL Benchmark Suite 1105
8.13.3 Using Your Own Benchmarks 1106
8.13.4 Measuring Performance with performance_schema 1106
8.14 Examining Server Thread (Process) Information 1106
8.14.1 Accessing the Process List 1107
8.14.2 Thread Command Values 1108
8.14.3 General Thread States 1110
8.14.4 Delayed-Insert Thread States 1116
8.14.5 Query Cache Thread States 1117
8.14.6 Replication Source Thread States 1118
8.14.7 Replication Replica I/O Thread States 1118
8.14.8 Replication Replica SQL Thread States 1119
8.14.9 Replication Replica Connection Thread States 1120
8.14.10 NDB Cluster Thread States 1120
8.14.11 Event Scheduler Thread States 1121
9 Language Structure 1123
9.1 Literal Values 1123
9.1.1 String Literals 1123
9.1.2 Numeric Literals 1126
9.1.3 Date and Time Literals 1126
9.1.4 Hexadecimal Literals 1128
9.1.5 Bit-Value Literals 1130
9.1.6 Boolean Literals 1131
9.1.7 NULL Values 1131
9.2 Schema Object Names 1132
9.2.1 Identifier Length Limits 1133
9.2.2 Identifier Qualifiers 1134
9.2.3 Identifier Case Sensitivity 1136
9.2.4 Mapping of Identifiers to File Names 1138
9.2.5 Function Name Parsing and Resolution 1140
9.3 Keywords and Reserved Words 1143
9.4 User-Defined Variables 1165
9.5 Expressions 1168
9.6 Comments 1172
10 Character Sets, Collations, Unicode 1175
10.1 Character Sets and Collations in General 1176
10.2 Character Sets and Collations in MySQL 1177
10.2.1 Character Set Repertoire 1178

10.2.2 UTF-8 for Metadata 1180
10.3 Specifying Character Sets and Collations 1181
10.3.1 Collation Naming Conventions 1181
10.3.2 Server Character Set and Collation 1182
10.3.3 Database Character Set and Collation 1183
10.3.4 Table Character Set and Collation 1184
10.3.5 Column Character Set and Collation 1185
10.3.6 Character String Literal Character Set and Collation 1186
10.3.7 The National Character Set 1188
10.3.8 Character Set Introducers 1188
10.3.9 Examples of Character Set and Collation Assignment 1190
10.3.10 Compatibility with Other DBMSs 1190
10.4 Connection Character Sets and Collations 1191
10.5 Configuring Application Character Set and Collation 1197
10.6 Error Message Character Set 1198
10.7 Column Character Set Conversion 1199
10.8 Collation Issues 1200
10.8.1 Using COLLATE in SQL Statements 1200
10.8.2 COLLATE Clause Precedence 1201
10.8.3 Character Set and Collation Compatibility 1201
10.8.4 Collation Coercibility in Expressions 1201
10.8.5 The binary Collation Compared to _bin Collations 1203
10.8.6 Examples of the Effect of Collation 1205
10.8.7 Using Collation in INFORMATION_SCHEMA Searches 1206
10.9 Unicode Support 1208
10.9.1 The utf8mb4 Character Set (4-Byte UTF-8 Unicode Encoding) 1209
10.9.2 The utf8mb3 Character Set (3-Byte UTF-8 Unicode Encoding) 1210
10.9.3 The utf8 Character Set (Alias for utf8mb3) 1210
10.9.4 The ucs2 Character Set (UCS-2 Unicode Encoding) 1211
10.9.5 The utf16 Character Set (UTF-16 Unicode Encoding) 1211
10.9.6 The utf16le Character Set (UTF-16LE Unicode Encoding) 1211
10.9.7 The utf32 Character Set (UTF-32 Unicode Encoding) 1211
10.9.8 Converting Between 3-Byte and 4-Byte Unicode Character Sets 1212
10.10 Supported Character Sets and Collations 1214
10.10.1 Unicode Character Sets 1215
10.10.2 West European Character Sets 1220
10.10.3 Central European Character Sets 1221
10.10.4 South European and Middle East Character Sets 1222
10.10.5 Baltic Character Sets 1223
10.10.6 Cyrillic Character Sets 1223
10.10.7 Asian Character Sets 1224
10.10.8 The Binary Character Set 1227
10.11 Restrictions on Character Sets 1228
10.12 Setting the Error Message Language 1228
10.13 Adding a Character Set 1229
10.13.1 Character Definition Arrays 1231
10.13.2 String Collating Support for Complex Character Sets 1232
10.13.3 Multi-Byte Character Support for Complex Character Sets 1232
10.14 Adding a Collation to a Character Set 1232
10.14.1 Collation Implementation Types 1233
10.14.2 Choosing a Collation ID 1236
10.14.3 Adding a Simple Collation to an 8-Bit Character Set 1237
10.14.4 Adding a UCA Collation to a Unicode Character Set 1238
10.15 Character Set Configuration 1244
10.16 MySQL Server Locale Support 1245
11 Data Types 1251
11.1 Numeric Data Types 1252
11.1.1 Numeric Data Type Syntax 1252

11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT 1255
11.1.3 Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC 1256
11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE 1256
11.1.5 Bit-Value Type - BIT 1257
11.1.6 Numeric Type Attributes 1257
11.1.7 Out-of-Range and Overflow Handling 1258
11.2 Date and Time Data Types 1259
11.2.1 Date and Time Data Type Syntax 1261
11.2.2 The DATE, DATETIME, and TIMESTAMP Types 1263
11.2.3 The TIME Type 1264
11.2.4 The YEAR Type 1265
11.2.5 2-Digit YEAR(2) Limitations and Migrating to 4-Digit YEAR 1266
11.2.6 Automatic Initialization and Updating for TIMESTAMP and DATETIME 1268
11.2.7 Fractional Seconds in Time Values 1272
11.2.8 Conversion Between Date and Time Types 1273
11.2.9 2-Digit Years in Dates 1274
11.3 String Data Types 1274
11.3.1 String Data Type Syntax 1274
11.3.2 The CHAR and VARCHAR Types 1278
11.3.3 The BINARY and VARBINARY Types 1279
11.3.4 The BLOB and TEXT Types 1280
11.3.5 The ENUM Type 1282
11.3.6 The SET Type 1285
11.4 Spatial Data Types 1287
11.4.1 Spatial Data Types 1289
11.4.2 The OpenGIS Geometry Model 1289
11.4.3 Supported Spatial Data Formats 1295
11.4.4 Creating Spatial Columns 1297
11.4.5 Populating Spatial Columns 1298
11.4.6 Fetching Spatial Data 1299
11.4.7 Optimizing Spatial Analysis 1299
11.4.8 Creating Spatial Indexes 1299
11.4.9 Using Spatial Indexes 1300
11.5 Data Type Default Values 1302
11.6 Data Type Storage Requirements 1303
11.7 Choosing the Right Type for a Column 1307
11.8 Using Data Types from Other Database Engines 1308
12 Functions and Operators 1309
12.1 SQL Function and Operator Reference 1310
12.2 User-Defined Function Reference 1320
12.3 Type Conversion in Expression Evaluation 1321
12.4 Operators 1324
12.4.1 Operator Precedence 1325
12.4.2 Comparison Functions and Operators 1326
12.4.3 Logical Operators 1332
12.4.4 Assignment Operators 1333
12.5 Flow Control Functions 1334
12.6 Numeric Functions and Operators 1336
12.6.1 Arithmetic Operators 1337
12.6.2 Mathematical Functions 1339
12.7 Date and Time Functions 1348
12.8 String Functions and Operators 1368
12.8.1 String Comparison Functions and Operators 1383
12.8.2 Regular Expressions 1386
12.8.3 Character Set and Collation of Function Results 1392
12.9 What Calendar Is Used By MySQL? 1392
12.10 Full-Text Search Functions 1393

12.10.1 Natural Language Full-Text Searches 1394
12.10.2 Boolean Full-Text Searches 1397
12.10.3 Full-Text Searches with Query Expansion 1402
12.10.4 Full-Text Stopwords 1403
12.10.5 Full-Text Restrictions 1407
12.10.6 Fine-Tuning MySQL Full-Text Search 1408
12.10.7 Adding a User-Defined Collation for Full-Text Indexing 1411
12.11 Cast Functions and Operators 1412
12.12 XML Functions 1417
12.13 Bit Functions and Operators 1428
12.14 Encryption and Compression Functions 1429
12.15 Locking Functions 1438
12.16 Information Functions 1439
12.17 Spatial Analysis Functions 1449
12.17.1 Spatial Function Reference 1449
12.17.2 Argument Handling by Spatial Functions 1452
12.17.3 Functions That Create Geometry Values from WKT Values 1453
12.17.4 Functions That Create Geometry Values from WKB Values 1454
12.17.5 MySQL-Specific Functions That Create Geometry Values 1456
12.17.6 Geometry Format Conversion Functions 1457
12.17.7 Geometry Property Functions 1457
12.17.8 Spatial Operator Functions 1465
12.17.9 Functions That Test Spatial Relations Between Geometry Objects 1466
12.18 Functions Used with Global Transaction IDs 1470
12.19 Aggregate Functions 1471
12.19.1 Aggregate Function Descriptions 1471
12.19.2 GROUP BY Modifiers 1476
12.19.3 MySQL Handling of GROUP BY 1479
12.20 Miscellaneous Functions 1480
12.21 Precision Math 1486
12.21.1 Types of Numeric Values 1487
12.21.2 DECIMAL Data Type Characteristics 1487
12.21.3 Expression Handling 1488
12.21.4 Rounding Behavior 1490
12.21.5 Precision Math Examples 1491
13 SQL Statements 1495
13.1 Data Definition Statements 1496
13.1.1 ALTER DATABASE Statement 1496
13.1.2 ALTER EVENT Statement 1497
13.1.3 ALTER FUNCTION Statement 1499
13.1.4 ALTER LOGFILE GROUP Statement 1499
13.1.5 ALTER PROCEDURE Statement 1500
13.1.6 ALTER SERVER Statement 1501
13.1.7 ALTER TABLE Statement 1501
13.1.8 ALTER TABLESPACE Statement 1520
13.1.9 ALTER VIEW Statement 1521
13.1.10 CREATE DATABASE Statement 1521
13.1.11 CREATE EVENT Statement 1522
13.1.12 CREATE FUNCTION Statement 1526
13.1.13 CREATE INDEX Statement 1526
13.1.14 CREATE LOGFILE GROUP Statement 1531
13.1.15 CREATE PROCEDURE and CREATE FUNCTION Statements 1533
13.1.16 CREATE SERVER Statement 1538
13.1.17 CREATE TABLE Statement 1539
13.1.18 CREATE TABLESPACE Statement 1572
13.1.19 CREATE TRIGGER Statement 1574
13.1.20 CREATE VIEW Statement 1576
13.1.21 DROP DATABASE Statement 1580

13.1.22 DROP EVENT Statement 1581
13.1.23 DROP FUNCTION Statement 1581
13.1.24 DROP INDEX Statement 1581
13.1.25 DROP LOGFILE GROUP Statement 1582
13.1.26 DROP PROCEDURE and DROP FUNCTION Statements 1582
13.1.27 DROP SERVER Statement 1583
13.1.28 DROP TABLE Statement 1583
13.1.29 DROP TABLESPACE Statement 1584
13.1.30 DROP TRIGGER Statement 1584
13.1.31 DROP VIEW Statement 1584
13.1.32 RENAME TABLE Statement 1585
13.1.33 TRUNCATE TABLE Statement 1586
13.2 Data Manipulation Statements 1587
13.2.1 CALL Statement 1587
13.2.2 DELETE Statement 1588
13.2.3 DO Statement 1592
13.2.4 HANDLER Statement 1593
13.2.5 INSERT Statement 1594
13.2.6 LOAD DATA Statement 1603
13.2.7 LOAD XML Statement 1613
13.2.8 REPLACE Statement 1620
13.2.9 SELECT Statement 1622
13.2.10 Subqueries 1637
13.2.11 UPDATE Statement 1649
13.3 Transactional and Locking Statements 1653
13.3.1 START TRANSACTION, COMMIT, and ROLLBACK Statements 1653
13.3.2 Statements That Cannot Be Rolled Back 1656
13.3.3 Statements That Cause an Implicit Commit 1656
13.3.4 SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Statements 1657
13.3.5 LOCK TABLES and UNLOCK TABLES Statements 1657
13.3.6 SET TRANSACTION Statement 1663
13.3.7 XA Transactions 1666
13.4 Replication Statements 1670
13.4.1 SQL Statements for Controlling Replication Source Servers 1670
13.4.2 SQL Statements for Controlling Replica Servers 1672
13.5 Prepared Statements 1682
13.5.1 PREPARE Statement 1685
13.5.2 EXECUTE Statement 1686
13.5.3 DEALLOCATE PREPARE Statement 1686
13.6 Compound Statements 1687
13.6.1 BEGIN ... END Compound Statement 1687
13.6.2 Statement Labels 1687
13.6.3 DECLARE Statement 1688
13.6.4 Variables in Stored Programs 1688
13.6.5 Flow Control Statements 1690
13.6.6 Cursors 1694
13.6.7 Condition Handling 1696
13.7 Database Administration Statements 1720
13.7.1 Account Management Statements 1720
13.7.2 Table Maintenance Statements 1739
13.7.3 Plugin and User-Defined Function Statements 1749
13.7.4 SET Statements 1752
13.7.5 SHOW Statements 1756
13.7.6 Other Administrative Statements 1804
13.8 Utility Statements 1814
13.8.1 DESCRIBE Statement 1814
13.8.2 EXPLAIN Statement 1814

13.8.3 HELP Statement 1816
13.8.4 USE Statement 1818
14 The InnoDB Storage Engine 1819
14.1 Introduction to InnoDB 1820
14.1.1 Benefits of Using InnoDB Tables 1822
14.1.2 Best Practices for InnoDB Tables 1823
14.1.3 Checking InnoDB Availability 1824
14.1.4 Testing and Benchmarking with InnoDB 1824
14.1.5 Turning Off InnoDB 1825
14.2 InnoDB and the ACID Model 1825
14.3 InnoDB Multi-Versioning 1827
14.4 InnoDB Architecture 1828
14.5 InnoDB In-Memory Structures 1829
14.5.1 Buffer Pool 1829
14.5.2 Change Buffer 1834
14.5.3 Adaptive Hash Index 1837
14.5.4 Log Buffer 1838
14.6 InnoDB On-Disk Structures 1838
14.6.1 Tables 1838
14.6.2 Indexes 1857
14.6.3 Tablespaces 1863
14.6.4 InnoDB Data Dictionary 1869
14.6.5 Doublewrite Buffer 1869
14.6.6 Redo Log 1870
14.6.7 Undo Logs 1871
14.7 InnoDB Locking and Transaction Model 1872
14.7.1 InnoDB Locking 1872
14.7.2 InnoDB Transaction Model 1876
14.7.3 Locks Set by Different SQL Statements in InnoDB 1883
14.7.4 Phantom Rows 1886
14.7.5 Deadlocks in InnoDB 1887
14.8 InnoDB Configuration 1890
14.8.1 InnoDB Startup Configuration 1890
14.8.2 Configuring InnoDB for Read-Only Operation 1895
14.8.3 InnoDB Buffer Pool Configuration 1896
14.8.4 Configuring the Memory Allocator for InnoDB 1902
14.8.5 Configuring Thread Concurrency for InnoDB 1903
14.8.6 Configuring the Number of Background InnoDB I/O Threads 1904
14.8.7 Using Asynchronous I/O on Linux 1904
14.8.8 Configuring InnoDB I/O Capacity 1905
14.8.9 Configuring Spin Lock Polling 1906
14.8.10 Purge Configuration 1907
14.8.11 Configuring Optimizer Statistics for InnoDB 1908
14.8.12 Configuring the Merge Threshold for Index Pages 1919
14.9 InnoDB Table Compression 1921
14.9.1 Overview of Table Compression 1921
14.9.2 Enabling Compression for a Table 1922
14.9.3 Tuning Compression for InnoDB Tables 1923
14.9.4 Monitoring InnoDB Table Compression at Runtime 1926
14.9.5 How Compression Works for InnoDB Tables 1927
14.9.6 Compression for OLTP Workloads 1930
14.9.7 SQL Compression Syntax Warnings and Errors 1931
14.10 InnoDB File-Format Management 1933
14.10.1 Enabling File Formats 1934
14.10.2 Verifying File Format Compatibility 1934
14.10.3 Identifying the File Format in Use 1937
14.10.4 Modifying the File Format 1938
14.11 InnoDB Row Formats 1938

14.12 InnoDB Disk I/O and File Space Management 1943
14.12.1 InnoDB Disk I/O 1943
14.12.2 File Space Management 1944
14.12.3 InnoDB Checkpoints 1945
14.12.4 Defragmenting a Table 1945
14.12.5 Reclaiming Disk Space with TRUNCATE TABLE 1946
14.13 InnoDB and Online DDL 1946
14.13.1 Online DDL Operations 1947
14.13.2 Online DDL Performance and Concurrency 1955
14.13.3 Online DDL Space Requirements 1958
14.13.4 Simplifying DDL Statements with Online DDL 1959
14.13.5 Online DDL Failure Conditions 1959
14.13.6 Online DDL Limitations 1960
14.14 InnoDB Startup Options and System Variables 1961
14.15 InnoDB INFORMATION_SCHEMA Tables 2028
14.15.1 InnoDB INFORMATION_SCHEMA Tables about Compression 2028
14.15.2 InnoDB INFORMATION_SCHEMA Transaction and Locking Information 2030
14.15.3 InnoDB INFORMATION_SCHEMA System Tables 2035
14.15.4 InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables 2040
14.15.5 InnoDB INFORMATION_SCHEMA Buffer Pool Tables 2043
14.15.6 InnoDB INFORMATION_SCHEMA Metrics Table 2047
14.16 InnoDB Integration with MySQL Performance Schema 2055
14.16.1 Monitoring InnoDB Mutex Waits Using Performance Schema 2057
14.17 InnoDB Monitors 2060
14.17.1 InnoDB Monitor Types 2060
14.17.2 Enabling InnoDB Monitors 2061
14.17.3 InnoDB Standard Monitor and Lock Monitor Output 2064
14.17.4 InnoDB Tablespace Monitor Output 2068
14.17.5 InnoDB Table Monitor Output 2071
14.18 InnoDB Backup and Recovery 2074
14.18.1 InnoDB Backup 2074
14.18.2 InnoDB Recovery 2075
14.19 InnoDB and MySQL Replication 2077
14.20 InnoDB memcached Plugin 2079
14.20.1 Benefits of the InnoDB memcached Plugin 2079
14.20.2 InnoDB memcached Architecture 2080
14.20.3 Setting Up the InnoDB memcached Plugin 2084
14.20.4 Security Considerations for the InnoDB memcached Plugin 2089
14.20.5 Writing Applications for the InnoDB memcached Plugin 2091
14.20.6 The InnoDB memcached Plugin and Replication 2103
14.20.7 InnoDB memcached Plugin Internals 2106
14.20.8 Troubleshooting the InnoDB memcached Plugin 2110
14.21 InnoDB Troubleshooting 2112
14.21.1 Troubleshooting InnoDB I/O Problems 2113
14.21.2 Forcing InnoDB Recovery 2113
14.21.3 Troubleshooting InnoDB Data Dictionary Operations 2115
14.21.4 InnoDB Error Handling 2118
14.22 InnoDB Limits 2119
14.23 InnoDB Restrictions and Limitations 2120
15 Alternative Storage Engines 2123
15.1 Setting the Storage Engine 2127
15.2 The MyISAM Storage Engine 2127
15.2.1 MyISAM Startup Options 2130
15.2.2 Space Needed for Keys 2131
15.2.3 MyISAM Table Storage Formats 2132
15.2.4 MyISAM Table Problems 2134
15.3 The MEMORY Storage Engine 2136
15.4 The CSV Storage Engine 2140

15.4.1 Repairing and Checking CSV Tables ... 2141
15.4.2 CSV Limitations ... 2141
15.5 The ARCHIVE Storage Engine ... 2141
15.6 The BLACKHOLE Storage Engine ... 2143
15.7 The MERGE Storage Engine ... 2145
15.7.1 MERGE Table Advantages and Disadvantages ... 2148
15.7.2 MERGE Table Problems ... 2148
15.8 The FEDERATED Storage Engine ... 2150
15.8.1 FEDERATED Storage Engine Overview ... 2150
15.8.2 How to Create FEDERATED Tables ... 2151
15.8.3 FEDERATED Storage Engine Notes and Tips ... 2154
15.8.4 FEDERATED Storage Engine Resources ... 2155
15.9 The EXAMPLE Storage Engine ... 2155
15.10 Other Storage Engines ... 2156
15.11 Overview of MySQL Storage Engine Architecture ... 2156
15.11.1 Pluggable Storage Engine Architecture ... 2156
15.11.2 The Common Database Server Layer ... 2157
16 High Availability and Scalability ... 2159
16.1 Using ZFS Replication ... 2161
16.1.1 Using ZFS for File System Replication ... 2162
16.1.2 Configuring MySQL for ZFS Replication ... 2163
16.1.3 Handling MySQL Recovery with ZFS ... 2163
16.2 Using MySQL with memcached ... 2164
16.2.1 Installing memcached ... 2165
16.2.2 Using memcached ... 2166
16.2.3 Developing a memcached Application ... 2185
16.2.4 Getting memcached Statistics ... 2208
16.2.5 memcached FAQ ... 2216
17 Replication ... 2221
17.1 Replication Configuration ... 2222
17.1.1 How to Set Up Replication ... 2223
17.1.2 Replication Formats ... 2233
17.1.3 Replication with Global Transaction Identifiers ... 2240
17.1.4 Replication and Binary Logging Options and Variables ... 2249
17.1.5 Common Replication Administration Tasks ... 2310
17.2 Replication Implementation ... 2313
17.2.1 Replication Threads ... 2313
17.2.2 Relay Log and Replication Metadata Repositories ... 2316
17.2.3 How Servers Evaluate Replication Filtering Rules ... 2323
17.3 Replication Solutions ... 2328
17.3.1 Using Replication for Backups ... 2329
17.3.2 Handling an Unexpected Halt of a Replica Server ... 2332
17.3.3 Using Replication with Different Source and Replica Storage Engines ... 2334
17.3.4 Using Replication for Scale-Out ... 2335
17.3.5 Replicating Different Databases to Different Replicas ... 2337
17.3.6 Improving Replication Performance ... 2338
17.3.7 Switching Sources During Failover ... 2339
17.3.8 Setting Up Replication to Use Encrypted Connections ... 2341
17.3.9 Semisynchronous Replication ... 2343
17.3.10 Delayed Replication ... 2347
17.4 Replication Notes and Tips ... 2348
17.4.1 Replication Features and Issues ... 2348
17.4.2 Replication Compatibility Between MySQL Versions ... 2370
17.4.3 Upgrading a Replication Setup ... 2371
17.4.4 Troubleshooting Replication ... 2373
17.4.5 How to Report Replication Bugs or Problems ... 2374
18 MySQL NDB Cluster 7.3 and NDB Cluster 7.4 ... 2377
18.1 NDB Cluster Overview ... 2380

18.1.1 NDB Cluster Core Concepts 2382
18.1.2 NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions 2384
18.1.3 NDB Cluster Hardware, Software, and Networking Requirements 2387
18.1.4 What is New in MySQL NDB Cluster 2388
18.1.5 NDB: Added, Deprecated, and Removed Options, Variables, and Parameters .. 2393
18.1.6 MySQL Server Using InnoDB Compared with NDB Cluster 2397
18.1.7 Known Limitations of NDB Cluster 2400
18.2 NDB Cluster Installation 2411
18.2.1 Installation of NDB Cluster on Linux 2413
18.2.2 Installing NDB Cluster on Windows 2419
18.2.3 Initial Configuration of NDB Cluster 2428
18.2.4 Initial Startup of NDB Cluster 2430
18.2.5 NDB Cluster Example with Tables and Data 2431
18.2.6 Safe Shutdown and Restart of NDB Cluster 2434
18.2.7 Upgrading and Downgrading NDB Cluster 2435
18.2.8 The NDB Cluster Auto-Installer (No longer supported) 2438
18.3 Configuration of NDB Cluster 2449
18.3.1 Quick Test Setup of NDB Cluster 2449
18.3.2 Overview of NDB Cluster Configuration Parameters, Options, and Variables 2451
18.3.3 NDB Cluster Configuration Files 2468
18.3.4 Using High-Speed Interconnects with NDB Cluster 2608
18.4 NDB Cluster Programs 2609
18.4.1 ndbd — The NDB Cluster Data Node Daemon 2609
18.4.2 ndbinfo_select_all — Select From ndbinfo Tables 2616
18.4.3 ndbmtd — The NDB Cluster Data Node Daemon (Multi-Threaded) 2618
18.4.4 ndb_mgmd — The NDB Cluster Management Server Daemon 2619
18.4.5 ndb_mgm — The NDB Cluster Management Client 2626
18.4.6 ndb_blob_tool — Check and Repair BLOB and TEXT columns of NDB Cluster Tables 2628
18.4.7 ndb_config — Extract NDB Cluster Configuration Information 2630
18.4.8 ndb_cpcd — Automate Testing for NDB Development 2639
18.4.9 ndb_delete_all — Delete All Rows from an NDB Table 2639
18.4.10 ndb_desc — Describe NDB Tables 2639
18.4.11 ndb_drop_index — Drop Index from an NDB Table 2644
18.4.12 ndb_drop_table — Drop an NDB Table 2645
18.4.13 ndb_error_reporter — NDB Error-Reporting Utility 2645
18.4.14 ndb_index_stat — NDB Index Statistics Utility 2647
18.4.15 ndb_move_data — NDB Data Copy Utility 2652
18.4.16 ndb_print_backup_file — Print NDB Backup File Contents 2654
18.4.17 ndb_print_file — Print NDB Disk Data File Contents 2654
18.4.18 ndb_print_frag_file — Print NDB Fragment List File Contents 2655
18.4.19 ndb_print_schema_file — Print NDB Schema File Contents 2656
18.4.20 ndb_print_sys_file — Print NDB System File Contents 2656
18.4.21 ndb_redo_log_reader — Check and Print Content of Cluster Redo Log 2656
18.4.22 ndb_restore — Restore an NDB Cluster Backup 2659
18.4.23 ndb_select_all — Print Rows from an NDB Table 2680
18.4.24 ndb_select_count — Print Row Counts for NDB Tables 2683
18.4.25 ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED) 2684
18.4.26 ndb_show_tables — Display List of NDB Tables 2687
18.4.27 ndb_size.pl — NDBCLUSTER Size Requirement Estimator 2688
18.4.28 ndb_waiter — Wait for NDB Cluster to Reach a Given Status 2691
18.4.29 Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs 2694
18.5 Management of NDB Cluster 2699
18.5.1 Commands in the NDB Cluster Management Client 2699
18.5.2 NDB Cluster Log Messages 2703
18.5.3 Event Reports Generated in NDB Cluster 2717

18.5.4 Summary of NDB Cluster Start Phases 2728
18.5.5 Performing a Rolling Restart of an NDB Cluster 2729
18.5.6 NDB Cluster Single User Mode 2731
18.5.7 Adding NDB Cluster Data Nodes Online 2732
18.5.8 Online Backup of NDB Cluster 2743
18.5.9 MySQL Server Usage for NDB Cluster 2747
18.5.10 NDB Cluster Disk Data Tables 2748
18.5.11 Online Operations with ALTER TABLE in NDB Cluster 2756
18.5.12 Distributed Privileges Using Shared Grant Tables 2759
18.5.13 NDB API Statistics Counters and Variables 2762
18.5.14 ndbinfo: The NDB Cluster Information Database 2772
18.5.15 Quick Reference: NDB Cluster SQL Statements 2812
18.5.16 NDB Cluster Security Issues 2817
18.6 NDB Cluster Replication 2824
18.6.1 NDB Cluster Replication: Abbreviations and Symbols 2826
18.6.2 General Requirements for NDB Cluster Replication 2826
18.6.3 Known Issues in NDB Cluster Replication 2827
18.6.4 NDB Cluster Replication Schema and Tables 2834
18.6.5 Preparing the NDB Cluster for Replication 2836
18.6.6 Starting NDB Cluster Replication (Single Replication Channel) 2838
18.6.7 Using Two Replication Channels for NDB Cluster Replication 2839
18.6.8 Implementing Failover with NDB Cluster Replication 2840
18.6.9 NDB Cluster Backups With NDB Cluster Replication 2842
18.6.10 NDB Cluster Replication: Bidirectional and Circular Replication 2848
18.6.11 NDB Cluster Replication Conflict Resolution 2851
18.7 NDB Cluster Release Notes 2865
19 Partitioning 2867
19.1 Overview of Partitioning in MySQL 2869
19.2 Partitioning Types 2871
19.2.1 RANGE Partitioning 2873
19.2.2 LIST Partitioning 2877
19.2.3 COLUMNS Partitioning 2879
19.2.4 HASH Partitioning 2887
19.2.5 KEY Partitioning 2890
19.2.6 Subpartitioning 2891
19.2.7 How MySQL Partitioning Handles NULL 2894
19.3 Partition Management 2899
19.3.1 Management of RANGE and LIST Partitions 2900
19.3.2 Management of HASH and KEY Partitions 2905
19.3.3 Exchanging Partitions and Subpartitions with Tables 2906
19.3.4 Maintenance of Partitions 2911
19.3.5 Obtaining Information About Partitions 2912
19.4 Partition Pruning 2914
19.5 Partition Selection 2918
19.6 Restrictions and Limitations on Partitioning 2923
19.6.1 Partitioning Keys, Primary Keys, and Unique Keys 2930
19.6.2 Partitioning Limitations Relating to Storage Engines 2933
19.6.3 Partitioning Limitations Relating to Functions 2934
19.6.4 Partitioning and Locking 2936
20 Stored Objects 2939
20.1 Defining Stored Programs 2940
20.2 Using Stored Routines 2941
20.2.1 Stored Routine Syntax 2942
20.2.2 Stored Routines and MySQL Privileges 2942
20.2.3 Stored Routine Metadata 2943
20.2.4 Stored Procedures, Functions, Triggers, and LAST_INSERT_ID() 2943
20.3 Using Triggers 2943
20.3.1 Trigger Syntax and Examples 2944

20.3.2 Trigger Metadata 2947
20.4 Using the Event Scheduler 2948
20.4.1 Event Scheduler Overview 2948
20.4.2 Event Scheduler Configuration 2949
20.4.3 Event Syntax 2951
20.4.4 Event Metadata 2951
20.4.5 Event Scheduler Status 2952
20.4.6 The Event Scheduler and MySQL Privileges 2953
20.5 Using Views 2955
20.5.1 View Syntax 2955
20.5.2 View Processing Algorithms 2956
20.5.3 Updatable and Insertable Views 2957
20.5.4 The View WITH CHECK OPTION Clause 2959
20.5.5 View Metadata 2959
20.6 Stored Object Access Control 2959
20.7 Stored Program Binary Logging 2963
20.8 Restrictions on Stored Programs 2968
20.9 Restrictions on Views 2972
21 INFORMATION_SCHEMA Tables 2975
21.1 Introduction 2976
21.2 The INFORMATION_SCHEMA CHARACTER_SETS Table 2979
21.3 The INFORMATION_SCHEMA COLLATIONS Table 2979
21.4 The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table 2980
21.5 The INFORMATION_SCHEMA COLUMNS Table 2980
21.6 The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table 2983
21.7 The INFORMATION_SCHEMA ENGINES Table 2983
21.8 The INFORMATION_SCHEMA EVENTS Table 2984
21.9 The INFORMATION_SCHEMA GLOBAL_STATUS and SESSION_STATUS Tables 2988
21.10 The INFORMATION_SCHEMA GLOBAL_VARIABLES and SESSION_VARIABLES Tables 2988
21.11 The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table 2988
21.12 The INFORMATION_SCHEMA OPTIMIZER_TRACE Table 2990
21.13 The INFORMATION_SCHEMA PARAMETERS Table 2990
21.14 The INFORMATION_SCHEMA PARTITIONS Table 2991
21.15 The INFORMATION_SCHEMA PLUGINS Table 2995
21.16 The INFORMATION_SCHEMA PROCESSLIST Table 2996
21.17 The INFORMATION_SCHEMA PROFILING Table 2997
21.18 The INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS Table 2998
21.19 The INFORMATION_SCHEMA ROUTINES Table 2999
21.20 The INFORMATION_SCHEMA SCHEMATA Table 3001
21.21 The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table 3002
21.22 The INFORMATION_SCHEMA STATISTICS Table 3003
21.23 The INFORMATION_SCHEMA TABLES Table 3004
21.24 The INFORMATION_SCHEMA TABLESPACES Table 3008
21.25 The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table 3008
21.26 The INFORMATION_SCHEMA TABLE_PRIVILEGES Table 3009
21.27 The INFORMATION_SCHEMA TRIGGERS Table 3009
21.28 The INFORMATION_SCHEMA USER_PRIVILEGES Table 3011
21.29 The INFORMATION_SCHEMA VIEWS Table 3012
21.30 INFORMATION_SCHEMA InnoDB Tables 3013
21.30.1 The INFORMATION_SCHEMA INNODB_BUFFER_PAGE Table 3013
21.30.2 The INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU Table 3016
21.30.3 The INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS Table 3019
21.30.4 The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables 3022
21.30.5 The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables 3023

21.30.6 The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables 3025
21.30.7 The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table 3026
21.30.8 The INFORMATION_SCHEMA INNODB_FT_CONFIG Table 3027
21.30.9 The INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD Table . 3028
21.30.10 The INFORMATION_SCHEMA INNODB_FT_DELETED Table 3029
21.30.11 The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table 3030
21.30.12 The INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE Table 3031
21.30.13 The INFORMATION_SCHEMA INNODB_LOCKS Table 3033
21.30.14 The INFORMATION_SCHEMA INNODB_LOCK_WAITS Table 3034
21.30.15 The INFORMATION_SCHEMA INNODB_METRICS Table 3035
21.30.16 The INFORMATION_SCHEMA INNODB_SYS_COLUMNS Table 3037
21.30.17 The INFORMATION_SCHEMA INNODB_SYS_DATAFILES Table 3038
21.30.18 The INFORMATION_SCHEMA INNODB_SYS_FIELDS Table 3038
21.30.19 The INFORMATION_SCHEMA INNODB_SYS_FOREIGN Table 3039
21.30.20 The INFORMATION_SCHEMA INNODB_SYS_FOREIGN_COLS Table 3040
21.30.21 The INFORMATION_SCHEMA INNODB_SYS_INDEXES Table 3040
21.30.22 The INFORMATION_SCHEMA INNODB_SYS_TABLES Table 3042
21.30.23 The INFORMATION_SCHEMA INNODB_SYS_TABLESPACES Table 3043
21.30.24 The INFORMATION_SCHEMA INNODB_SYS_TABLESTATS View 3044
21.30.25 The INFORMATION_SCHEMA INNODB_TRX Table 3045
21.31 INFORMATION_SCHEMA NDB Cluster Tables 3048
21.31.1 The INFORMATION_SCHEMA FILES Table 3048
21.31.2 The INFORMATION_SCHEMA ndb_transid_mysql_connection_map Table 3053
21.32 INFORMATION_SCHEMA Thread Pool Tables 3054
21.32.1 The INFORMATION_SCHEMA TP_THREAD_GROUP_STATE Table 3055
21.32.2 The INFORMATION_SCHEMA TP_THREAD_GROUP_STATS Table 3056
21.32.3 The INFORMATION_SCHEMA TP_THREAD_STATE Table 3058
21.33 INFORMATION_SCHEMA Connection-Control Tables 3059
21.33.1 The INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS Table 3059
21.34 INFORMATION_SCHEMA MySQL Enterprise Firewall Tables 3059
21.34.1 The INFORMATION_SCHEMA MYSQL_FIREWALL_USERS Table 3060
21.34.2 The INFORMATION_SCHEMA MYSQL_FIREWALL_WHITELIST Table 3060
21.35 Extensions to SHOW Statements 3060
22 MySQL Performance Schema 3063
22.1 Performance Schema Quick Start 3065
22.2 Performance Schema Build Configuration 3070
22.3 Performance Schema Startup Configuration 3071
22.4 Performance Schema Runtime Configuration 3073
22.4.1 Performance Schema Event Timing 3074
22.4.2 Performance Schema Event Filtering 3077
22.4.3 Event Pre-Filtering 3079
22.4.4 Pre-Filtering by Instrument 3079
22.4.5 Pre-Filtering by Object 3081
22.4.6 Pre-Filtering by Thread 3082
22.4.7 Pre-Filtering by Consumer 3083
22.4.8 Example Consumer Configurations 3086
22.4.9 Naming Instruments or Consumers for Filtering Operations 3090
22.4.10 Determining What Is Instrumented 3091
22.5 Performance Schema Queries 3091
22.6 Performance Schema Instrument Naming Conventions 3091
22.7 Performance Schema Status Monitoring 3094
22.8 Performance Schema Atom and Molecule Events 3097
22.9 Performance Schema Tables for Current and Historical Events 3097
22.10 Performance Schema Statement Digests 3099
22.11 Performance Schema General Table Characteristics 3102
22.12 Performance Schema Table Descriptions 3103

22.12.1 Performance Schema Table Index 3104
22.12.2 Performance Schema Setup Tables 3105
22.12.3 Performance Schema Instance Tables 3110
22.12.4 Performance Schema Wait Event Tables 3114
22.12.5 Performance Schema Stage Event Tables 3119
22.12.6 Performance Schema Statement Event Tables 3123
22.12.7 Performance Schema Connection Tables 3131
22.12.8 Performance Schema Connection Attribute Tables 3134
22.12.9 Performance Schema Summary Tables 3137
22.12.10 Performance Schema Miscellaneous Tables 3149
22.13 Performance Schema Option and Variable Reference 3156
22.14 Performance Schema Command Options 3158
22.15 Performance Schema System Variables 3159
22.16 Performance Schema Status Variables 3169
22.17 Performance Schema and Plugins 3171
22.18 Using the Performance Schema to Diagnose Problems 3172
22.18.1 Query Profiling Using Performance Schema 3173
22.19 Restrictions on Performance Schema 3174
23 Connectors and APIs 3177
23.1 MySQL Connector/C++ 3179
23.2 MySQL Connector/J 3179
23.3 MySQL Connector/NET 3180
23.4 MySQL Connector/ODBC 3180
23.5 MySQL Connector/Python 3180
23.6 libmysqld, the Embedded MySQL Server Library 3180
23.6.1 Compiling Programs with libmysqld 3181
23.6.2 Restrictions When Using the Embedded MySQL Server 3181
23.6.3 Options with the Embedded Server 3182
23.6.4 Embedded Server Examples 3182
23.7 MySQL C API 3185
23.8 MySQL PHP API 3185
23.9 MySQL Perl API 3185
23.10 MySQL Python API 3186
23.11 MySQL Ruby APIs 3186
23.11.1 The MySQL/Ruby API 3187
23.11.2 The Ruby/MySQL API 3187
23.12 MySQL Tcl API 3187
23.13 MySQL Eiffel Wrapper 3187
24 MySQL Enterprise Edition 3189
24.1 MySQL Enterprise Monitor Overview 3189
24.2 MySQL Enterprise Backup Overview 3190
24.3 MySQL Enterprise Security Overview 3191
24.4 MySQL Enterprise Encryption Overview 3191
24.5 MySQL Enterprise Audit Overview 3191
24.6 MySQL Enterprise Firewall Overview 3192
24.7 MySQL Enterprise Thread Pool Overview 3192
24.8 MySQL Enterprise Data Masking and De-Identification Overview 3192
25 MySQL Workbench 3193
A MySQL 5.6 Frequently Asked Questions 3195
A.1 MySQL 5.6 FAQ: General 3195
A.2 MySQL 5.6 FAQ: Storage Engines 3197
A.3 MySQL 5.6 FAQ: Server SQL Mode 3197
A.4 MySQL 5.6 FAQ: Stored Procedures and Functions 3198
A.5 MySQL 5.6 FAQ: Triggers 3202
A.6 MySQL 5.6 FAQ: Views 3204
A.7 MySQL 5.6 FAQ: INFORMATION_SCHEMA 3204
A.8 MySQL 5.6 FAQ: Migration 3205
A.9 MySQL 5.6 FAQ: Security 3205

A.10 MySQL 5.6 FAQ: NDB Cluster 3206
A.11 MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets 3219
A.12 MySQL 5.6 FAQ: Connectors & APIs 3231
A.13 MySQL 5.6 FAQ: C API, libmysql 3231
A.14 MySQL 5.6 FAQ: Replication 3232
A.15 MySQL 5.6 FAQ: MySQL Enterprise Thread Pool 3236
A.16 MySQL 5.6 FAQ: InnoDB Change Buffer 3237
A.17 MySQL 5.6 FAQ: Virtualization Support 3239
B Error Messages and Common Problems 3241
B.1 Error Message Sources and Elements 3241
B.2 Error Information Interfaces 3243
B.3 Problems and Common Errors 3244
B.3.1 How to Determine What Is Causing a Problem 3244
B.3.2 Common Errors When Using MySQL Programs 3245
B.3.3 Administration-Related Issues 3257
B.3.4 Query-Related Issues 3265
B.3.5 Optimizer-Related Issues 3271
B.3.6 Table Definition-Related Issues 3272
B.3.7 Known Issues in MySQL 3273
C Indexes 3277
MySQL Glossary 3829

Preface and Legal Notices

This is the Reference Manual for the MySQL Database System, version 5.6, through release 5.6.51. Differences between minor versions of MySQL 5.6 are noted in the present text with reference to release numbers (5.6.*x*). For license information, see the Legal Notices.

This manual is not intended for use with older versions of the MySQL software due to the many functional and other differences between MySQL 5.6 and previous versions. If you are using an earlier release of the MySQL software, please refer to the appropriate manual. For example, *MySQL 5.5 Reference Manual* covers the 5.5 series of MySQL software releases.

If you are using MySQL 5.7, please refer to the *MySQL 5.7 Reference Manual*.

Licensing information—MySQL 5.6. This product may include third-party software, used under license. If you are using a *Commercial* release of MySQL 5.6, see the MySQL 5.6 Commercial Release License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this Commercial release. If you are using a *Community* release of MySQL 5.6, see the MySQL 5.6 Community Release License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this Community release.

Licensing information—MySQL NDB Cluster 7.3. This product may include third-party software, used under license. If you are using a *Commercial* release of NDB Cluster 7.3, see the MySQL NDB Cluster 7.3 Commercial Release License Information User Manual for licensing information relating to third-party software that may be included in this Commercial release. If you are using a *Community* release of NDB Cluster 7.3, see the MySQL NDB Cluster 7.3 Community Release License Information User Manual for licensing information relating to third-party software that may be included in this Community release.

Licensing information—MySQL NDB Cluster 7.4. This product may include third-party software, used under license. If you are using a *Commercial* release of NDB Cluster 7.4, see the MySQL NDB Cluster 7.4 Commercial Release License Information User Manual for licensing information relating to third-party software that may be included in this Commercial release. If you are using a *Community* release of NDB Cluster 7.4, see the MySQL NDB Cluster 7.4 Community Release License Information User Manual for licensing information relating to third-party software that may be included in this Community release.

Legal Notices

Copyright © 1997, 2021, Oracle and/or its affiliates.

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

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

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

U.S. GOVERNMENT END USERS: Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific

supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government's use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

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

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

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

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

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

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

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at
https://www.oracle.com/corporate/accessibility/.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit
https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.

Chapter 1 General Information

Table of Contents

1.1 About This Manual 2
1.2 Overview of the MySQL Database Management System 4
    1.2.1 What is MySQL? 4
    1.2.2 The Main Features of MySQL 5
    1.2.3 History of MySQL 8
1.3 What Is New in MySQL 5.6 8
1.4 Server and Status Variables and Options Added, Deprecated, or Removed in MySQL 5.6 20
1.5 MySQL Information Sources 34
1.6 How to Report Bugs or Problems 35
1.7 MySQL Standards Compliance 39
    1.7.1 MySQL Extensions to Standard SQL 41
    1.7.2 MySQL Differences from Standard SQL 43
    1.7.3 How MySQL Deals with Constraints 45
1.8 Credits 48
    1.8.1 Contributors to MySQL 48
    1.8.2 Documenters and translators 53
    1.8.3 Packages that support MySQL 54
    1.8.4 Tools that were used to create MySQL 55
    1.8.5 Supporters of MySQL 55

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

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

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

- For a discussion of MySQL Database Server capabilities, see Section 1.2.2, “The Main Features of MySQL”.
- For an overview of new MySQL features, see Section 1.3, “What Is New in MySQL 5.6”. For information about the changes in each version, see the Release Notes.
- For installation instructions, see Chapter 2, *Installing and Upgrading MySQL*. For information about upgrading MySQL, see Section 2.11, “Upgrading MySQL”.
- For a tutorial introduction to the MySQL Database Server, see Chapter 3, *Tutorial*.
- For information about configuring and administering MySQL Server, see Chapter 5, *MySQL Server Administration*.
- For information about security in MySQL, see Chapter 6, *Security*.
- For information about setting up replication servers, see Chapter 17, *Replication*.
- For information about MySQL Enterprise, the commercial MySQL release with advanced features and management tools, see Chapter 24, *MySQL Enterprise Edition*.

- For answers to a number of questions that are often asked concerning the MySQL Database Server and its capabilities, see Appendix A, *MySQL 5.6 Frequently Asked Questions*.
- For a history of new features and bug fixes, see the Release Notes.

Important

To report problems or bugs, please use the instructions at Section 1.6, "How to Report Bugs or Problems". If you find a security bug in MySQL Server, please let us know immediately by sending an email message to `<secalert_us@oracle.com>`. Exception: Support customers should report all problems, including security bugs, to Oracle Support.

1.1 About This Manual

This is the Reference Manual for the MySQL Database System, version 5.6, through release 5.6.51. Differences between minor versions of MySQL 5.6 are noted in the present text with reference to release numbers (5.6.*x*). For license information, see the Legal Notices.

This manual is not intended for use with older versions of the MySQL software due to the many functional and other differences between MySQL 5.6 and previous versions. If you are using an earlier release of the MySQL software, please refer to the appropriate manual. For example, *MySQL 5.5 Reference Manual* covers the 5.5 series of MySQL software releases.

If you are using MySQL 5.7, please refer to the *MySQL 5.7 Reference Manual*.

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

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

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

Typographical and Syntax Conventions

This manual uses certain typographical conventions:

- `Text in this style` is used for SQL statements; database, table, and column names; program listings and source code; and environment variables. Example: "To reload the grant tables, use the `FLUSH PRIVILEGES` statement."
- **`Text in this style`** indicates input that you type in examples.
- `Text in this style` indicates the names of executable programs and scripts, examples being `mysql` (the MySQL command-line client program) and `mysqld` (the MySQL server executable).
- *`Text in this style`* is used for variable input for which you should substitute a value of your own choosing.
- *Text in this style* is used for emphasis.
- **Text in this style** is used in table headings and to convey especially strong emphasis.
- `Text in this style` is used to indicate a program option that affects how the program is executed, or that supplies information that is needed for the program to function in a certain way.

Example: “The `--host` option (short form `-h`) tells the `mysql` client program the hostname or IP address of the MySQL server that it should connect to”.

- File names and directory names are written like this: “The global `my.cnf` file is located in the `/etc` directory.”
- Character sequences are written like this: “To specify a wildcard, use the ‘`%`’ character.”

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

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

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

```
source> type a mysql command on the replication source here
replica> type a mysql command on the replica here
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Manual Authorship

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

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

1.2 Overview of the MySQL Database Management System

1.2.1 What is MySQL?

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

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

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

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

- **MySQL databases are relational.**

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

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

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

- **MySQL software is Open Source.**

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

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

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

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

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

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

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

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

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

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

1.2.2 The Main Features of MySQL

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

- MySQL 8.0: What Is New in MySQL 8.0
- MySQL 5.7: What Is New in MySQL 5.7
- MySQL 5.6: Section 1.3, “What Is New in MySQL 5.6”

Internals and Portability

- Written in C and C++.
- Tested with a broad range of different compilers.
- Works on many different platforms. See https://www.mysql.com/support/supportedplatforms/database.html.
- For portability, configured using `CMake`.
- Tested with Purify (a commercial memory leakage detector) as well as with Valgrind, a GPL tool (http://developer.kde.org/~sewardj/).
- Uses multi-layered server design with independent modules.
- Designed to be fully multithreaded using kernel threads, to easily use multiple CPUs if they are available.
- Provides transactional and nontransactional storage engines.
- Uses very fast B-tree disk tables (`MyISAM`) with index compression.
- Designed to make it relatively easy to add other storage engines. This is useful if you want to provide an SQL interface for an in-house database.
- Uses a very fast thread-based memory allocation system.
- Executes very fast joins using an optimized nested-loop join.
- Implements in-memory hash tables, which are used as temporary tables.
- Implements SQL functions using a highly optimized class library that should be as fast as possible. Usually there is no memory allocation at all after query initialization.
- Provides the server as a separate program for use in a client/server networked environment, and as a library that can be embedded (linked) into standalone applications. Such applications can be used in isolation or in environments where no network is available.

Data Types

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

Statements and Functions

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

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

- Full support for SQL `GROUP BY` and `ORDER BY` clauses. Support for group functions (`COUNT()`, `AVG()`, `STD()`, `SUM()`, `MAX()`, `MIN()`, and `GROUP_CONCAT()`).
- Support for `LEFT OUTER JOIN` and `RIGHT OUTER JOIN` with both standard SQL and ODBC syntax.
- Support for aliases on tables and columns as required by standard SQL.
- Support for `DELETE`, `INSERT`, `REPLACE`, and `UPDATE` to return the number of rows that were changed (affected), or to return the number of rows matched instead by setting a flag when connecting to the server.

- Support for MySQL-specific `SHOW` statements that retrieve information about databases, storage engines, tables, and indexes. Support for the `INFORMATION_SCHEMA` database, implemented according to standard SQL.
- An `EXPLAIN` statement to show how the optimizer resolves a query.
- Independence of function names from table or column names. For example, `ABS` is a valid column name. The only restriction is that for a function call, no spaces are permitted between the function name and the "`(`" that follows it. See Section 9.3, "Keywords and Reserved Words".
- You can refer to tables from different databases in the same statement.

Security

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

Scalability and Limits

- Support for large databases. We use MySQL Server with databases that contain 50 million records. We also know of users who use MySQL Server with 200,000 tables and about 5,000,000,000 rows.
- Support for up to 64 indexes per table. Each index may consist of 1 to 16 columns or parts of columns. The maximum index width for `InnoDB` tables is either 767 bytes or 3072 bytes. See Section 14.22, "InnoDB Limits". The maximum index width for `MyISAM` tables is 1000 bytes. See Section 15.2, "The MyISAM Storage Engine". An index may use a prefix of a column for `CHAR`, `VARCHAR`, `BLOB`, or `TEXT` column types.

Connectivity

- Clients can connect to MySQL Server using several protocols:
  - Clients can connect using TCP/IP sockets on any platform.
  - On Windows systems, clients can connect using named pipes if the server is started with the `named_pipe` system variable enabled. Windows servers also support shared-memory connections if started with the `shared_memory` system variable enabled. Clients can connect through shared memory by using the `--protocol=memory` option.
  - On Unix systems, clients can connect using Unix domain socket files.
- MySQL client programs can be written in many languages. A client library written in C is available for clients written in C or C++, or for any language that provides C bindings.
- APIs for C, C++, Eiffel, Java, Perl, PHP, Python, Ruby, and Tcl are available, enabling MySQL clients to be written in many languages. See Chapter 23, *Connectors and APIs*.
- The Connector/ODBC (MyODBC) interface provides MySQL support for client programs that use ODBC (Open Database Connectivity) connections. For example, you can use MS Access to connect to your MySQL server. Clients can be run on Windows or Unix. Connector/ODBC source is available. All ODBC 2.5 functions are supported, as are many others. See MySQL Connector/ODBC Developer Guide.
- The Connector/J interface provides MySQL support for Java client programs that use JDBC connections. Clients can be run on Windows or Unix. Connector/J source is available. See MySQL Connector/J 5.1 Developer Guide.
- MySQL Connector/NET enables developers to easily create .NET applications that require secure, high-performance data connectivity with MySQL. It implements the required ADO.NET interfaces and

integrates into ADO.NET aware tools. Developers can build applications using their choice of .NET languages. MySQL Connector/NET is a fully managed ADO.NET driver written in 100% pure C#. See MySQL Connector/NET Developer Guide.

Localization

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

Clients and Tools

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

1.2.3 History of MySQL

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

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

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

1.3 What Is New in MySQL 5.6

This section summarizes what has been added to, deprecated in, and removed from MySQL 5.6. A companion section lists MySQL server options and variables that have been added, deprecated, or removed in MySQL 5.6; see Section 1.4, “Server and Status Variables and Options Added, Deprecated, or Removed in MySQL 5.6”.

- Features Added in MySQL 5.6
- Features Deprecated in MySQL 5.6

- Features Removed in MySQL 5.6

Features Added in MySQL 5.6

The following features have been added to MySQL 5.6:

- **Security improvements.** These security improvements were made:

  - MySQL now provides a method for storing authentication credentials encrypted in an option file named `.mylogin.cnf`. To create the file, use the `mysql_config_editor` utility. The file can be read later by MySQL client programs to obtain authentication credentials for connecting to a MySQL server. `mysql_config_editor` writes the `.mylogin.cnf` file using encryption so the credentials are not stored as clear text, and its contents when decrypted by client programs are used only in memory. In this way, passwords can be stored in a file in non-cleartext format and used later without ever needing to be exposed on the command line or in an environment variable. For more information, see Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”.

  - MySQL now supports stronger encryption for user account passwords, available through an authentication plugin named `sha256_password` that implements SHA-256 password hashing. This plugin is built in, so it is always available and need not be loaded explicitly. For more information, including instructions for creating accounts that use SHA-256 passwords, see Section 6.4.1.4, “SHA-256 Pluggable Authentication”.

  - The `mysql.user` system table now has a `password_expired` column. Its default value is `'N'`, but can be set to `'Y'` with the new `ALTER USER` statement. After an account's password has been expired, all operations performed in subsequent connections to the server using the account result in an error until the user issues a `SET PASSWORD` statement to establish a new account password. For more information, see Section 13.7.1.1, “ALTER USER Statement”, and Section 6.2.10, “Server Handling of Expired Passwords”.

  - MySQL now has provision for checking password security:

    - In statements that assign a password supplied as a cleartext value, the value is checked against the current password policy and rejected if it is weak (the statement returns an `ER_NOT_VALID_PASSWORD` error). This affects the `CREATE USER`, `GRANT`, and `SET PASSWORD` statements. Passwords given as arguments to the `PASSWORD()` and `OLD_PASSWORD()` functions are checked as well.

    - The strength of potential passwords can be assessed using the new `VALIDATE_PASSWORD_STRENGTH()` SQL function, which takes a password argument and returns an integer from 0 (weak) to 100 (strong).

    Both capabilities are implemented by the `validate_password` plugin. For more information, see Section 6.4.3, “The Password Validation Plugin”.

  - `mysql_upgrade` now produces a warning if it finds user accounts with passwords hashed with the older pre-4.1 hashing method. Such accounts should be updated to use more secure password hashing. See Section 6.1.2.4, “Password Hashing in MySQL”

  - On Unix platforms, `mysql_install_db` supports a new option, `--random-passwords`, that provides for more secure MySQL installation. Invoking `mysql_install_db` with `--random-passwords` causes it to assign a random password to the MySQL `root` accounts, set the “password expired” flag for those accounts, and remove the anonymous-user MySQL accounts. For additional details, see Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”.

  - Logging has been modified so that passwords do not appear in plain text in statements written to the general query log, slow query log, and binary log. See Section 6.1.2.3, “Passwords and Logging”.

The `mysql` client no longer logs to its history file statements that refer to passwords. See Section 4.5.1.3, “mysql Client Logging”.

- `START SLAVE` syntax has been modified to permit connection parameters to be specified for connecting to the source. This provides an alternative to storing the password in the `master.info` file. See Section 13.4.2.5, “START SLAVE Statement”.
- MySQL now sets the access control granted to clients on the named pipe to the minimum necessary for successful communication on Windows. Newer MySQL client software can open named pipe connections without any additional configuration. If older client software cannot be upgraded immediately, the new `named_pipe_full_access_group` system variable can be used to give a Windows group the necessary permissions to open a named pipe connection. Membership in the full-access group should be restricted and temporary.

- **MySQL Enterprise.** The format of the file generated by the audit log plugin was changed for better compatibility with Oracle Audit Vault. See Section 6.4.4, “MySQL Enterprise Audit”, and Section 6.4.4.3, “Audit Log File Formats”.

  The audit log plugin included in MySQL Enterprise Edition now has the capability of filtering audited events based on user account and event status. Several new system variables provide DBAs with filtering control. In addition, audit log plugin reporting capability has been improved by the addition of several status variables. For more information, see Section 6.4.4.4, “Audit Log Logging Control”, and Audit Log Plugin Status Variables.

  MySQL Enterprise Edition now includes a set of encryption functions based on the OpenSSL library that expose OpenSSL capabilities at the SQL level. These functions enable Enterprise applications to perform the following operations:

  - Implement added data protection using public-key asymmetric cryptography
  - Create public and private keys and digital signatures
  - Perform asymmetric encryption and decryption
  - Use cryptographic hashing for digital signing and data verification and validation

  For more information, see Section 6.5, “MySQL Enterprise Encryption”.

  MySQL Enterprise Edition now includes MySQL Enterprise Firewall, an application-level firewall that enables database administrators to permit or deny SQL statement execution based on matching against allowlists of accepted statement patterns. This helps harden MySQL Server against attacks such as SQL injection or attempts to exploit applications by using them outside of their legitimate query workload characteristics. For more information, see Section 6.4.5, “MySQL Enterprise Firewall”.

- **Changes to server defaults.** Beginning with MySQL 5.6.6, several MySQL Server parameter defaults differ from the defaults in previous releases. The motivation for these changes is to provide better out-of-box performance and to reduce the need for database administrators to change settings manually. For more information, see Section 5.1.2.1, “Changes to Server Defaults”.

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

  - You can create `FULLTEXT` indexes on `InnoDB` tables, and query them using the `MATCH() ... AGAINST` syntax. This feature includes a new proximity search operator (`@`) and several new configuration options and `INFORMATION_SCHEMA` tables: See Section 14.6.2.3, “InnoDB FULLTEXT Indexes” for more information.
  - Several `ALTER TABLE` operations can be performed without copying the table, without blocking inserts, updates, and deletes to the table, or both. These enhancements are known collectively as online DDL. See Section 14.13, “InnoDB and Online DDL” for details.

- `InnoDB` now supports the `DATA DIRECTORY='directory'` clause of the `CREATE TABLE` statement, which permits creating tables outside of the data directory. This enhancement provides the flexibility to createtables in locations that better suit your server environment. For example, you can place busy tables on an SSD device, or large tables on a high-capacity HDD device.

  For more information, see Section 14.6.1.2, “Creating Tables Externally”.

- `InnoDB` now supports the notion of “transportable tablespaces”, allowing file-per-table tablespaces (`.ibd` files) to be exported from a running MySQL instance and imported into another running instance without inconsistencies or mismatches caused by buffered data, in-progress transactions, and internal bookkeeping details such as the space ID and LSN.

  The `FOR EXPORT` clause of the `FLUSH TABLE` command writes any unsaved changes from `InnoDB` memory buffers to the `.ibd` file. After copying the `.ibd` file and a separate metadata file to the other server, the `DISCARD TABLESPACE` and `IMPORT TABLESPACE` clauses of the `ALTER TABLE` statement are used to bring the table data into a different MySQL instance.

  This enhancement provides the flexibility to move tables that reside in file-per-table tablespaces around to better suit your server environment. For example, you could move busy tables to an SSD device, or move large tables to a high-capacity HDD device. For more information, see Section 14.6.1.3, “Importing InnoDB Tables”.

- You can now set the `InnoDB` page size for uncompressed tables to 8KB or 4KB, as an alternative to the default 16KB. This setting is controlled by the `innodb_page_size` configuration option. You specify the size when creating the MySQL instance. All `InnoDB` tablespaces within an instance share the same page size. Smaller page sizes can help to avoid redundant or inefficient I/O for certain combinations of workload and storage devices, particularly SSD devices with small block sizes.

- Improvements to the algorithms for adaptive flushing make I/O operations more efficient and consistent under a variety of workloads. The new algorithm and default configuration values are expected to improve performance and concurrency for most users. Advanced users can fine-tune their I/O responsiveness through several configuration options. See Section 14.8.3.4, “Configuring Buffer Pool Flushing” for details.

- You can code MySQL applications that access `InnoDB` tables through a NoSQL-style API. This feature uses the popular `memcached` daemon to relay requests such as `ADD`, `SET`, and `GET` for key-value pairs. These simple operations to store and retrieve data avoid the SQL overhead such as parsing and constructing a query execution plan. You can access the same data through the NoSQL API and SQL. For example, you might use the NoSQL API for fast updates and lookups, and SQL for complex queries and compatibility with existing applications. See Section 14.20, “InnoDB memcached Plugin” for details.

- Optimizer statistics for `InnoDB` tables are gathered at more predictable intervals and can persist across server restarts, for improved plan stability. You can also control the amount of sampling done for `InnoDB` indexes, to make the optimizer statistics more accurate and improve the query execution plan. See Section 14.8.11.1, “Configuring Persistent Optimizer Statistics Parameters” for details.

- New optimizations apply to read-only transactions, improving performance and concurrency for ad-hoc queries and report-generating applications. These optimizations are applied automatically when practical, or you can specify `START TRANSACTION READ ONLY` to ensure the transaction is read-only. See Section 8.5.3, “Optimizing InnoDB Read-Only Transactions” for details.

- You can move the `InnoDB` undo log out of the system tablespace into one or more separate tablespaces. The I/O patterns for the undo log make these new tablespaces good candidates to move to SSD storage, while keeping the system tablespace on hard disk storage. For details, see Section 14.6.3.3, “Undo Tablespaces”.

- You can improve the efficiency of the InnoDB checksum feature by specifying the configuration option innodb_checksum_algorithm=crc32, which turns on a faster checksum algorithm. This option replaces the innodb_checksums option. Data written using the old checksum algorithm (option value innodb) is fully upward-compatible; tablespaces modified using the new checksum algorithm (option value crc32) cannot be downgraded to an earlier version of MySQL that does not support the innodb_checksum_algorithm option.

- The InnoDB redo log files now have a maximum combined size of 512GB, increased from 4GB. You can specify the larger values through the innodb_log_file_size option. The startup behavior now automatically handles the situation where the size of the existing redo log files does not match the size specified by innodb_log_file_size and innodb_log_files_in_group.

- The --innodb-read-only option lets you run a MySQL server in read-only mode. You can access InnoDB tables on read-only media such as a DVD or CD, or set up a data warehouse with multiple instances all sharing the same data directory. See Section 14.8.2, “Configuring InnoDB for Read-Only Operation” for usage details.

- A new configuration option, innodb_compression_level, allows you to select a compression level for InnoDB compressed tables, from the familiar range of 0-9 used by zlib. You can also control whether compressed pages in the buffer pool are stored in the redo log when an update operation causes pages to be compressed again. This behavior is controlled by the innodb_log_compressed_pages configuration option.

- Data blocks in an InnoDB compressed table contain a certain amount of empty space (padding) to allow DML operations to modify the row data without re-compressing the new values. Too much padding can increase the chance of a compression failure, requiring a page split, when the data does need to be re-compressed after extensive changes. The amount of padding can now be adjusted dynamically, so that DBAs can reduce the rate of compression failures without re-creating the entire table with new parameters, or re-creating the entire instance with a different page size. The associated new configuration options are innodb_compression_failure_threshold_pct, innodb_compression_pad_pct_max.

- Several new InnoDB-related INFORMATION_SCHEMA tables provide information about the InnoDB buffer pool, metadata about tables, indexes, and foreign keys from the InnoDB data dictionary, and low-level information about performance metrics that complements the information from the Performance Schema tables.

- To ease the memory load on systems with huge numbers of tables, InnoDB now frees up the memory associated with an opened table using an LRU algorithm to select tables that have gone the longest without being accessed. To reserve more memory to hold metadata for open InnoDB tables, increase the value of the table_definition_cache configuration option. InnoDB treats this value as a “soft limit” for the number of open table instances in the InnoDB data dictionary cache. For additional information, refer to the table_definition_cache documentation.

- InnoDB has several internal performance enhancements, including reducing contention by splitting the kernel mutex, moving flushing operations from the main thread to a separate thread, enabling multiple purge threads, and reducing contention for the buffer pool on large-memory systems.

- InnoDB uses a new, faster algorithm to detect deadlocks. Information about all InnoDB deadlocks can be written to the MySQL server error log, to help diagnose application issues.

- To avoid a lengthy warmup period after restarting the server, particularly for instances with large InnoDB buffer pools, you can reload pages into the buffer pool immediately after a restart. MySQL can dump a compact data file at shutdown, then consult that data file to find the pages to reload on the next restart. You can also manually dump or reload the buffer pool at any time, for example during benchmarking or after complex report-generation queries. See Section 14.8.3.5, “Saving and Restoring the Buffer Pool State” for details.

- As of MySQL 5.6.16, `innochecksum` provides support for files greater than 2GB in size. Previously, `innochecksum` only supported files up to 2GB in size.
- As of MySQL 5.6.16, new global configuration parameters, `innodb_status_output` and `innodb_status_output_locks`, allow you to dynamically enable and disable the standard `InnoDB` Monitor and `InnoDB` Lock Monitor for periodic output. Enabling and disabling monitors for periodic output by creating and dropping specially named tables is deprecated and may be removed in a future release. For additional information, see Section 14.17, “InnoDB Monitors”.
- As of MySQL 5.6.17, Online DDL support is extended to the following operations for regular and partitioned `InnoDB` tables:
  - `OPTIMIZE TABLE`
  - `ALTER TABLE ... FORCE`
  - `ALTER TABLE ... ENGINE=INNODB` (when run on an `InnoDB` table)

  Online DDL support reduces table rebuild time and permits concurrent DML. See Section 14.13, “InnoDB and Online DDL”.
- As of MySQL 5.6.42, the zlib library version bundled with MySQL was raised from version 1.2.3 to version 1.2.11. MySQL implements compression with the help of the zlib library.

  If you use `InnoDB` compressed tables, see Section 2.11.3, “Changes in MySQL 5.6” for related upgrade implications.

- **Partitioning.** These table-partitioning enhancements were added:
  - The maximum number of partitions is increased to 8192. This number includes all partitions and all subpartitions of the table.
  - It is now possible to exchange a partition of a partitioned table or a subpartition of a subpartitioned table with a nonpartitioned table that otherwise has the same structure using the `ALTER TABLE ... EXCHANGE PARTITION` statement. This can be used, for example, to import and export partitions. For more information and examples, see Section 19.3.3, “Exchanging Partitions and Subpartitions with Tables”.
  - Explicit selection of one or more partitions or subpartitions is now supported for queries, as well as for many data modification statements, that act on partitioned tables. For example, assume a table `t` with some integer column `c` has 4 partitions named `p0`, `p1`, `p2`, and `p3`. Then the query `SELECT * FROM t PARTITION (p0, p1) WHERE c < 5` returns only those rows from partitions `p0` and `p1` for which `c` is less than 5.

    The following statements support explicit partition selection:
    - `SELECT`
    - `DELETE`
    - `INSERT`
    - `REPLACE`
    - `UPDATE`
    - `LOAD DATA`.

- `LOAD XML`.

  For syntax, see the descriptions of the individual statements. For additional information and examples, see Section 19.5, “Partition Selection”.

- Partition lock pruning greatly improves performance of many DML and DDL statements acting on tables with many partitions by helping to eliminate locks on partitions that are not affected by these statements. Such statements include many `SELECT`, `SELECT ... PARTITION`, `UPDATE`, `REPLACE`, `INSERT`, as well as many other statements. For more information, including a complete listing of the statements whose performance has thus been improved, see Section 19.6.4, “Partitioning and Locking”.

- **Performance Schema.** The Performance Schema includes several new features:

  - Instrumentation for table input and output. Instrumented operations include row-level accesses to persistent base tables or temporary tables. Operations that affect rows are fetch, insert, update, and delete.

  - Event filtering by table, based on schema and/or table names.

  - Event filtering by thread. More information is collected for threads.

  - Summary tables for table and index I/O, and for table locks.

  - Instrumentation for statements and stages within statements.

  - Configuration of instruments and consumers at server startup, which previously was possible only at runtime.

- **MySQL NDB Cluster.** MySQL NDB Cluster is released as a separate product; the most recent GA releases are based on MySQL 5.6 and use version 7.3 of the `NDB` storage engine. Clustering support is not available in mainline MySQL Server 5.6 releases. For more information about MySQL NDB Cluster 7.3, see Chapter 18, *MySQL NDB Cluster 7.3 and NDB Cluster 7.4*. The latest current development version is MySQL NDB Cluster 7.4, based on version 7.4 of the `NDB` storage engine and MySQL Server 5.6. MySQL NDB Cluster 7.4 is currently available for testing and evaluation. The most recent MySQL NDB Cluster 7.4 release can be obtained from https://dev.mysql.com/downloads/cluster/.

  For more information and an overview of improvements made in MySQL NDB Cluster 7.4, see Section 18.1.4.2, “What is New in NDB Cluster 7.4”.

  MySQL NDB Cluster 7.2, the previous GA release, is based on MySQL Server 5.5, although we recommend that new deployments use MySQL NDB Cluster 7.3.

- **Replication and logging.** These replication enhancements were added:

  - MySQL now supports transaction-based replication using *global transaction identifiers* (also known as “GTIDs”). This makes it possible to identify and track each transaction when it is committed on the originating server and as it is applied by any replicas.

    Enabling of GTIDs in a replication setup is done primarily using the new `gtid_mode` and `enforce_gtid_consistency` system variables. For information about additional options and variables introduced in support of GTIDs, see Section 17.1.4.5, “Global Transaction ID Options and Variables”.

    When using GTIDs it is not necessary to refer to log files or positions within those files when starting a new replica or failing over to a new source, which greatly simplifies these tasks. For

more information about provisioning servers for GTID replication with or without referring to binary log files, see Section 17.1.3.3, “Using GTIDs for Failover and Scaleout”.

GTID-based replication is completely transaction-based, which makes it simple to check the consistency of sources and replicas. If all transactions committed on a given source are also committed on a given replica, consistency between the two servers is guaranteed.

For more complete information about the implementation and use of GTIDs in MySQL Replication, see Section 17.1.3, “Replication with Global Transaction Identifiers”.

- MySQL row-based replication now supports row image control. By logging only those columns required for uniquely identifying and executing changes on each row (as opposed to all columns) for each row change, it is possible to save disk space, network resources, and memory usage. You can determine whether full or minimal rows are logged by setting the `binlog_row_image` server system variable to one of the values `minimal` (log required columns only), `full` (log all columns), or `noblob` (log all columns except for unneeded `BLOB` or `TEXT` columns). See System Variables Used with Binary Logging, for more information.

- Binary logs written and read by the MySQL Server are now crash-safe, because only complete events (or transactions) are logged or read back. By default, the server logs the length of the event as well as the event itself and uses this information to verify that the event was written correctly. You can also cause the server to write checksums for the events using CRC32 checksums by setting the `binlog_checksum` system variable. To cause the server to read checksums from the binary log, use the `master_verify_checksum` system variable. The `--slave-sql-verify-checksum` system variable causes the replica SQL thread to read checksums from the relay log.

- MySQL now supports logging of source connection information and of replica relay log information to tables as well as files. Use of these tables can be controlled independently, by the `master_info_repository` and `relay_log_info_repository` system variable system variables. Setting `master_info_repository` to `TABLE` causes connection information logging to the `slave_master_info` table. Setting `relay_log_info_repository` to `TABLE` causes relay log information logging to the `slave_relay_log_info` table. Both tables are created automatically in the `mysql` system database.

  In order for replication to be resilient to unexpected halts, the `slave_master_info` and `slave_relay_log_info` tables must each use a transactional storage engine, and beginning with MySQL 5.6.6, these tables are created using `InnoDB` for this reason. (Bug #13538891) If you are using a previous MySQL 5.6 release in which both of these tables use `MyISAM`, this means that, prior to starting replication, you must convert both of them to a transactional storage engine (such as `InnoDB`) if you wish for replication to be resilient to unexpected halts. You can do this in such cases by means of the appropriate `ALTER TABLE ... ENGINE=...` statements. You should *not* attempt to change the storage engine used by either of these tables while replication is actually running.

  See Section 17.3.2, “Handling an Unexpected Halt of a Replica Server”, for more information.

- `mysqlbinlog` now has the capability to back up a binary log in its original binary format. When invoked with the `--read-from-remote-server` and `--raw` options, `mysqlbinlog` connects to a server, requests the log files, and writes output files in the same format as the originals. See Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”.

- MySQL now supports delayed replication such that a replica server deliberately lags behind the source by at least a specified amount of time. The default delay is 0 seconds. Use the new `MASTER_DELAY` option for `CHANGE MASTER TO` to set the delay.

  Delayed replication can be used for purposes such as protecting against user mistakes on the source (a DBA can roll back a delayed replica to the time just before the disaster) or testing how the system behaves when there is a lag. See Section 17.3.10, “Delayed Replication”.

- A replica having multiple network interfaces can now be caused to use only one of these (to the exclusion of the others) by using the `MASTER_BIND` option when issuing a `CHANGE MASTER TO` statement.

- The `log_bin_basename` system variable has been added. This variable contains the complete filename and path to the binary log file. Whereas the `log_bin` system variable shows only whether or not binary logging is enabled, `log_bin_basename` reflects the name set with the `--log-bin` server option.

  Similarly, the `relay_log_basename` system variable shows the filename and complete path to the relay log file.

- MySQL Replication now supports parallel execution of transactions with multithreading on the replica. When parallel execution is enabled, the replica SQL thread acts as the coordinator for a number of replica worker threads as determined by the value of the `slave_parallel_workers` server system variable. The current implementation of multithreading on the replica assumes that data and updates are partitioned on a per-database basis, and that updates within a given database occur in the same relative order as they do on the source. However, it is not necessary to coordinate transactions between different databases. Transactions can then also be distributed per database, which means that a worker thread on the replica can process successive transactions on a given database without waiting for updates to other databases to complete.

  Since transactions on different databases can occur in a different order on the replica than on the source, simply checking for the most recently executed transaction is not a guarantee that all previous transactions on the source have been executed on the replica. This has implications for logging and recovery when using a multithreaded replica. For information about how to interpret binary logging information when using multithreading on the replica, see Section 13.7.5.35, “SHOW SLAVE STATUS Statement”.

- **Optimizer enhancements.** These query optimizer improvements were implemented:

  - The optimizer now more efficiently handles queries (and subqueries) of the following form:

    ```
    SELECT ... FROM single_table ... ORDER BY non_index_column [DESC] LIMIT [M,]N;
    ```

    That type of query is common in web applications that display only a few rows from a larger result set. For example:

    ```
    SELECT col1, ... FROM t1 ... ORDER BY name LIMIT 10;
    SELECT col1, ... FROM t1 ... ORDER BY RAND() LIMIT 15;
    ```

    The sort buffer has a size of `sort_buffer_size`. If the sort elements for *`N`* rows are small enough to fit in the sort buffer (*`M`*+*`N`* rows if *`M`* was specified), the server can avoid using a merge file and perform the sort entirely in memory. For details, see Section 8.2.1.16, “LIMIT Query Optimization”.

  - The optimizer implements Disk-Sweep Multi-Range Read. Reading rows using a range scan on a secondary index can result in many random disk accesses to the base table when the table is large and not stored in the storage engine's cache. With the Disk-Sweep Multi-Range Read (MRR) optimization, MySQL tries to reduce the number of random disk access for range scans by first scanning the index only and collecting the keys for the relevant rows. Then the keys are sorted and finally the rows are retrieved from the base table using the order of the primary key. The motivation for Disk-sweep MRR is to reduce the number of random disk accesses and instead achieve a more sequential scan of the base table data. For more information, see Section 8.2.1.10, “Multi-Range Read Optimization”.

  - The optimizer implements Index Condition Pushdown (ICP), an optimization for the case where MySQL retrieves rows from a table using an index. Without ICP, the storage engine traverses the index to locate rows in the base table and returns them to the MySQL server which evaluates the `WHERE` condition for the rows. With ICP enabled, and if parts of the `WHERE` condition can be

evaluated by using only fields from the index, the MySQL server pushes this part of the `WHERE` condition down to the storage engine. The storage engine then evaluates the pushed index condition by using the index entry and only if this is satisfied is base row be read. ICP can reduce the number of accesses the storage engine has to do against the base table and the number of accesses the MySQL server has to do against the storage engine. For more information, see Section 8.2.1.5, “Index Condition Pushdown Optimization”.

- The `EXPLAIN` statement now provides execution plan information for `DELETE`, `INSERT`, `REPLACE`, and `UPDATE` statements. Previously, `EXPLAIN` provided information only for `SELECT` statements. In addition, the `EXPLAIN` statement now can produce output in JSON format. See Section 13.8.2, “EXPLAIN Statement”.

- The optimizer more efficiently handles subqueries in the `FROM` clause (that is, derived tables). Materialization of subqueries in the `FROM` clause is postponed until their contents are needed during query execution, which improves performance. In addition, during query execution, the optimizer may add an index to a derived table to speed up row retrieval from it. For more information, see Section 8.2.2.4, “Optimizing Derived Tables”.

- The optimizer uses semijoin and materialization strategies to optimize subquery execution. See Section 8.2.2.1, “Optimizing Subqueries with Semijoin Transformations”, and Section 8.2.2.2, “Optimizing Subqueries with Materialization”.

- A Batched Key Access (BKA) join algorithm is now available that uses both index access to the joined table and a join buffer. The BKA algorithm supports inner join, outer join, and semijoin operations, including nested outer joins and nested semijoins. Benefits of BKA include improved join performance due to more efficient table scanning. For more information, see Section 8.2.1.11, “Block Nested-Loop and Batched Key Access Joins”.

- The optimizer now has a tracing capability, primarily for use by developers. The interface is provided by a set of `optimizer_trace_`*`xxx`* system variables and the `INFORMATION_SCHEMA.OPTIMIZER_TRACE` table. For details, see MySQL Internals: Tracing the Optimizer.

- **Condition handling.** MySQL now supports the `GET DIAGNOSTICS` statement. `GET DIAGNOSTICS` provides applications a standardized way to obtain information from the diagnostics area, such as whether the previous SQL statement produced an exception and what it was. For more information, see Section 13.6.7.3, “GET DIAGNOSTICS Statement”.

  In addition, several deficiencies in condition handler processing rules were corrected so that MySQL behavior is more like standard SQL:

  - Block scope is used in determining which handler to select. Previously, a stored program was treated as having a single scope for handler selection.

  - Condition precedence is more accurately resolved.

  - Diagnostics area clearing has changed. Bug #55843 caused handled conditions to be cleared from the diagnostics area before activating the handler. This made condition information unavailable within the handler. Now condition information is available to the handler, which can inspect it with the `GET DIAGNOSTICS` statement. The condition information is cleared when the handler exits, if it has not already been cleared during handler execution.

  - Previously, handlers were activated as soon as a condition occurred. Now they are not activated until the statement in which the condition occurred finishes execution, at which point the most appropriate handler is chosen. This can make a difference for statements that raise multiple conditions, if a condition raised later during statement execution has higher precedence than an earlier condition and there are handlers in the same scope for both conditions. Previously, the handler for the first condition raised would be chosen, even if it had a lower precedence than other handlers. Now the handler for the condition with highest precedence is chosen, even if it is not the first condition raised by the statement.

For more information, see Section 13.6.7.6, “Scope Rules for Handlers”.

- **Data types.** These data type changes have been implemented:
  - MySQL now permits fractional seconds for `TIME`, `DATETIME`, and `TIMESTAMP` values, with up to microseconds (6 digits) precision. See Section 11.2.7, “Fractional Seconds in Time Values”.
  - Previously, at most one `TIMESTAMP` column per table could be automatically initialized or updated to the current date and time. This restriction has been lifted. Any `TIMESTAMP` column definition can have any combination of `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` clauses. In addition, these clauses now can be used with `DATETIME` column definitions. For more information, see Section 11.2.6, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”.
  - In MySQL, the `TIMESTAMP` data type differs in nonstandard ways from other data types in terms of default value and assignment of automatic initialization and update attributes. These behaviors remain the default but now are deprecated, and can be turned off by enabling the `explicit_defaults_for_timestamp` system variable at server startup. See Section 11.2.6, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”, and Section 5.1.7, “Server System Variables”.
- **Host cache.** MySQL now provides more information about the causes of errors that occur when clients connect to the server, as well as improved access to the host cache, which contains client IP address and host name information and is used to avoid DNS lookups. These changes have been implemented:
  - New `Connection_errors_xxx` status variables provide information about connection errors that do not apply to specific client IP addresses.
  - Counters have been added to the host cache to track errors that do apply to specific IP addresses, and a new `host_cache` Performance Schema table exposes the contents of the host cache so that it can be examined using `SELECT` statements. Access to host cache contents makes it possible to answer questions such as how many hosts are cached, what kinds of connection errors are occurring for which hosts, or how close host error counts are to reaching the `max_connect_errors` system variable limit.
  - The host cache size now is configurable using the `host_cache_size` system variable.

  For more information, see Section 5.1.11.2, “DNS Lookups and the Host Cache”, and Section 22.12.10.1, “The host_cache Table”.
- **OpenGIS.** The OpenGIS specification defines functions that test the relationship between two geometry values. MySQL originally implemented these functions such that they used object bounding rectangles and returned the same result as the corresponding MBR-based functions. Corresponding versions are now available that use precise object shapes. These versions are named with an `ST_` prefix. For example, `Contains()` uses object bounding rectangles, whereas `ST_Contains()` uses object shapes. For more information, see Section 12.17.9, “Functions That Test Spatial Relations Between Geometry Objects”.

Features Deprecated in MySQL 5.6

The following features are deprecated in MySQL 5.6; you should expect them to be removed in a future series. Where alternatives are shown, applications should be updated to use them.

For applications that use features deprecated in MySQL 5.6 that have been removed in a higher MySQL series, statements may fail when replicated from a MySQL 5.6 source to a higher-series replica, or may have different effects on source and replica. To avoid such problems, applications that use features deprecated in 5.6 should be revised to avoid them and use alternatives when possible.

- The `ERROR_FOR_DIVISION_BY_ZERO`, `NO_ZERO_DATE`, and `NO_ZERO_IN_DATE` SQL modes are deprecated and setting the `sql_mode` value to include any of them generates a warning. In MySQL 5.7, these modes do nothing. Instead, their effects are included in the effects of strict SQL mode (`STRICT_ALL_TABLES` or `STRICT_TRANS_TABLES`). The motivation for the change in MySQL 5.7 is to reduce the number of SQL modes with an effect dependent on strict mode and make them part of strict mode itself.

  To make advance preparation for an upgrade to MySQL 5.7, see SQL Mode Changes in MySQL 5.7. That discussion provides guidelines to assess whether your applications are affected by the SQL mode changes in MySQL 5.7.

- Relying on implicit `GROUP BY` sorting in MySQL 5.6 is deprecated. To achieve a specific sort order of grouped results, it is preferable to use an explicit `ORDER BY` clause. `GROUP BY` sorting is a MySQL extension that may change in a future release; for example, to make it possible for the optimizer to order groupings in whatever manner it deems most efficient and to avoid the sorting overhead.

- Pre-4.1 passwords and the `mysql_old_password` authentication plugin. Passwords stored in the older hash format used before MySQL 4.1 are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords and the `mysql_old_password` authentication plugin are now deprecated. To prevent connections using accounts that have pre-4.1 password hashes, the `secure_auth` system variable is now enabled by default. (To permit connections for accounts that have such password hashes, start the server with `--secure_auth=0`. However, because pre-4.1 passwords are deprecated, disabling `secure_auth` is also deprecated.)

  DBAs are advised to convert accounts that use the `mysql_old_password` authentication plugin to use `mysql_native_password` instead. For account upgrade instructions, see Section 6.4.1.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”.

  The `OLD_PASSWORD()` function generates pre-4.1 password hashes, as does `PASSWORD()` if the `old_passwords` system variable is set to 1. `OLD_PASSWORD()` and `old_passwords=1` are deprecated.

- The `--skip-innodb` option and its synonyms (`--innodb=OFF`, `--disable-innodb`, and so forth).

- The `innodb_locks_unsafe_for_binlog` system variable.

- The `date_format`, `datetime_format`, and `time_format` system variables, which are unused.

- The `have_profiling`, `profiling`, and `profiling_history_size` system variables.

- The `innodb_use_sys_malloc` and `innodb_additional_mem_pool_size` system variables.

- The `timed_mutexes` system variable. It does nothing and has no effect.

- The `--language` option. Use the `--lc-messages-dir` and `--lc-messages` options instead.

- The `IGNORE` clause for `ALTER TABLE`. `ALTER IGNORE TABLE` causes problems for replication, prevents online `ALTER TABLE` for unique index creation, and causes problems with foreign keys (rows removed in the parent table).

- The `msql2mysql`, `mysql_convert_table_format`, `mysql_find_rows`, `mysql_fix_extensions`, `mysql_setpermission`, `mysql_waitpid`, `mysql_zap`, `mysqlaccess`, and `mysqlbug` utilities.

- The `mysqlhotcopy` utility. Alternatives include `mysqldump` and MySQL Enterprise Backup.

Features Removed in MySQL 5.6

The following items are obsolete and have been removed in MySQL 5.6. Where alternatives are shown, applications should be updated to use them.

For MySQL 5.5 applications that use features removed in MySQL 5.6, statements may fail when replicated from a MySQL 5.5 source to a MySQL 5.6 replica, or may have different effects on source and replica. To avoid such problems, applications that use features removed in MySQL 5.6 should be revised to avoid them and use alternatives when possible.

- The `--log` server option and the `log` system variable. Instead, use the `general_log` system variable to enable the general query log and the `general_log_file` system variable to set the general query log file name.
- The the `log_slow_queries` system variable. Instead, use the `slow_query_log` system variable to enable the slow query log and the `slow_query_log_file` system variable to set the slow query log file name.
- The `--one-thread` server option. Use `--thread_handling=no-threads` instead.
- The `--safe-mode` server option.
- The `--skip-thread-priority` server option.
- The `--table-cache` server option. Use the `table_open_cache` system variable instead.
- The `--init-rpl-role` and `--rpl-recovery-rank` options, the `rpl_recovery_rank` system variable, and the `Rpl_status` status variable.
- The `engine_condition_pushdown` system variable. Use the `engine_condition_pushdown` flag of the `optimizer_switch` variable instead.
- The `have_csv`, `have_innodb`, `have_ndbcluster`, and `have_partitioning` system variables. Use `SHOW PLUGINS` or query the `PLUGINS` table in the `INFORMATION_SCHEMA` database instead.
- The `sql_big_tables` system variable. Use `big_tables` instead.
- The `sql_low_priority_updates` system variable. Use `low_priority_updates` instead.
- The `sql_max_join_size` system variable. Use `max_join_size` instead.
- The `max_long_data_size` system variable. Use `max_allowed_packet` instead.
- The `FLUSH MASTER` and `FLUSH SLAVE` statements. Use the `RESET MASTER` and `RESET SLAVE` statements instead.
- The `SLAVE START` and `SLAVE STOP` statements. Use The `START SLAVE` and `STOP SLAVE` statements.
- The `SHOW AUTHORS` and `SHOW CONTRIBUTORS` statements.
- The `OPTION` and `ONE_SHOT` modifiers for the `SET` statement.
- It is explicitly disallowed to assign the value `DEFAULT` to stored procedure or function parameters or stored program local variables (for example with a `SET var_name = DEFAULT` statement). It remains permissible to assign `DEFAULT` to system variables, as before.
- Most `SHOW ENGINE INNODB MUTEX` output is removed in 5.6.14. `SHOW ENGINE INNODB MUTEX` output is removed entirely in MySQL 5.7.2. Comparable information can be generated by creating views on Performance Schema tables.

1.4 Server and Status Variables and Options Added, Deprecated, or Removed in MySQL 5.6

- Options and Variables Introduced in MySQL 5.6
- Options and Variables Deprecated in MySQL 5.6
- Options and Variables Removed in MySQL 5.6

This section lists server variables, status variables, and options that were added for the first time, have been deprecated, or have been removed in MySQL 5.6.

Options and Variables Introduced in MySQL 5.6

The following system variables, status variables, and server options have been added in MySQL 5.6.

- `Audit_log_current_size`: Audit log file current size. Added in MySQL 5.6.20.
- `Audit_log_event_max_drop_size`: Size of largest dropped audited event. Added in MySQL 5.6.20.
- `Audit_log_events`: Number of handled audited events. Added in MySQL 5.6.20.
- `Audit_log_events_filtered`: Number of filtered audited events. Added in MySQL 5.6.20.
- `Audit_log_events_lost`: Number of dropped audited events. Added in MySQL 5.6.20.
- `Audit_log_events_written`: Number of written audited events. Added in MySQL 5.6.20.
- `Audit_log_total_size`: Combined size of written audited events. Added in MySQL 5.6.20.
- `Audit_log_write_waits`: Number of write-delayed audited events. Added in MySQL 5.6.20.
- `Binlog_stmt_cache_disk_use`: Number of nontransactional statements that used temporary file instead of binary log statement cache. Added in MySQL 5.6.1.
- `Binlog_stmt_cache_use`: Number of statements that used temporary binary log statement cache. Added in MySQL 5.6.1.
- `Com_alter_user`: Count of ALTER USER statements. Added in MySQL 5.6.6.
- `Com_get_diagnostics`: Count of GET DIAGNOSTICS statements. Added in MySQL 5.6.4.
- `Connection_control_delay_generated`: How many times server delayed connection request. Added in MySQL 5.6.35.
- `Connection_errors_accept`: Number of errors calling accept on listening port. Added in MySQL 5.6.5.
- `Connection_errors_internal`: Number of connections refused due to internal errors. Added in MySQL 5.6.5.
- `Connection_errors_max_connections`: Number of connections refused due to max_connections limit. Added in MySQL 5.6.5.
- `Connection_errors_peer_address`: Number of errors searching for connection client IP addresses. Added in MySQL 5.6.5.
- `Connection_errors_select`: Number of errors calling select/poll on listening port. Added in MySQL 5.6.5.
- `Connection_errors_tcpwrap`: Number of connections refused by libwrap. Added in MySQL 5.6.5.
- `Firewall_access_denied`: Number of statements rejected by MySQL Enterprise Firewall. Added in MySQL 5.6.24.
- `Firewall_access_granted`: Number of statements accepted by MySQL Enterprise Firewall. Added in MySQL 5.6.24.
- `Firewall_cached_entries`: Number of statements recorded by MySQL Enterprise Firewall. Added in MySQL 5.6.24.
- `Handler_external_lock`: Number of locks started while statement executed. Added in MySQL 5.6.2.

- `Handler_mrr_init`: Number of times storage engine MRR implementation is used for table access. Added in MySQL 5.6.1.
- `Handler_read_last`: Number of requests to read last index entry. Added in MySQL 5.6.1.
- `Innodb_available_undo_logs`: Total number of InnoDB rollback segments; different from innodb_rollback_segments, which displays number of active rollback segments. Added in MySQL 5.6.5.
- `Innodb_buffer_pool_bytes_data`: Number of bytes containing data (dirty or clean) in buffer pool. Added in MySQL 5.6.10.
- `Innodb_buffer_pool_bytes_dirty`: Number of bytes currently dirty in buffer pool. Added in MySQL 5.6.10.
- `Innodb_buffer_pool_dump_status`: Display status of buffer pool recording operation triggered by innodb_buffer_pool_dump_at_shutdown or innodb_buffer_pool_dump_now. Added in MySQL 5.6.3.
- `Innodb_buffer_pool_load_status`: Display status of buffer pool warmup operation triggered by innodb_buffer_pool_load_at_startup or innodb_buffer_pool_load_now. Added in MySQL 5.6.3.
- `Innodb_num_open_files`: Number of physical files currently opened by InnoDB. Added in MySQL 5.6.2.
- `Last_query_partial_plans`: Number of iterations in execution plan construction for previous statement. Added in MySQL 5.6.5.
- `Performance_schema_accounts_lost`: How many accounts table rows could not be added. Added in MySQL 5.6.3.
- `Performance_schema_digest_lost`: How many digests could not be instrumented. Added in MySQL 5.6.5.
- `Performance_schema_hosts_lost`: How many hosts table rows could not be added. Added in MySQL 5.6.3.
- `Performance_schema_session_connect_attrs_lost`: Number of connections for which attribute strings could not be created. Added in MySQL 5.6.6.
- `Performance_schema_socket_classes_lost`: How many socket instruments could not be loaded. Added in MySQL 5.6.3.
- `Performance_schema_socket_instances_lost`: How many socket instrument instances could not be created. Added in MySQL 5.6.3.
- `Performance_schema_stage_classes_lost`: How many stage instruments could not be loaded. Added in MySQL 5.6.3.
- `Performance_schema_statement_classes_lost`: How many statement instruments could not be loaded. Added in MySQL 5.6.3.
- `Performance_schema_users_lost`: How many users table rows could not be added. Added in MySQL 5.6.3.
- `Rsa_public_key`: sha256_password authentication plugin RSA public key value. Added in MySQL 5.6.6.
- `Slave_last_heartbeat`: Shows when latest heartbeat signal was received, in TIMESTAMP format. Added in MySQL 5.6.1.
- `Slave_rows_last_search_algorithm_used`: Search algorithm most recently used by this replica to locate rows for row-based replication (index, table, or hash scan). Added in MySQL 5.6.6.
- `Ssl_server_not_after`: SSL certificate last valid date. Added in MySQL 5.6.3.

- `Ssl_server_not_before`: SSL certificate first valid date. Added in MySQL 5.6.3.
- `Table_open_cache_hits`: Number of hits for open tables cache lookups. Added in MySQL 5.6.6.
- `Table_open_cache_misses`: Number of misses for open tables cache lookups. Added in MySQL 5.6.6.
- `Table_open_cache_overflows`: Number of overflows for open tables cache. Added in MySQL 5.6.6.
- `audit-log`: Whether to activate audit log plugin. Added in MySQL 5.6.10.
- `audit_log_buffer_size`: Size of audit log buffer. Added in MySQL 5.6.10.
- `audit_log_connection_policy`: Audit logging policy for connection-related events. Added in MySQL 5.6.20.
- `audit_log_current_session`: Whether to audit current session. Added in MySQL 5.6.20.
- `audit_log_exclude_accounts`: Accounts not to audit. Added in MySQL 5.6.20.
- `audit_log_file`: Name of audit log file. Added in MySQL 5.6.10.
- `audit_log_flush`: Close and reopen audit log file. Added in MySQL 5.6.10.
- `audit_log_format`: Audit log file format. Added in MySQL 5.6.14.
- `audit_log_include_accounts`: Accounts to audit. Added in MySQL 5.6.20.
- `audit_log_policy`: Audit logging policy. Added in MySQL 5.6.10.
- `audit_log_rotate_on_size`: Close and reopen audit log file at this size. Added in MySQL 5.6.10.
- `audit_log_statement_policy`: Audit logging policy for statement-related events. Added in MySQL 5.6.20.
- `audit_log_strategy`: Audit logging strategy. Added in MySQL 5.6.10.
- `authentication_windows_log_level`: Windows authentication plugin logging level. Added in MySQL 5.6.10.
- `authentication_windows_use_principal_name`: Whether to use Windows authentication plugin principal name. Added in MySQL 5.6.10.
- `avoid_temporal_upgrade`: Whether ALTER TABLE should upgrade pre-5.6.4 temporal columns. Added in MySQL 5.6.24.
- `bind_address`: IP address or host name to bind to. Added in MySQL 5.6.1.
- `binlog-checksum`: Enable/disable binary log checksums. Added in MySQL 5.6.2.
- `binlog_checksum`: Enable/disable binary log checksums. Added in MySQL 5.6.2.
- `binlog_error_action`: Controls what happens when server cannot write to binary log. Added in MySQL 5.6.22.
- `binlog_gtid_simple_recovery`: Controls how binary logs are iterated during GTID recovery. Added in MySQL 5.6.23.
- `binlog_max_flush_queue_time`: How long to read transactions before flushing to binary log. Added in MySQL 5.6.6.
- `binlog_order_commits`: Whether to commit in same order as writes to binary log. Added in MySQL 5.6.6.

- binlog_row_image: Use full or minimal images when logging row changes. Added in MySQL 5.6.2.
- binlog_rows_query_log_events: When enabled, enables logging of rows query log events when using row-based logging. Disabled by default. Do not enable when producing logs for pre-5.6 replicas/readers. Added in MySQL 5.6.2.
- binlog_stmt_cache_size: Size of cache to hold nontransactional statements for binary log during transaction. Added in MySQL 5.6.1.
- binlogging_impossible_mode: Deprecated and later removed. Use binlog_error_action instead. Added in MySQL 5.6.20.
- block_encryption_mode: Mode for block-based encryption algorithms. Added in MySQL 5.6.17.
- connection_control_failed_connections_threshold: Consecutive failed connection attempts before delays occur. Added in MySQL 5.6.35.
- connection_control_max_connection_delay: Maximum delay (milliseconds) for server response to failed connection attempts. Added in MySQL 5.6.35.
- connection_control_min_connection_delay: Minimum delay (milliseconds) for server response to failed connection attempts. Added in MySQL 5.6.35.
- core_file: Write core file on server crashes. Added in MySQL 5.6.2.
- daemon_memcached_enable_binlog: . Added in MySQL 5.6.6.
- daemon_memcached_engine_lib_name: Shared library implementing InnoDB memcached plugin. Added in MySQL 5.6.6.
- daemon_memcached_engine_lib_path: Directory which contains shared library implementing InnoDB memcached plugin. Added in MySQL 5.6.6.
- daemon_memcached_option: Space-separated options which are passed to underlying memcached daemon on startup. Added in MySQL 5.6.6.
- daemon_memcached_r_batch_size: Specifies how many memcached read operations to perform before doing COMMIT to start new transaction. Added in MySQL 5.6.6.
- daemon_memcached_w_batch_size: Specifies how many memcached write operations to perform before doing COMMIT to start new transaction. Added in MySQL 5.6.6.
- default-authentication-plugin: Default authentication plugin. Added in MySQL 5.6.6.
- default_tmp_storage_engine: Default storage engine (table type) for TEMPORARY tables. Added in MySQL 5.6.3.
- disable-gtid-unsafe-statements: Obsolete: Replaced by enforce_gtid_consistency in MySQL 5.6.9. Added in MySQL 5.6.5.
- disable_gtid_unsafe_statements: Obsolete: Replaced by enforce_gtid_consistency in MySQL 5.6.9. Added in MySQL 5.6.5.
- disconnect_on_expired_password: Whether server disconnects clients with expired passwords if clients cannot handle such accounts. Added in MySQL 5.6.10.
- end_markers_in_json: Whether optimizer JSON output should add end markers. Added in MySQL 5.6.5.
- enforce_gtid_consistency: Prevents execution of statements that cannot be logged in transactionally safe manner. Added in MySQL 5.6.9.
- eq_range_index_dive_limit: Cutoff for switching from index dives to index statistics. Added in MySQL 5.6.5.

- explicit_defaults_for_timestamp: Whether TIMESTAMP columns are nullable and have DEFAULT NULL. Added in MySQL 5.6.6.
- gtid_done: Obsolete: Replaced by gtid_executed in MySQL 5.6.9. Added in MySQL 5.6.5.
- gtid_executed: Global: All GTIDs in binary log (global) or current transaction (session). Read-only. Added in MySQL 5.6.9.
- gtid_lost: Obsolete: Replaced by gtid_purged in MySQL 5.6.9. Added in MySQL 5.6.5.
- gtid_mode: Controls whether GTID based logging is enabled and what type of transactions logs can contain. Added in MySQL 5.6.5.
- gtid_next: Specifies GTID for next statement to execute; see documentation for details. Added in MySQL 5.6.5.
- gtid_owned: Set of GTIDs owned by this client (session), or by all clients, together with thread ID of owner (global). Read-only. Added in MySQL 5.6.5.
- gtid_purged: Set of all GTIDs that have been purged from binary log. Added in MySQL 5.6.9.
- host_cache_size: Size of host cache. Added in MySQL 5.6.5.
- ignore-db-dir: Treat directory as nondatabase directory. Added in MySQL 5.6.3.
- ignore_db_dirs: Directories treated as nondatabase directories. Added in MySQL 5.6.3.
- innodb_adaptive_flushing_lwm: Low water mark representing percentage of redo log capacity at which adaptive flushing is enabled. Added in MySQL 5.6.6.
- innodb_adaptive_max_sleep_delay: Allows InnoDB to adjust value of innodb_thread_sleep_delay up or down automatically, according to current workload. Added in MySQL 5.6.3.
- innodb_api_bk_commit_interval: How often to auto-commit idle connections which use InnoDB memcached interface, in seconds. Added in MySQL 5.6.7.
- innodb_api_disable_rowlock: . Added in MySQL 5.6.6.
- innodb_api_enable_binlog: Allows use of InnoDB memcached plugin with MySQL binary log. Added in MySQL 5.6.6.
- innodb_api_enable_mdl: Locks table used by InnoDB memcached plugin, so that it cannot be dropped or altered by DDL through SQL interface. Added in MySQL 5.6.6.
- innodb_api_trx_level: Allows control of transaction isolation level on queries processed by memcached interface. Added in MySQL 5.6.6.
- innodb_buffer_pool_dump_at_shutdown: Specifies whether to record pages cached in InnoDB buffer pool when MySQL server is shut down, to shorten warmup process at next restart. Added in MySQL 5.6.3.
- innodb_buffer_pool_dump_now: Immediately record pages cached in InnoDB buffer pool. Added in MySQL 5.6.3.
- innodb_buffer_pool_filename: Specifies file containing list of page numbers produced by innodb_buffer_pool_dump_at_shutdown or innodb_buffer_pool_dump_now. Added in MySQL 5.6.3.
- innodb_buffer_pool_load_abort: Interrupts process of restoring InnoDB buffer pool contents triggered by innodb_buffer_pool_load_at_startup or innodb_buffer_pool_load_now. Added in MySQL 5.6.3.
- innodb_buffer_pool_load_at_startup: Specifies that, on MySQL server startup, InnoDB buffer pool is automatically "warmed up" by loading same pages it held at earlier time. Added in MySQL 5.6.3.

- `innodb_buffer_pool_load_now`: Immediately "warms up" InnoDB buffer pool by loading set of data pages, without waiting for server restart. Added in MySQL 5.6.3.
- `innodb_change_buffer_max_size`: Maximum size for InnoDB change buffer, as percentage of total size of buffer pool. Added in MySQL 5.6.2.
- `innodb_checksum_algorithm`: Specifies how to generate and verify checksum stored in each disk block of each InnoDB tablespace. Added in MySQL 5.6.3.
- `innodb_cmp_per_index_enabled`: Enables per-index compression-related statistics in INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX table. Added in MySQL 5.6.7.
- `innodb_compression_failure_threshold_pct`: Cutoff point at which MySQL begins adding padding within compressed pages to avoid expensive compression failures. Added in MySQL 5.6.7.
- `innodb_compression_level`: Level of zlib compression to use for InnoDB compressed tables and indexes. Added in MySQL 5.6.7.
- `innodb_compression_pad_pct_max`: Maximum percentage that can be reserved as free space within each compressed page, to avoid compression failures when tightly packed data is recompressed. Added in MySQL 5.6.7.
- `innodb_disable_sort_file_cache`: Disable OS file system cache for merge-sort temporary files. Added in MySQL 5.6.4.
- `innodb_fil_make_page_dirty_debug`: Dirties first page of specified tablespace. Added in MySQL 5.6.17.
- `innodb_flush_log_at_timeout`: Write and flush logs every N seconds. Added in MySQL 5.6.6.
- `innodb_flush_neighbors`: Specifies whether or not flushing page from InnoDB buffer pool also flushes other dirty pages in same extent. Added in MySQL 5.6.3.
- `innodb_flushing_avg_loops`: Number of iterations for which InnoDB keeps previously calculated snapshot of flushing state, controlling how quickly adaptive flushing responds to changing workloads. Added in MySQL 5.6.6.
- `innodb_force_load_corrupted`: Lets InnoDB load tables at startup that are marked as corrupted; use only during troubleshooting. Added in MySQL 5.6.3.
- `innodb_ft_aux_table`: Qualified name of InnoDB table containing FULLTEXT index for diagnostic purposes. Added in MySQL 5.6.4.
- `innodb_ft_cache_size`: Size of cache holding parsed document in memory while creating InnoDB FULLTEXT index. Added in MySQL 5.6.4.
- `innodb_ft_enable_diag_print`: Whether to enable additional full-text search diagnostic output. Added in MySQL 5.6.4.
- `innodb_ft_enable_stopword`: Omit stop words from search index during creation of InnoDB FULLTEXT index. Added in MySQL 5.6.4.
- `innodb_ft_max_token_size`: Maximum length of words that are stored in InnoDB FULLTEXT index. Added in MySQL 5.6.4.
- `innodb_ft_min_token_size`: Minimum length of words that are stored in InnoDB FULLTEXT index. Added in MySQL 5.6.4.
- `innodb_ft_num_word_optimize`: Number of words to process during each OPTIMIZE TABLE operation on InnoDB FULLTEXT index. Added in MySQL 5.6.4.
- `innodb_ft_result_cache_limit`: InnoDB FULLTEXT search query result cache limit. Added in MySQL 5.6.13.

- `innodb_ft_server_stopword_table`: Specifies table holding list of stopwords for InnoDB FULLTEXT indexes, overriding default stopword list; can be overridden by innodb_ft_user_stopword_table. Added in MySQL 5.6.4.
- `innodb_ft_sort_pll_degree`: Number of threads used to create InnoDB FULLTEXT index in parallel, when building search index for large table. Added in MySQL 5.6.4.
- `innodb_ft_total_cache_size`: Total memory allocated for InnoDB FULLTEXT search index cache. Added in MySQL 5.6.13.
- `innodb_ft_user_stopword_table`: Specifies table holding list of stopwords for InnoDB FULLTEXT indexes, overriding default stopword list as well as innodb_ft_server_stopword_table. Added in MySQL 5.6.4.
- `innodb_io_capacity_max`: Limit up to which InnoDB is allowed to extend innodb_io_capacity setting in case of emergency. Added in MySQL 5.6.6.
- `innodb_large_prefix`: Enables longer keys for column prefix indexes. Added in MySQL 5.6.3.
- `innodb_log_checkpoint_now`: Debug option that forces InnoDB to write checkpoint. Added in MySQL 5.6.12.
- `innodb_log_compressed_pages`: Specifies whether images of re-compressed pages are stored in InnoDB redo logs. Added in MySQL 5.6.11.
- `innodb_lru_scan_depth`: Influences algorithms and heuristics for flush operation for InnoDB buffer pool. Added in MySQL 5.6.3.
- `innodb_max_dirty_pages_pct_lwm`: Low water mark representing percentage of dirty pages where preflushing is enabled to control dirty page ratio. Added in MySQL 5.6.6.
- `innodb_max_purge_lag_delay`: Maximum delay in milliseconds for formula calculated using innodb_max_purge_lag configuration option. Added in MySQL 5.6.5.
- `innodb_monitor_disable`: Turns off one or more counters in information_schema.innodb_metrics table. Added in MySQL 5.6.2.
- `innodb_monitor_enable`: Turns on one or more counters in information_schema.innodb_metrics table. Added in MySQL 5.6.2.
- `innodb_monitor_reset`: Resets count value for one or more counters in information_schema.innodb_metrics table to zero. Added in MySQL 5.6.2.
- `innodb_monitor_reset_all`: Resets all values (minimum, maximum, and so on) for one or more counters in information_schema.innodb_metrics table. Added in MySQL 5.6.2.
- `innodb_numa_interleave`: Enables NUMA MPOL_INTERLEAVE memory policy for allocation of InnoDB buffer pool. Added in MySQL 5.6.27.
- `innodb_online_alter_log_max_size`: Upper limit on size of temporary log files used during online DDL operations for InnoDB tables. Added in MySQL 5.6.6.
- `innodb_optimize_fulltext_only`: Causes OPTIMIZE TABLE statement for InnoDB table to process newly added, deleted, and updated token data for FULLTEXT index, rather than to reorganize data in clustered index of table. Added in MySQL 5.6.4.
- `innodb_page_size`: Specifies page size for all InnoDB tablespaces in instance. Added in MySQL 5.6.4.
- `innodb_print_all_deadlocks`: During shutdown, prints information about all InnoDB deadlocks to server error log. Added in MySQL 5.6.2.
- `innodb_random_read_ahead`: Enables random read-ahead technique for optimizing InnoDB I/O. Added in MySQL 5.6.3.

- `innodb_read_only`: Starts server in read-only mode. Added in MySQL 5.6.7.
- `innodb_rollback_segments`: Defines number of rollback segments in system tablespace for InnoDB to use within transaction. Added in MySQL 5.6.2.
- `innodb_saved_page_number_debug`: Saves page number. Added in MySQL 5.6.17.
- `innodb_sort_buffer_size`: Specifies size of buffer used for sorting data during creation of InnoDB index. Added in MySQL 5.6.4.
- `innodb_stats_auto_recalc`: Causes InnoDB to recalculate persistent statistics automatically after table data is changed substantially. Added in MySQL 5.6.6.
- `innodb_stats_include_delete_marked`: Include delete-marked records when calculating persistent InnoDB statistics. Added in MySQL 5.6.35.
- `innodb_stats_method`: Specifies how InnoDB index statistics collection code should treat NULLs. Added in MySQL 5.6.2.
- `innodb_stats_persistent`: Turns on InnoDB persistent statistics feature. Added in MySQL 5.6.6.
- `innodb_stats_persistent_sample_pages`: Number of pages to sample in each InnoDB index, when persistent statistics feature is also enabled. Added in MySQL 5.6.2.
- `innodb_stats_transient_sample_pages`: Number of pages to sample in each InnoDB index, when persistent statistics feature is turned off (default). Added in MySQL 5.6.2.
- `innodb_status_output`: Used to enable or disable periodic output for standard InnoDB Monitor. Also used in combination with innodb_status_output_locks to enable and disable periodic output for InnoDB Lock Monitor. Added in MySQL 5.6.16.
- `innodb_status_output_locks`: Used to enable or disable periodic output for standard InnoDB Lock Monitor. innodb_status_output must also be enabled to produce periodic output for InnoDB Lock Monitor. Added in MySQL 5.6.16.
- `innodb_sync_array_size`: Splits internal data structure used to coordinate threads, for higher concurrency in workloads with large numbers of waiting threads. Added in MySQL 5.6.3.
- `innodb_tmpdir`: Directory location for temporary table files created during online ALTER TABLE operations. Added in MySQL 5.6.29.
- `innodb_undo_directory`: Relative or absolute directory path where InnoDB creates separate tablespaces for undo logs; typically used to place those logs on different storage device. Added in MySQL 5.6.3.
- `innodb_undo_logs`: Number of undo logs (rollback segments) used by InnoDB; alias for innodb_rollback_segments. Added in MySQL 5.6.3.
- `innodb_undo_tablespaces`: Number of tablespace files that rollback segments are divided between. Added in MySQL 5.6.3.
- `log-raw`: Whether to log queries without rewriting. Added in MySQL 5.6.3.
- `log_bin_basename`: Path and base name for binary log files. Added in MySQL 5.6.2.
- `log_bin_use_v1_row_events`: Whether server is using version 1 binary log row events. Added in MySQL 5.6.6.
- `log_slow_admin_statements`: Log slow OPTIMIZE, ANALYZE, ALTER and other administrative statements to slow query log if it is open. Added in MySQL 5.6.11.
- `log_slow_slave_statements`: Cause slow statements as executed by replica to be written to slow query log. Added in MySQL 5.6.11.

- `log_throttle_queries_not_using_indexes`: Throttle write rate to slow log for queries not using indexes slow query log if it is open. Added in MySQL 5.6.5.
- `master_info_repository`: Whether to write connection metadata repository, containing source information and replication I/O thread location in source's binary log, to file or table. Added in MySQL 5.6.2.
- `master_verify_checksum`: Cause source to examine checksums when reading from binary log. Added in MySQL 5.6.2.
- `max_binlog_stmt_cache_size`: Can be used to restrict total size used to cache all nontransactional statements during transaction. Added in MySQL 5.6.1.
- `max_digest_length`: Maximum digest size in bytes. Added in MySQL 5.6.24.
- `metadata_locks_cache_size`: Size of metadata locks cache. Added in MySQL 5.6.4.
- `metadata_locks_hash_instances`: Number of metadata lock hashes. Added in MySQL 5.6.8.
- `mysql_firewall_max_query_size`: Maximum size of recorded statements. Added in MySQL 5.6.24.
- `mysql_firewall_mode`: Whether MySQL Enterprise Firewall is operational. Added in MySQL 5.6.24.
- `mysql_firewall_trace`: Whether to enable firewall trace. Added in MySQL 5.6.24.
- `named_pipe_full_access_group`: Name of Windows group granted full access to named pipe. Added in MySQL 5.6.43.
- `optimizer_join_cache_level`: How join buffers are used. Added in MySQL 5.6.1.
- `optimizer_trace`: Control optimizer tracing. Added in MySQL 5.6.3.
- `optimizer_trace_features`: Control optimizer tracing. Added in MySQL 5.6.3.
- `optimizer_trace_limit`: Control optimizer tracing. Added in MySQL 5.6.3.
- `optimizer_trace_max_mem_size`: Control optimizer tracing. Added in MySQL 5.6.3.
- `optimizer_trace_offset`: Control optimizer tracing. Added in MySQL 5.6.3.
- `performance-schema-consumer-events-stages-current`: Configure events-stages-current consumer. Added in MySQL 5.6.4.
- `performance-schema-consumer-events-stages-history`: Configure events-stages-history consumer. Added in MySQL 5.6.4.
- `performance-schema-consumer-events-stages-history-long`: Configure events-stages-history-long consumer. Added in MySQL 5.6.4.
- `performance-schema-consumer-events-statements-current`: Configure events-statements-current consumer. Added in MySQL 5.6.4.
- `performance-schema-consumer-events-statements-history`: Configure events-statements-history consumer. Added in MySQL 5.6.4.
- `performance-schema-consumer-events-statements-history-long`: Configure events-statements-history-long consumer. Added in MySQL 5.6.4.
- `performance-schema-consumer-events-waits-current`: Configure events-waits-current consumer. Added in MySQL 5.6.4.
- `performance-schema-consumer-events-waits-history`: Configure events-waits-history consumer. Added in MySQL 5.6.4.

- performance-schema-consumer-events-waits-history-long: Configure events-waits-history-long consumer. Added in MySQL 5.6.4.
- performance-schema-consumer-global-instrumentation: Configure global-instrumentation consumer. Added in MySQL 5.6.4.
- performance-schema-consumer-statements-digest: Configure statements-digest consumer. Added in MySQL 5.6.5.
- performance-schema-consumer-thread-instrumentation: Configure thread-instrumentation consumer. Added in MySQL 5.6.4.
- performance-schema-instrument: Configure Performance Schema instrument. Added in MySQL 5.6.4.
- performance_schema_accounts_size: Number of rows in accounts table. Added in MySQL 5.6.3.
- performance_schema_digests_size: Number of rows in events_statements_summary_by_digest table. Added in MySQL 5.6.5.
- performance_schema_events_stages_history_long_size: Number of rows in events_stages_history_long table. Added in MySQL 5.6.3.
- performance_schema_events_stages_history_size: Number of rows per thread in events_stages_history table. Added in MySQL 5.6.3.
- performance_schema_events_statements_history_long_size: Number of rows in events_statements_history_long table. Added in MySQL 5.6.3.
- performance_schema_events_statements_history_size: Number of rows per thread in events_statements_history table. Added in MySQL 5.6.3.
- performance_schema_hosts_size: Number of rows in hosts table. Added in MySQL 5.6.3.
- performance_schema_max_digest_length: Maximum Performance Schema digest size in bytes. Added in MySQL 5.6.26.
- performance_schema_max_socket_classes: Maximum number of socket instruments. Added in MySQL 5.6.3.
- performance_schema_max_socket_instances: Maximum number of instrumented socket objects. Added in MySQL 5.6.3.
- performance_schema_max_stage_classes: Maximum number of stage instruments. Added in MySQL 5.6.3.
- performance_schema_max_statement_classes: Maximum number of statement instruments. Added in MySQL 5.6.3.
- performance_schema_session_connect_attrs_size: Size of connection attribute strings buffer per thread. Added in MySQL 5.6.6.
- performance_schema_setup_actors_size: Number of rows in setup_actors table. Added in MySQL 5.6.1.
- performance_schema_setup_objects_size: Number of rows in setup_objects table. Added in MySQL 5.6.1.
- performance_schema_users_size: Number of rows in users table. Added in MySQL 5.6.3.
- plugin-load-add: Add to list of plugins to load at startup. Added in MySQL 5.6.3.
- pseudo_slave_mode: For internal server use. Added in MySQL 5.6.10.

- `relay_log_basename`: Complete path to relay log, including file name. Added in MySQL 5.6.2.
- `relay_log_info_repository`: Whether to write location of replication SQL thread in relay logs to file or table. Added in MySQL 5.6.2.
- `rpl_stop_slave_timeout`: Number of seconds that STOP REPLICA or STOP SLAVE waits before timing out. Added in MySQL 5.6.13.
- `sha256_password_private_key_path`: SHA256 authentication plugin private key path name. Added in MySQL 5.6.6.
- `sha256_password_public_key_path`: SHA256 authentication plugin public key path name. Added in MySQL 5.6.6.
- `show_old_temporals`: Whether SHOW CREATE TABLE should indicate pre-5.6.4 temporal columns. Added in MySQL 5.6.24.
- `simplified_binlog_gtid_recovery`: Renamed to binlog_gtid_simple_recovery. Added in MySQL 5.6.21.
- `slave-sql-verify-checksum`: Cause replica to examine checksums when reading from relay log. Added in MySQL 5.6.2.
- `slave_checkpoint_group`: Maximum number of transactions processed by multithreaded replica before checkpoint operation is called to update progress status. Not supported by NDB Cluster. Added in MySQL 5.6.3.
- `slave_checkpoint_period`: Update progress status of multithreaded replica and flush relay log info to disk after this number of milliseconds. Not supported by NDB Cluster. Added in MySQL 5.6.3.
- `slave_max_allowed_packet`: Maximum size, in bytes, of packet that can be sent from replication source server to replica; overrides max_allowed_packet. Added in MySQL 5.6.6.
- `slave_parallel_workers`: Number of applier threads for executing replication transactions in parallel. 0 disables replica multithreading. Not supported by MySQL Cluster. Added in MySQL 5.6.3.
- `slave_pending_jobs_size_max`: Maximum size of replica worker queues holding events not yet applied. Added in MySQL 5.6.3.
- `slave_rows_search_algorithms`: Determines search algorithms used for replica update batching. Any 2 or 3 from this list: INDEX_SEARCH, TABLE_SCAN, HASH_SCAN. Added in MySQL 5.6.6.
- `slave_sql_verify_checksum`: Cause replica to examine checksums when reading from relay log. Added in MySQL 5.6.2.
- `slow-start-timeout`: Wait before trying to kill windows service at startup. Added in MySQL 5.6.5.
- `ssl_crl`: File that contains certificate revocation lists. Added in MySQL 5.6.3.
- `ssl_crlpath`: Directory that contains certificate revocation list files. Added in MySQL 5.6.3.
- `stored_program_cache`: Sets "soft" upper limit for number of cached stored routines per connection. Stored procedures and stored functions are cached separately; this variable sets size for both of these. Added in MySQL 5.6.5.
- `table_open_cache_instances`: Number of open tables cache instances. Added in MySQL 5.6.6.
- `thread_pool_algorithm`: Thread pool algorithm. Added in MySQL 5.6.10.
- `thread_pool_high_priority_connection`: Whether current session is high priority. Added in MySQL 5.6.10.
- `thread_pool_max_unused_threads`: Maximum permissible number of unused threads. Added in MySQL 5.6.10.

- `thread_pool_prio_kickup_timer`: How long before statement is moved to high-priority execution. Added in MySQL 5.6.10.
- `thread_pool_size`: Number of thread groups in thread pool. Added in MySQL 5.6.10.
- `thread_pool_stall_limit`: How long before statement is defined as stalled. Added in MySQL 5.6.10.
- `transaction-read-only`: Default transaction access mode. Added in MySQL 5.6.5.
- `tx_read_only`: Default transaction access mode. Added in MySQL 5.6.5.
- `validate-password`: Whether to activate password validation plugin. Added in MySQL 5.6.6.
- `validate_password_dictionary_file`: validate_password dictionary file. Added in MySQL 5.6.6.
- `validate_password_dictionary_file_last_parsed`: When dictionary file was last parsed. Added in MySQL 5.6.26.
- `validate_password_dictionary_file_words_count`: Number of words in dictionary file. Added in MySQL 5.6.26.
- `validate_password_length`: validate_password required password length. Added in MySQL 5.6.6.
- `validate_password_mixed_case_count`: validate_password required number of uppercase/lowercase characters. Added in MySQL 5.6.6.
- `validate_password_number_count`: validate_password required number of digit characters. Added in MySQL 5.6.6.
- `validate_password_policy`: validate_password password policy. Added in MySQL 5.6.6.
- `validate_password_special_char_count`: validate_password required number of special characters. Added in MySQL 5.6.6.
- `validate_user_plugins`: Whether to perform additional validation of user plugins. Added in MySQL 5.6.11.

Options and Variables Deprecated in MySQL 5.6

The following system variables, status variables, and options have been deprecated in MySQL 5.6.

- `Delayed_errors`: Number of rows written with INSERT DELAYED for which some error occurred. Deprecated as of MySQL 5.6.7.
- `Delayed_insert_threads`: Number of INSERT DELAYED thread handlers in use. Deprecated as of MySQL 5.6.7.
- `Delayed_writes`: Number of INSERT DELAYED rows written. Deprecated as of MySQL 5.6.7.
- `Not_flushed_delayed_rows`: Number of rows waiting to be written in INSERT DELAY queues. Deprecated as of MySQL 5.6.7.
- `avoid_temporal_upgrade`: Whether ALTER TABLE should upgrade pre-5.6.4 temporal columns. Deprecated as of MySQL 5.6.24.
- `binlogging_impossible_mode`: Deprecated and later removed. Use binlog_error_action instead. Deprecated as of MySQL 5.6.22.
- `delayed_insert_limit`: After inserting delayed_insert_limit rows, INSERT DELAYED handler checks whether there are any SELECT statements pending. If so, it allows these to execute before continuing. Deprecated as of MySQL 5.6.7.

- `delayed_insert_timeout`: How many seconds INSERT DELAYED thread should wait for INSERT statements before terminating. Deprecated as of MySQL 5.6.7.
- `delayed_queue_size`: What size queue (in rows) should be allocated for handling INSERT DELAYED. Deprecated as of MySQL 5.6.7.
- `explicit_defaults_for_timestamp`: Whether TIMESTAMP columns are nullable and have DEFAULT NULL. Deprecated as of MySQL 5.6.6.
- `have_profiling`: Whether statement profiling capability is available. Deprecated as of MySQL 5.6.8.
- `innodb`: Enable InnoDB (if this version of MySQL supports it). Deprecated as of MySQL 5.6.21.
- `innodb_additional_mem_pool_size`: Size of memory pool InnoDB uses to store data dictionary information and other internal data structures. Deprecated as of MySQL 5.6.3.
- `innodb_checksums`: Enable InnoDB checksums validation. Deprecated as of MySQL 5.6.3.
- `innodb_locks_unsafe_for_binlog`: Force InnoDB not to use next-key locking. Instead use only row-level locking. Deprecated as of MySQL 5.6.3.
- `innodb_stats_sample_pages`: Number of index pages to sample for index distribution statistics. Deprecated as of MySQL 5.6.3.
- `innodb_use_sys_malloc`: Whether InnoDB uses OS or own memory allocator. Deprecated as of MySQL 5.6.3.
- `language`: Client error messages in given language. May be given as full path. Deprecated as of MySQL 5.6.1.
- `master-retry-count`: Number of tries replica makes to connect to source before giving up. Deprecated as of MySQL 5.6.1.
- `max_delayed_threads`: Do not start more than this number of threads to handle INSERT DELAYED statements. If set to zero, which means INSERT DELAYED is not used. Deprecated as of MySQL 5.6.7.
- `max_insert_delayed_threads`: Synonym for max_delayed_threads. Deprecated as of MySQL 5.6.7.
- `max_tmp_tables`: Unused. Deprecated as of MySQL 5.6.7.
- `multi_range_count`: Maximum number of ranges to send to table handler at once during range selects. Deprecated as of MySQL 5.6.7.
- `profiling`: Enable or disable statement profiling. Deprecated as of MySQL 5.6.8.
- `profiling_history_size`: How many statements to maintain profiling information for. Deprecated as of MySQL 5.6.8.
- `show_old_temporals`: Whether SHOW CREATE TABLE should indicate pre-5.6.4 temporal columns. Deprecated as of MySQL 5.6.24.
- `simplified_binlog_gtid_recovery`: Renamed to binlog_gtid_simple_recovery. Deprecated as of MySQL 5.6.23.
- `thread_concurrency`: Permits application to provide hint to threads system for desired number of threads which should be run at one time. Deprecated as of MySQL 5.6.1.
- `timed_mutexes`: Specify whether to time mutexes (only InnoDB mutexes are currently supported). Deprecated as of MySQL 5.6.20.

Options and Variables Removed in MySQL 5.6

The following system variables, status variables, and options have been removed in MySQL 5.6.

- `Com_show_new_master`: Count of SHOW NEW MASTER statements. Removed in MySQL 5.6.2.
- `bind-address`: IP address or host name to bind to. Removed in MySQL 5.6.1.
- `disable-gtid-unsafe-statements`: Obsolete: Replaced by enforce_gtid_consistency in MySQL 5.6.9. Removed in MySQL 5.6.9.
- `disable_gtid_unsafe_statements`: Obsolete: Replaced by enforce_gtid_consistency in MySQL 5.6.9. Removed in MySQL 5.6.9.
- `engine_condition_pushdown`: Push supported query conditions to storage engine. Removed in MySQL 5.6.1.
- `gtid_done`: Obsolete: Replaced by gtid_executed in MySQL 5.6.9. Removed in MySQL 5.6.9.
- `gtid_lost`: Obsolete: Replaced by gtid_purged in MySQL 5.6.9. Removed in MySQL 5.6.9.
- `have_csv`: Whether mysqld supports csv tables. Removed in MySQL 5.6.1.
- `have_innodb`: Whether mysqld supports InnoDB tables. Removed in MySQL 5.6.1.
- `have_partitioning`: Whether mysqld supports partitioning. Removed in MySQL 5.6.1.
- `log`: Log connections and queries to file. Removed in MySQL 5.6.1.
- `log-slow-admin-statements`: Log slow OPTIMIZE, ANALYZE, ALTER and other administrative statements to slow query log if it is open. Removed in MySQL 5.6.11.
- `log-slow-slave-statements`: Cause slow statements as executed by replica to be written to slow query log. Removed in MySQL 5.6.11.
- `log_slow_queries`: Whether to log slow queries. Logging defaults to hostname-slow.log file. Must be enabled to activate other slow query log options. Removed in MySQL 5.6.1.
- `mysql_firewall_max_query_size`: Maximum size of recorded statements. Removed in MySQL 5.6.26.
- `one-thread`: Only use one thread (for debugging under Linux). Removed in MySQL 5.6.1.
- `optimizer_join_cache_level`: How join buffers are used. Removed in MySQL 5.6.3.
- `safe-mode`: Skip some optimization stages (for testing). Removed in MySQL 5.6.6.
- `skip-thread-priority`: Do not give threads different priorities. Removed in MySQL 5.6.1.
- `sql_big_tables`: This variable is deprecated, and is mapped to big_tables. Removed in MySQL 5.6.1.
- `sql_low_priority_updates`: This variable is deprecated, and is mapped to low_priority_updates. Removed in MySQL 5.6.1.
- `sql_max_join_size`: This variable is deprecated, and is mapped to max_join_size. Removed in MySQL 5.6.1.

1.5 MySQL Information Sources

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

- MySQL Websites
- MySQL Community Support at the MySQL Forums

- MySQL Enterprise

MySQL Websites

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

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

MySQL Community Support at the MySQL Forums

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

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

MySQL Enterprise

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

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

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

1.6 How to Report Bugs or Problems

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

- Start by searching the MySQL online manual at https://dev.mysql.com/doc/. We try to keep the manual up to date by updating it frequently with solutions to newly found problems. In addition, the release notes accompanying the manual can be particularly useful since it is quite possible that a newer version contains a solution to your problem. The release notes are available at the location just given for the manual.

- If you get a parse error for an SQL statement, please check your syntax closely. If you cannot find something wrong with it, it is extremely likely that your current version of MySQL Server doesn't support the syntax you are using. If you are using the current version and the manual doesn't cover the syntax that you are using, MySQL Server doesn't support your statement.

  If the manual covers the syntax you are using, but you have an older version of MySQL Server, you should check the MySQL change history to see when the syntax was implemented. In this case, you have the option of upgrading to a newer version of MySQL Server.

- For solutions to some common problems, see Section B.3, “Problems and Common Errors”.

- Search the bugs database at http://bugs.mysql.com/ to see whether the bug has been reported and fixed.

- You can also use http://www.mysql.com/search/ to search all the Web pages (including the manual) that are located at the MySQL website.

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

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

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

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

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

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

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

It is most helpful when a good description of the problem is included in the bug report. That is, give a good example of everything you did that led to the problem and describe, in exact detail, the problem itself. The best reports are those that include a full example showing how to reproduce the bug or problem. See Section 5.8, “Debugging MySQL”.

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

The most common errors made in bug reports are (a) not including the version number of the MySQL distribution that you use, and (b) not fully describing the platform on which the MySQL server is installed (including the platform type and version number). These are highly relevant pieces of information, and in 99 cases out of 100, the bug report is useless without them. Very often we get questions like, “Why doesn't this work for me?” Then we find that the feature requested wasn't

implemented in that MySQL version, or that a bug described in a report has been fixed in newer MySQL versions. Errors often are platform-dependent. In such cases, it is next to impossible for us to fix anything without knowing the operating system and the version number of the platform.

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

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

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

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

Please include the following information in your report:

- The version number of the MySQL distribution you are using (for example, MySQL 5.7.10). You can find out which version you are running by executing `mysqladmin version`. The `mysqladmin` program can be found in the `bin` directory under your MySQL installation directory.
- The manufacturer and model of the machine on which you experience the problem.
- The operating system name and version. If you work with Windows, you can usually get the name and version number by double-clicking your My Computer icon and pulling down the “Help/About Windows” menu. For most Unix-like operating systems, you can get this information by executing the command `uname -a`.
- Sometimes the amount of memory (real and virtual) is relevant. If in doubt, include these values.
- The contents of the `docs/INFO_BIN` file from your MySQL installation. This file contains information about how MySQL was configured and compiled.
- If you are using a source distribution of the MySQL software, include the name and version number of the compiler that you used. If you have a binary distribution, include the distribution name.
- If the problem occurs during compilation, include the exact error messages and also a few lines of context around the offending code in the file where the error occurs.
- If `mysqld` died, you should also report the statement that caused `mysqld` to unexpectedly exit. You can usually get this information by running `mysqld` with query logging enabled, and then looking in the log after `mysqld` exits. See Section 5.8, “Debugging MySQL”.
- If a database table is related to the problem, include the output from the `SHOW CREATE TABLE` *`db_name.tbl_name`* statement in the bug report. This is a very easy way to get the definition of any table in a database. The information helps us create a situation matching the one that you have experienced.
- The SQL mode in effect when the problem occurred can be significant, so please report the value of the `sql_mode` system variable. For stored procedure, stored function, and trigger objects, the relevant `sql_mode` value is the one in effect when the object was created. For a stored procedure or function, the `SHOW CREATE PROCEDURE` or `SHOW CREATE FUNCTION` statement shows the relevant SQL mode, or you can query `INFORMATION_SCHEMA` for the information:

```
SELECT ROUTINE_SCHEMA, ROUTINE_NAME, SQL_MODE
```

```
FROM INFORMATION_SCHEMA.ROUTINES;
```

For triggers, you can use this statement:

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

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

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

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

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

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

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

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

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

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

that the program uses. For example, if you have a Perl script that uses the `DBI` and `DBD::mysql` modules, include the version numbers for Perl, `DBI`, and `DBD::mysql`.

- If your question is related to the privilege system, please include the output of `mysqladmin reload`, and all the error messages you get when trying to connect. When you test your privileges, you should execute `mysqladmin reload version` and try to connect with the program that gives you trouble.
- If you have a patch for a bug, do include it. But do not assume that the patch is all we need, or that we can use it, if you do not provide some necessary information such as test cases showing the bug that your patch fixes. We might find problems with your patch or we might not understand it at all. If so, we cannot use it.

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

  If you are running Windows, please verify the value of `lower_case_table_names` using the `SHOW VARIABLES LIKE 'lower_case_table_names'` statement. This variable affects how the server handles lettercase of database and table names. Its effect for a given value should be as described in Section 9.2.3, “Identifier Case Sensitivity”.
- If you often get corrupted tables, you should try to find out when and why this happens. In this case, the error log in the MySQL data directory may contain some information about what happened. (This is the file with the `.err` suffix in the name.) See Section 5.4.2, “The Error Log”. Please include any relevant information from this file in your bug report. Normally `mysqld` should *never* corrupt a table if nothing killed it in the middle of an update. If you can find the cause of `mysqld` dying, it is much easier for us to provide you with a fix for the problem. See Section B.3.1, “How to Determine What Is Causing a Problem”.
- If possible, download and install the most recent version of MySQL Server and check whether it solves your problem. All versions of the MySQL software are thoroughly tested and should work without problems. We believe in making everything as backward-compatible as possible, and you should be able to switch MySQL versions without difficulty. See Section 2.1.2, “Which MySQL Version and Distribution to Install”.

1.7 MySQL Standards Compliance

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

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

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

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

MySQL Server was originally designed to work with medium-sized databases (10-100 million rows, or about 100MB per table) on small computer systems. Today MySQL Server handles terabyte-sized databases, but the code can also be compiled in a reduced version suitable for hand-held and embedded devices. The compact design of the MySQL server makes development in both directions possible without any conflicts in the source tree.

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

MySQL supports ODBC levels 0 to 3.51.

MySQL supports high-availability database clustering using the `NDBCLUSTER` storage engine. See Chapter 18, *MySQL NDB Cluster 7.3 and NDB Cluster 7.4*.

We implement XML functionality which supports most of the W3C XPath standard. See Section 12.12, “XML Functions”.

Selecting SQL Modes

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

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

For more information on setting the SQL mode, see Section 5.1.10, “Server SQL Modes”.

Running MySQL in ANSI Mode

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

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

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

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

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

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

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

See Section 5.1.6, “Server Command Options”.

1.7.1 MySQL Extensions to Standard SQL

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

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

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

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

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

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

The following descriptions list MySQL extensions, organized by category.

- Organization of data on disk

  MySQL Server maps each database to a directory under the MySQL data directory, and maps tables within a database to file names in the database directory. This has a few implications:

  - Database and table names are case-sensitive in MySQL Server on operating systems that have case-sensitive file names (such as most Unix systems). See Section 9.2.3, “Identifier Case Sensitivity”.
  - You can use standard system commands to back up, rename, move, delete, and copy tables that are managed by the `MyISAM` storage engine. For example, it is possible to rename a `MyISAM` table by renaming the `.MYD`, `.MYI`, and `.frm` files to which the table corresponds. (Nevertheless, it is preferable to use `RENAME TABLE` or `ALTER TABLE ... RENAME` and let the server rename the files.)

- General language syntax

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

- SQL statement syntax

  - The `ANALYZE TABLE`, `CHECK TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements.
  - The `CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE` statements. See Section 13.1.10, “CREATE DATABASE Statement”, Section 13.1.21, “DROP DATABASE Statement”, and Section 13.1.1, “ALTER DATABASE Statement”.

- The `DO` statement.
- `EXPLAIN SELECT` to obtain a description of how tables are processed by the query optimizer.
- The `FLUSH` and `RESET` statements.
- The `SET` statement. See Section 13.7.4.1, “SET Syntax for Variable Assignment”.
- The `SHOW` statement. See Section 13.7.5, “SHOW Statements”. The information produced by many of the MySQL-specific `SHOW` statements can be obtained in more standard fashion by using `SELECT` to query `INFORMATION_SCHEMA`. See Chapter 21, *INFORMATION_SCHEMA Tables*.
- Use of `LOAD DATA`. In many cases, this syntax is compatible with Oracle `LOAD DATA`. See Section 13.2.6, “LOAD DATA Statement”.
- Use of `RENAME TABLE`. See Section 13.1.32, “RENAME TABLE Statement”.
- Use of `REPLACE` instead of `DELETE` plus `INSERT`. See Section 13.2.8, “REPLACE Statement”.
- Use of `CHANGE col_name`, `DROP col_name`, or `DROP INDEX`, `IGNORE` or `RENAME` in `ALTER TABLE` statements. Use of multiple `ADD`, `ALTER`, `DROP`, or `CHANGE` clauses in an `ALTER TABLE` statement. See Section 13.1.7, “ALTER TABLE Statement”.
- Use of index names, indexes on a prefix of a column, and use of `INDEX` or `KEY` in `CREATE TABLE` statements. See Section 13.1.17, “CREATE TABLE Statement”.
- Use of `TEMPORARY` or `IF NOT EXISTS` with `CREATE TABLE`.
- Use of `IF EXISTS` with `DROP TABLE` and `DROP DATABASE`.
- The capability of dropping multiple tables with a single `DROP TABLE` statement.
- The `ORDER BY` and `LIMIT` clauses of the `UPDATE` and `DELETE` statements.
- `INSERT INTO tbl_name SET col_name = ...` syntax.
- The `DELAYED` clause of the `INSERT` and `REPLACE` statements.
- The `LOW_PRIORITY` clause of the `INSERT`, `REPLACE`, `DELETE`, and `UPDATE` statements.
- Use of `INTO OUTFILE` or `INTO DUMPFILE` in `SELECT` statements. See Section 13.2.9, “SELECT Statement”.
- Options such as `STRAIGHT_JOIN` or `SQL_SMALL_RESULT` in `SELECT` statements.
- You don't need to name all selected columns in the `GROUP BY` clause. This gives better performance for some very specific, but quite normal queries. See Section 12.19, “Aggregate Functions”.
- You can specify `ASC` and `DESC` with `GROUP BY`, not just with `ORDER BY`.
- The ability to set variables in a statement with the `:=` assignment operator. See Section 9.4, “User-Defined Variables”.

- Data types
  - The `MEDIUMINT`, `SET`, and `ENUM` data types, and the various `BLOB` and `TEXT` data types.
  - The `AUTO_INCREMENT`, `BINARY`, `NULL`, `UNSIGNED`, and `ZEROFILL` data type attributes.

- Functions and operators
  - To make it easier for users who migrate from other SQL environments, MySQL Server supports aliases for many functions. For example, all string functions support both standard SQL syntax and ODBC syntax.
  - MySQL Server understands the `||` and `&&` operators to mean logical OR and AND, as in the C programming language. In MySQL Server, `||` and `OR` are synonyms, as are `&&` and `AND`. Because of this nice syntax, MySQL Server doesn't support the standard SQL `||` operator for string concatenation; use `CONCAT()` instead. Because `CONCAT()` takes any number of arguments, it is easy to convert use of the `||` operator to MySQL Server.
  - Use of `COUNT(DISTINCT value_list)` where *`value_list`* has more than one element.
  - String comparisons are case-insensitive by default, with sort ordering determined by the collation of the current character set, which is `latin1` (cp1252 West European) by default. To perform case-sensitive comparisons instead, you should declare your columns with the `BINARY` attribute or use the `BINARY` cast, which causes comparisons to be done using the underlying character code values rather than a lexical ordering.
  - The `%` operator is a synonym for `MOD()`. That is, *`N % M`* is equivalent to `MOD(N,M)`. `%` is supported for C programmers and for compatibility with PostgreSQL.
  - The `=`, `<>`, `<=`, `<`, `>=`, `>`, `<<`, `>>`, `<=>`, `AND`, `OR`, or `LIKE` operators may be used in expressions in the output column list (to the left of the `FROM`) in `SELECT` statements. For example:

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

  - The `LAST_INSERT_ID()` function returns the most recent `AUTO_INCREMENT` value. See Section 12.16, “Information Functions”.
  - `LIKE` is permitted on numeric values.
  - The `REGEXP` and `NOT REGEXP` extended regular expression operators.
  - `CONCAT()` or `CHAR()` with one argument or more than two arguments. (In MySQL Server, these functions can take a variable number of arguments.)
  - The `BIT_COUNT()`, `CASE`, `ELT()`, `FROM_DAYS()`, `FORMAT()`, `IF()`, `PASSWORD()`, `ENCRYPT()`, `MD5()`, `ENCODE()`, `DECODE()`, `PERIOD_ADD()`, `PERIOD_DIFF()`, `TO_DAYS()`, and `WEEKDAY()` functions.
  - Use of `TRIM()` to trim substrings. Standard SQL supports removal of single characters only.
  - The `GROUP BY` functions `STD()`, `BIT_OR()`, `BIT_AND()`, `BIT_XOR()`, and `GROUP_CONCAT()`. See Section 12.19, “Aggregate Functions”.

1.7.2 MySQL Differences from Standard SQL

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

- There are several differences between the MySQL and standard SQL privilege systems. For example, in MySQL, privileges for a table are not automatically revoked when you delete a table. You must explicitly issue a `REVOKE` statement to revoke privileges for a table. For more information, see Section 13.7.1.6, “REVOKE Statement”.
- The `CAST()` function does not support cast to `REAL` or `BIGINT`. See Section 12.11, “Cast Functions and Operators”.

1.7.2.1 SELECT INTO TABLE Differences

MySQL Server doesn't support the `SELECT ... INTO TABLE` Sybase SQL extension. Instead, MySQL Server supports the `INSERT INTO ... SELECT` standard SQL syntax, which is basically the same thing. See Section 13.2.5.1, “INSERT ... SELECT Statement”. For example:

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

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

You can use `SELECT ... INTO` with user-defined variables. The same syntax can also be used inside stored routines using cursors and local variables. See Section 13.2.9.1, “SELECT ... INTO Statement”.

1.7.2.2 UPDATE Differences

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

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

1.7.2.3 FOREIGN KEY Constraint Differences

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

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

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

- In an SQL statement that inserts, deletes, or updates many rows, foreign key constraints (like unique constraints) are checked row-by-row. When performing foreign key checks, `InnoDB` sets shared row-level locks on child or parent records that it must examine. MySQL checks foreign key constraints immediately; the check is not deferred to transaction commit. According to the SQL standard, the default behavior should be deferred checking. That is, constraints are only checked after the *entire SQL statement* has been processed. This means that it is not possible to delete a row that refers to itself using a foreign key.

- No storage engine, including `InnoDB`, recognizes or enforces the `MATCH` clause used in referential-integrity constraint definitions. Use of an explicit `MATCH` clause does not have the specified effect, and it causes `ON DELETE` and `ON UPDATE` clauses to be ignored. Specifying the `MATCH` should be avoided.

  The `MATCH` clause in the SQL standard controls how `NULL` values in a composite (multiple-column) foreign key are handled when comparing to a primary key in the referenced table. MySQL essentially implements the semantics defined by `MATCH SIMPLE`, which permits a foreign key to be all or partially `NULL`. In that case, a (child table) row containing such a foreign key can be inserted even though it does not match any row in the referenced (parent) table. (It is possible to implement other semantics using triggers.)

- MySQL requires that the referenced columns be indexed for performance reasons. However, MySQL does not enforce a requirement that the referenced columns be `UNIQUE` or be declared `NOT NULL`.

A `FOREIGN KEY` constraint that references a non-`UNIQUE` key is not standard SQL but rather an `InnoDB` extension. The `NDB` storage engine, on the other hand, requires an explicit unique key (or primary key) on any column referenced as a foreign key.

The handling of foreign key references to nonunique keys or keys that contain `NULL` values is not well defined for operations such as `UPDATE` or `DELETE CASCADE`. You are advised to use foreign keys that reference only `UNIQUE` (including `PRIMARY`) and `NOT NULL` keys.

- MySQL parses but ignores “inline `REFERENCES` specifications” (as defined in the SQL standard) where the references are defined as part of the column specification. MySQL accepts `REFERENCES` clauses only when specified as part of a separate `FOREIGN KEY` specification. For storage engines that do not support foreign keys (such as `MyISAM`), MySQL Server parses and ignores foreign key specifications.

For information about foreign key constraints, see Section 13.1.17.5, “FOREIGN KEY Constraints”.

1.7.2.4 '--' as the Start of a Comment

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

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

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

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

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

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

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

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

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

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

1.7.3 How MySQL Deals with Constraints

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

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

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

Several SQL mode options are available to provide greater control over handling of bad data values and whether to continue statement execution or abort when errors occur. Using these options, you can configure MySQL Server to act in a more traditional fashion that is like other DBMSs that reject improper input. The SQL mode can be set globally at server startup to affect all clients. Individual clients can set the SQL mode at runtime, which enables each client to select the behavior most appropriate for its requirements. See Section 5.1.10, “Server SQL Modes”.

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

1.7.3.1 PRIMARY KEY and UNIQUE Index Constraints

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

MySQL supports an `IGNORE` keyword for `INSERT`, `UPDATE`, and so forth. If you use it, MySQL ignores primary-key or unique-key violations and continues processing with the next row. See the section for the statement that you are using (Section 13.2.5, “INSERT Statement”, Section 13.2.11, “UPDATE Statement”, and so forth).

You can get information about the number of rows actually inserted or updated with the `mysql_info()` C API function. You can also use the `SHOW WARNINGS` statement. See mysql_info(), and Section 13.7.5.41, “SHOW WARNINGS Statement”.

Only `InnoDB` tables support foreign keys. See Section 13.1.17.5, “FOREIGN KEY Constraints”.

1.7.3.2 FOREIGN KEY Constraints

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

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

Note

`NDB` does not support `ON UPDATE CASCADE` actions where the referenced column is the parent table's primary key.

`SET DEFAULT` is also supported by the MySQL Server but is currently rejected as invalid by `InnoDB` and `NDB`. Since MySQL does not support deferred constraint checking, `NO ACTION` is treated as `RESTRICT`. For the exact syntax supported by MySQL for foreign keys, see Section 13.1.17.5, “FOREIGN KEY Constraints”.

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

MySQL requires that foreign key columns be indexed; if you create a table with a foreign key constraint but no index on a given column, an index is created. *Exception*: NDB Cluster requires an explicit unique key (or primary key) on the foreign key column.

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

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

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

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

1.7.3.3 Constraints on Invalid Data

By default, MySQL is forgiving of invalid or improper data values and coerces them to valid values for data entry. However, you can enable strict SQL mode such that the server rejects invalid values and aborts the statement in which they occur. See Section 5.1.10, “Server SQL Modes”.

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

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

- If you try to store an out of range value into a numeric column, MySQL Server instead stores zero, the smallest possible value, or the largest possible value, whichever is closest to the invalid value.
- For strings, MySQL stores either the empty string or as much of the string as can be stored in the column.
- If you try to store a string that does not start with a number into a numeric column, MySQL Server stores 0.
- Invalid values for `ENUM` and `SET` columns are handled as described in Section 1.7.3.4, “ENUM and SET Constraints”.
- MySQL permits you to store certain incorrect date values into `DATE` and `DATETIME` columns (such as `'2000-02-31'` or `'2000-02-00'`). In this case, when an application has not enabled strict SQL mode, it up to the application to validate the dates before storing them. If MySQL can store a date value and retrieve exactly the same value, MySQL stores it as given. If the date is totally wrong (outside the server's ability to store it), the special “zero” date value `'0000-00-00'` is stored in the column instead.
- If you try to store `NULL` into a column that doesn't take `NULL` values, an error occurs for single-row `INSERT` statements. For multiple-row `INSERT` statements or for `INSERT INTO ... SELECT` statements, MySQL Server stores the implicit default value for the column data type. In general, this is `0` for numeric types, the empty string (`''`) for string types, and the “zero” value for date and time types. Implicit default values are discussed in Section 11.5, “Data Type Default Values”.
- If an `INSERT` statement specifies no value for a column, MySQL inserts its default value if the column definition includes an explicit `DEFAULT` clause. If the definition has no such `DEFAULT` clause, MySQL inserts the implicit default value for the column data type.

The reason for using the preceding rules when strict mode is not in effect is that we cannot check these conditions until the statement has begun executing. We cannot just roll back if we encounter a problem after updating a few rows, because the storage engine may not support rollback. The option of terminating the statement is not that good; in this case, the update would be “half done,” which is probably the worst possible scenario. In this case, it is better to “do the best you can” and then continue as if nothing happened.

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

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

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

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

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

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

1.7.3.4 ENUM and SET Constraints

`ENUM` and `SET` columns provide an efficient way to define columns that can contain only a given set of values. See Section 11.3.5, “The ENUM Type”, and Section 11.3.6, “The SET Type”.

With strict mode enabled (see Section 5.1.10, “Server SQL Modes”), the definition of a `ENUM` or `SET` column acts as a constraint on values entered into the column. An error occurs for values that do not satisfy these conditions:

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

Errors for invalid values can be suppressed in strict mode if you use `INSERT IGNORE` or `UPDATE IGNORE`. In this case, a warning is generated rather than an error. For `ENUM`, the value is inserted as the error member (`0`). For `SET`, the value is inserted as given except that any invalid substrings are deleted. For example, `'a,x,b,y'` results in a value of `'a,b'`.

1.8 Credits

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

1.8.1 Contributors to MySQL

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

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

  The initial port to Win32/NT.

- Per Eric Olsson

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

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

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

- David J. Hughes

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

- Patrick Lynch

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

- Fred Lindberg

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

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

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

- Yuri Dario

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

- Tim Bunce

  Author of mysqlhotcopy.

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

  Sorting for Slovenian language.

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

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

- Joshua Chamas <joshua@chamas.com>

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

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

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

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

  For one of the early JDBC drivers.

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

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

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

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

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

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

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

  For providing RPM packages of MySQL for Red Hat Linux.

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

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

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

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

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

  For providing RPM versions for MySQL 3.21.

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

  Ideas for secure checking of DNS host names.

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

  Some support for Chinese(BIG5) characters.

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

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

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

  Czech sorting order.

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

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

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

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

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

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

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

  The `AGGREGATE` extension to user-defined functions.

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

  Help to set up InstallShield for MySQL-Win32.

- Jethro Wright III <jetman@li.net>

  The `libmysql.dll` library.

- James Pereria <jpereira@iafrica.com>

  Mysqlmanager, a Win32 GUI tool for administering MySQL Servers.

- Curt Sampson <cjs@portal.ca>

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

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

  Examples in the MySQL Tutorial.

- Steve Harvey

  For making `mysqlaccess` more secure.

- Konark IA-64 Centre of Persistent Systems Private Limited

  Help with the Win64 port of the MySQL server.

- Albert Chin-A-Young.

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

- John Birrell

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

- Benjamin Pflugmann

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

- Jocelyn Fournier

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

- Marc Liyanage

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

- Robert Rutherford

  Providing invaluable information and feedback about the QNX port.

- Previous developers of NDB Cluster

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

- Google Inc.

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

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

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

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

- Daniel Koch <dkoch@amcity.com>

  Irix setup.

- Luuk de Boer <luuk@wxs.nl>

  Benchmark questions.

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

  DBD::mysql questions.

- Boyd Lynn Gerber <gerberb@zenez.com>

  SCO-related questions.

- Richard Mehalick <RM186061@shellus.com>

  xmysql-related questions and basic installation questions.

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

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

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

  General questions.

- Jonathan J Smith <jsmith@wtp.net>

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

- David Sklar <sklar@student.net>

  Using MySQL from PHP and Perl.

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

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

- John Lyon <jlyon@imag.net>

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

- Lorvid Ltd. <lorvid@WOLFENET.com>

  Simple billing/license/support/copyright issues.

- Patrick Sherrill <patrick@coconet.com>

  ODBC and VisualC++ interface questions.

- Randy Harmon <rjharmon@uptimecomputers.com>

  DBD, Linux, some SQL syntax questions.

1.8.2 Documenters and translators

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

- Kim Aldale

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

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

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

- Yan Cailin

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

- Jay Flaherty <fty@mediapulse.com>

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

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

  Proof-reading of the Reference Manual.

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

  French error messages.

- Petr Snajdr, <snajdr@pvt.net>

  Czech error messages.

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

  Polish error messages.

- Miguel Angel Fernandez Roiz

  Spanish error messages.

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

  Norwegian error messages and testing of MySQL 3.21.xx.

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

  Russian error messages.

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

  Italian error messages.

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

German error messages.

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

  Slovak error messages.

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

  Romanian error messages.

- Peter Feher

  Hungarian error messages.

- Roberto M. Serqueira

  Portuguese error messages.

- Carsten H. Pedersen

  Danish error messages.

- Arjen Lentz

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

1.8.3 Packages that support MySQL

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

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

- Tim Bunce, Alligator Descartes

  For the `DBD` (Perl) interface.

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

  For the Perl interface for MySQL Server.

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

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

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

  For porting PHP for MySQL Server.

- Georg Richter

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

- Giovanni Maruzzelli <maruzz@matrice.it>

  For porting iODBC (Unix ODBC).

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

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

1.8.4 Tools that were used to create MySQL

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

- Free Software Foundation

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

- Free Software Foundation & The XEmacs development team

  For a really great editor/environment.

- Julian Seward

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

- Dorothea Lütkehaus and Andreas Zeller

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

1.8.5 Supporters of MySQL

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

- VA Linux / Andover.net

  Funded replication.

- NuSphere

  Editing of the MySQL manual.

- Stork Design studio

  The MySQL website in use between 1998-2000.

- Intel

  Contributed to development on Windows and Linux platforms.

- Compaq

  Contributed to Development on Linux/Alpha.

- SWSoft

  Development on the embedded `mysqld` version.

- FutureQuest

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

Chapter 2 Installing and Upgrading MySQL

Table of Contents

2.1 General Installation Guidance 59
2.1.1 Supported Platforms 60
2.1.2 Which MySQL Version and Distribution to Install 60
2.1.3 How to Get MySQL 61
2.1.4 Verifying Package Integrity Using MD5 Checksums or GnuPG 61
2.1.5 Installation Layouts 74
2.1.6 Compiler-Specific Build Characteristics 74
2.2 Installing MySQL on Unix/Linux Using Generic Binaries 74
2.3 Installing MySQL on Microsoft Windows 77
2.3.1 MySQL Installation Layout on Microsoft Windows 79
2.3.2 Choosing an Installation Package 80
2.3.3 MySQL Installer for Windows 81
2.3.4 Installing MySQL on Microsoft Windows Using a `noinstall` ZIP Archive 104
2.3.5 Troubleshooting a Microsoft Windows MySQL Server Installation 112
2.3.6 Windows Postinstallation Procedures 113
2.3.7 Windows Platform Restrictions 115
2.4 Installing MySQL on OS X 116
2.4.1 General Notes on Installing MySQL on OS X 117
2.4.2 Installing MySQL on OS X Using Native Packages 118
2.4.3 Installing a MySQL Launch Daemon 123
2.4.4 Installing and Using the MySQL Preference Pane 125
2.5 Installing MySQL on Linux 130
2.5.1 Installing MySQL on Linux Using the MySQL Yum Repository 131
2.5.2 Replacing a Third-Party Distribution of MySQL Using the MySQL Yum Repository 134
2.5.3 Installing MySQL on Linux Using the MySQL APT Repository 137
2.5.4 Installing MySQL on Linux Using the MySQL SLES Repository 137
2.5.5 Installing MySQL on Linux Using RPM Packages from Oracle 137
2.5.6 Installing MySQL on Linux Using Debian Packages from Oracle 141
2.5.7 Installing MySQL on Linux from the Native Software Repositories 142
2.5.8 Deploying MySQL on Linux with Docker 146
2.5.9 Installing MySQL on Linux with Juju 153
2.6 Installing MySQL Using Unbreakable Linux Network (ULN) 153
2.7 Installing MySQL on Solaris 153
2.7.1 Installing MySQL on Solaris Using a Solaris PKG 154
2.8 Installing MySQL on FreeBSD 155
2.9 Installing MySQL from Source 156
2.9.1 Source Installation Methods 156
2.9.2 Source Installation Prerequisites 157
2.9.3 MySQL Layout for Source Installation 158
2.9.4 Installing MySQL Using a Standard Source Distribution 158
2.9.5 Installing MySQL Using a Development Source Tree 162
2.9.6 Configuring SSL Library Support 164
2.9.7 MySQL Source-Configuration Options 165
2.9.8 Dealing with Problems Compiling MySQL 179
2.9.9 MySQL Configuration and Third-Party Tools 180
2.10 Postinstallation Setup and Testing 181
2.10.1 Initializing the Data Directory 181
2.10.2 Starting the Server 184
2.10.3 Testing the Server 187
2.10.4 Securing the Initial MySQL Accounts 189
2.10.5 Starting and Stopping MySQL Automatically 193
2.11 Upgrading MySQL 194

2.11.1 Before You Begin 194
2.11.2 Upgrade Paths 195
2.11.3 Changes in MySQL 5.6 195
2.11.4 Upgrading MySQL Binary or Package-based Installations on Unix/Linux 202
2.11.5 Upgrading MySQL with the MySQL Yum Repository 204
2.11.6 Upgrading MySQL with the MySQL APT Repository 206
2.11.7 Upgrading MySQL with the MySQL SLES Repository 206
2.11.8 Upgrading MySQL on Windows 206
2.11.9 Upgrade Troubleshooting 208
2.11.10 Rebuilding or Repairing Tables or Indexes 208
2.11.11 Copying MySQL Databases to Another Machine 210
2.12 Downgrading MySQL 211
2.12.1 Before You Begin 211
2.12.2 Downgrade Paths 211
2.12.3 Downgrade Notes 212
2.12.4 Downgrading Binary and Package-based Installations on Unix/Linux 213
2.12.5 Downgrade Troubleshooting 215
2.13 Perl Installation Notes 215
2.13.1 Installing Perl on Unix 216
2.13.2 Installing ActiveState Perl on Windows 216
2.13.3 Problems Using the Perl DBI/DBD Interface 217

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

If you are interested in migrating to MySQL from another database system, see Section A.8, “MySQL 5.6 FAQ: Migration”, which contains answers to some common questions concerning migration issues.

Installation of MySQL generally follows the steps outlined here:

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

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

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

   Several versions of MySQL are available, and most are available in several distribution formats. You can choose from pre-packaged distributions containing binary (precompiled) programs or source code. When in doubt, use a binary distribution. Oracle also provides access to the MySQL source code for those who want to see recent developments and test new code. To determine which version and type of distribution you should use, see Section 2.1.2, “Which MySQL Version and Distribution to Install”.

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

   For instructions, see Section 2.1.3, “How to Get MySQL”. To verify the integrity of the distribution, use the instructions in Section 2.1.4, “Verifying Package Integrity Using MD5 Checksums or GnuPG”.

4. **Install the distribution.**

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

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

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

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

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

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

- **Unix, Linux, FreeBSD**

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

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

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

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

- **Microsoft Windows**

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

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

- **macOS**

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

  For information on making use of an macOS Launch Daemon to automatically start and stop MySQL, see Section 2.4.3, “Installing a MySQL Launch Daemon”.

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

2.1 General Installation Guidance

The immediately following sections contain the information necessary to choose, download, and verify your distribution. The instructions in later sections of the chapter describe how to install the distribution

that you choose. For binary distributions, see the instructions at Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries” or the corresponding section for your platform if available. To build MySQL from source, use the instructions in Section 2.9, “Installing MySQL from Source”.

2.1.1 Supported Platforms

MySQL platform support evolves over time; please refer to https://www.mysql.com/support/supportedplatforms/database.html for the latest updates.

2.1.2 Which MySQL Version and Distribution to Install

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

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

The naming scheme in MySQL 5.6 uses release names that consist of three numbers and an optional suffix; for example, **mysql-5.6.1-m1**. The numbers within the release name are interpreted as follows:

- The first number (**5**) is the major version number.
- The second number (**6**) is the minor version number. Taken together, the major and minor numbers constitute the release series number. The series number describes the stable feature set.
- The third number (**1**) is the version number within the release series. This is incremented for each new bugfix release. In most cases, the most recent version within a series is the best choice.

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

- **m*N*** (for example, **m1**, **m2**, **m3**, ...) indicates a milestone number. MySQL development uses a milestone model, in which each milestone introduces a small subset of thoroughly tested features. Following the releases for one milestone, development proceeds with another small number of releases that focuses on the next set of features. From one milestone to the next, feature interfaces may change or features may even be removed, based on feedback provided by community members who try these early releases. Features within milestone releases may be considered to be of pre-production quality.
- **rc** indicates a Release Candidate (RC). Release candidates are believed to be stable, having passed all of MySQL's internal testing. New features may still be introduced in RC releases, but the focus shifts to fixing bugs to stabilize features introduced earlier within the series.
- Absence of a suffix indicates a General Availability (GA) or Production release. GA releases are stable, having successfully passed through the earlier release stages, and are believed to be reliable, free of serious bugs, and suitable for use in production systems.

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

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

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

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

  For additional information, see Section 2.9.7, “MySQL Source-Configuration Options”.
- You want to configure `mysqld` without some features that are included in the standard binary distributions. For example, distributions normally are compiled with support for all character sets. If you want a smaller MySQL server, you can recompile it with support for only the character sets you need.
- You want to read or modify the C and C++ code that makes up MySQL. For this purpose, obtain a source distribution.
- Source distributions contain more tests and examples than binary distributions.

2.1.3 How to Get MySQL

Check our downloads page at https://dev.mysql.com/downloads/ for information about the current version of MySQL and for downloading instructions.

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

For Debian-based Linux platforms, MySQL can be installed using the MySQL APT Repository. See Section 2.5.3, “Installing MySQL on Linux Using the MySQL APT Repository” for details.

For SUSE Linux Enterprise Server (SLES) platforms, MySQL can be installed using the MySQL SLES Repository. See Section 2.5.4, “Installing MySQL on Linux Using the MySQL SLES Repository” for details.

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

2.1.4 Verifying Package Integrity Using MD5 Checksums or GnuPG

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

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

The following sections describe how to use these methods.

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

2.1.4.1 Verifying the MD5 Checksum

After you have downloaded a MySQL package, you should make sure that its MD5 checksum matches the one provided on the MySQL download pages. Each package has an individual checksum that

you can verify against the package that you downloaded. The correct MD5 checksum is listed on the downloads page for each MySQL product, and you compare it against the MD5 checksum of the file (product) that you download.

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

Linux and Microsoft Windows examples:

```
shell> md5sum mysql-standard-5.6.51-linux-i686.tar.gz
aaab65abbec64d5e907dcd41b8699945  mysql-standard-5.6.51-linux-i686.tar.gz
```

```
shell> md5.exe mysql-installer-community-5.6.51.msi
aaab65abbec64d5e907dcd41b8699945  mysql-installer-community-5.6.51.msi
```

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

Note

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

2.1.4.2 Signature Checking Using GnuPG

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

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

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

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

mQGiBD4+owwRBAC14GIfUfCyEDSIePvEW3SAFUdJBtoQHH/nJKZyQT7h9bPlUWC3
RODjQReyCITRrdwyrKUGku2FmeVGwn2u2WmDMNABLnpprWPkBdCk96+OmSLN9brZ
fw2vOUgCmYv2hW0hyDHuvYlQA/BThQoADgj8AW6/0Lo7V1W9/8VuHP0gQwCgvzV3
BqOxRznNCRCRxAuAuVztHRcEAJooQK1+iSiunZMYD1WufeXfshc57S/+yeJkegNW
hxwR9pRWVArNYJdDRT+rf2RUe3vpquKNQU/hnEIUHJRQqYHo8gTxvxXNQc7fJYLV
K2HtkrPbP72vwsEKMYhhr0eKCbtLGfls9krjJ6sBgACyP/Vb7hiPwxh6rDZ7ITnE
kYpXBACmWpP8NJTkamEnPCia2ZoOHODANwpUkP43I7jsDmgtobZX9qnrAXw+uNDI
QJEXM6FSbi0LLtZciNlYsafwAPEOMDKpMqAK6IyisNtPvaLd8lH0bPAnWqcyefep
rv0sxxqUEMcM3o7wwgfN83POkDasDbs3pjwPhxvhz6//62zQJ7Q2TXlTUUwgUmVs
ZWFzZSBFbmdpbmVlcmluZyA8bXlzcWwtYnVpbGRAb3NzLm9yYWNsZS5jb20+iGwE
ExECACwCGyMCHgECF4ACGQEGCwkIBwMCBhUKCQgCAwUWAgMBAAUCXEBY+wUJI87e
5AAKCRCMcY07UHLh9RZPAJ9uvm0zlzfCN+DHxHVaoFLFjdVYTQCfborsC9tmEZYa
whhogjeBkZkorbyIaQQTEQIAKQIbIwYLCQgHAwIEFQIIAwQWAgMBAh4BAheAAhkB
BQJTAdRmBQkaZsvLAAoJEIxxjTtQcuH1X4MAoKNLWAbCBUj96637kv6Xa/fJuX5m
AJwPtmgDfjUe2iuhXdTrFEPT19SB6ohmBBMRAgAmAhsjBgsJCAcDAgQVAggDBBYC
AwECHgECF4AFAk53PioFCRP7AhUACgkQjHGNO1By4fUmzACeJdfqgc9gWTUhgmcM
AOmG4RjwuxcAoKfM+U8yMOGELi+TRif7MtKEms6piGkEExECACkCGyMGCwkIBwMC
BBUCCAMEFgIDAQIeAQIXgAIZAQUCUZSROgUJFTchqgAKCRCMcY07UHLh9YtAAJ9X
```

```
rA/ymlmozPZn+A9ls8/uwMcTsQCfaQMNq1dNkhH2kyByc3Rx9/W2xfqJARwEEAEC
AAYFAlAS6+UACgkQ8aIC+GoXHivrWwf/dtLk/x+NC2VMDlg+vOeM0qgG1IlhXZfi
NsEisvvGaz4m8fSFRGe+1bvvfDoKRhxiGXU48RusjixzvBb6KTMuY6JpOVfz9Dj3
H9spYriHa+i6rYySXZIpOhfLiMnTy7NH2OvYCyNzSS/ciIUACIfH/2NH8zNT5CNF
1uPNRs7HsHzzz7pOlTjtTWiF4cq/Ij6Z6CNrmdj+SiMvjYN9u6sdEKGtoNtpycgD
5HGKR+I7Nd/7v56yhaUe4FpuvsNXig86K9tI6MUFS8CUyy7Hj3kVBZOUWVBM053k
nGdALSygQr50DA3jMGKVl4ZnHje2RVWRmFTr5YWoRTMxUSQPMLpBNIkBHAQQAQIA
BgUCU1B+vQAKCRAohbcD0zcc8dWwCACWXXWDXIcAWRUw+j3ph8dr9u3SItljn3wB
c7clpclKWPuLvTz7lGgzlVB0s8hH4xgkSA+zLzl6u56mpUzskFl7f1I3Ac9GGpM4
0M5vmmR9hwlD1HdZtGfbD+wkjlqgitNLoRcGdRf/+U7x09GhSS7Bf339sunIX6sM
gXSC4L32D3zDjF5icGdb0kj+3lCrRmp853dGyA3ff9yUiBkxcKNawpi7Vz3D2ddU
pOF3BP+8NKPg4P2+srKgkFbd4HidcISQCt3rY4vaTkEkLKg0nNA6U4r0YgOa7wIT
SsxFlntMMzaRg53QtK0+YkH0KuZR3GY8B7pi+tlgycyVR7mIFo7riQEcBBABCAAG
BQJWgVd0AAoJEEZu4b/gk4UKk9MH/Rnt7EccPjSJC5CrB2AU5LY2Dsr+PePI2ubP
WsEdG82qSjjGpbhIH8LSg/PzQoGHiFWMmmZWJktRT+dcgLbs3b2VwCNAwCE8jOHd
UkQhEowgomdNvHiBHKHjP4/lF68KOPiO/2mxYYkmpM7BWf3kB57DJ5CTi3/JLoN7
zF40qIs/p09ePvnwStpglbbtUn7XPO+1/Ee8VHzimABom52PkQIuxNiVUzLVn3bS
Wqrd5ecuqLk6yzjPXd2XhDHWC9Twpl68GePru6EzQtusi0m6S/sHgEXqh/IxrFZV
JlljF75JvosZq5zeulr0i6kOij+Y1p6MFffihITZ1gTmk+CLvK2JASIEEAECAAwF
Ak53QS4FAwASdQAACgkQlxC4m8pXrXwJ8Qf/be/UO9mqfoc2sMyhwMpN4/fdBWwf
LkA12FXQDOQMvwH9HsmEjnfUgYKXschZRi+DuHXe1P7l8G2aQLubhBsQf9ejKvRF
TzuWMQkdIq+6Koulxv6ofkCcv3d1xtO2W7nb5yxcpVBPrRfGFGebJvZa58DymCNg
yGtAU6AOz4veavNmI2+GIDQsY66+tYDvZ+CxwzdYu+HDV9HmrJfc6deM0mnBn7SR
jqzxJPgoTQhihTav6q/R5/2p5NvQ/H84OgS6GjosfGc2duUDzCP/kheMRKfzuyKC
OHQPtJuIj8++gfpHtEU7IDUX1So3c9n0PdpeBvclsDbpRnCNxQWU4mBot4kBIgQQ
AQIADAUCToi2GQUDABJ1AAAKCRCXELibyletfLZAB/9oRqx+NC98UQD/wlxCRytz
vi/MuPnbgQUPLHEap10tvEi33S/H/xDR/tcGofY4cjAvo5skZXXeWq93Av7PACUb
zkg0X0eSr2oL6wy66xfov72AwSuX+iUK68qtKaLqRLitM02y8aNRV/ggKvt7UMvG
mOvs5yLaYlobyvGaFC2ClfkNOt2MlVnQZCmnYBCwOktPGkExiu2yZMifcYGxQcpH
KVFG59KeF2cM2d4xYM8HJqkSGGW306LFVSyeRwG+wbttgLpD5bM/T2b3fF/J35ra
CSMLZearRTq8aygPl+XM7MM2eR946aw6jmOsgNBErbvvIdQj6LudAZj+8imcXV2K
iQEiBBABAgAMBQJOmdnRBQMAEnUAAAoJEJcQuJvKV618AvIIAIEF1ZJ+Ry7WOdKF
5oeQ/ynaYUigzN92fW/9zB8yuQlngkFJGidYMbci1tR1siziIVJFusR3ZonqAPGK
/SUta9Y6KWLhmc7c5UnEHklq/NfdMZ2WVSIykXlctqw0sbb+z1ecEd4G8u9j5ill
MO1B36rQayYAPoeXLX8dY4VyFLVGaQ00rWQBYFZrpw16ATWbWGJP332NSfCk4zZq
6kXEW07q0st3YBgAAGdNQyEeZCa4d4pBRSX6189Kjg6GDnIcaiOF6HO6PLr9fRlL
r5ObCgU+G9gEhfiVwDEV9E+7/Bq2pYZ9whhkBqWQzdpXTNTM24uaEhE01EPO5zeC
O214q6mJASIEEAECAAwFAk6rpgEFAwASdQAACgkQlxC4m8pXrXzAhwf/f9O99z16
3Y5FZVIxexyqXQ/Mct9uKHuXEVnRFYbA49dQLD4S73N+zN7gn9jFeQcBo4w8qVUV
94U/ta/VbLkdtNREyplPM4XY8YE5Wfd9bfyg3q1PbEiVjk995sBF+2+To99YYKst
gXPqjlH0jUfEyDmexOj+hsp8Rc63kvkIx36VBa4ONRYFefGAhKDMigL2YAhc1UkG
tkGTuLmlCGwIV6lviDZD3RJf5375VFnaHv7eXfwQxCwE+BxG3CURrjfxjaxMTmMP
yAG2rhDp5oTUEvqDYNbko5UxYOmrSjvF4FzXwqerElXJUkUzSh0pp7RxHB/1lCxD
s7D1F1hlgFQuNIkBIgQQAQIADAUCTrzZHAUDABJ1AAAKCRCXELibyletfMUpB/4s
07dREULIBnA1D6qr3fHsQJNZqbAuyDlvgGGLWzoyEDs+1JMFFlaa+EeLIo1386GU
2DammDC23p3IB79uQhJeD2Z1TcVg4cA64SfF/CHca5coeRSrdAiudzU/cgLGtXIP
/OaFamXgdMxAhloLFbSHPCZkyb00phVa8+xeIVDrK1HByZsNIXy/SSK8U26S2PVZ
2o14fWvKbJ1Aga8N6DuWY/D8P2mi3RAbiuZgfzkmKL5idH/wSKfnFKdTgJzssdCc
1jZEGVk5rFYcWOrJARHeP/tsnb/UxKBEsNtO7e3N2e/rLVnEykVIO066hz7xZK/V
NBSpx3k3qj4XPK41IHy2iQEiBBABAgAMBQJOzqO8BQMAEnUAAAoJEJcQuJvKV618
2twH/0IzjXLxN45nvIfEjC75a+i9ZSLlqR8lsHL4GpEScFKI0a0lT4IVAIY2RKG+
MAs2eHm0UfKuwGs5jluRZ9RqKrc61sY0XQV9/7znY9Db16ghX04JjknOKs/fPi87
rvKkB/QxJWS8qbb/erRmW+cPNjbRxTFPS5JIwFWHA16ieFEpvdAgKV6nfvJVTq1r
jPDcnIA9CJN2SmUFx9Qx3SRc6ITbam1hjFnY6sCh6AUhxLI2f1mq1xH9PqEy42Um
68prRqTyJ7Iox1g/UDDkeeUcAg7T1viTz7uXpS3Wrq4zzo4yOpaJfLDR3pI5g2Zk
SNGTMo6aySE4OABt8i1Pc1Pm6AmJASIEEAECAAwFAk7yPFYFAwASdQAACgkQlxC4
m8pXrXzXiAf9FrXe0lgcPM+tYOWMLhv5gXJi2VUBaLxpyRXm/kJcmxInKq1GCd3y
D4/FLHNu3ZcCz/uklPAbZXWI0O6ewq0LWsRtklmJjWiedH+hGyaTv95VklojRIBd
8nBaJ6M98rljMBHTFwWvjQFVf4FLRJQZqHlvjcCkq2Dd9BWJpGXvr/gpKkmMJYNK
/ftfZRcChb35NI19WRpOhj9u808OPcqKVvZBcPwFGV5cEBzmAC94J7JcD8+S8Ik8
iUJMQGGL3QcmZOBozovh86hj7KTSEBHlLXl832z89H1hLeuLbnXoGLv3zeUFSxkv
1h35LhZLqIMDQRXLuUzxGHMBpLhPyGWRJ4kBIgQQAQIADAUCTwQJFwUDABJ1AAAK
CRCXELibyletfABvB/9Cy69cjOqLGywITs3Cpg//40jmdhSAVxilJivP6J5bubFH
DJlVTx541Dv5h4hTG2BQuueQ4q1VCpSGW+rHcdhPyvmZGRz1rxdQQGh1Dv0Bod2c
3PJVSYPSrRSwCZJkJHOtVRBdjK4mkZb5aFTza+Tor9kxzj4FcXVd4KAS+hHQHYHc
Ar8tt2eOLzqdEFTULeGiSoNn+PVzvzdfhndphK+8F2jfQ2UKuc01O7k0Yn9xZVx0
OG6fE1gStzLv7C5amWLRd8+xh+MN0G8MgNglpBoExsEMMlPBYSUHa6lxpdMNMuib
rIyVncE9X8QOhImt8K0sNn/EdbuldJNGYbDLt7O4iQEiBBABAgAMBQJPFdTcBQMA
EnUAAAoJEJcQuJvKV6184owH+wZ/uLpezXnSxigeH1sig72QEXMrNd5DVHCJdig3
bo+K5YmmN710/m5z+63XKUEWpd6/knajObgckThzWftNeK1SSFQGPmoYZP9EZnSU
7L+/dSUpExbj842G5LYagrCyMGtlxRywWEmbi72TKS/JOK0jLiOdvVy+PHrZSu0D
TVQ7cJh1BmPsbz7zzxjmcI5l+7B7K7RHZHq45nDLoIabwDacj7BXvBK0Ajqz4QyJ
```

```
GQUjXC7q+88I+ptPvOXlE5nI/NbiCJOMI6d/bWN1KwYrC80fZuFaznfQFcPyUaDw
yRaun+K3kEji2wXecq+yMmLUEp01TKsUeOL50HD6hHH07W+JASIEEAECAAwFAk85
bQsFAwASdQAACgkQlxC4m8pXrXwKPQgAlkbUsTr7nkq+haOk0jKpaHWEbRMEGMrB
I3F7E+RDO6V/8y4Jtn04EYDc8GgZMBah+mOgeINq3y8jRMYV5jVtZXv2MWYFUcjM
kVBKeqhi/pGEjmUdmdt3DlPv3Z+fMTMRmAocI981iY/go8PVPg/+nrR6cFK2xxnO
R8TacikJBFeSfkkORg1tDzjjYv1B5ZIEkpplepl5ahJBBq7cpYhTdY6Yk0Sz0J8w
EdffLSaNxrRuWLrRhWzZU7p9bFzfb/7OHc21dJnB7wKv5VvtgE+jiQw9tOKaf5hc
SgRYuF6heu+B25gc5Uu88lo409mZ7oxQ6hDCn7JHvzh0rhmSN+Kid4kBIgQQAQIA
DAUCT0qQrQUDABJ1AAAKCRCXELibyletfC9UB/4o2ggJYM0CLxEpP0GU8UKOh3+/
zm1DN7Qe4kY2iCtF1plKHQaTgt5FlgRCFaiXcVv7WzGz/FnmxonR1leLl+kfRlwy
PPnoI/AWPCy/NO4Cl5KnjsSmsdDUpObwZ4KYsdilZR7ViJu2swdAIgnXBUwrlRJR
7CK4TAKrTeonRgVSrVx8Vt//8/cYj73CLq8oY/KK0iHiQrSwo44uyhdiFIAssjyX
n6/2E+w0zgvPexNSNNROHQ8pjbq+NTY6GwKIGsaej3UTRwQ7psvKXz8y7xdzmOAr
/khGvxB5gjkx02pimjeia8v66aH6rbnojJMAovNUS4EHdHnulv4rovC8Kf9iiQEi
BBABAgAMBQJPVdsaBQMAEnUAAAoJEJcQuJvKV618vVEIALFXPBzcAO1SnQarBLzy
YMVZZumPvSXKnUHAO+6kjApXPJ+qFRdUaSNshZxVKY9Zryblu4ol/fLUTt0CliSD
IxD6L4GXEm4VYYCl4lPO3bVsJnGITLFwQGHM27EmjVoTiD8Ch7kPq2EXr3dMRgzj
pdz+6aHGSUfOdLTPXufDvW83bEWGaRVuTJKw+wIrcuRqQ+ucWJgJGwcE4zeHjZad
Jx1XUm1X+BbI73uiQussyjhhQVVNU7QEdrjyuscaZ/H38wjUwNbylxDPB4I8quC1
knQ0wSHr7gKpM+E9nhiS14poRqU18u78/sJ2MUPXnQA6533IC238/LP8JgqB+BiQ
BTSJASIEEAECAAwFAk9ng3cFAwASdQAACgkQlxC4m8pXrXxQRAf/UZlkkpFJj1om
9hIRz7gS+l7YvTaKSzpo+TBcx3C7aqKJpir6TlMK9cb9HGTHo2Xp1N3FtQL72NvO
6CcJpBURbvSyb4i0hrm/YcbUC4Y3eajWhkRS3iVfGNFbc/rHthViz0r6Y5lhXX16
aVkDv5CIFWaF3BiUK0FnHrZiy4FPacUXCwEjv3uf8MpxV5oEmo8Vs1h4TL3obyUz
qrImFrEMYE/12lkE8iR5KWCaF8eFyl56HL3PPl90JMQBXzhwsFoWCPuwjfM5w6sW
Ll//zynwxtlJ9CRz9c2vK6aJ8DRu3OfBKN1iiEcNEynksDnNXErn5xXKz3p5pYdq
e9BLzUQCDYkBIgQQAQIADAUCT3inRgUDABJ1AAAKCRCXELibyletfGMKCADJ97qk
geBntQ+tZtKSFyXznAugYQmbzJld8U6eGSQnQkM40Vd62UZLdA8MjlWKS8y4A4L2
0cI14zs5tKG9Q72BxQOw5xkxlLASw1/8WeYEbw7ZA+sPG//q9v3kIkru3sv64mMA
enZtxsykexRGyCumxLjzlAcL1drWJGUYE2Kl6uzQS7jb+3PNBloQvz6nb3YRZ+Cg
Ly9D41SIK+fpnV8r4iqhu7r4LmAQ7Q1DF9aoGaYvn2+xLGyWHxJAUet4xkMNOLp6
k9RF1nbNe4I/sqeCB25CZhCTEvHdjSGTD2yJR5jfoWkwO9w8DZG1Q9WrWqki4hSB
l0cmcvO34pC1SJYziQEiBBABAgAMBQJPinQFBQMAEnUAAAoJEJcQuJvKV618CFEI
AJp5BbcV7+JBMRSvkoUcAWDoJSP2ug9zGw5FB8J90PDefKWCKs5Tjayf2TvM5ntq
5DE9SGaXbloIwa74FoZlgqlhMZ4AtY9Br+oyPJ5S844wpAmWMFc6NnEPFaHQkQ+b
dJYpRVNd9lzagJP261P3S+S9T2UeHVdOJBgWIq9Mbs4lnZzWsnZfQ4Lsz0aPqe48
tkU8hw+nflby994qIwNOlk/u+I/lJbNz5zDY91oscXTRl2jV1qBgKYwwCXxyB3j9
fyVpRl+7QnqbTWcCICVFL+uuYpP0HjdoKNqhzEguAUQQLOB9msPTXfa2hG+32ZYg
5pzI5V7GCHq0KO6u5Ctj3TGJASIEEAECAAwFAk+cQEEFAwASdQAACgkQlxC4m8pX
rXzi7AgAx8wJzNdD7UlgdKmrAK//YqH7arSssb33Xf45sVHDpUVA454DXeBrZpi+
zEuo03o5BhAuf38cwfbkV6jN1mC2N0FZfpy4v7RxHKLYr7tr6r+DRn1L1giX5ybx
CgY0fLAxkwscWUKGKABWxkz9b/beEXaO2rMt+7DBUdpAOP5FNRQ8WLRWBcMGQiaT
S4YcNDAiNkrSP8CMLQP+04hQjahxwCgBnksylciqz3Y5/MreybNnTOrdjVDsF0Oe
t0uLOiWXUZV1FfaGIdb/oBQLg+e1B74p5+q3aF8YI97qAZpPa1qiQzWIDX8LX9QX
EFyZ3mvqzGrxkFoocXleNPgWT8fRuokBIgQQAQIADAUCT64N/QUDABJ1AAAKCRCX
ELibyletfDOGCACKfcjQlSxrWlEUrYYZpoBP7DE+YdlIGumt5l6vBmxmt/5OEhqr
+dWwuoiyC5tm9CvJbuZup8anWfFzTTJmPRPsmE4z7Ek+3CNMVM2wIynsLOt1pRFK
4/5RNjRLbwI6EtoCQfpLcZJ//SB56sK4DoFKH28Ok4cplESPnoMqA3QafdSEA/FL
qvZV/iPgtTz7vjQkMgrXAIUM4fvKe3iXkAExGXtmgdXHVFoKmHrxJ2DTSvM7/19z
jGJeu2MhIKHyqEmCk6hLjxyCE5pAH59KlbAQOP1bS28xlRskBApm2wN+LOZWzC62
HhEReQ50inCGuuubK0PqUQnyYc+lUFxrFpcliQEiBBABAgAMBQJPv9lVBQMAEnUA
AAoJEJcQuJvKV618AzgH/iRFFCi4qjvoqji1fi7yNPZVOMMO2H13Ks+AfcjRtHuV
aa30u50ND7TH+XQe6yerTapLh3aAm/sNP99aTxIuwRSlyKEoDs93+XVSgRqPBgbF
/vxv0ykok3p6L9DxFO/w5cL8JrBhMZoJrEkIBFkwN8tWlcXPRFQvcdBYv3M3DTZU
qY+UHnOxHvSzsl+LJ0S9Xcd9C5bvYfabmYJvG5eRS3pj1L/y3a6yw6hvY+JtnQAk
t05TdeHMIgQH/zb8V9wxDzmE0un8LyoC2Jx5TpikQsJSejwK6b3coxVBlngku6+C
qDAimObZLw6H9xYYIK0FoJs7j5bQZEwUO7OLBgjcMOqJASIEEAECAAwFAk/Rpc8F
AwASdQAACgkQlxC4m8pXrXw49Qf/TdNbun2htQ+cRWarszOx8BLEiW/x6PVyUQpZ
nV/0qvhKzlJUjM9hQPcA0AsOjhqtCN6Cy8KXbK/TvPm9D/Nk6HWwD1PomzrJVFk2
ywGFIuTR+lluKSp7mzm5ym0wJs5cPq731Im31RUQU8ndjLrq9YOf5FVL8NqmcOAU
4E8d68BbmVCQC5MMr0901FKwKznShfpy7VYN25/BASj8dhnynBYQErqToOJB6Cnd
JhdTlbfR4SirqAYZZg3XeqGhByytEHE1x7FMWWFYhdNtsnAVhYBbWqAzBs8lF9Jd
Mhaf0VQU/4z10gVrRtXLR/ixrCi+P4cM/fOQkqd6pwqWkaXt6okBIgQQAQIADAUC
T+NxIAUDABJ1AAAKCRCXELibyletfFBBCAC6+0TUJDcNaqOxOG1KViY6KYg9NCL8
pwNK+RKNK/N1V+WGJQH7qDMwRoOn3yogrHax4xIeOWiILrvHK0O6drS1DjsymIhR
Sm2XbE/8pYmEbuJ9vHh3b/FTChmSAO7dDjSKdWD3dvaY8lSsuDDqPdTX8FzOfrXC
M22C/YPg7oUG2A5svE1b+yismP4KmVNWAepEuPZcnEMPFgop3haHg9X2+mj/btDB
Yr6p9kAgIY17nigtNTNjtI0dMLu43aIzedCYHqOlNHiB049jkJs54fMGBjF9qPtc
m0k44xyKd1/JXWMdNUmtwKsChAXJS3YOciMgIx6tqYUTndrP4I6q1rfriQEiBBAB
AgAMBQJP9T1VBQMAEnUAAAoJEJcQuJvKV618J9wIAI1lId9SMbEHF6PKXRe154lE
pap5imMU/lGTj+9ZcXmlf8o2PoMMmb3/E1k+EZUaeSBoOmjS8C2gwd5XFwRrlwAD
RlK/pG5XsL4h5wmN2fj1ororrJXvqH427PLRQK9yzdwG4+9HTBOxjoS8qZT9plyK
```

```
AJZzAydAMqyseRHgNo0vMwlgrs4ojo+GcFGQHrF3IaUjvVfUPOmIj7afopFdIZmI
GaSF0TXBzqcZ1chFv/eTBcIuIKRvlaDee5FgV7+nLH2nKOARCLvV/+8uDi2zbr83
Ip5x2tD3XuUZ0ZWxD0AQWcrLdmGb4lkxbGxvCtsaJHaLXWQ2m760RjIUcwVMEBKJ
ASIEEAECAAwFAlAGYWsFAwASdQAACgkQlxC4m8pXrXwyVAgAvuvEl6yuGkniWOlv
uHEusUv/+2GCBg6qV+IEpVtbTCCgiFjYR5GasSp1gpZ5r4BocOlbGdjdJGHTpyK8
xD1i+6qZWUYhNRg2POXUVzcNEl2hhouwPLOifcmTwAKU76TEv3L5STviL3hWgUR2
yEUZ3Ut0IGVV6uPER9jpR3qd6O3PeuFkwf+NaGTye4jioLAy3aYwtZCUXzvYmNLP
90K4y+5yauZteLmNeq26miKC/NQu4snNFClPbGRjHD1ex9KDiAMttOgN4WEq7srT
rYgtT531WY4deHpNgoPlHPuAfC0H+S6YWuMbgfcb6dV+Rrd8Ij6zM3B/PcjmsYUf
OPdPtIkBIgQQAQIADAUCUBgtfQUDABJ1AAAKCRCXELibyletfAm3CACQlw21Lfeg
d8RmIITsfnFG/sfM3MvZcjVfEAtsY3fTK9NiyU0B3yX0PU3ei37qEW+50BzqiStf
5VhNvLfbZR+yPou7o2MAP31mq3Uc6grpTV64BRIkCmRWg40WMjNI1hv7AN/0atgj
ATYQXgnEw7mfFb0XZtMTD6cmrz/A9nTPVgZDxzopOMgCCC1ZK4Vpq9FKdCYUaHpX
3sqnDf+gpVIHkTCMgWLYQOeX5Nl+fgnq6JppaQ3ySZRUDr+uFUs0uvDRvI/cn+ur
ri92wdDnczjFumKvz/cLJAg5TG2Jv1Jx3wecALsVqQ3gL7f7vr1OMaqhI5FEBqdN
29L9cZe/ZmkriQEiBBIBCgAMBQJVoNxyBYMHhh+AAAoJEEoz7NUmyPxLD1EH/2eh
7a4+8A1lPLy2L9xcNt2bifLfFP2pEjcG6ulBoMKpHvuTCgtX6ZPdHpM7uUOje/F1
CCN0IPB533U1NIoWIKndwNUJjughtoRM+caMUdYyc4kQm29Se6hMPDfyswXE5Bwe
PmoOm4xWPVOH/cVN04zyLuxdlQZNQF/nJg6PMsz4w5z+K6NGGm24NEPcc72iv+6R
Uc/ry/7v5cVu4hO5+r104mmNV5yLecQF13cHy2JlngIHXPSlxTZbeJX7qqxE7TQh
5nviSPgdk89oB5jFSx4g1efXiwtLlP7lbDlxHduomyQuH9yqmPZMbkJt9uZDc8Zz
MYsDDwlc7BIe5bGKfjqJAhwEEAECAAYFAlSanFIACgkQdzHqU52lcqLdvg//cAEP
qdN5VTKWEoDFjDS4I6t8+0KzdDWDacVFwKJ8RAo1M2SklDxnIvnzysZd2VHp5Pq7
i4LYCZo5lDkertQ6LwaQxc4X6myKY4LTA652ObFqsSfgh9kW+aJBBAyeahPQ8CDD
+Yl23+MY5wTsj4qt7KffNzy78vLbYnVnvRQ3/CboVix0SRzg0I3Oi7n3B0lihvXy
5goy9ikjzZevejMEfjfeRCgoryy9j5RvHH9PF3fJVtUtHCS4f+kxLmbQJ1XqNDVD
hlFzjz8oUzz/8YXy3im5MY7Zuq4P4wWiI7rkIFMjTYSpz/evxkVlkR74qOngT2pY
VHLyJkqwh56i0aXcjMZiuu2cymUt2LB9IsaMyWBNJjXr2doRGMAfjuR5ZaittmML
yZwix9mWVk7tkwlIxmT/IW6Np0qMhDZcWYqPRpf7+MqY3ZYMK4552b8aDMjhXrnO
OwLsz+UI4bZa1r9dguIWIt2C2b5C1RQ9AsQBPwg7h5P+HhRuFAuDKK+vgV8FRuzR
JeKkFqwB4y0Nv7BzKbFKmP+V+/krRv+/Dyz9Bz/jyAQgw02u1tPupH9BGhlRyluN
yCJFTSNj7G+OLU0/l4XNph5OOC7sy+AMZcsL/gsT/TXCizRcCuApNTPDaenACpbv
g8OoIzmNWhh4LXbAUHCKmY//hEw9PvTZA1xKHgyJAhwEEgECAAYFAlJYsKQACgkQ
oirk60MpxUV2XQ//b2/uvThkkbeOegusDC4AZfjnL/V3mgk4iYy4AC9hum0R9oNl
XDR51P1TEw9mC1btHj+7m7Iq1a5ke5wIC7ENZiilr0yPqeWgL5+LC98dz/L85hqA
wIoGeOfMhrlaVbAZEj4yQTAJDA35vZHVsQmp87il0m+fZX04OBLXBzw86EoAAZ7Q
EoH4qFcT9k1T363tvNnIm3mEvkQ5WjE1R9uchJa1g7hdlNQlVkjFmPZrJK9fl4z5
6Dto89Po4Sge48jDH0pias4HATYHsxW819nz5jZzGcxLnFRRR5iITVZi9qzsHP7N
bUh3qxuWCHS9xziXpOcSZY848xXw63Y5jDJfpzupzu/KHj6CzXYJUEEqp9MluoGb
/BCCEPzdZ0ovyxFutM/BRcc6DvE6sTDF/UES21ROqfuwtJ6qJYWX+lBIgyCJvj4o
RdbzxUleePuzqCzmwrIXtoOKW0Rlj4SCeF9yCwUMBTGW5/nCLmN4dwf1KW2RP2Eg
4ERbuUy7QnwRP5UCl+0ISZJyYUISfg8fmPIdQsetUK9Cj+Q5jpB2GXwELXWnIK6h
K/6jXp+EGEXSqdIE53vAFe7LwfHiP/D5M71D2h62sdIOmUm3lm7xMOnM5tKlBiV+
4jJSUmriCT62zo710+6iLGqmUUYlEll6Ppvo8yuanXkYRCFJpSSP7VP0bBqIZgQT
EQIAJgUCTnc9dgIbIwUJEPPzpwYLCQgHAwIEFQIIAwQWAgMBAh4BAheAAAoJEIxx
jTtQcuH1Ut4AoIKjhdf70899d+7JFq3LD7zeeyI0AJ9Z+YyE1HZSnzYi73brScil
bIV6sbQ7TXlTUUwgUGFja2FnZSBzaWduaW5nIGtleSAod3d3Lm15c3FsLmNvbSkg
PGJ1aWxkQG15c3FsLmNvbT6IbwQwEQIALwUCTnc9rSgdIGJ1aWxkQG15c3FsLmNv
bSB3aWxsIHN0b3Agd29ya2luZyBzb29uAAoJEIxxjTtQcuH1tT0An3EMrSjEkUv2
9OX05JkLiVfQr0DPAJwKtL1ycnLPv15pGMvSzav8JyWN3IhlBBMRAgAdBQJHrJS0
BQkNMFioBQsHCgMEAxUDAgMWAgECF4AAEgkQjHGNO1By4fUHZUdQRwABAa6SAJ9/
PgZQSPNeQ6LvVVzCALEBJOBt7QCffgs+vWP18JutdZc7XiawgAN9vmmITAQTEQIA
DAUCPj6j0QWDCWYAuwAKCRBJUOEqsnKR8iThAJ9ZsR4o37dNGyl77nEqP6RAlJqa
YgCeNTPTEVY+VXHR/yjfyo0bVurRxT2ITAQTEQIADAUCPkKCAwWDCWIiiQAKCRC2
9c1NxrokP5aRAKCIaaegaMyiPKenmmm8xeTJSR+fKQCgrv0TqHyvCRINmi6LPucx
GKwfy7KIRgQQEQIABgUCP6zjrwAKCRCvxSNIeIN0D/aWAKDbUiEgwwAFNh2n8gGJ
Sw/8lAuISgCdHMzLAS26NDP8T2iejsfUOR5sNriIRgQQEQIABgUCP7RDdwAKCRCF
lq+rMHNOZsbDAJ0WoPV+tWILtZG3wYqg5LuHM03faQCeKuVvCmdPtro06xDzeeTX
VrZ14+GIRgQQEQIABgUCQ1uz6gAKCRCL2C5vMLlLXH90AJ0QsqhdAqTAk3SBnO2w
zuSOwiDIUwCdFExsdDtXf1cL3Q4ilo+OTdrTW2CIRgQTEQIABgUCRPEzJgAKCRD2
ScT0YJNTDApxAKCJtqT9LCHFYfWKNGGBgKjka0zi9wCcCG3MvnvBzDUqDVebudUZ
61Sont+ITAQQEQIADAUCQYHLAQWDBiLZiwAKCRAYWdAfZ3uh7EKNAJwPywk0Nz+Z
Lybw4YNQ7H1UxZycaQCePVhY4P5CHGjeYj9SX2gQCE2SNx+ITAQQEQIADAUCQYHL
NAWDBiLZWAAKCRCBwvfr4hO2kiIjAJ0VU1VQHzF7yYVeg+bh31nng9OOkwCeJI8D
9mx8neg4wspqvgXRA8+t2saITAQQEQIADAUCQYHLYgWDBiLZKgAKCRBrcOzZXcP0
cwmqAJsFjOvkY9c5eA/zyMrOZ1uPB6pd4QCdGyzgbYb/eoPu6FMvVI9PVIeNZReI
TAQQEQIADAUCQdCTJAWDBdQRaAAKCRB9JcoKwSmnwmJVAKCG9a+Q+qjCzDzDtZKx
5NzDW1+W+QCeL68seX8OoiXLQuRlifmPMrV2m9+ITAQQEQIADAUCQitbugWDBXlI
0gAKCRDmG6SJFeu5q/MTAKCTMvlCQtLKlzD0sYdwVLHXJrRUvgCffmdeS6aDpwIn
U0/yvYjg1xlYiuqITAQSEQIADAUCQCpZOgWDB3pLUgAKCRA8oR80lPr4YSZcAJwP
4DncDk4YzvDvnRbXW6SriJn1yQCdEy+d0CqfdhM7HGUs+PZQ9mJKBKqITAQSEQIA
DAUCQD36ugWDB2ap0gAKCRDy11xj45xlnLLfAKC0NzCVqrbTDRw25cUss14RRoUV
```

```
PACeLpEc3zSahJUB0NNGTNlpwlTczlCITAQSEQIADAUCQQ4KhAWDBpaaCAAKCRA5
yiv0PWqKX/zdAJ4hNn3AijtcAyMLrLhlZQvib551mwCgw6FEhGLjZ+as0W681luc
wZ6PzW+ITAQSEQIADAUCQoClNAWDBSP/WAAKCRAEDcCFfIOfqOMkAJwPUDhS1eTz
gnXclDKgf353LbjvXgCeLCWyyj/2d0gIk6SqzaPl2UcWrqiITAQTEQIADAUCPk1N
hAWDCVdXCAAKCRAtu3a/rdTJMwUMAKCVPkbk1Up/kyPrlsVKU/Nv3bOTZACfW5za
HX38jDCuxsjIr/084n4kw/uITAQTEQIADAUCQdeAdgWDBc0kFgAKCRBm79vIzYL9
Pj+8AJ9d7rvGJIcHzTCSYVnaStv6jP+AEACeNHa5yltqieRBCCcLcacGqYK81omI
TAQTEQIADAUCQhiBDgWDBYwjfgAKCRB2wQMcojFuoaDuAJ9CLYdysef7IsW42UfW
hI6HjxkzSgCfeEpXS4hEmmGicdpRiJQ/W21aB0GIZQQTEQIAHQULBwoDBAMVAwID
FgIBAheABQJLcC/KBQkQ8/OnABIHZUdQRwABAQkQjHGNO1By4fWw2wCeJilgEarL
8eEyfDdYTyRdqE45HkoAnjFSZY8Zg/iXeErHI0r04BRukNVgiHsEMBECADsFAkJ3
NfU0HQBPb3BzLi4uIHNob3VsZCBoYXZlIGJlZW4gbG9jYWwhIEknbSAqc28qIHN0
dXBpZC4uLgAKCRA5yiv0PWqKX+9HAJ0WjTx/rqgouK4QCrOV/2IOU+jMQQCfYSC8
JgsIIeN8aiyuStTdYrk0VWCIjwQwEQIATwUCRW8Av0gdAFNob3VsZCBoYXZlIGJl
ZW4gYSBsb2NhbCBzaWduYXR1cmUsIG9yIHNvbWV0aGluZyAtIFdURiB3YXMgSSB0
aGlua2luZz8ACgkQOcor9D1qil+g+wCfcFWoo5qUl4XTE9K8tH3Q+xGWeYYAnjii
KxjtOXc0ls+BlqXxbfZ9uqBsiQIiBBABAgAMBQJBgcuFBYMGItkHAAoJEKrj5s5m
oURoqC8QAIISudocbJRhrTAROOPoMsReyp46Jdp3iL1oFDGcPfkZSBwWh8L+cJjh
dycIwwSeZ1D2h9S5Tc4EnoE0khsS6wBpuAuih5s//coRqIIiLKEdhTmNqulkCH5m
imCzc5zXWZDW0hpLr2InGsZMuh2QCwAkB4RTBM+r18cUXMLV4YHKyjIVaDhsiPP/
MKUj6rJNsUDmDq1GiJdOjySjtCFjYADlQYSD7zcd1vpqQLThnZBESvEoCqumEfOP
xemNU6xAB0CL+pUpB40pE6Un6Krr5h6yZxYZ/N5vzt0Y3B5UUMkgYDSpjbulNvaU
TFiOxEU3gJvXc1+h0BsxM7FwBZnuMA8LEA+UdQb76YcyuFBcROhmcEUTiducLu84
E2BZ2NSBdymRQKSinhvXsEWlH6Txm1gtJLynYsvPi4B4JxKbb+awnFPusL8W+gfz
jbygeKdyqzYgKj3M79R3geaY7Q75Kxl1UogiOKcbI5VZvg47OQCWeeERnejqEAdx
EQiwGA/ARhVOP/1l0LQA7jg2P1xTtrBqqC2ufDB+v+jhXaCXxstKSW1lTbv/b0d6
454UaOUV7RisN39pE2zFvJvY7bwfiwbUJVmYLm4rWJAEOJLIDtDRtt2h8JahDObm
3CWkpadjw57S5v1c/mn+xV9yTgVx5YUfC/788L1HNKXfeVDq8zbAiQIiBBMBAgAM
BQJCnwocBYMFBZpwAAoJENjCCglaJFfPIT4P/25zvPp8ixqV85igs3rRqMBtBsj+
5EoEW6DJnlGhoi26yf1nasC2frVasWG7i4JIm0U3WfLZERGDjR/nqlOCEqsP5gS3
43N7r4UpDkBsYh0WxH/ZtST5llFK3zd7XgtxvqKL98l/OSgijH2W2SJ9DGpjtO+T
iegq7igtJzw7Vax9z/LQH2xhRQKZR9yernwMSYaJ72i9SyWbK3k0+e95fGnlR5pF
zlGq320rYHgD7v9yoQ2t1klsAxK6e3b7Z+RiJG6cAU8o8F0kGxjWzF4v8D1op7S+
IoRdB0Bap01ko0KLyt3+g4/33/2UxsW50BtfqcvYNJvU4bZns1YSqAgDOOanBhg8
Ip5XPlDxH6J/3997n5JNj/nk5ojfd8nYfe/5TjflWNiput6tZ7frEki1wl6pTNbv
V9C1eLUJMSXfDZyHtUXmiP9DKNpsucCUeBKWRKLqnsHLkLYydsIeUJ8+ciKc+EWh
FxEY+Ml72cXAaz5BuW9L8KHNzZZfez/ZJabiARQpFfjOwAnmhzJ9r++TEKRLEr96
taUI9/8nVPvT6LnBpcM38Td6dJ639YvuH3ilAqmPPw50YvglIEe4BUYD5r52Seqc
8XQowouGOuBX4vs7zgWFuYA/s9ebfGaIw+uJd/56Xl9ll6q5CghqB/yt1EceFEnF
CAjQc2SeRo6qzx22iEYEEBECAAYFAkSAbycACgkQCywYeUxD5vWDcACfQsVk/XGi
ITFyFVQ3IR/3Wt7zqBMAoNhso/cX8VUfs2BzxPvvGS3y+5Q9iEYEEBECAAYFAkUw
ntcACgkQOI4l6LNBlYkyFgCbBcw5gIii0RTDJsdNiuJDcu/NPqEAniSq9iTaLjgF
HZbaizUU8arsVCB5iEYEEBECAAYFAkWho2sACgkQu9u2hBuwKr6bjwCfa7ZK6O+X
mT08Sysg4DEoZnK4L9UAoLWgHuYg35wbZYx+ZUTh98diGU/miF0EExECAB0FAj4+
owwFCQlmAYAFCwcKAwQDFQMCAxYCAQIXgAAKCRCMcY07UHLh9XGOAJ4pVME15/DG
rUDohtGv2z8a7yv4AgCeKIp0jWUWE525QocBWms7ezxd6syIXQQTEQIAHQUCR6yU
zwUJDTBYqAULBwoDBAMVAwIDFgIBAheAAAoJEIxxjTtQcuH1dCoAoLC6RtsD9K3N
7NOxcp3PYOzH2oqzAKCFHn0jSqxk7E8by3sh+Ay8yVv0BYhdBBMRAgAdBQsHCgME
AxUDAgMWAgECF4AFAkequSEFCQ0ufRUACgkQjHGNO1By4fUdtwCfRNcueXikBMy7
tE2BbfwEyTLBTFAAnifQGbkmcARVS7nqauGhe1ED/vdgiF0EExECAB0FCwcKAwQD
FQMCAxYCAQIXgAUCS3AuZQUJEPPyWQAKCRCMcY07UHLh9aA+AKCHDkOBKBrGb8tO
g9BIub3LFhMvHQCeIOOot1hHHUlsTIXAUrD8+ubIeZaJARwEEgECAAYFAkvCIgMA
CgkQ3PTrHsNvDi8eQgf/dSx0R9Klozz8iK79w00NOsdoJY0Na0NTFmTbqHg30XJo
G62cXYgc3+TJnd+pYhYi5gyBixF/L8k/kPVPzX9W0YfwChZDsfTw0iDVmGxOswiN
jzSo0lhWq86/nEL30Khl9AhCC1XFNRw8WZYq9Z1qUXHHJ2rDARaedvpKHOjzRY0N
dx6R2zNyHDx2mlfCQ9wDchWEuJdAv0uHrQ0HV9+xq7lW/Q3L/V5AuU0tiowyAbBL
PPYrB6x9vt2ZcXS7BOy8SfQ1i8W2QDQ/Toork4YwBiv6WCW/ociy7paAoPOWV/Nf
2S6hDispeecbk7wqpbUj5klDmwrlgB/jmoAXWEnbsYkBIgQQAQIADAUCSSpooAUD
ABJ1AAAKCRCXELibyletfFOMCACpP+OVZ7lH/cNY+373c4FnSI0/S5PXS0ABgdd4
BFWRFWKrWBeXBGc8sZfHOzVEwkzV96iyHbpddeAOAkEA4OVPW1MMFCmlHxi2s9/N
JrSrTPVfQOH5fR9hn7Hbpq/ETw0IoX1FKo7vndMnHZnFEnI+PDXLcdMYQgljYzhT
xER4vYY0UKu8ekSshUy4zOX7XSJxwqPUvps8qs/TvojIF+vDJvgFYHVkgvS+shp8
Oh/exg9vKETBlgU87Jgsqn/SN2LrR/Jhl0aLd0G0iQ+/wHmVYdQUMFaCZwk/BKNa
XPzmGZEUZ3RNbYa19Mo7hcE3js76nh5YMxFvxbTggVu4kdFkiQEiBBABAgAMBQJK
M06IBQMAEnUAAAoJEJcQuJvKV618F4gH/innejIHffGMk8jYix4ZZT7pW6ApyoI+
N9Iy85H4L+8rVQrtcTHyq0VkcN3wPSwtfZszUF/0qP6P8sLJNJ1BtrHxLORYjJPm
gveeyHPzA2oJl6imqWUTiW822fyjY/azwhvZFzxmvbFJ+r5N/Z57+Ia4t9LTSqTN
HzMUYaXKDaAqzZeK7P0E6XUaaeygbjWjBLQ1O0ezozAy+Kk/gXApmDCGFuHSFe7Z
mgtFcbXLM2XFQpMUooETD2R8MUsd+xnQsff/k6pQOLxi+jUEsWSr/iqmvlk6gZ4D
pemBjuhcXYlxJYjUaX9Zmn5s+ofF4GFxRqXoY7l9Z+tCM9AX37lm6S+JASIEEAEC
AAwFAkpEcgoFAwASdQAACgkQlxC4m8pXrXz2mgf/RQkpmMM+5r8znx2TpRAGHi5w
ktvdFxlvPaOBWE28NDwTrpcoMqo9kzAiuvEQjVNihbP21wR3kvnQ84rTAH0mlC2I
```

```
uyybggpqwzOUl+Wi0o+vk8ZA0A0dStWRN8uqneCsd1XnqDe1rvqC4/9yY223tLmA
kPvz54ka2vX9GdJ3kxMWewhrVQSLCktQpygU0dujGTDqJtnk0WcBhVF9T87lv3W2
eGdPielzHU5trXezmGFj21d56G5ZFK8co7RrTt4qdznt80glh1BTGmhLlzjMPLTe
dcMusm3D1QB9ITogcG94ghSf9tEKmmRJ6OnnWM5Kn9KcL63E5oj2/lY9H54wSYkB
IgQQAQIADAUCSlY+RwUDABJ1AAAKCRCXELibyletfOOQB/0dyJBiBjgf+8d3yNID
pDktLhZYw8crIjPBVdOgX12xaUYBTGcQITRVHSggzffDA5BQXeUuWhpL4QB0uz1c
EPPwSMiWiXlBtwF5q6RVf3PZGJ9fmFuTkPRO7SruZeVDo9WP8HjbQtOLukYf566e
grzAYR9p74UgWftpDtmrqrRTobiuvsFBxosbeRCvEQCrN0n+p5D9hCVB88tUPHnO
WA4mlduAFZDxQWTApKQ92frHiBqy+M1JFezz2OM3fYN+Dqo/Cb7ZwOAA/2dbwS7o
y4sXEHbfWonjskgPQwFYB23tsFUuM4uZwVEbJg+bveglDsDStbDlfgArXSL/0+ak
lFcHiQEiBBABAgAMBQJKaAqEBQMAEnUAAAoJEJcQuJvKV618rH0H/iCciD4U6YZN
JBj0GN7/Xt851t9FWocmcaC+qtuXnkFhplXkxZVOCU4VBMs4GBoqfIvagbBTyfV4
Di+W8Uxr+/1jiu3l/HvoFxwdwNkGG6zNBhWSjdwQpGwPvh5ryV1OfLX/mgQgdDmx
vqz5+kFDUj4m7uLaeuU2j1T0lR4zU0yAsbt7J3hwfqJCXHOc9bm5nvJwMrSm+sdC
TP5HjUlwHr9mTe8xuZvj6sO/w0P4AqIMxjC9W7pT9q0ofG2KSTwt7wFbh05sbG4U
QYOJe4+Soh3+KjAa1c0cvmIh4cKX9qfCWwhhdeNfh1A9VTHhnl5zTv/UjvnQtjhl
H/Fq1eBSKcSJASIEEAECAAwFAkp5LgoFAwASdQAACgkQlxC4m8pXrXwY6wgAg3f8
76L3qDZTYlFAWs3pXBl8GsUr1DEkTlEDZMZKDM3wPmhaWBR1hMA3y6p3aaCUyJIJ
BEneXzgyU9uqCxXpC78d5qc3xs/Jd/SswzNYuvuzLYOw5wN5L31SLmQTQ8KqE0uo
RynBmtDCQ4M2UKifSnv+0+3mPh85LVAS481GNpL+VVfCYtKesWNu40+98Yg6L9NG
WwRTfsQbcdokZo44Jz7Y7f81ObC4r/X1DgPj2+d4AU/plzDcdrbINOyprs+7340e
cnaGO4Lsgd19b1CvcgJgltRquu3kRvd+Ero2RYpDv6GVK8Ea0Lto4+b/Ae8cLXAh
QnaWQCEWmw+AU4Jbz4kBIgQQAQIADAUCSo5fvQUDABJ1AAAKCRCXELibyletfA08
B/9w8yJdc8K+k07U30wR/RUg3Yb2lBDygmy091mVsyB0RGixBDXEPOXBqGKAXiV1
QSMAXM2VKRsuKahY2HFkPbyhZtjbdTa7Pr/bSnPvRhAh9GNWvvRg2Kp3qXDdjv9x
ywEghKVxcEIVXtNRvpbqRoKmHzIExvUQck5DM1VwfREeYIoxgs4035WADhVMdngQ
S2Gt8P2WaU/p8EZhFGg6X8KtOlD68zGboaJe0hj2VDc+Jc+KdjRfE3fW5ITOid/o
DkUaIW6tB3WkXb0g6D/2hrEJbX3headChHKSB8eQdOR9bcCJDhhU8csd501qmrhC
ctmvlpeWQZdIQdk6sABPWeeCiQEiBBABAgAMBQJKoBJHBQMAEnUAAAoJEJcQuJvK
V618Ml8H/1D88/g/p9fSVor4Wu5WlMbg8zEAik3BIxQruEFWda6nART6M9E7e+P1
++UHZsWYs6l9ROpWxRLG1Yy9jLec2Y3nUtb20m65p+IVeKR2a9PHW35WZDV9dOYP
GZabKkO1clLeWLVgp9LRjZ+AeRG+ljHqsULXro1dwewLTB/gg9I2vgNv6dKxyKak
nM/GrqZLATAq2KoaE/u/6lzRFZIzZnLtjZh8X7+nS+V8v9IiY4ntrpkrbvFk30U6
WJp79oBIWwnW/84RbxutRoEwSar/TLwVRkcZyRXeJTapbnLGnQ/lDO1o1d7+Vbjd
q/Sg/cKHHf7NthCwkQNsCnHL0f51gZCJASIEEAECAAwFAkqoEAAFAwASdQAACgkQ
lxC4m8pXrXwE/Af/XD4R/A5R6Ir/nCvKwCTKJmalajssuAcLEa2pMnFZYO/8rzLO
+Gp8p0qFH9C4LFwA0NvR5q6X/swuROf4zxljSvNcdlQVaAfJ2ZDEgJ5GXzsPplrv
SAI9jS3LL7fSWDZgKuUe0a4qx7A0NgyGMUYGhP+QlRFa8vWEBI9fANd/0mMqAeBV
qQyOH0X1FiW1Ca2Jn4NKfuMy9GEvRddVIbB1LvoNVtXPNzeeKMyNb9Jdx1MFWssy
COBP2DayJKTmjvqPEc/YOjOowoN5sJ/jn4mVSTvvlTooLiReSs6GSCAjMVxN7eYS
/Oyq6Iu1JDcJvmB8N2WixAZtAVgF8OA7CWXKVYkBIgQQAQIADAUCSrnHiQUDABJ1
AAAKCRCXELibyletfPChB/9uECti1dZeNuFsd0/RuGyRUVlrrhJE6WCcOrLO9par
rPbewbKBmjSzB0MygJXGvcC06mPNuquJ7/WpxKsFmfg4vJBPlADFKtgRUy9BLzjC
eotWchPHFBVW9ftPbaQViSUu7d89NLjDDM5xrh80puDIApxoQLDoIrh3T1kpZx56
jSWv0gelFUMbXAzmqkJSyL4Xdh1aqzgUbREd7Xf2ICzuh0sV6V7c/AwWtjWEGEsA
HZaiQDywZwbC18GwrMLiAzGWb/AScFDQRCZKJDjL+Ql8YT6z+ZMVr8gb7CIU5PKY
dhiIf2UVTQwLAoW7lNRCQQAqcGjK3IMIz7SO/yk4HmVUiQEiBBABAgAMBQJK3gjG
BQMAEnUAAAoJEJcQuJvKV618jkEH+wb0Zv9z7xQgpLMowVuBFQVu8/z7P5ASumyB
PUO3+0JVxSHBhlCKQK7n11m1fhuGt2fCxXhSU6LzXj36rsKRY53lGZ9QhvqFUtQH
3Xb2IQLIJC4UKjG2jSSCdcuA/x98bwp2v7O03rn7ndCS16CwXnRV3geQoNipRKMS
DajKPpZv1RiZm8pMKqEb8WSw352xWoOcxuffjlsOEwvJ85SEGCAZ9tmIlkZOc7Ai
QONDvii9b8AYhQ60RIQC0HP2ASSmK0V92VeFPxHmAygdDQgZNVtbVxgnnt7oTNEu
VRXNY+z4OfBArp7R+cTsvijDRZY4kML1n22hUybwoxUEvjqZV2+JASIEEAECAAwF
AkrvOlQFAwASdQAACgkQlxC4m8pXrXxrPAgArXiNgZirNuBhfNCXlkzkCHLx5wnV
e4SmTpbWzTwWw7+qk7d4l9hlWtdImISORINzo7f4ShSUzJX2GciNaXhaHRo7+y5O
Zbu82jQb09aQQj/nibKYuqxqUrobTEm+DuYz3JUQZm2PsPcHLS8mX9cxvrJUncPG
nXEV0DRaq71SGWDprtkvBbp6i38aY3sIhYgz8wM5m1szKDtjywmBYcFehIdozt9z
hm7wZshzRWQX1+Rf/pIsnk+OzBIa34crSemTnacbV/B7278z2XAyziPNFuqz0xu+
iltOmYmayfNWAmumuw9NcuwWMlth6Mc2HLrpo0ZBheJ6iuDMPsHnwqdB/4kBIgQQ
AQIADAUCSwBd2gUDABJ1AAAKCRCXELibyletfP6tB/4m1w0BtlkJgtS6E+B/ns14
z4A4PGors+n+MYm05qzvi+EnDF/sytCmVcKeimrtvDcfoDtKAFFvJjcYXfnJdGWm
Pu0SJMRL5KKCirAKwZmU/saxOgoB5QLNw+DHPteJ3w9GmWlGxIqG1r15WC5duzBC
y3FsnjJYG3jaLnHOO9yXXb5h0kUTORfUKdvAr1gxF2KoatZWqGoaPPnHoqb88rjt
zk8I7gDqoXnzh8wLxa0ZYvfTC/McxdWTrwXLft+krmMQ18iIZEne2hvVLNJVuluU
oiWLeHA8iNCQ4W4WTdLc1mCnCjGTMX/MN41uLH0C9Ka4R6wEaqj4lPDk1B/1TV+Q
iQEiBBABAgAMBQJLEYGrBQMAEnUAAAoJEJcQuJvKV618naIH/2t9aH5mBTKBN6fU
qhrf79vIsjtI/QNS5qisBISZMX3/1/0Gu6WnxkPSfdCUJMWCjMcnVj7KU2wxTHHG
VpAStd9r2afUNxRyqZwzwyytktuZok0XngAEDYDDBS3ssu2R4uWLCsC2ysXEqO/5
tI5YrTWJZrfeIphTaYP5hxrMujvqy3kEwKKbiMz91cDeiLS+YCBcalj5n/1dMYf7
8U8C6ieurxAg/L8h6x25VM4Ilx4MmG2T8QGtkkUXd+Fd/KYWmf0LE5LLPknf0Hhw
oVslPXeinp4FsHK/5wzviv4YZpzuTqs9NlKcMsa4IuuPOB0FDf0pn+OFQbEg9QwY
2gCozK+JASIEEAECAAwFAksjTdQFAwASdQAACgkQlxC4m8pXrXwlogf/XBGbXRVX
```

```
LMaRN4SczOjwT3/tUCriTkb3v+zKjRG90zFhYAccjn7w+7jKQicjq6quQG1EH2X4
/Su6ps1lDLqGHHhiJW3ZhxQScLZmhdAYsh2qG4GP/UW3QjXG7c61t+H3olvWg2cr
wqCxxFZAgkAAkr9xcHWFZJEQeXoob6cCZObaUnHSANdmC6s5lUxXYa2bmL7Q3UB4
4KCzDvAfbPZKJOw9k0qb3lc11zx+vGdyZFbm4R0+3LPp/vT0b3GlSbbF9lU1GOXh
VaphrgFFa76dmjfHCkPplXAkK1VSIU/aPGAefduTFMdlSZpdMtJ5AULjGcszBDlR
pLlPxvqVa0ZpgIkBIgQQAQIADAUCSycmkgUDABJ1AAAKCRCXELibyletfHlNCACp
1YespiHfQt2alcscE5zgfETEHHic8Ai6pNkU9HT4TeWcFHEDe5QqfYcpjLrQvBXS
kSvxEittbyRdv+e+j5Z+HyHjiG8nAQBL6qy9eHqQE4+d7gYs6DTk7sG9ZMYphREb
ltzD+F4hVCQdLT8LNr0eVFN7ehqECScDaCG8/Qyti+l/0M902/Yn+mz0ilOiUdWJ
9x6LPaIINtb1gsYDEylLjwGIZmI0r5Kh9wYoV4vnNezFbxO1uRiW0B7iaPjIEsbt
OOKp7wx2aX+DM3N9F3BtaIY8XnzcnomNm83SNsgmgrZljpQltUnNqIhNM8DupQ+I
WOV5gtl6pTC7CgeVTVyRiQEiBBABAgAMBQJLOGXuBQMAEnUAAAoJEJcQuJvKV618
ll4IAKJ9mm4jb0c8fe9+uDI8eCJRbzNbVXm8zWzpA8GUtQAakwxoKv332QP1Wa1P
odni/e3EMhsSREOZJJv79YqGxGRBTE9Kb/VjM34nas4XSnXKW28XWhKyIw+XwQAi
nY2swFHh+83Htr/mwTdJfS2aEYl2zboBvd/JZCdhOGU2GH737S/3uEczoKkfVQ/w
OTM8X1xWwlYWqx23k/DsGcuDs9lA2g7Mx7DSqBtVjaTkn9h0zATzXLDkmP4SAUVj
cZ83WDpFre5WnizZjdXlBMM5OCexp5WpmzyHLTnaBFK4jEmnsk5C2Rnoyp8Ivz6g
Ecg1tRbEXijRw++d2TFYlJwLKtiJASIEEAECAAwFAktKMicFAwASdQAACgkQlxC4
m8pXrXxqHQgAuYY5scKrh0m/GS9EYnyC9494lOlO6iytU0CpE6oBC31M3hfX/Dbj
UbcS5szZNU+2CPYo4ujQLZ7suN7+tTjG6pZFfMevajT9+jsL+NPMF8RLdLOVYmbl
TmSQGNO+XGEYaKYH5oZIeIW5AKCgi2ozkdFlBBLAx7Kqo/FyybhkURFEcvEyVmgf
3KLV7IIiX/fYLfoCMCJ/Lcm9/llSFB1n8Nvg66Xd533DKoHjueD3jyaNAVlo2mq/
sIAv++kntvOiB3GDK5pfwHZ78WWiCpsWZpE5gzAnzJ1Y0WEigRo0PVLu3cLO0jLG
23d+H/CbfZ8rkajHJeCDQF7YVmP0t0nYpYkBIgQQAQIADAUCS1v+ZgUDABJ1AAAK
CRCXELibyletfNS/CACqt2TkB86mjqM+cJ74+dWBvJ2aFuURuxzm95i9Q/W/hU08
2iMbC3+0k2oD8CrTOe61P+3oRyLjv/UEDUNzLncNe2YsA9JeV+4hvPwH5Vp3Om13
089fCKZUbqslXNKkHiWYU+zAaZJXEuGRmRz0HbQIeAMOWF4oa226uo1e4ws1Jhc+
F3E/ApCRyFBqBUdL05hapQLditYpsBjIdiBGpjzidMLE2wX2W4ZpAdN0U6BIyIqR
mTPjbSkvzS9kSWFmfhQgnBDKEYJpVZgE1sN52rYC1sDeGeiuKxlzjVov9MMhYMWa
Zo3R5o3F2iIM/BK6FbC252lf/Mhu3ICuXujNBZNYiQEiBBABAgAMBQJLbSH4BQMA
EnUAAAoJEJcQuJvKV618kd0IAJLLwDH6gvgAlBFklQJXqQxUdcSOOVMAWtlHgWOy
ozjgomZZBkRL8dtCDr9YBMcj5czcQ3qpmLJdppXhKB+kJV2iUXfDMSFXwJ4wLfIs
8FNnXw8H5U01oBkGH/Ku6ngL9Vwt+MjYHtCWkw9QueUKZnDudX9qIzLAIt+mwSTu
A6+fY4VWIg40AA0v3exaQM55YR/UhlKunpGG9o8Qkq77dMEbTMpOmBoLbOMRB3Dd
MAvVU6G2l6Pcb7KobVCuOBnb6batXARV/G8sw+nzfJ16fr/KobZT2A6m+Jrqk4dl
F14ljLbz16O5JGUPAryN2G2ddBdSAy7dtFSVhWWiWC9n88q5Ag0EPj6jHRAIAO/h
iX8WzHWOMLJT54x/axeDdqn1rBDf5cWmaCWHN2ujNNlgpx5emoU9v7QStsNUCOGB
bXkeO4Ar7YG+jtSR33zqNh3y5kQ0YkY3dQ0wh6nsl+wh4XIIY/3TUZVtmdJeUBRH
JlfVNFYad2hX1guFI37Ny1PoZAFsxO82g+XB/Se8r/+sbmVcONdcdIeFKrE3FjLt
IjNQcxC6l9Q2Oy8KDxG/zvUZG3+H5i3tdRMyGgmuD6gEV0GXOHYUopzLeit1+Aa0
bCk36Mwbu+BeOw/CJW3+b0mB27hOaf9aCA855IP6fJFvtxcblq8nHIqhU3Dc9tec
sl9/S1xZ5S8ylG/xeRsAAwUH/i8KqmvAhq0X7DgCcYputwh37cuZlHOa1Ep07JRm
BCDgkdQXkGrsj2Wzw7Aw/TGdWWkmn2pxb8BRui5cfcZFO7c6vryi6FpJuLucX975
+eVY50ndWkPXkJ1HF4i+HJwRqE2zliN/RHMs4LJcwXQvvjD43EE3AO6eiVFbD+qA
AdxUFoOeLblKNBHPG7DPG9xL+Ni5rkE+TXShxsB7F0z7ZdJJZOG0JODmox7IstQT
GoaU9u41oyZTIiXPiFidJoIZCh7fdurP8pn3X+R5HUNXMr7M+ba8lSNxce/F3kmH
0L7rsKqdh9d/aVxhJINJ+inVDnrXWVoXu9GBjT8Nco1iU9SIVAQYEQIADAUCTnc9
7QUJE/sBuAASB2VHUEcAAQEJEIxxjTtQcuH1FJsAmwWK9vmwRJ/y9gTnJ8PWf0BV
roUTAKClYAhZuX2nUNwH4vlEJQHDqYa5yQ==
=ghXk
-----END PGP PUBLIC KEY BLOCK-----
```

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

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

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

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

```
gpg:           new user IDs: 1
gpg:         new signatures: 53
```

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

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

If you experience problems or require RPM specific information, see Section 2.1.4.4, “Signature Checking Using RPM”.

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

Table 2.1 MySQL Package and Signature Files for Source files

File Type	File Name
Distribution file	`mysql-standard-5.6.51-linux-i686.tar.gz`
Signature file	`mysql-standard-5.6.51-linux-i686.tar.gz.asc`

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

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

If the downloaded package is valid, you should see a `Good signature` message similar to this one:

```
shell> gpg --verify mysql-standard-5.6.51-linux-i686.tar.gz.asc
gpg: Signature made Tue 01 Feb 2011 02:38:30 AM CST using DSA key ID 5072E1F5
gpg: Good signature from "MySQL Release Engineering <mysql-build@oss.oracle.com>"
```

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

```
shell> gpg --verify mysql-standard-5.6.51-linux-i686.tar.gz.asc
gpg: Signature made Wed 23 Jan 2013 02:25:45 AM PST using DSA key ID 5072E1F5
gpg: checking the trustdb
gpg: no ultimately trusted keys found
gpg: Good signature from "MySQL Release Engineering <mysql-build@oss.oracle.com>"
gpg: WARNING: This key is not certified with a trusted signature!
gpg:          There is no indication that the signature belongs to the owner.
Primary key fingerprint: A4A9 4068 76FC BD3C 4567  70C8 8C71 8D3B 5072 E1F5
```

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

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

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

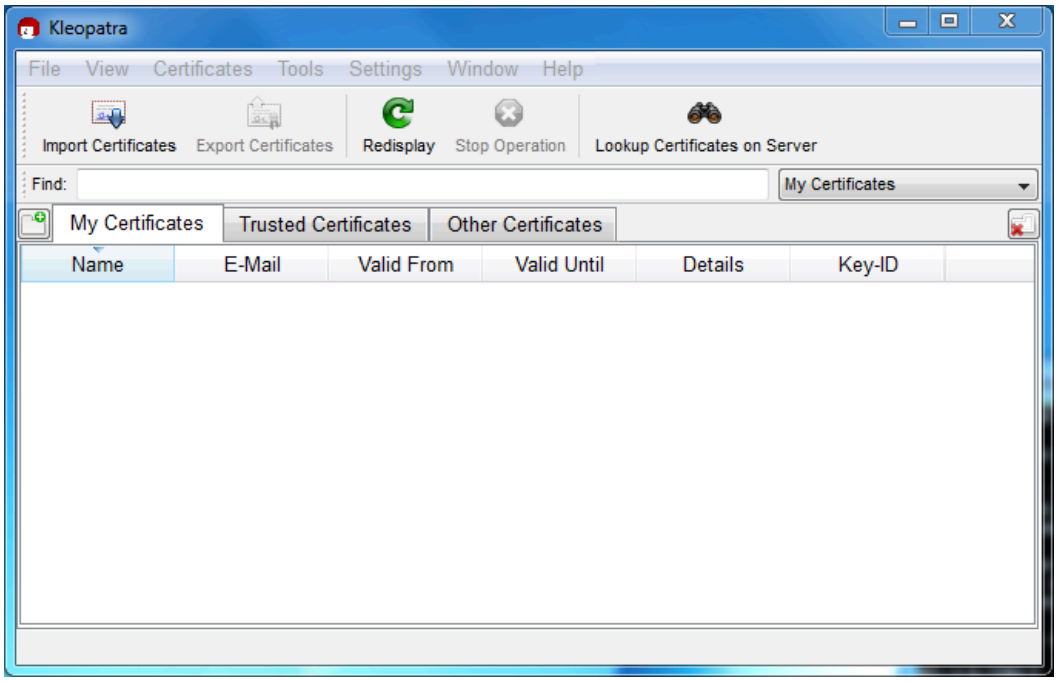
2.1.4.3 Signature Checking Using Gpg4win for Windows

The Section 2.1.4.2, “Signature Checking Using GnuPG” section describes how to verify MySQL downloads using GPG. That guide also applies to Microsoft Windows, but another option is to use a

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

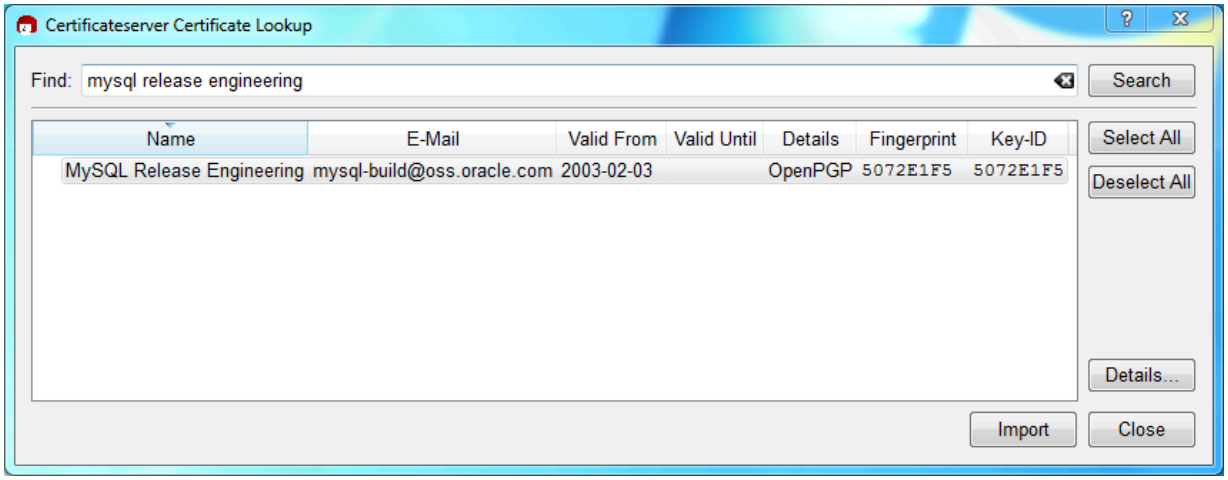
Download and install Gpg4win, and then load Kleopatra. The dialog should look similar to:

Figure 2.1 Kleopatra: Initial Screen



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

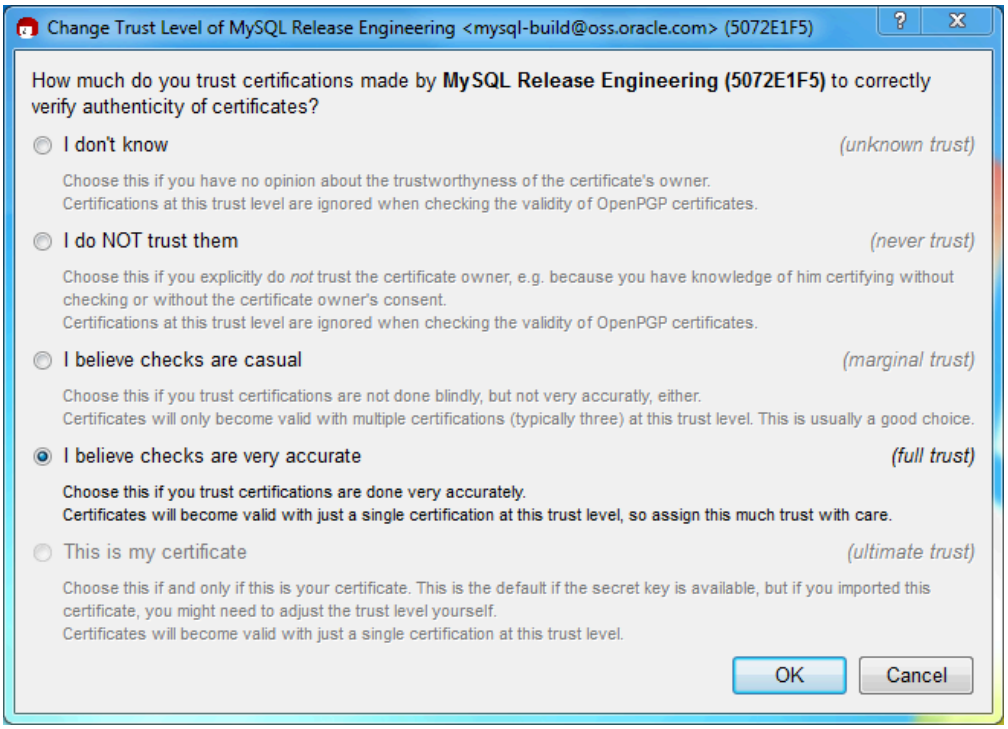
Figure 2.2 Kleopatra: Lookup Certificates on Server Wizard: Finding a Certificate



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

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

Figure 2.3 Kleopatra: Change Trust level for MySQL Release Engineering



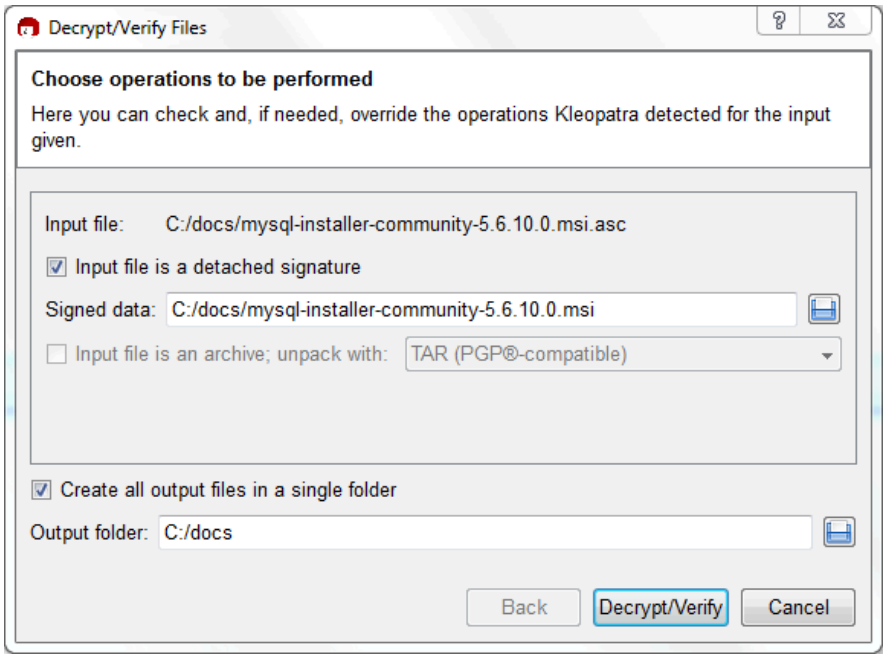
Next, verify the downloaded MySQL package file. This requires files for both the packaged file, and the signature. The signature file must have the same name as the packaged file but with an appended `.asc` extension, as shown by the example in the following table. The signature is linked to on the downloads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

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

File Type	File Name
Distribution file	`mysql-installer-community-5.6.51.msi`
Signature file	`mysql-installer-community-5.6.51.msi.asc`

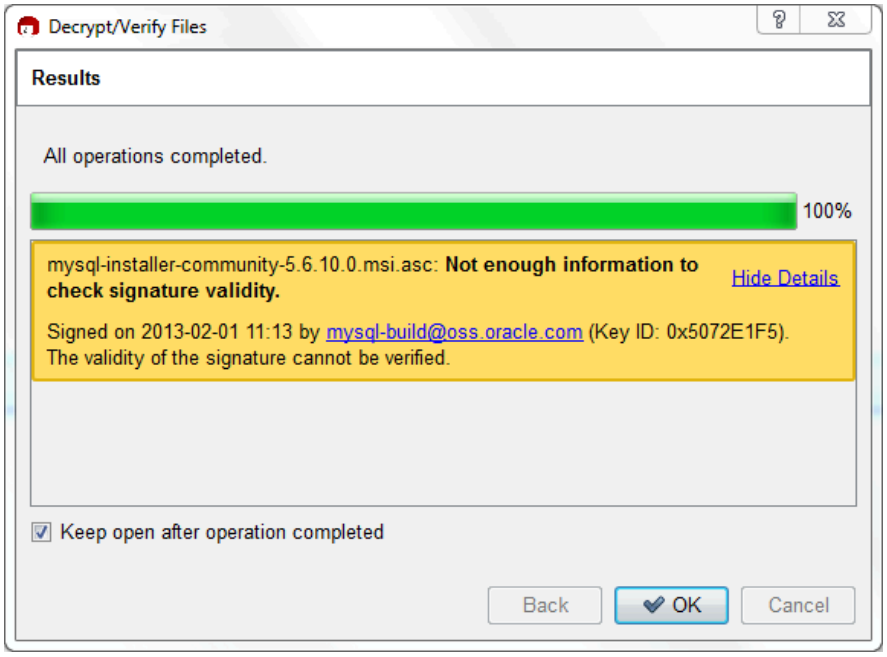
Make sure that both files are stored in the same directory and then run the following command to verify the signature for the distribution file. Either drag and drop the signature (`.asc`) file into Kleopatra, or load the dialog from **File**, **Decrypt/Verify Files...**, and then choose either the `.msi`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

Figure 2.4 Kleopatra: The Decrypt and Verify Files Dialog



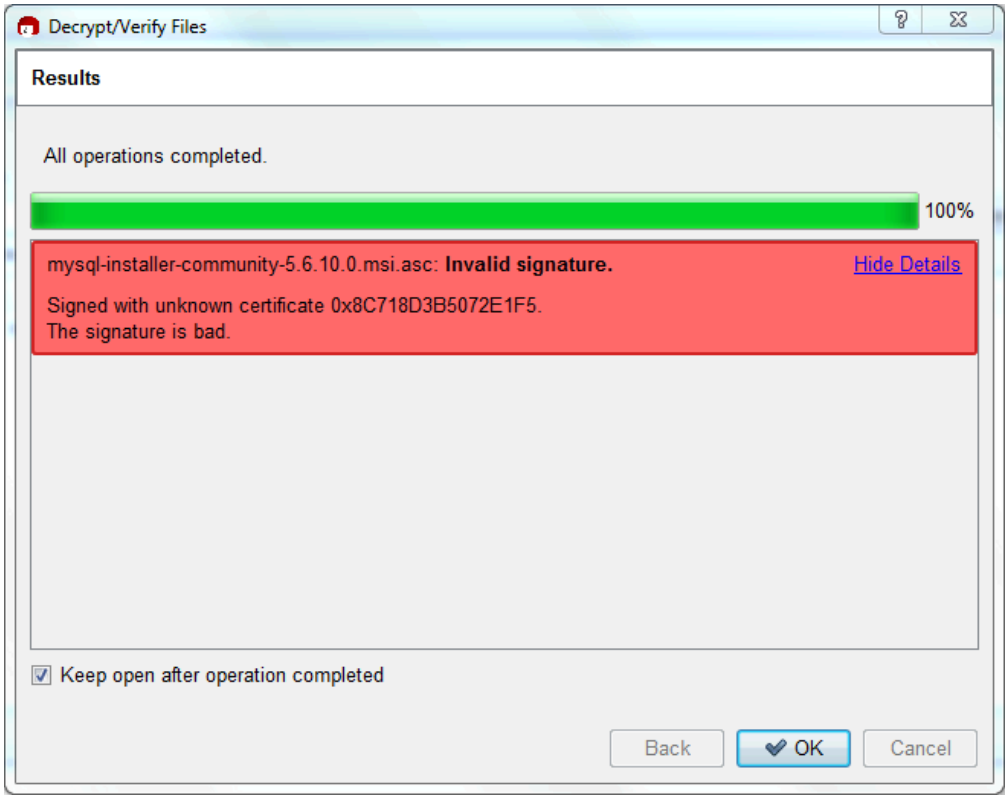
Click **Decrypt/Verify** to check the file. The two most common results should look like the following, and although the yellow warning looks problematic, the following means that the file check passed with success. You may now run this installer.

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



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

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



The Section 2.1.4.2, “Signature Checking Using GnuPG” section explains why you probably don't see a green `Good signature` result.

2.1.4.4 Signature Checking Using RPM

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

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

Example:

```
shell> rpm --checksig MySQL-server-5.6.51-0.linux_glibc2.5.i386.rpm
MySQL-server-5.6.51-0.linux_glibc2.5.i386.rpm: md5 gpg OK
```

Note

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

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

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

```
shell> rpm --import https://dev.mysql.com/doc/refman/5.6/en/checking-gpg-signature.html
```

If you need to obtain the MySQL public key, see Section 2.1.4.2, “Signature Checking Using GnuPG”.

2.1.5 Installation Layouts

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

- Section 2.3.1, “MySQL Installation Layout on Microsoft Windows”
- Section 2.9.3, “MySQL Layout for Source Installation”
- Table 2.3, “MySQL Installation Layout for Generic Unix/Linux Binary Package”
- Table 2.11, “MySQL Installation Layout for Linux RPM Packages from the MySQL Developer Zone”
- Table 2.7, “MySQL Installation Layout on OS X”

2.1.6 Compiler-Specific Build Characteristics

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

`icc` (Intel C++ Compiler) Builds

A server built with `icc` has these characteristics:

- SSL support is not included.

2.2 Installing MySQL on Unix/Linux Using Generic Binaries

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

This section covers the installation of MySQL from a compressed `tar` file binary distribution on Unix/Linux platforms. For other platform-specific binary package formats, see the other platform-specific sections in this manual. For example, for Windows distributions, see Section 2.3, “Installing MySQL on Microsoft Windows”. See Section 2.1.3, “How to Get MySQL” on how to obtain MySQL in different distribution formats.

MySQL compressed `tar` file binary distributions have names of the form `mysql-`*`VERSION`*`-`*`OS`*`.tar.gz`, where *`VERSION`* is a number (for example, `5.6.51`), and *`OS`* indicates the type of operating system for which the distribution is intended (for example, `pc-linux-i686` or `winx64`).

Warnings

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

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

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

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

  Or, on APT-based systems:

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

- **SLES 11**: As of MySQL 5.6.37, the Linux Generic tarball package format is EL6 instead of EL5. As a side effect, the MySQL client `bin/mysql` needs `libtinfo.so.5`.

  A workaround is to create a symlink, such as `ln -s libncurses.so.5.6 /lib64/libtinfo.so.5` on 64-bit systems or `ln -s libncurses.so.5.6 /lib/libtinfo.so.5` on 32-bit systems.

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

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

Directory	Contents of Directory
`bin`, `scripts`	`mysqld` server, client and utility programs
`data`	Log files, databases
`docs`	MySQL manual in Info format
`include`	Include (header) files
`lib`	Libraries
`mysql-test`	Test suite
`man`	Unix manual pages
`share`	Error messages, dictionary, and SQL for database installation
`sql-bench`	Benchmarks
`support-files`	Miscellaneous support files, including sample configuration files

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

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

```
shell> groupadd mysql
shell> useradd -r -g mysql -s /bin/false mysql
shell> cd /usr/local
shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
shell> ln -s full-path-to-mysql-VERSION-OS mysql
shell> cd mysql
shell> scripts/mysql_install_db --user=mysql
shell> bin/mysqld_safe --user=mysql &
# Next command is optional
shell> cp support-files/mysql.server /etc/init.d/mysql.server
```

Note

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

Note

The procedure does not assign passwords to MySQL accounts. To do so, use the instructions in Section 2.10.4, “Securing the Initial MySQL Accounts”.

`mysql_install_db` creates a default option file named `my.cnf` in the base installation directory. This file is created from a template included in the distribution package named `my-default.cnf`. For more information, see Section 5.1.2.2, “Using a Sample Default Server Configuration File”.

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

Create a mysql User and Group

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

```
shell> groupadd mysql
shell> useradd -r -g mysql -s /bin/false mysql
```

Note

Because the user is required only for ownership purposes, not login purposes, the `useradd` command uses the `-r` and `-s /bin/false` options to create a user that does not have login permissions to your server host. Omit these options if your `useradd` does not support them.

Obtain and Unpack the Distribution

Pick the directory under which you want to unpack the distribution and change location into it. The example here unpacks the distribution under `/usr/local`. The instructions, therefore, assume that you have permission to create files and directories in `/usr/local`. If that directory is protected, you must perform the installation as `root`.

```
shell> cd /usr/local
```

Obtain a distribution file using the instructions in Section 2.1.3, “How to Get MySQL”. For a given release, binary distributions for all platforms are built from the same MySQL source distribution.

Unpack the distribution, which creates the installation directory. `tar` can uncompress and unpack the distribution if it has `z` option support:

```
shell> tar zxvf /path/to/mysql-VERSION-OS.tar.gz
```

The `tar` command creates a directory named `mysql-VERSION-OS`.

To install MySQL from a compressed `tar` file binary distribution, your system must have GNU `gunzip` to uncompress the distribution and a reasonable `tar` to unpack it. If your `tar` program supports the `z` option, it can both uncompress and unpack the file.

GNU `tar` is known to work. The standard `tar` provided with some operating systems is not able to unpack the long file names in the MySQL distribution. You should download and install GNU `tar`, or if available, use a preinstalled version of GNU tar. Usually this is available as `gnutar`, `gtar`, or as `tar` within a GNU or Free Software directory, such as `/usr/sfw/bin` or `/usr/local/bin`. GNU `tar` is available from http://www.gnu.org/software/tar/.

If your `tar` does not have `z` option support, use `gunzip` to unpack the distribution and `tar` to unpack it. Replace the preceding `tar` command with the following alternative command to uncompress and extract the distribution:

```
shell> gunzip < /path/to/mysql-VERSION-OS.tar.gz | tar xvf -
```

Next, create a symbolic link to the installation directory created by `tar`:

```
shell> ln -s full-path-to-mysql-VERSION-OS mysql
```

The `ln` command makes a symbolic link to the installation directory. This enables you to refer more easily to it as `/usr/local/mysql`. To avoid having to type the path name of client programs always when you are working with MySQL, you can add the `/usr/local/mysql/bin` directory to your `PATH` variable:

```
shell> export PATH=$PATH:/usr/local/mysql/bin
```

Perform Postinstallation Setup

The remainder of the installation process involves setting distribution ownership and access permissions, initializing the data directory, starting the MySQL server, and setting up the configuration file. For instructions, see Section 2.10, “Postinstallation Setup and Testing”.

2.3 Installing MySQL on Microsoft Windows

Important

MySQL Community 5.6 Server requires the Microsoft Visual C++ 2010 Redistributable Package to run on Windows platforms. Users should make sure the package has been installed on the system before installing the server. The package is available at the Microsoft Download Center.

MySQL is available for Microsoft Windows, for both 32-bit and 64-bit versions. For supported Windows platform information, see https://www.mysql.com/support/supportedplatforms/database.html.

There are different methods to install MySQL on Microsoft Windows.

MySQL Installer Method

The simplest and recommended method is to download MySQL Installer (for Windows) and let it install and configure all of the MySQL products on your system. Here is how:

1. Download MySQL Installer from https://dev.mysql.com/downloads/installer/ and execute it.

   Note

   Unlike the standard MySQL Installer, the smaller "web-community" version does not bundle any MySQL applications but rather downloads the MySQL products you choose to install.

2. Choose the appropriate **Setup Type** for your system. Typically you should choose **Developer Default** to install the MySQL server and other MySQL tools related to MySQL development, and helpful tools like MySQL Workbench. Or, choose the **Custom** setup type to select the desired MySQL products manually.

   Note

   Multiple versions of MySQL server can exist on a single system. You can choose one or multiple versions.

3. Complete the installation process by following the instructions. This installs several MySQL products and starts the MySQL server.

MySQL is now installed. If you configured MySQL as a service, then Windows automatically starts the MySQL server every time you restart your system.

Note

You probably also installed other helpful MySQL products like MySQL Workbench on your system. Consider loading Chapter 25, *MySQL Workbench* to check your new MySQL server connection. By default, this program automatically start after installing MySQL.

This process also installs the MySQL Installer application on your system, and later you can use MySQL Installer to upgrade or reconfigure your MySQL products.

Additional Installation Information

It is possible to run MySQL as a standard application or as a Windows service. By using a service, you can monitor and control the operation of the server through the standard Windows service management tools. For more information, see Section 2.3.4.7, “Starting MySQL as a Windows Service”.

Generally, you should install MySQL on Windows using an account that has administrator rights. Otherwise, you may encounter problems with certain operations such as editing the `PATH` environment variable or accessing the `Service Control Manager`. When installed, MySQL does not need to be executed using a user with Administrator privileges.

For a list of limitations on the use of MySQL on the Windows platform, see Section 2.3.7, “Windows Platform Restrictions”.

In addition to the MySQL Server package, you may need or want additional components to use MySQL with your application or development environment. These include, but are not limited to:

- To connect to the MySQL server using ODBC, you must have a Connector/ODBC driver. For more information, including installation and configuration instructions, see MySQL Connector/ODBC Developer Guide.

  Note

  MySQL Installer installs and configures Connector/ODBC for you.

- To use MySQL server with .NET applications, you must have the Connector/NET driver. For more information, including installation and configuration instructions, see MySQL Connector/NET Developer Guide.

  Note

  MySQL Installer installs and configures MySQL Connector/NET for you.

MySQL distributions for Windows can be downloaded from https://dev.mysql.com/downloads/. See Section 2.1.3, “How to Get MySQL”.

MySQL for Windows is available in several distribution formats, detailed here. Generally speaking, you should use MySQL Installer. It contains more features and MySQL products than the older MSI, is simpler to use than the compressed file, and you need no additional tools to get MySQL up and running. MySQL Installer automatically installs MySQL Server and additional MySQL products, creates an options file, starts the server, and enables you to create default user accounts. For more information on choosing a package, see Section 2.3.2, “Choosing an Installation Package”.

- A MySQL Installer distribution includes MySQL Server and additional MySQL products, including MySQL Workbench. MySQL Installer can also be used to upgrade this product in the future.

For instructions on installing MySQL using MySQL Installer, see Section 2.3.3, “MySQL Installer for Windows”.

- The standard binary distribution (packaged as a compressed file) contains all of the necessary files that you unpack into your chosen location. This package contains all of the files in the full Windows MSI Installer package, but does not include an installation program.

  For instructions on installing MySQL using the compressed file, see Section 2.3.4, “Installing MySQL on Microsoft Windows Using a `noinstall` ZIP Archive”.

- The source distribution format contains all the code and support files for building the executables using the Visual Studio compiler system.

  For instructions on building MySQL from source on Windows, see Section 2.9, “Installing MySQL from Source”.

MySQL on Windows Considerations

- **Large Table Support**

  If you need tables with a size larger than 4 GB, install MySQL on an NTFS or newer file system. Do not forget to use `MAX_ROWS` and `AVG_ROW_LENGTH` when you create tables. See Section 13.1.17, “CREATE TABLE Statement”.

  Note

  InnoDB tablespace files cannot exceed 4 GB on Windows 32-bit systems.

- **MySQL and Virus Checking Software**

  Virus-scanning software such as Norton/Symantec Anti-Virus on directories containing MySQL data and temporary tables can cause issues, both in terms of the performance of MySQL and the virus-scanning software misidentifying the contents of the files as containing spam. This is due to the fingerprinting mechanism used by the virus-scanning software, and the way in which MySQL rapidly updates different files, which may be identified as a potential security risk.

  After installing MySQL Server, it is recommended that you disable virus scanning on the main directory (`datadir`) used to store your MySQL table data. There is usually a system built into the virus-scanning software to enable specific directories to be ignored.

  In addition, by default, MySQL creates temporary files in the standard Windows temporary directory. To prevent the temporary files also being scanned, configure a separate temporary directory for MySQL temporary files and add this directory to the virus scanning exclusion list. To do this, add a configuration option for the `tmpdir` parameter to your `my.ini` configuration file. For more information, see Section 2.3.4.2, “Creating an Option File”.

- **Running MySQL on a 4K Sector Hard Drive**

  Running the MySQL server on a 4K sector hard drive on Windows is not supported with `innodb_flush_method=async_unbuffered`, which is the default setting. The workaround is to use `innodb_flush_method=normal`.

2.3.1 MySQL Installation Layout on Microsoft Windows

For MySQL 5.6 on Windows, the default installation directory is `C:\Program Files\MySQL\MySQL Server 5.6` for installations performed with MySQL Installer. If you use the ZIP archive method to install MySQL, you may prefer to install in `C:\mysql`. However, the layout of the subdirectories remains similar (exceptions are indicated).

All of the files are located within this parent directory, using the structure shown in the following table.

Table 2.4 Default MySQL Installation Layout for Microsoft Windows

Directory	Contents of Directory	Notes
`bin`, `scripts`	`mysqld` server, client and utility programs	
`%PROGRAMDATA%\MySQL\MySQL Server 5.6\`	Log files, databases	The Windows system variable `%PROGRAMDATA%` defaults to `C:\ProgramData`.
`data`	Pristine templates	
`docs`	Release documentation	With MySQL Installer, use the `Modify` operation to select this optional folder.
`include`	Include (header) files	
`lib`	Libraries	
`share`	Miscellaneous support files, including error messages, character set files, sample configuration files, SQL for database installation	
`mysql-test`, `scripts`, and `sql-bench`	Debug binaries and test suite	ZIP archive only.

The packages create and set up the data directory that the installed server uses and creates a pristine "template" data directory named `data` under the installation directory. After an installation has been performed using this package, the template data directory can be copied to set up additional MySQL instances. See Section 5.7, "Running Multiple MySQL Instances on One Machine".

2.3.2 Choosing an Installation Package

For MySQL 5.6, there are multiple installation package formats to choose from when installing MySQL on Windows. The package formats described in this section are:

- MySQL Installer
- MySQL noinstall ZIP Archives
- MySQL Docker Images

Program Database (PDB) files (with file name extension `pdb`) provide information for debugging your MySQL installation in the event of a problem. These files are included in ZIP Archive distributions (but not MSI distributions) of MySQL.

MySQL Installer

This package has a file name similar to `mysql-installer-community-5.6.51.0.msi` or `mysql-installer-commercial-5.6.51.0.msi`, and utilizes MSIs to automatically install MySQL server and other products. MySQL Installer downloads and applies updates to itself, and for each of the installed products. It also configures the installed MySQL server (including a sandbox InnoDB cluster test setup) and MySQL Router. MySQL Installer is recommended for most users.

MySQL Installer can install and manage (add, modify, upgrade, and remove) many other MySQL products, including:

- Applications – MySQL Workbench, MySQL for Visual Studio, MySQL Utilities, MySQL Shell, MySQL Router
- Connectors – MySQL Connector/C++, MySQL Connector/NET, Connector/ODBC, MySQL Connector/Python, MySQL Connector/J, MySQL Connector/Node.js

- Documentation – MySQL Manual (PDF format), samples and examples

MySQL Installer operates on all MySQL supported versions of Windows (see https://www.mysql.com/support/supportedplatforms/database.html).

Note

Because MySQL Installer is not a native component of Microsoft Windows and depends on .NET, it does not work on minimal installations like the Server Core version of Windows Server.

For instructions on how to install MySQL using MySQL Installer, see Section 2.3.3, “MySQL Installer for Windows”.

MySQL noinstall ZIP Archives

These packages contain the files found in the complete MySQL Server installation package, with the exception of the GUI. This format does not include an automated installer, and must be manually installed and configured.

The `noinstall` ZIP archives are split into two separate compressed files. The main package is named `mysql-VERSION-winx64.zip` for 64-bit and `mysql-VERSION-win32.zip` for 32-bit. This contains the components needed to use MySQL on your system. The optional MySQL test suite, MySQL benchmark suite, and debugging binaries/information components (including PDB files) are in a separate compressed file named `mysql-VERSION-winx64-debug-test.zip` for 64-bit and `mysql-VERSION-win32-debug-test.zip` for 32-bit.

If you choose to install a `noinstall` ZIP archive, see Section 2.3.4, “Installing MySQL on Microsoft Windows Using a `noinstall` ZIP Archive”.

MySQL Docker Images

For information on using the MySQL Docker images provided by Oracle on Windows platform, see Section 2.5.8.3, “Deploying MySQL on Windows and Other Non-Linux Platforms with Docker”.

Warning

The MySQL Docker images provided by Oracle are built specifically for Linux platforms. Other platforms are not supported, and users running the MySQL Docker images from Oracle on them are doing so at their own risk.

2.3.3 MySQL Installer for Windows

MySQL Installer is a standalone application designed to ease the complexity of installing and configuring MySQL products that run on Microsoft Windows. It supports the following MySQL products:

- MySQL Servers

  MySQL Installer can install and manage multiple, separate MySQL server instances on the same host at the same time. For example, MySQL Installer can install, configure, and upgrade a separate instance of MySQL 5.6, MySQL 5.7, and MySQL 8.0 on the same host. MySQL Installer does not permit server upgrades between major and minor version numbers, but does permit upgrades within a release series (such as 8.0.21 to 8.0.22).

  Note

  MySQL Installer cannot install both *Community* and *Commercial* releases of MySQL server on the same host. If you require both releases on the same host, consider using the ZIP archive distribution to install one of the releases.

- MySQL Applications

MySQL Workbench, MySQL Shell, MySQL Router, and MySQL for Visual Studio.

- MySQL Connectors

  MySQL Connector/NET, MySQL Connector/Python, MySQL Connector/ODBC, MySQL Connector/J, and MySQL Connector/C++. To install MySQL Connector/Node.js, see https://dev.mysql.com/downloads/connector/nodejs/.

- Documentation and Samples

  MySQL Reference Manuals (by version) in PDF format and MySQL database samples (by version).

Installation Requirements

MySQL Installer requires Microsoft .NET Framework 4.5.2 or later. If this version is not installed on the host computer, you can download it by visiting the Microsoft website.

An internet connection is required to download a manifest containing metadata for the latest MySQL products that are not part of a full bundle. MySQL Installer attempts to download the manifest when you start the application for the first time and then periodically in configurable intervals (see MySQL Installer options). Alternatively, you can retrieve an updated manifest manually by clicking **Catalog** in the MySQL Installer dashboard.

Note

If the first-time or subsequent manifest download is unsuccessful, an error is logged and you may have limited access to MySQL products during your session. MySQL Installer attempts to download the manifest with each startup until the initial manifest structure is updated. For help finding a product, see Locating Products to Install.

MySQL Installer Community Release

Download software from https://dev.mysql.com/downloads/installer/ to install the Community release of all MySQL products for Windows. Select one of the following MySQL Installer package options:

- *Web*: Contains MySQL Installer and configuration files only. The web package option downloads only the MySQL products you select to install, but it requires an internet connection for each download. The size of this file is approximately 2 MB. The file name has the form `mysql-installer-community-web-VERSION.N.msi` in which *`VERSION`* is the MySQL server version number such as 8.0 and `N` is the package number, which begins at 0.

- *Full or Current Bundle*: Bundles all of the MySQL products for Windows (including the MySQL server). The file size is over 300 MB, and the name has the form `mysql-installer-community-VERSION.N.msi` in which *`VERSION`* is the MySQL Server version number such as 8.0 and `N` is the package number, which begins at 0.

MySQL Installer Commercial Release

Download software from https://edelivery.oracle.com/ to install the Commercial release (Standard or Enterprise Edition) of MySQL products for Windows. If you are logged in to your My Oracle Support (MOS) account, the Commercial release includes all of the current and previous GA versions available in the Community release, but it excludes development-milestone versions. When you are not logged in, you see only the list of bundled products that you downloaded already.

The Commercial release also includes the following products:

- Workbench SE/EE

- MySQL Enterprise Backup

- MySQL Enterprise Firewall

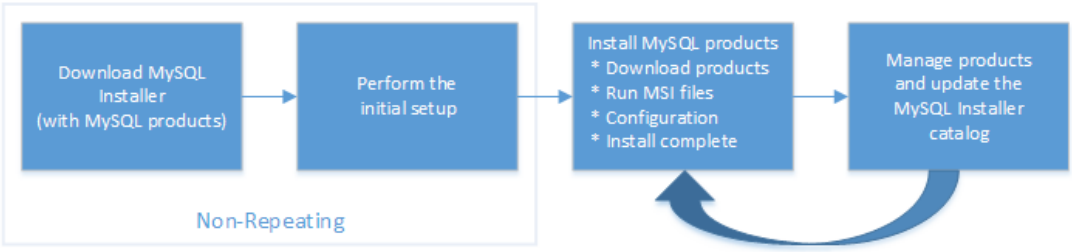
The Commercial release integrates with your MOS account. For knowledge-base content and patches, see My Oracle Support.

2.3.3.1 MySQL Installer Initial Setup

- Choosing a Setup Type
- Path Conflicts
- Check Requirements
- MySQL Installer Configuration Files

When you download MySQL Installer for the first time, a setup wizard guides you through the initial installation of MySQL products. As the following figure shows, the initial setup is a one-time activity in the overall process. MySQL Installer detects existing MySQL products installed on the host during its initial setup and adds them to the list of products to be managed.

Figure 2.7 MySQL Installer Process Overview



MySQL Installer extracts configuration files (described later) to the hard drive of the host during the initial setup. Although MySQL Installer is a 32-bit application, it can install both 32-bit and 64-bit binaries.

The initial setup adds a link to the Start menu under the **MySQL** group. Click **Start**, **All Programs**, **MySQL**, **MySQL Installer** to open MySQL Installer.

Choosing a Setup Type

During the initial setup, you are prompted to select the MySQL products to be installed on the host. One alternative is to use a predetermined setup type that matches your setup requirements. By default, both GA and pre-release products are included in the download and installation with the **Developer Default**, **Client only**, and **Full** setup types. Select the **Only install GA products** option to restrict the product set to include GA products only when using these setup types.

Choosing one of the following setup types determines the initial installation only and does not limit your ability to install or update MySQL products for Windows later:

- **Developer Default**: Install the following products that compliment application development with MySQL:
  - MySQL Server (Installs the version that you selected when you downloaded MySQL Installer.)
  - MySQL Shell
  - MySQL Router
  - MySQL Workbench
  - MySQL for Visual Studio
  - MySQL Connectors (for .NET / Python / ODBC / Java / C++)

- MySQL Documentation
- MySQL Samples and Examples

- **Server only**: Only install the MySQL server. This setup type installs the general availability (GA) or development release server that you selected when you downloaded MySQL Installer. It uses the default installation and data paths.
- **Client only**: Only install the most recent MySQL applications and MySQL connectors. This setup type is similar to the `Developer Default` type, except that it does not include MySQL server or the client programs typically bundled with the server, such as `mysql` or `mysqladmin`.
- **Full**: Install all available MySQL products.
- **Custom**: The custom setup type enables you to filter and select individual MySQL products from the MySQL Installer catalog.

Note

For MySQL Server versions 8.0.20 (and earlier), 5.7, and 5.6, the account you use to run MySQL Installer may not have adequate permission to install the server data files and this can interrupt the installation because the `ExecSecureObjects` MSI action cannot be executed. To proceed, deselect the **Server data files** feature before attempting to install the server again. For help, see Product Features To Install).

The **Server data files** check box was removed from the feature tree for MySQL Server 8.0.21 (and higher).

Use the `Custom` setup type to install:

- A product or product version that is not available from the usual download locations. The catalog contains all product releases, including the other releases between pre-release (or development) and GA.
- An instance of MySQL server using an alternative installation path, data path, or both. For instructions on how to adjust the paths, see Section 2.3.3.2, “Setting Alternative Server Paths with MySQL Installer”.
- Two or more MySQL server versions on the same host at the same time (for example, 5.6, 5.7, and 8.0).
- A specific combination of products and features not offered as a predetermine setup type. For example, you can install a single product, such as MySQL Workbench, instead of installing all client applications for Windows.

Path Conflicts

When the default installation or data folder (required by MySQL server) for a product to be installed already exists on the host, the wizard displays the **Path Conflict** step to identify each conflict and enable you to take action to avoid having files in the existing folder overwritten by the new installation. You see this step in the initial setup only when MySQL Installer detects a conflict.

To resolve the path conflict, do one of the following:

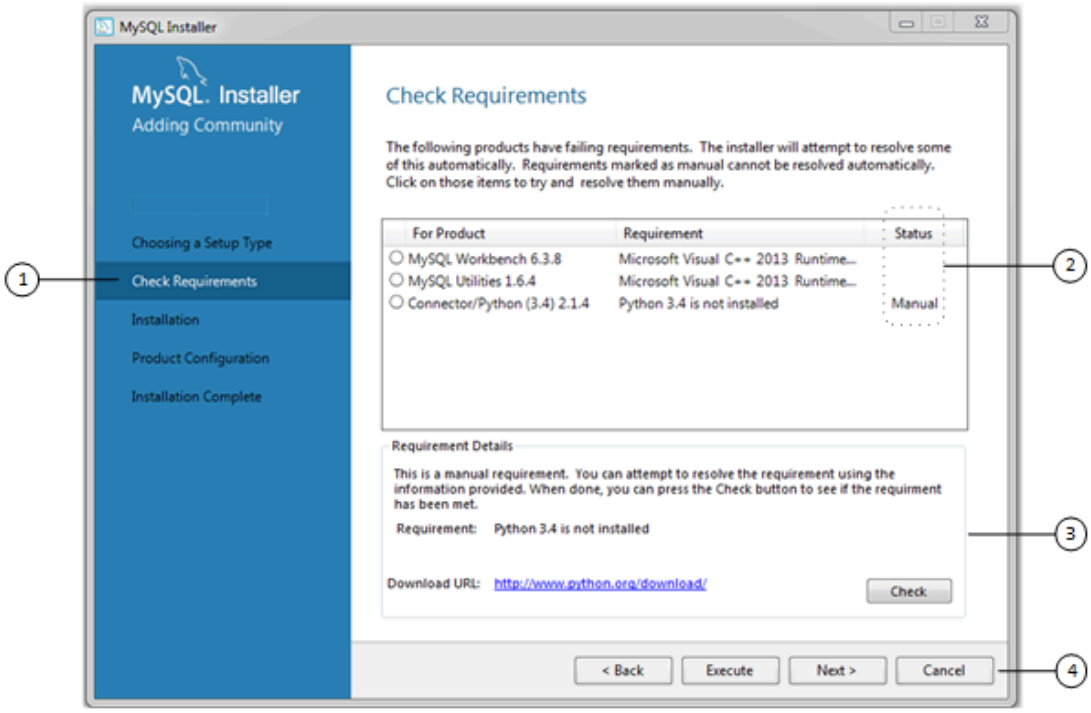
- Select a product from the list to display the conflict options. A warning symbol indicates which path is in conflict. Use the browse button to choose a new path and then click **Next**.
- Click **Back** to choose a different setup type or product version, if applicable. The `Custom` setup type enables you to select individual product versions.

- Click **Next** to ignore the conflict and overwrite files in the existing folder.
- Delete the existing product. Click **Cancel** to stop the initial setup and close MySQL Installer. Open MySQL Installer again from the Start menu and delete the installed product from the host using the Delete operation from the MySQL Installer dashboard.

Check Requirements

MySQL Installer uses entries in the `package-rules.xml` file to determine whether the prerequisite software for each product is installed on the host. When the requirements check fails, MySQL Installer displays the **Check Requirements** step to help you update the host. Requirements are evaluated each time you download a new product (or version) for installation. The following figure identifies and describes the key areas of this step.

Figure 2.8 Check Requirements



Description of Check Requirements Elements

1. Shows the current step in the initial setup. Steps in this list may change slightly depending on the products already installed on the host, the availability of prerequisite software, and the products to be installed on the host.
2. Lists all pending installation requirements by product and indicates the status as follows:
   - A blank space in the **Status** column means that MySQL Installer can attempt to download and install the required software for you.
   - The word *Manual* in the **Status** column means that you must satisfy the requirement manually. Select each product in the list to see its requirement details.
3. Describes the requirement in detail to assist you with each manual resolution. When possible, a download URL is provided. After you download and install the required software, click **Check** to verify that the requirement has been met.
4. Provides the following set operations to proceed:
   - **Back** – Return to the previous step. This action enables you to select a different the setup type.

- **Execute** – Have MySQL Installer attempt to download and install the required software for all items without a manual status. Manual requirements are resolved by you and verified by clicking **Check**.
- **Next** – Do not execute the request to apply the requirements automatically and proceed to the installation without including the products that fail the check requirements step.
- **Cancel** – Stop the installation of MySQL products. Because MySQL Installer is already installed, the initial setup begins again when you open MySQL Installer from the Start menu and click **Add** from the dashboard. For a description of the available management operations, see Product Catalog.

MySQL Installer Configuration Files

All MySQL Installer files are located within the `C:\Program Files (x86)` and `C:\ProgramData` folders. The following table describes the files and folders that define MySQL Installer as a standalone application.

Note

Installed MySQL products are neither altered nor removed when you update or uninstall MySQL Installer.

Table 2.5 MySQL Installer Configuration Files

File or Folder	Description	Folder Hierarchy
`MySQL Installer for Windows`	This folder contains all of the files needed to run MySQL Installer and MySQLInstallerConsole.exe, a command-line program with similar functionality.	`C:\Program Files (x86)`
`Templates`	The `Templates` folder has one file for each version of MySQL server. Template files contain keys and formulas to calculate some values dynamically.	`C:\ProgramData\MySQL \MySQL Installer for Windows\Manifest`
`package-rules.xml`	This file contains the prerequisites for every product to be installed.	`C:\ProgramData\MySQL \MySQL Installer for Windows\Manifest`
`produts.xml`	The `products` file (or product catalog) contains a list of all products available for download.	`C:\ProgramData\MySQL \MySQL Installer for Windows\Manifest`
`Product Cache`	The `Product Cache` folder contains all standalone `.msi` files bundled with the full package or downloaded afterward.	`C:\ProgramData\MySQL \MySQL Installer for Windows`

2.3.3.2 Setting Alternative Server Paths with MySQL Installer

You can change the default installation path, the data path, or both when you install MySQL server. After you have installed the server, the paths cannot be altered without removing and reinstalling the server instance.

To change paths for MySQL server

1. Identify the MySQL server to change and enable the **Advanced Options** link.
   a. Navigate to the **Select Products** page by doing one of the following:

i. If this is an initial setup of MySQL Installer, select the `Custom` setup type and click **Next**.

ii. If MySQL Installer is installed already, launch it from the Start menu and then click **Add** from the dashboard.

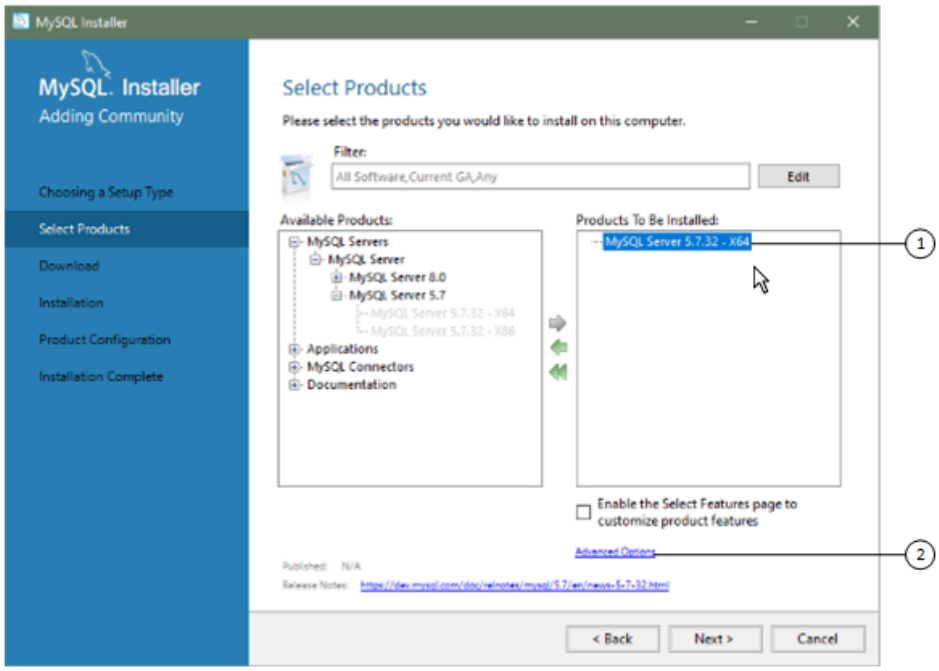
b. Click **Edit** to apply a filter on the product list shown in **Available Products** (see Locating Products to Install.

c. With the server instance selected, use the arrow to move the selected server to the **Products To Be Installed** list.

d. Click the server to select it. When you select the server, the **Advanced Options** link is enabled below the list of products to be installed (see the following figure).

2. Click **Advanced Options** to open a dialog box where you can enter alternative path names. After the path names are validated, click **Next** to continue with the configuration steps.

Figure 2.9 Change MySQL Server Path



2.3.3.3 Installation Workflows with MySQL Installer

MySQL Installer provides a wizard-like tool to install and configure new MySQL products for Windows. Unlike the initial setup, which runs only once, MySQL Installer invokes the wizard each time you download or install a new product. For first-time installations, the steps of the initial setup proceed directly into the steps of the installation. For assistance with product selection, see Locating Products to Install.

Note

Full permissions are granted to the user executing MySQL Installer to all generated files, such as `my.ini`. This does not apply to files and directories for specific products, such as the MySQL server data directory in `%ProgramData%` that is owned by `SYSTEM`.

Products installed and configured on a host follow a general pattern that might require your input during the various steps. If you attempt to install a product that is incompatible with the existing MySQL server version (or a version selected for upgrade), you are alerted about the possible mismatch.

MySQL Installer provides the following sequence of actions that apply to different workflows:

- **Select Products.** If you selected the `Custom` setup type during the initial setup or clicked **Add** from the MySQL Installer dashboard, MySQL Installer includes this action in the sidebar. From this page, you can apply a filter to modify the Available Products list and then select one or more products to move (using arrow keys) to the Products To Be Installed list.

  Select the check box on this page to activate the Select Features action where you can customize the products features after the product is downloaded.

- **Download.** If you installed the full (not web) MySQL Installer package, all `.msi` files were loaded to the `Product Cache` folder during the initial setup and are not downloaded again. Otherwise, click **Execute** to begin the download. The status of each product changes from `Ready to Download`, to `Downloading`, and then to `Downloaded`.

- **Select Features To Install (disabled by default).** After MySQL Installer downloads a product's `.msi` file, you can customize the features if you enabled the optional check box previously during the Select Products action.

  To customize product features after the installation, click **Modify** in the MySQL Installer dashboard.

- **Installation.** The status of each product in the list changes from `Ready to Install`, to `Installing`, and lastly to `Complete`. During the process, click **Show Details** to view the installation actions.

  If you cancel the installation at this point, the products are installed, but the server (if installed) is not yet configured. To restart the server configuration, open MySQL Installer from the Start menu and click **Reconfigure** next to the appropriate server in the dashboard.

- **Product configuration.** This step applies to MySQL Server, MySQL Router, and samples only. The status for each item in the list should indicate `Ready to Configure`. Click **Next** to start the configuration wizard for all items in the list. The configuration options presented during this step are specific to the version of database or router that you selected to install.

  Click **Execute** to begin applying the configuration options or click **Back** (repeatedly) to return to each configuration page.

- **Installation complete.** This step finalizes the installation for products that do not require configuration. It enables you to copy the log to a clipboard and to start certain applications, such as MySQL Workbench and MySQL Shell. Click **Finish** to open the MySQL Installer dashboard.

MySQL Server Configuration with MySQL Installer

MySQL Installer performs the initial configuration of the MySQL server. For example:

- It creates the configuration file (`my.ini`) that is used to configure the MySQL server. The values written to this file are influenced by choices you make during the installation process. Some definitions are host dependent. For example, query_cache is enabled if the host has fewer than three cores.

  Note

  Query cache was deprecated in MySQL 5.7 and removed in MySQL 8.0 (and later).

- By default, a Windows service for the MySQL server is added.

- Provides default installation and data paths for MySQL server. For instructions on how to change the default paths, see Section 2.3.3.2, “Setting Alternative Server Paths with MySQL Installer”.

- It can optionally create MySQL server user accounts with configurable permissions based on general roles, such as DB Administrator, DB Designer, and Backup Admin. It optionally creates a Windows user named `MysqlSys` with limited privileges, which would then run the MySQL Server.

User accounts may also be added and configured in MySQL Workbench.

- Checking **Show Advanced Options** enables additional **Logging Options** to be set. This includes defining custom file paths for the error log, general log, slow query log (including the configuration of seconds it requires to execute a query), and the binary log.

During the configuration process, click **Next** to proceed to the next step or **Back** to return to the previous step. Click **Execute** at the final step to apply the server configuration.

The sections that follow describe the server configuration options that apply to MySQL server on Windows. The server version you installed will determine which steps and options you can configure. Configuring MySQL server may include some or all of the steps.

Type and Networking

- Server Configuration Type

  Choose the MySQL server configuration type that describes your setup. This setting defines the amount of system resources (memory) to assign to your MySQL server instance.

  - **Development**: A computer that hosts many other applications, and typically this is your personal workstation. This setting configures MySQL to use the least amount of memory.
  - **Server**: Several other applications are expected to run on this computer, such as a web server. The Server setting configures MySQL to use a medium amount of memory.
  - **Dedicated**: A computer that is dedicated to running the MySQL server. Because no other major applications run on this server, this setting configures MySQL to use the majority of available memory.

- Connectivity

  Connectivity options control how the connection to MySQL is made. Options include:

  - **TCP/IP**: This option is selected by default. You may disable TCP/IP Networking to permit local host connections only. With the TCP/IP connection option selected, you can modify the following items:
    - **Port** for classic MySQL protocol connections. The default value is `3306`.
    - **X Protocol Port** shown when configuring MySQL 8.0 server only. The default value is `33060`
    - **Open Windows Firewall port for network access**, which is selected by default for TCP/IP connections.

    If a port number is in use already, you will see the information icon ( ) next to the default value and **Next** is disabled until you provide a new port number.
  - **Named Pipe**: Enable and define the pipe name, similar to setting the `named_pipe` system variable. The default name is `MySQL`.
  - **Shared Memory**: Enable and define the memory name, similar to setting the `shared_memory` system variable. The default name is `MySQL`.

- Advanced Configuration

  Check **Show Advanced and Logging Options** to set custom logging and advanced options in later steps. The Logging Options step enables you to define custom file paths for the error log, general log, slow query log (including the configuration of seconds it requires to execute a query), and the binary log. The Advanced Options step enables you to set the unique server ID required when binary logging is enabled in a replication topology.

- MySQL Enterprise Firewall (Enterprise Edition only)

  The **Enable MySQL Enterprise Firewall** check box is deselected by default. Select this option to enable a security list that offers protection against certain types of attacks. Additional post-installation configuration is required (see Section 6.4.5, “MySQL Enterprise Firewall”).

Important

There is an issue for MySQL 8.0.19 that prevents the server from starting if MySQL Enterprise Firewall is selected during the server configuration steps. If the server startup operation fails, click **Cancel** to end the configuration process and return to the dashboard. You must uninstall the server.

The workaround is to run MySQL Installer without MySQL Enterprise Firewall selected. (That is, do not select the **Enable MySQL Enterprise Firewall** check box.) Then install MySQL Enterprise Firewall afterward using the instructions for manual installation (see Section 6.4.5.2, “Installing or Uninstalling MySQL Enterprise Firewall”).

Authentication Method

The **Authentication Method** step is visible only during the installation or upgrade of MySQL 8.0.4 or higher. It introduces a choice between two server-side authentication options. The MySQL user accounts that you create in the next step will use the authentication method that you select in this step.

MySQL 8.0 connectors and community drivers that use `libmysqlclient` 8.0 now support the `mysql_native_password` default authentication plugin. However, if you are unable to update your clients and applications to support this new authentication method, you can configure the MySQL server to use `mysql_native_password` for legacy authentication. For more information about the implications of this change, see caching_sha2_password as the Preferred Authentication Plugin.

If you are installing or upgrading to MySQL 8.0.4 or higher, select one of the following authentication methods:

- Use Strong Password Encryption for Authentication (RECOMMENDED)

  MySQL 8.0 supports a new authentication based on improved, stronger SHA256-based password methods. It is recommended that all new MySQL server installations use this method going forward.

Important

The `caching_sha2_password` authentication plugin on the server requires new versions of connectors and clients, which add support for the new MySQL 8.0 default authentication.

- Use Legacy Authentication Method (Retain MySQL 5.x Compatibility)

  Using the old MySQL 5.x legacy authentication method should be considered only in the following cases:

  - Applications cannot be updated to use MySQL 8.0 connectors and drivers.
  - Recompilation of an existing application is not feasible.
  - An updated, language-specific connector or driver is not available yet.

Accounts and Roles

- Root Account Password

  Assigning a root password is required and you will be asked for it when performing other MySQL Installer operations. Password strength is evaluated when you repeat the password in the box

provided. For descriptive information regarding password requirements or status, move your mouse pointer over the information icon ( ) when it appears.

- MySQL User Accounts (Optional)

  Click **Add User** or **Edit User** to create or modify MySQL user accounts with predefined roles. Next, enter the required account credentials:

  - **User Name:** MySQL user names can be up to 32 characters long.
  - **Host:** Select `localhost` for local connections only or `<All Hosts (%)>` when remote connections to the server are required.
  - **Role:** Each predefined role, such as `DB Admin`, is configured with its own set of privileges. For example, the `DB Admin` role has more privileges than the `DB Designer` role. The **Role** drop-down list contains a description of each role.
  - **Password:** Password strength assessment is performed while you type the password. Passwords must be confirmed. MySQL permits a blank or empty password (considered to be insecure).

  MySQL Installer Commercial Release Only: MySQL Enterprise Edition for Windows, a commercial product, also supports an authentication method that performs external authentication on Windows. Accounts authenticated by the Windows operating system can access the MySQL server without providing an additional password.

  To create a new MySQL account that uses Windows authentication, enter the user name and then select a value for **Host** and **Role**. Click **Windows** authentication to enable the `authentication_windows` plugin. In the Windows Security Tokens area, enter a token for each Windows user (or group) who can authenticate with the MySQL user name. MySQL accounts can include security tokens for both local Windows users and Windows users that belong to a domain. Multiple security tokens are separated by the semicolon character (`;`) and use the following format for local and domain accounts:

  - Local account

    Enter the simple Windows user name as the security token for each local user or group; for example, **`finley;jeffrey;admin`**.

  - Domain account

    Use standard Windows syntax (*`domain\domainuser`*) or MySQL syntax (*`domain\\domainuser`*) to enter Windows domain users and groups.

    For domain accounts, you may need to use the credentials of an administrator within the domain if the account running MySQL Installer lacks the permissions to query the Active Directory. If this is the case, select **Validate Active Directory users with** to activate the domain administrator credentials.

  Windows authentication permits you to test all of the security tokens each time you add or modify a token. Click **Test Security Tokens** to validate (or revalidate) each token. Invalid tokens generate a descriptive error message along with a red `X` icon and red token text. When all tokens resolve as valid (green text without an `X` icon), you can click **OK** to save the changes.

Windows Service

On the Windows platform, MySQL server can run as a named service managed by the operating system and be configured to start up automatically when Windows starts. Alternatively, you can configure MySQL server to run as an executable program that requires manual configuration.

- **Configure MySQL server as a Windows service** (Selected by default.)

When the default configuration option is selected, you can also select the following:

- **Start the MySQL Server at System Startup**

  When selected (default), the service startup type is set to Automatic; otherwise, the startup type is set to Manual.

- **Run Windows Service as**

  When **Standard System Account** is selected (default), the service logs on as Network Service.

  The **Custom User** option must have privileges to log on to Microsoft Windows as a service. The **Next** button will be disabled until this user is configured with the required privileges.

  A custom user account is configured in Windows by searching for "local security policy" in the Start menu. In the Local Security Policy window, select **Local Policies**, **User Rights Assignment**, and then **Log On As A Service** to open the property dialog. Click **Add User or Group** to add the custom user and then click **OK** in each dialog to save the changes.

- Deselect the Windows Service option

Logging Options

This step is available if the **Show Advanced Configuration** check box was selected during the **Type and Networking** step. To enable this step now, click **Back** to return to the **Type and Networking** step and select the check box.

Advanced configuration options are related to the following MySQL log files:

- Error Log
- General Log
- Slow Query Log
- Bin Log

Note

The binary log is enabled by default for MySQL 5.7 and higher.

Advanced Options

This step is available if the **Show Advanced Configuration** check box was selected during the **Type and Networking** step. To enable this step now, click **Back** to return to the **Type and Networking** step and select the check box.

The advanced-configuration options include:

- **Server ID**

  Set the unique identifier used in a replication topology. If binary logging is enabled, you must specify a server ID. The default ID value depends on the server version. For more information, see the description of the `server_id` system variable.

- **Table Names Case**

  You can set the following options during the initial and subsequent configuration the server. For the MySQL 8.0 release series, these options apply only to the initial configuration of the server.

  - Lower Case

Sets the `lower_case_table_names` option value to 1 (default), in which table names are stored in lowercase on disk and comparisons are not case-sensitive.

- Preserve Given Case

  Sets the `lower_case_table_names` option value to 2, in which table names are stored as given but compared in lowercase.

Apply Server Configuration

All configuration settings are applied to the MySQL server when you click **Execute**. Use the **Configuration Steps** tab to follow the progress of each action; the icon for each toggles from white to green (with a check mark) on success. Otherwise, the process stops and displays an error message if an individual action times out. Click the **Log** tab to view the log.

When the installation completes successfully and you click **Finish**, MySQL Installer and the installed MySQL products are added to the Microsoft Windows Start menu under the `MySQL` group. Opening MySQL Installer loads the dashboard where installed MySQL products are listed and other MySQL Installer operations are available.

MySQL Router Configuration with MySQL Installer

MySQL Installer downloads and installs a suite of tools for developing and managing business-critical applications on Windows. The suite consists of applications, connectors, documentation, and samples.

During the initial setup, choose any predetermined setup type, except `Server only`, to install the latest GA version of the tools. Use the `Custom` setup type to install an individual tool or specific version. If MySQL Installer is installed on the host already, use the **Add** operation to select and install tools from the MySQL Installer dashboard.

MySQL Router Configuration

MySQL Installer provides a configuration wizard that can bootstrap an installed instance of MySQL Router 8.0 to direct traffic between MySQL applications and an InnoDB Cluster. When configured, MySQL Router runs as a local Windows service.

Note

You are prompted to configure MySQL Router after the initial installation and when you reconfigure an installed router explicitly. In contrast, the upgrade operation does not require or prompt you to configure the upgraded product.

To configure MySQL Router, do the following:

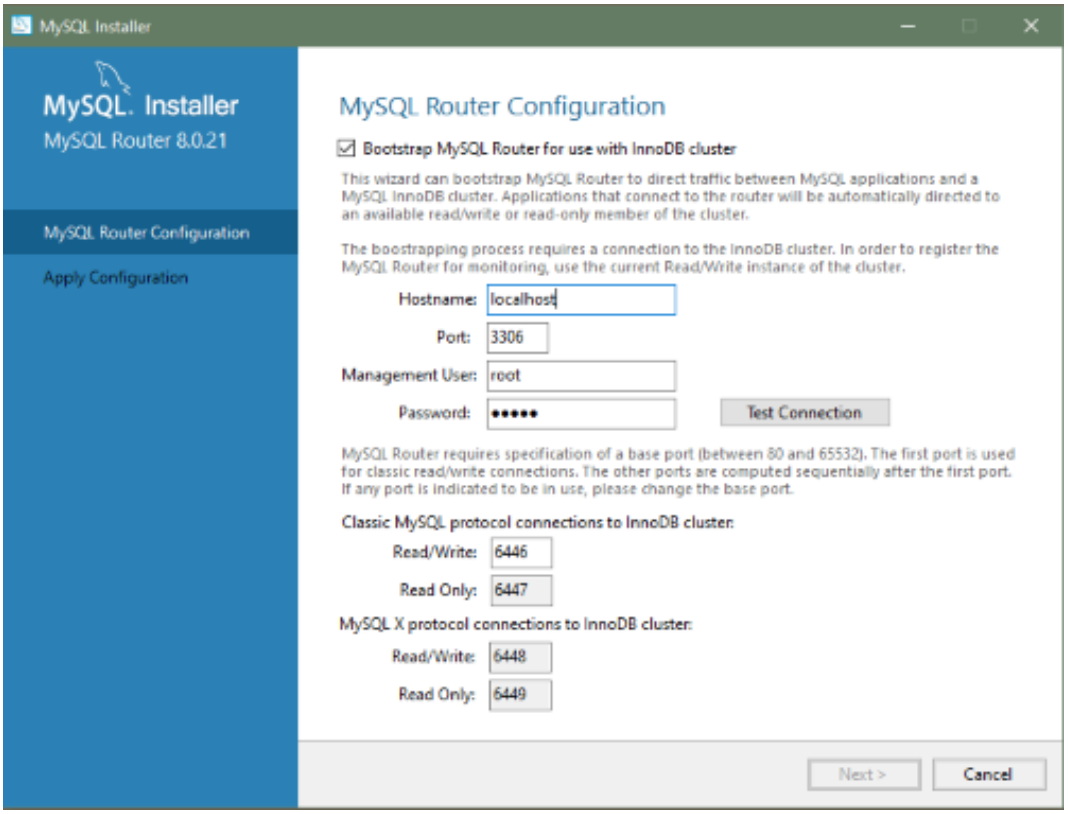
1. Set up InnoDB Cluster.
2. Using MySQL Installer, download and install the MySQL Router application. After the installation finishes, the configuration wizard prompts you for information. Select the **Configure MySQL Router for InnoDB Cluster** check box to begin the configuration and provide the following configuration values:

   - **Hostname:** Host name of the primary (seed) server in the InnoDB Cluster (`localhost` by default).
   - **Port:** The port number of the primary (seed) server in the InnoDB Cluster (`3306` by default).
   - **Management User:** An administrative user with root-level privileges.
   - **Password:** The password for the management user.
   - **Classic MySQL protocol connections to InnoDB Cluster**

Read/Write: Set the first base port number to one that is unused (between 80 and 65532) and the wizard will select the remaining ports for you.

The figure that follows shows an example of the MySQL Router configuration page, with the first base port number specified as 6446 and the remaining ports set by the wizard to 6447, 6448, and 6449.

Figure 2.10 MySQL Router Configuration



3. Click **Next** and then **Execute** to apply the configuration. Click **Finish** to close MySQL Installer or return to the MySQL Installer dashboard.

After configuring MySQL Router, the root account exists in the user table as `root@localhost` (local) only, instead of `root@%` (remote). Regardless of where the router and client are located, even if both are located on the same host as the seed server, any connection that passes through the router is viewed by server as being remote, not local. As a result, a connection made to the server using the local host (see the example that follows), does not authenticate.

```
shell> \c root@localhost:6446
```

2.3.3.4 MySQL Installer Product Catalog and Dashboard

This section describes the MySQL Installer product catalog, the dashboard, and other actions related to product selection and upgrades.

- Product Catalog
- MySQL Installer Dashboard
- Locating Products to Install
- Upgrading MySQL Server
- Removing MySQL Server

- Upgrading MySQL Installer

Product Catalog

The product catalog stores the complete list of released MySQL products for Microsoft Windows that are available to download from MySQL Downloads. By default, and when an Internet connection is present, MySQL Installer attempts to update the catalog at startup every seven days. You can also update the catalog manually from the dashboard (described later).

An up-to-date catalog performs the following actions:

- Populates the **Available Products** pane of the Select Products page. This step appears when you select:
  - The `Custom` setup type during the initial setup.
  - The **Add** operation from the dashboard.
- Identifies when product updates are available for the installed products listed in the dashboard.

The catalog includes all development releases (Pre-Release), general releases (Current GA), and minor releases (Other Releases). Products in the catalog will vary somewhat, depending on the MySQL Installer release that you download.

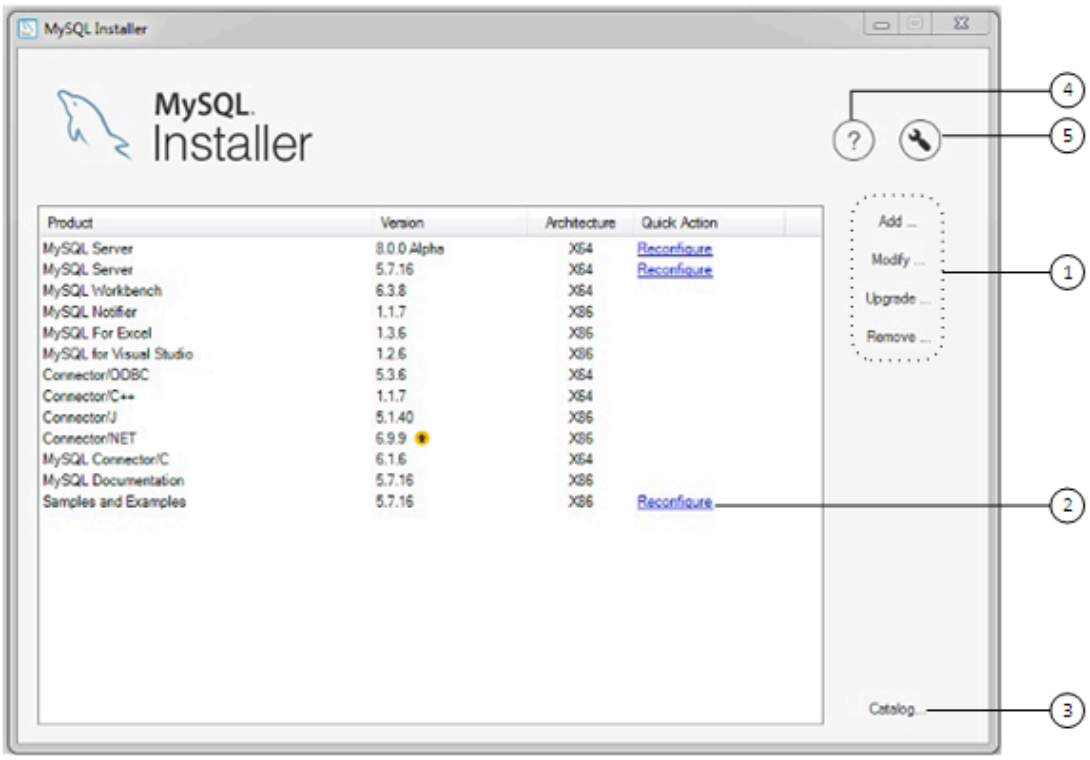
MySQL Installer Dashboard

The MySQL Installer dashboard is the default view that you see when you start MySQL Installer after the initial setup finishes. If you closed MySQL Installer before the setup was finished, MySQL Installer resumes the initial setup before it displays the dashboard.

Note

Products covered under Oracle Lifetime Sustaining Support, if installed, may appear in the dashboard. These products, such as MySQL for Excel and MySQL Notifier, can be modified or removed only.

Figure 2.11 MySQL Installer Dashboard Elements



Description of MySQL Installer Dashboard Elements

1. MySQL Installer dashboard operations provide a variety of actions that apply to installed products or products listed in the catalog. To initiate the following operations, first click the operation link and then select the product or products to manage:

   - **Add**: This operation opens the Select Products page. From there you can adjust the filter, select one or more products to download (as needed), and begin the installation. For hints about using the filter, see Locating Products to Install.

     Use the directional arrows to move each product from the **Available Products** column to the **Products To Be Installed** column. To enable the Product Features page where you can customize features, click the related check box (disabled by default).

     Note

     For MySQL Server versions 8.0.20 (and earlier), 5.7, and 5.6, the account you use to run MySQL Installer may not have adequate permission to install the server data files and this can interrupt the installation because the `ExecSecureObjects` MSI action cannot be executed. To proceed, deselect the **Server data files** feature before attempting to install the server again.

     The **Server data files** check box was removed from the feature tree for MySQL Server 8.0.21 (or higher).

   - **Modify**: Use this operation to add or remove the features associated with installed products. Features that you can modify vary in complexity by product. When the **Program Shortcut** check box is selected, the product appears in the Start menu under the `MySQL` group.

   - **Upgrade**: This operation loads the Select Products to Upgrade page and populates it with all the upgrade candidates. An installed product can have more than one upgrade version and the operation requires a current product catalog. MySQL Installer upgrades all of the selected products in one action. Click **Show Details** to view the actions performed by MySQL Installer.

   - **Remove**: This operation opens the Remove Products page and populates it with the MySQL products installed on the host. Select the MySQL products you want to remove (uninstall) and then click **Execute** to begin the removal process. During the operation, an indicator shows the number of steps that are executed as a percentage of all steps.

     To select products to remove, do one of the following:

     - Select the check box for one or more products.
     - Select the **Product** check box to select all products.

2. The **Reconfigure** link in the Quick Action column next to each installed server loads the current configuration values for the server and then cycles through all configuration steps enabling you to change the options and values. You must provide credentials with root privileges to reconfigure these items. Click the **Log** tab to show the output of each configuration step performed by MySQL Installer.

   On completion, MySQL Installer stops the server, applies the configuration changes, and restarts the server for you. For a description of each configuration option, see MySQL Server Configuration with MySQL Installer. Installed `Samples and Examples` associated with a specific MySQL server version can be also be reconfigured to apply new feature settings, if any.

3. The **Catalog** link enables you to download the latest catalog of MySQL products manually and then to integrate those product changes with MySQL Installer. The catalog-download action does not perform an upgrade of the products already installed on the host. Instead, it returns to the

dashboard and adds an arrow icon to the Version column for each installed product that has a newer version. Use the **Upgrade** operation to install the newer product version.

You can also use the **Catalog** link to display the current change history of each product without downloading the new catalog. Select the **Do not update at this time** check box to view the change history only.

4. The MySQL Installer About icon ( ) shows the current version of MySQL Installer and general information about MySQL. The version number is located above the **Back** button.

   Tip

   Always include this version number when reporting a problem with MySQL Installer.

   In addition to the About MySQL information ( ), you can also select the following icons from the side panel:

   - License icon ( ) for MySQL Installer.

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

   - Resource links icon ( ) to the latest MySQL product documentation, blogs, webinars, and more.

5. The MySQL Installer Options icon ( ) includes the following tabs:

   - **Product Catalog**: Manages the automatic catalog updates. By default, MySQL Installer checks for catalog updates at startup every seven days. When new products or product versions are available, MySQL Installer adds them to the catalog and then inserts an arrow icon ( ) next to the version number of installed products listed in the dashboard.

     Use the product catalog option to enable or disable automatic updates and to reset the number of days between automatic catalog downloads. At startup, MySQL Installer uses the number of days you set to determine whether a download should be attempted. This action is repeated during next startup if MySQL Installer encounters an error downloading the catalog.

   - **Connectivity Settings**: Several operations performed by MySQL Installer require internet access. This option enables you to use a default value to validate the connection or to use a different URL, one selected from a list or added by you manually. With the **Manual** option selected, new URLs can be added and all URLs in the list can be moved or deleted. When the **Automatic** option is selected, MySQL Installer attempts to connect to each default URL in the list (in order) until a connection is made. If no connection can be made, it raises an error.

Locating Products to Install

MySQL products in the catalog are listed by category: MySQL Servers, Applications, MySQL Connectors, and Documentation. Only the latest GA versions appear in the **Available Products** pane

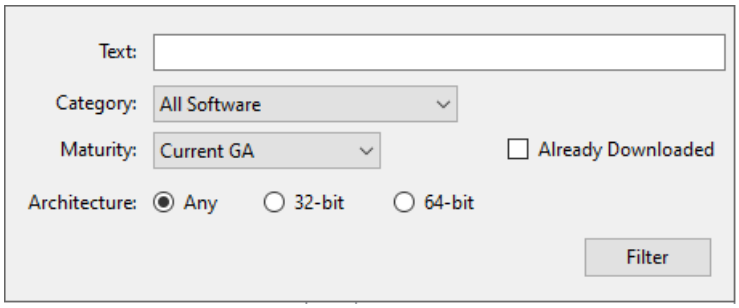
by default. If you are looking for a pre-release or older version of a product, it may not be visible in the default list.

Note

Keep the product catalog up-to-date. Click **Catalog** on the MySQL Installer dashboard to download the latest manifest.

To change the default product list, click **Add** in the dashboard to open the Select Products page, and then click **Edit** to open the dialog box shown in the figure that follows. Modify the settings and then click **Filter**.

Figure 2.12 Filter Available Products



Reset one or more of the following fields to modify the list of available products:

- Text: Filter by text.
- Category: All Software (default), MySQL Servers, Applications, MySQL Connectors, or Documentation (for samples and documentation).
- Maturity: Current Bundle (appears initially with the full package only), Pre-Release, Current GA, or Other Releases. If you see a warning, confirm that you have the most recent product manifest by clicking **Catalog** on the MySQL Installer dashboard. If MySQL Installer is unable to download the manifest, the range of products you see is limited to bundled products, standalone product MSIs located in the `Product Cache` folder already, or both.

Note

The Commercial release of MySQL Installer does not display any MySQL products when you select the Pre-Release maturity filter. Products in development are available from the Community release of MySQL Installer only.

- Already Downloaded (the check box is deselected by default). Permits you to view and manage downloaded products only.
- Architecture: Any (default), 32-bit, or 64-bit.

Upgrading MySQL Server

Important server upgrade conditions:

- MySQL Installer does not permit server upgrades between major release versions or minor release versions, but does permit upgrades within a release series, such as an upgrade from 5.7.18 to 5.7.19.
- Upgrades between milestone releases (or from a milestone release to a GA release) are not supported. Significant development changes take place in milestone releases and you may encounter compatibility issues or problems starting the server.
- For upgrades to MySQL 8.0.16 server and higher, a check box enables you to skip the upgrade check and process for system tables, while checking and processing data dictionary tables normally.

MySQL Installer does not prompt you with the check box when the previous server upgrade was skipped or when the server was configured as a sandbox InnoDB Cluster. This behavior represents a change in how MySQL Server performs an upgrade (see What the MySQL Upgrade Process Upgrades) and it alters the sequence of steps that MySQL Installer applies to the configuration process.

If you select **Skip system tables upgrade check and process. (Not recommended)**, MySQL Installer starts the upgraded server with the `--upgrade=MINIMAL` server option, which upgrades the data dictionary only. If you stop and then restart the server without the `--upgrade=MINIMAL` option, the server upgrades the system tables automatically, if needed.

The following information appears in the **Log** tab and log file after the upgrade configuration (with system tables skipped) is complete:

```
WARNING: The system tables upgrade was skipped after upgrading MySQL Server. The
server will be started now with the --upgrade=MINIMAL option, but then each
time the server is started it will attempt to upgrade the system tables, unless
you modify the Windows service (command line) to add --upgrade=MINIMAL to bypass
the upgrade.

FOR THE BEST RESULTS: Run mysqld.exe --upgrade=FORCE on the command line to upgrade
the system tables manually.
```

To choose a new server version:

1. Click **Upgrade**. Confirm that the check box next to product name in the **Upgradeable Products** pane has a check mark. Deselect the products that you do not intend to upgrade at this time.

Note

For server milestone releases in the same release series, MySQL Installer deselects the server upgrade and displays a warning to indicate that the upgrade is not supported, identifies the risks of continuing, and provides a summary of the steps to perform a logical upgrade manually. You can reselect server upgrade at your own risk. For instructions on how to perform a logical upgrade with a milestone release, see Logical Upgrade.

2. Click a product in the list to highlight it. This action populates the **Upgradeable Versions** pane with the details of each available version for the selected product: version number, published date, and a `Changes` link to open the release notes for that version.

Removing MySQL Server

To remove a local MySQL server:

1. Determine whether the local data directory should be removed. If you retain the data directory, another server installation can reuse the data. This option is enabled by default (removes the data directory).
2. Click **Execute** to begin uninstalling the local server. Note that all products that you selected to remove are also uninstalled at this time.
3. (Optional) Click the **Log** tab to display the current actions performed by MySQL Installer.

Upgrading MySQL Installer

MySQL Installer remains installed on your computer, and like other software, MySQL Installer can be upgraded from the previous version. In some cases, other MySQL software may require that you upgrade MySQL Installer for compatibility. This section describes how to identify the current version of MySQL Installer and how to upgrade MySQL Installer manually.

To locate the installed version of MySQL Installer:

1. Start MySQL Installer from the search menu. The MySQL Installer dashboard opens.

2. Click the MySQL Installer About icon ( ). The version number is located above the **Back** button.

To initiate an on-demand upgrade of MySQL Installer:

1. Connect the computer with MySQL Installer installed to the internet.
2. Start MySQL Installer from the search menu. The MySQL Installer dashboard opens.
3. Click **Catalog** on the bottom of the dashboard to open the Update Catalog window.
4. Click **Execute** to begin the process. If the installed version of MySQL Installer can be upgraded, you will be prompted to start the upgrade.
5. Click **Next** to review all changes to the catalog and then click **Finish** to return to the dashboard.
6. Verify the (new) installed version of MySQL Installer (see the previous procedure).

2.3.3.5 MySQLInstallerConsole Reference

`MySQLInstallerConsole.exe` provides command-line functionality that is similar to MySQL Installer. It is installed when MySQL Installer is initially executed and then available within the `MySQL Installer for Windows` directory. By default, that is in `C:\Program Files (x86)\MySQL\MySQL Installer for Windows`, and the console must be executed with administrative privileges.

To use, invoke the command prompt with administrative privileges by choosing **Start**, **Accessories**, then right-click on **Command Prompt** and choose `Run as administrator`. And from the command line, optionally change the directory to where `MySQLInstallerConsole.exe` is located:

```
C:\> cd Program Files (x86)\MySQL\MySQL Installer for Windows
C:\Program Files (x86)\MySQL\MySQL Installer for Windows> MySQLInstallerConsole.exe help
=================== Start Initialization ===================
MySQL Installer is running in Community mode

Attempting to update manifest.
Initializing product requirements
Loading product catalog
Checking for product catalog snippets
Checking for product packages in the bundle
Categorizing product catalog
Finding all installed packages.
Your product catalog was last updated at 11/1/2016 4:10:38 PM
=================== End Initialization ===================

The following commands are available:

Configure - Configures one or more of your installed programs.
Help      - Provides list of available commands.
Install   - Install and configure one or more available MySQL programs.
List      - Provides an interactive way to list all products available.
Modify    - Modifies the features of installed products.
Remove    - Removes one or more products from your system.
Status    - Shows the status of all installed products.
Update    - Update the current product catalog.
Upgrade   - Upgrades one or more of your installed programs.
```

MySQL Product Names

Many of the `MySQLInstallerConsole` commands accept one or more keywords that represent a MySQL product (or products) in the catalog. The current set of valid keywords for use with commands is shown in the following table.

Table 2.6 MySQL Product Keywords for MySQLInstallerConsole

Keyword	MySQL Product
`server`	MySQL Server

Keyword	MySQL Product
workbench	MySQL Workbench
shell	MySQL Shell
visual	MySQL for Visual Studio
router	MySQL Router
backup	MySQL Enterprise Backup
net	MySQL Connector/NET
odbc	MySQL Connector/ODBC
c++	MySQL Connector/C++
python	MySQL Connector/Python
j	MySQL Connector/J
documentation	MySQL Server Documentation
samples	MySQL Samples (sakila and world databases)

`MySQLInstallerConsole` Command Options

`MySQLInstallerConsole.exe` supports the following command options:

Note

Configuration block values that contain a colon character (`:`) must be wrapped in quotation marks. For example, `installdir="C:\MySQL\MySQL Server 8.0"`.

- `configure [product1]:[setting]=[value]; [product2]:[setting]=[value]; [...]`

  Configures one or more MySQL products on your system. Multiple setting=value pairs can be configured for each product.

  Switches include:

  `-showsettings` — Displays the available options for the selected product, by passing in the product name after `-showsettings`.

  `-silent` — Disables confirmation prompts.

```
C:\> MySQLInstallerConsole configure -showsettings server
C:\> MySQLInstallerConsole configure server:port=3307
```

- `help [command]`

  Displays a help message with usage examples and then exits. Pass in an additional command to receive help specific to that command.

```
C:\> MySQLInstallerConsole help
C:\> MySQLInstallerConsole help install
```

- `install [product]:[features]:[config block]:[config block]:[config block]; [...]`

  Installs one or more MySQL products on your system. If pre-release products are available, both GA and pre-release products are installed when the value of the `-type` switch is `Developer`, `Client`, or `Full`. Use the `-only_ga_products` switch to restrict the product set to GA products only when using these setup types.

  Switches and syntax options include:

`-only_ga_products`	Restricts the product set to include GA products only.
`-type=[SetupType]`	Installs a predefined set of software. The setup type can be one of the following: • `Developer`: Installs a complete development environment. • `Server`: Installs a single MySQL server • `Client`: Installs client programs and libraries • `Full`: Installs everything • `Custom`: Installs user-selected products. This is the default option. **Note** Non-custom setup types are valid only when no other MySQL products are installed.
`-showsettings`	Displays the available options for the selected product, by passing in the product name after `-showsettings`.
`-silent`	Disable confirmation prompts.
`[product]`	Each product can be specified by a product keyword with or without a semicolon-separated version qualifier. Passing in a product keyword alone selects the latest version of the product. If multiple architectures are available for that version of the product, the command returns the first one in the manifest list for interactive confirmation. Alternatively, you can pass in the exact version and architecture (`x86` or `x64`) after the product keyword using the `-silent` switch.
`[features]`	All features associated with a MySQL product are installed by default. The feature block is a semicolon-separated list of features or an asterisk character (`*`) that selects all features. To remove a feature, use the `modify` command.
`[config block]`	One or more configuration blocks can be specified. Each configuration block is a semicolon-separated list of key-value pairs. A block can include either a `config` or `user` type key; `config` is the default type if one is not defined. Configuration block values that contain a colon character (`:`) must be wrapped in quotation marks. For example, `installdir="C:\MySQL\MySQL Server 8.0"`. Only one configuration type block can be defined for each product. A user block should be defined for each user to be created during the product installation. **Note** The `user` type key is not supported when a product is being reconfigured.

```
C:\> MySQLInstallerConsole install server;5.6.25:*:port=3307;serverid=2:type=user;username=foo;password=b
```

```
C:\> MySQLInstallerConsole install server;5.6.25;x64 -silent
```

An example that passes in additional configuration blocks, separated by ^ to fit:

```
C:\> MySQLInstallerConsole install server;5.6.25;x64:*:type=config;openfirewall=true; ^
         generallog=true;binlog=true;serverid=3306;enable_tcpip=true;port=3306;rootpasswd=pass; ^
         installdir="C:\MySQL\MySQL Server 5.6":type=user;datadir="C:\MySQL\data";username=foo;passw
```

- `list`

  Lists an interactive console where all of the available MySQL products can be searched. Execute `MySQLInstallerConsole list` to launch the console and enter in a substring to search.

  ```
  C:\> MySQLInstallerConsole list
  ```

- `modify [product1:-removelist|+addlist] [product2:-removelist|+addlist] [...]`

  Modifies or displays features of a previously installed MySQL product. To display the features of a product, append the product keyword to the command, for example:

  ```
  C:\> MySQLInstallerConsole modify server
  ```

  The syntax option for this command:

`-silent`	Disable confirmation prompts.

  ```
  C:\> MySQLInstallerConsole modify server:+documentation
  C:\> MySQLInstallerConsole modify server:-debug
  ```

- `remove [product1] [product2] [...]`

  Removes one ore more products from your system. Switches and syntax options include:

`*`	Pass in * to remove all of the MySQL products.
`-continue`	Continue the operation even if an error occurs.
`-silent`	Disable confirmation prompts.

  ```
  C:\> MySQLInstallerConsole remove *
  C:\> MySQLInstallerConsole remove server
  ```

- `status`

  Provides a quick overview of the MySQL products that are installed on the system. Information includes product name and version, architecture, date installed, and install location.

  ```
  C:\> MySQLInstallerConsole status
  ```

- `update`

  Downloads the latest MySQL product catalog to your system. On success, the catalog is applied the next time either `MySQLInstaller` or `MySQLInstallerConsole` is executed.

  ```
  C:\> MySQLInstallerConsole update
  ```

  Note

  The **Automatic Catalog Update** GUI option executes this command from the Windows Task Scheduler.

- `upgrade [product1:version] [product2:version] [...]`

  Upgrades one or more products on your system. Syntax options include:

*	Pass in * to upgrade all products to the latest version, or pass in specific products.
!	Pass in ! as a version number to upgrade the MySQL product to its latest version.
-silent	Disable confirmation prompts.

```
C:\> MySQLInstallerConsole upgrade *
C:\> MySQLInstallerConsole upgrade workbench:8.0.21
C:\> MySQLInstallerConsole upgrade workbench:!
C:\> MySQLInstallerConsole upgrade workbench:8.0.21 visual:1.2.9
```

2.3.4 Installing MySQL on Microsoft Windows Using a noinstall ZIP Archive

Users who are installing from the noinstall package can use the instructions in this section to manually install MySQL. The process for installing MySQL from a ZIP Archive package is as follows:

1. Extract the archive to the desired install directory
2. Create an option file
3. Choose a MySQL server type
4. Start the MySQL server
5. Secure the default user accounts

This process is described in the sections that follow.

2.3.4.1 Extracting the Install Archive

To install MySQL manually, do the following:

1. If you are upgrading from a previous version please refer to Section 2.11.8, “Upgrading MySQL on Windows”, before beginning the upgrade process.
2. Make sure that you are logged in as a user with administrator privileges.
3. Choose an installation location. Traditionally, the MySQL server is installed in C:\mysql. If you do not install MySQL at C:\mysql, you must specify the path to the install directory during startup or in an option file. See Section 2.3.4.2, “Creating an Option File”.

Note

The MySQL Installer installs MySQL under C:\Program Files\MySQL.

4. Extract the install archive to the chosen installation location using your preferred file-compression tool. Some tools may extract the archive to a folder within your chosen installation location. If this occurs, you can move the contents of the subfolder into the chosen installation location.

2.3.4.2 Creating an Option File

If you need to specify startup options when you run the server, you can indicate them on the command line or place them in an option file. For options that are used every time the server starts, you may find it most convenient to use an option file to specify your MySQL configuration. This is particularly true under the following circumstances:

- The installation or data directory locations are different from the default locations (C:\Program Files\MySQL\MySQL Server 5.6 and C:\Program Files\MySQL\MySQL Server 5.6\data).

- You need to tune the server settings, such as memory, cache, or InnoDB configuration information.

When the MySQL server starts on Windows, it looks for option files in several locations, such as the Windows directory, C:\, and the MySQL installation directory (for the full list of locations, see Section 4.2.2.2, “Using Option Files”). The Windows directory typically is named something like C:\WINDOWS. You can determine its exact location from the value of the WINDIR environment variable using the following command:

```
C:\> echo %WINDIR%
```

MySQL looks for options in each location first in the my.ini file, and then in the my.cnf file. However, to avoid confusion, it is best if you use only one file. If your PC uses a boot loader where C: is not the boot drive, your only option is to use the my.ini file. Whichever option file you use, it must be a plain text file.

Note

When using the MySQL Installer to install MySQL Server, it creates my.ini at the default location. As of MySQL 5.5.27, the user running MySQL Installer is granted full permissions to this new my.ini.

In other words, be sure that the MySQL Server user has permission to read the my.ini file.

You can also make use of the example option files included with your MySQL distribution; see Section 5.1.2, “Server Configuration Defaults”.

An option file can be created and modified with any text editor, such as Notepad. For example, if MySQL is installed in E:\mysql and the data directory is in E:\mydata\data, you can create an option file containing a [mysqld] section to specify values for the basedir and datadir options:

```
[mysqld]
# set basedir to your installation path
basedir=E:/mysql
# set datadir to the location of your data directory
datadir=E:/mydata/data
```

Microsoft Windows path names are specified in option files using (forward) slashes rather than backslashes. If you do use backslashes, double them:

```
[mysqld]
# set basedir to your installation path
basedir=E:\\mysql
# set datadir to the location of your data directory
datadir=E:\\mydata\\data
```

The rules for use of backslash in option file values are given in Section 4.2.2.2, “Using Option Files”.

The data directory is located within the AppData directory for the user running MySQL.

If you would like to use a data directory in a different location, you should copy the entire contents of the data directory to the new location. For example, if you want to use E:\mydata as the data directory instead, you must do two things:

1. Move the entire data directory and all of its contents from the default location (for example C:\Program Files\MySQL\MySQL Server 5.6\data) to E:\mydata.
2. Use a --datadir option to specify the new data directory location each time you start the server.

2.3.4.3 Selecting a MySQL Server Type

The following table shows the available servers for Windows in MySQL 5.6.

Binary	Description
mysqld	Optimized binary with named-pipe support

Binary	Description
mysqld-debug	Like mysqld, but compiled with full debugging and automatic memory allocation checking

All of the preceding binaries are optimized for modern Intel processors, but should work on any Intel i386-class or higher processor.

Each of the servers in a distribution support the same set of storage engines. The SHOW ENGINES statement displays which engines a given server supports.

All Windows MySQL 5.6 servers have support for symbolic linking of database directories.

MySQL supports TCP/IP on all Windows platforms. MySQL servers on Windows also support named pipes, if you start the server with the named_pipe system variable enabled. It is necessary to enable this variable explicitly because some users have experienced problems with shutting down the MySQL server when named pipes were used. The default is to use TCP/IP regardless of platform because named pipes are slower than TCP/IP in many Windows configurations.

2.3.4.4 Starting the Server for the First Time

This section gives a general overview of starting the MySQL server. The following sections provide more specific information for starting the MySQL server from the command line or as a Windows service.

The information here applies primarily if you installed MySQL using the noinstall version, or if you wish to configure and test MySQL manually rather than with the GUI tools.

Note

MySQL server starts automatically after using MySQL Installer.

The examples in these sections assume that MySQL is installed under the default location of C:\Program Files\MySQL\MySQL Server 5.6. Adjust the path names shown in the examples if you have MySQL installed in a different location.

Clients have two options. They can use TCP/IP, or they can use a named pipe if the server supports named-pipe connections.

MySQL for Windows also supports shared-memory connections if the server is started with the shared_memory system variable enabled. Clients can connect through shared memory by using the --protocol=MEMORY option.

For information about which server binary to run, see Section 2.3.4.3, “Selecting a MySQL Server Type”.

Testing is best done from a command prompt in a console window (or “DOS window”). In this way you can have the server display status messages in the window where they are easy to see. If something is wrong with your configuration, these messages make it easier for you to identify and fix any problems.

To start the server, enter this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.6\bin\mysqld" --console
```

For a server that includes InnoDB support, you should see the messages similar to those following as it starts (the path names and sizes may differ):

```
InnoDB: The first specified datafile c:\ibdata\ibdata1 did not exist:
InnoDB: a new database to be created!
InnoDB: Setting file c:\ibdata\ibdata1 size to 209715200
InnoDB: Database physically writes the file full: wait...
InnoDB: Log file c:\iblogs\ib_logfile0 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile0 size to 31457280
```

```
InnoDB: Log file c:\iblogs\ib_logfile1 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile1 size to 31457280
InnoDB: Log file c:\iblogs\ib_logfile2 did not exist: new to be created
InnoDB: Setting log file c:\iblogs\ib_logfile2 size to 31457280
InnoDB: Doublewrite buffer not found: creating new
InnoDB: Doublewrite buffer created
InnoDB: creating foreign key constraint system tables
InnoDB: foreign key constraint system tables created
011024 10:58:25  InnoDB: Started
```

When the server finishes its startup sequence, you should see something like this, which indicates that the server is ready to service client connections:

```
mysqld: ready for connections
Version: '5.6.51'  socket: ''  port: 3306
```

The server continues to write to the console any further diagnostic output it produces. You can open a new console window in which to run client programs.

If you omit the `--console` option, the server writes diagnostic output to the error log in the data directory (`C:\Program Files\MySQL\MySQL Server 5.6\data` by default). The error log is the file with the `.err` extension, and may be set using the `--log-error` option.

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.10.4, “Securing the Initial MySQL Accounts”.

2.3.4.5 Starting MySQL from the Windows Command Line

The MySQL server can be started manually from the command line. This can be done on any version of Windows.

To start the `mysqld` server from the command line, you should start a console window (or “DOS window”) and enter this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.6\bin\mysqld"
```

The path to `mysqld` may vary depending on the install location of MySQL on your system.

You can stop the MySQL server by executing this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.6\bin\mysqladmin" -u root shutdown
```

Note

If the MySQL `root` user account has a password, you need to invoke `mysqladmin` with the `-p` option and supply the password when prompted.

This command invokes the MySQL administrative utility `mysqladmin` to connect to the server and tell it to shut down. The command connects as the MySQL `root` user, which is the default administrative account in the MySQL grant system.

Note

Users in the MySQL grant system are wholly independent from any operating system users under Microsoft Windows.

If `mysqld` doesn't start, check the error log to see whether the server wrote any messages there to indicate the cause of the problem. By default, the error log is located in the `C:\Program Files\MySQL\MySQL Server 5.6\data` directory. It is the file with a suffix of `.err`, or may be specified by passing in the `--log-error` option. Alternatively, you can try to start the server with the `--console` option; in this case, the server may display some useful information on the screen that may help solve the problem.

The last option is to start mysqld with the --standalone and --debug options. In this case, mysqld writes a log file C:\mysqld.trace that should contain the reason why mysqld doesn't start. See Section 5.8.3, “The DBUG Package”.

Use mysqld --verbose --help to display all the options that mysqld supports.

2.3.4.6 Customizing the PATH for MySQL Tools

Warning

You must exercise great care when editing your system PATH by hand; accidental deletion or modification of any portion of the existing PATH value can leave you with a malfunctioning or even unusable system.

To make it easier to invoke MySQL programs, you can add the path name of the MySQL bin directory to your Windows system PATH environment variable:

- On the Windows desktop, right-click the **My Computer** icon, and select **Properties**.
- Next select the **Advanced** tab from the **System Properties** menu that appears, and click the **Environment Variables** button.
- Under **System Variables**, select **Path**, and then click the **Edit** button. The **Edit System Variable** dialogue should appear.
- Place your cursor at the end of the text appearing in the space marked **Variable Value**. (Use the **End** key to ensure that your cursor is positioned at the very end of the text in this space.) Then enter the complete path name of your MySQL bin directory (for example, C:\Program Files\MySQL\MySQL Server 5.6\bin)

  Note

  There must be a semicolon separating this path from any values present in this field.

  Dismiss this dialogue, and each dialogue in turn, by clicking **OK** until all of the dialogues that were opened have been dismissed. The new PATH value should now be available to any new command shell you open, allowing you to invoke any MySQL executable program by typing its name at the DOS prompt from any directory on the system, without having to supply the path. This includes the servers, the mysql client, and all MySQL command-line utilities such as mysqladmin and mysqldump.

You should not add the MySQL bin directory to your Windows PATH if you are running multiple MySQL servers on the same machine.

2.3.4.7 Starting MySQL as a Windows Service

On Windows, the recommended way to run MySQL is to install it as a Windows service, so that MySQL starts and stops automatically when Windows starts and stops. A MySQL server installed as a service can also be controlled from the command line using NET commands, or with the graphical Services utility. Generally, to install MySQL as a Windows service you should be logged in using an account that has administrator rights.

The Services utility (the Windows Service Control Manager) can be found in the Windows Control Panel. To avoid conflicts, it is advisable to close the Services utility while performing server installation or removal operations from the command line.

Installing the service

Before installing MySQL as a Windows service, you should first stop the current server if it is running by using the following command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.6\bin\mysqladmin"
         -u root shutdown
```

Note

If the MySQL root user account has a password, you need to invoke mysqladmin with the -p option and supply the password when prompted.

This command invokes the MySQL administrative utility mysqladmin to connect to the server and tell it to shut down. The command connects as the MySQL root user, which is the default administrative account in the MySQL grant system.

Note

Users in the MySQL grant system are wholly independent from any operating system users under Windows.

Install the server as a service using this command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.6\bin\mysqld" --install
```

The service-installation command does not start the server. Instructions for that are given later in this section.

To make it easier to invoke MySQL programs, you can add the path name of the MySQL bin directory to your Windows system PATH environment variable:

- On the Windows desktop, right-click the **My Computer** icon, and select **Properties**.
- Next select the **Advanced** tab from the **System Properties** menu that appears, and click the **Environment Variables** button.
- Under **System Variables**, select **Path**, and then click the **Edit** button. The **Edit System Variable** dialogue should appear.
- Place your cursor at the end of the text appearing in the space marked **Variable Value**. (Use the **End** key to ensure that your cursor is positioned at the very end of the text in this space.) Then enter the complete path name of your MySQL bin directory (for example, C:\Program Files\MySQL\MySQL Server 5.6\bin), and there should be a semicolon separating this path from any values present in this field. Dismiss this dialogue, and each dialogue in turn, by clicking **OK** until all of the dialogues that were opened have been dismissed. You should now be able to invoke any MySQL executable program by typing its name at the DOS prompt from any directory on the system, without having to supply the path. This includes the servers, the mysql client, and all MySQL command-line utilities such as mysqladmin and mysqldump.

  You should not add the MySQL bin directory to your Windows PATH if you are running multiple MySQL servers on the same machine.

Warning

You must exercise great care when editing your system PATH by hand; accidental deletion or modification of any portion of the existing PATH value can leave you with a malfunctioning or even unusable system.

The following additional arguments can be used when installing the service:

- You can specify a service name immediately following the --install option. The default service name is MySQL.
- If a service name is given, it can be followed by a single option. By convention, this should be --defaults-file=*file_name* to specify the name of an option file from which the server should read options when it starts.

The use of a single option other than --defaults-file is possible but discouraged. --defaults-file is more flexible because it enables you to specify multiple startup options for the server by placing them in the named option file.

- You can also specify a --local-service option following the service name. This causes the server to run using the LocalService Windows account that has limited system privileges. If both --defaults-file and --local-service are given following the service name, they can be in any order.

For a MySQL server that is installed as a Windows service, the following rules determine the service name and option files that the server uses:

- If the service-installation command specifies no service name or the default service name (MySQL) following the --install option, the server uses the service name of MySQL and reads options from the [mysqld] group in the standard option files.
- If the service-installation command specifies a service name other than MySQL following the --install option, the server uses that service name. It reads options from the [mysqld] group and the group that has the same name as the service in the standard option files. This enables you to use the [mysqld] group for options that should be used by all MySQL services, and an option group with the service name for use by the server installed with that service name.
- If the service-installation command specifies a --defaults-file option after the service name, the server reads options the same way as described in the previous item, except that it reads options only from the named file and ignores the standard option files.

As a more complex example, consider the following command:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.6\bin\mysqld"
         --install MySQL --defaults-file=C:\my-opts.cnf
```

Here, the default service name (MySQL) is given after the --install option. If no --defaults-file option had been given, this command would have the effect of causing the server to read the [mysqld] group from the standard option files. However, because the --defaults-file option is present, the server reads options from the [mysqld] option group, and only from the named file.

Note

On Windows, if the server is started with the --defaults-file and --install options, --install must be first. Otherwise, mysqld.exe attempts to start the MySQL server.

You can also specify options as Start parameters in the Windows Services utility before you start the MySQL service.

Finally, before trying to start the MySQL service, make sure the user variables %TEMP% and %TMP% (and also %TMPDIR%, if it has ever been set) for the operating system user who is to run the service are pointing to a folder to which the user has write access. The default user for running the MySQL service is LocalSystem, and the default value for its %TEMP% and %TMP% is C:\Windows\Temp, a directory LocalSystem has write access to by default. However, if there are any changes to that default setup (for example, changes to the user who runs the service or to the mentioned user variables, or the --tmpdir option has been used to put the temporary directory somewhere else), the MySQL service might fail to run because write access to the temporary directory has not been granted to the proper user.

Starting the service

After a MySQL server instance has been installed as a service, Windows starts the service automatically whenever Windows starts. The service also can be started immediately from

the Services utility, or by using an sc start *mysqld_service_name* or NET START *mysqld_service_name* command. SC and NET commands are not case-sensitive.

When run as a service, mysqld has no access to a console window, so no messages can be seen there. If mysqld does not start, check the error log to see whether the server wrote any messages there to indicate the cause of the problem. The error log is located in the MySQL data directory (for example, C:\Program Files\MySQL\MySQL Server 5.6\data). It is the file with a suffix of .err.

When a MySQL server has been installed as a service, and the service is running, Windows stops the service automatically when Windows shuts down. The server also can be stopped manually using the Services utility, the sc stop *mysqld_service_name* command, the NET START *mysqld_service_name* command, or the mysqladmin shutdown command.

You also have the choice of installing the server as a manual service if you do not wish for the service to be started automatically during the boot process. To do this, use the --install-manual option rather than the --install option:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.6\bin\mysqld" --install-manual
```

Removing the service

To remove a server that is installed as a service, first stop it if it is running by executing SC STOP *mysqld_service_name* or NET STOP *mysqld_service_name*. Then use SC DELETE *mysqld_service_name* to remove it:

```
C:\> SC DELETE mysql
```

Alternatively, use the mysqld --remove option to remove the service.

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.6\bin\mysqld" --remove
```

If mysqld is not running as a service, you can start it from the command line. For instructions, see Section 2.3.4.5, “Starting MySQL from the Windows Command Line”.

If you encounter difficulties during installation, see Section 2.3.5, “Troubleshooting a Microsoft Windows MySQL Server Installation”.

For more information about stopping or removing a Windows service, see Section 5.7.2.2, “Starting Multiple MySQL Instances as Windows Services”.

2.3.4.8 Testing The MySQL Installation

You can test whether the MySQL server is working by executing any of the following commands:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.6\bin\mysqlshow"
C:\> "C:\Program Files\MySQL\MySQL Server 5.6\bin\mysqlshow" -u root mysql
C:\> "C:\Program Files\MySQL\MySQL Server 5.6\bin\mysqladmin" version status proc
C:\> "C:\Program Files\MySQL\MySQL Server 5.6\bin\mysql" test
```

If mysqld is slow to respond to TCP/IP connections from client programs, there is probably a problem with your DNS. In this case, start mysqld with the skip_name_resolve system variable enabled and use only localhost and IP addresses in the Host column of the MySQL grant tables. (Be sure that an account exists that specifies an IP address or you may not be able to connect.)

You can force a MySQL client to use a named-pipe connection rather than TCP/IP by specifying the --pipe or --protocol=PIPE option, or by specifying . (period) as the host name. Use the --socket option to specify the name of the pipe if you do not want to use the default pipe name.

If you have set a password for the root account, deleted the anonymous account, or created a new user account, then to connect to the MySQL server you must use the appropriate -u and -p options

with the commands shown previously. See Section 4.2.4, “Connecting to the MySQL Server Using Command Options”.

For more information about `mysqlshow`, see Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”.

2.3.5 Troubleshooting a Microsoft Windows MySQL Server Installation

When installing and running MySQL for the first time, you may encounter certain errors that prevent the MySQL server from starting. This section helps you diagnose and correct some of these errors.

Your first resource when troubleshooting server issues is the error log. The MySQL server uses the error log to record information relevant to the error that prevents the server from starting. The error log is located in the data directory specified in your `my.ini` file. The default data directory location is `C:\Program Files\MySQL\MySQL Server 5.6\data`, or `C:\ProgramData\Mysql` on Windows 7 and Windows Server 2008. The `C:\ProgramData` directory is hidden by default. You need to change your folder options to see the directory and contents. For more information on the error log and understanding the content, see Section 5.4.2, “The Error Log”.

For information regarding possible errors, also consult the console messages displayed when the MySQL service is starting. Use the `SC START mysqld_service_name` or `NET START mysqld_service_name` command from the command line after installing `mysqld` as a service to see any error messages regarding the starting of the MySQL server as a service. See Section 2.3.4.7, “Starting MySQL as a Windows Service”.

The following examples show other common error messages you might encounter when installing MySQL and starting the server for the first time:

- If the MySQL server cannot find the `mysql` privileges database or other critical files, it displays these messages:

```
System error 1067 has occurred.
Fatal error: Can't open and lock privilege tables:
Table 'mysql.user' doesn't exist
```

  These messages often occur when the MySQL base or data directories are installed in different locations than the default locations (`C:\Program Files\MySQL\MySQL Server 5.6` and `C:\Program Files\MySQL\MySQL Server 5.6\data`, respectively).

  This situation can occur when MySQL is upgraded and installed to a new location, but the configuration file is not updated to reflect the new location. In addition, old and new configuration files might conflict. Be sure to delete or rename any old configuration files when upgrading MySQL.

  If you have installed MySQL to a directory other than `C:\Program Files\MySQL\MySQL Server 5.6`, ensure that the MySQL server is aware of this through the use of a configuration (`my.ini`) file. Put the `my.ini` file in your Windows directory, typically `C:\WINDOWS`. To determine its exact location from the value of the `WINDIR` environment variable, issue the following command from the command prompt:

```
C:\> echo %WINDIR%
```

  You can create or modify an option file with any text editor, such as Notepad. For example, if MySQL is installed in `E:\mysql` and the data directory is `D:\MySQLdata`, you can create the option file and set up a `[mysqld]` section to specify values for the `basedir` and `datadir` options:

```
[mysqld]
# set basedir to your installation path
basedir=E:/mysql
# set datadir to the location of your data directory
datadir=D:/MySQLdata
```

  Microsoft Windows path names are specified in option files using (forward) slashes rather than backslashes. If you do use backslashes, double them:

```
[mysqld]
# set basedir to your installation path
basedir=C:\\Program Files\\MySQL\\MySQL Server 5.6
# set datadir to the location of your data directory
datadir=D:\\MySQLdata
```

The rules for use of backslash in option file values are given in Section 4.2.2.2, “Using Option Files”.

If you change the `datadir` value in your MySQL configuration file, you must move the contents of the existing MySQL data directory before restarting the MySQL server.

See Section 2.3.4.2, “Creating an Option File”.

- If you reinstall or upgrade MySQL without first stopping and removing the existing MySQL service and install MySQL using the MySQL Installer, you might see this error:

```
Error: Cannot create Windows service for MySql. Error: 0
```

This occurs when the Configuration Wizard tries to install the service and finds an existing service with the same name.

One solution to this problem is to choose a service name other than `mysql` when using the configuration wizard. This enables the new service to be installed correctly, but leaves the outdated service in place. Although this is harmless, it is best to remove old services that are no longer in use.

To permanently remove the old `mysql` service, execute the following command as a user with administrative privileges, on the command line:

```
C:\> SC DELETE mysql
[SC] DeleteService SUCCESS
```

If the `SC` utility is not available for your version of Windows, download the `delsrv` utility from http://www.microsoft.com/windows2000/techinfo/reskit/tools/existing/delsrv-o.asp and use the `delsrv mysql` syntax.

2.3.6 Windows Postinstallation Procedures

GUI tools exist that perform most of the tasks described in this section, including:

- MySQL Installer: Used to install and upgrade MySQL products.
- MySQL Workbench: Manages the MySQL server and edits SQL statements.

On Windows, you need not create the data directory and the grant tables. MySQL distributions for Windows include the grant tables with a set of preinitialized accounts in the `mysql` database under the data directory.

Regarding passwords, if you installed MySQL using the MySQL Installer, you may have already assigned passwords to the accounts. (See Section 2.3.3, “MySQL Installer for Windows”.) Otherwise, use the password-assignment procedure given in Section 2.10.4, “Securing the Initial MySQL Accounts”.

Before assigning passwords, you might want to try running some client programs to make sure that you can connect to the server and that it is operating properly. Make sure that the server is running (see Section 2.3.4.4, “Starting the Server for the First Time”). You can also set up a MySQL service that runs automatically when Windows starts (see Section 2.3.4.7, “Starting MySQL as a Windows Service”).

These instructions assume that your current location is the MySQL installation directory and that it has a `bin` subdirectory containing the MySQL programs used here. If that is not true, adjust the command path names accordingly.

If you installed MySQL using MySQL Installer (see Section 2.3.3, “MySQL Installer for Windows”), the default installation directory is `C:\Program Files\MySQL\MySQL Server 5.6`:

```
C:\> cd "C:\Program Files\MySQL\MySQL Server 5.6"
```

A common installation location for installation from a ZIP archive is `C:\mysql`:

```
C:\> cd C:\mysql
```

Alternatively, add the `bin` directory to your `PATH` environment variable setting. That enables your command interpreter to find MySQL programs properly, so that you can run a program by typing only its name, not its path name. See Section 2.3.4.6, “Customizing the PATH for MySQL Tools”.

With the server running, issue the following commands to verify that you can retrieve information from the server. The output should be similar to that shown here.

Use `mysqlshow` to see what databases exist:

```
C:\> bin\mysqlshow
+--------------------+
|     Databases      |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
```

The list of installed databases may vary, but always includes at a minimum `mysql` and `information_schema`.

The preceding command (and commands for other MySQL programs such as `mysql`) may not work if the correct MySQL account does not exist. For example, the program may fail with an error, or you may not be able to view all databases. If you installed MySQL using MySQL Installer, the `root` user was created automatically with the password you supplied. In this case, you should use the `-u root` and `-p` options. (You must use those options if you have already secured the initial MySQL accounts.) With `-p`, the client program prompts you for the `root` password. For example:

```
C:\> bin\mysqlshow -u root -p
Enter password: (enter root password here)
+--------------------+
|     Databases      |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
```

If you specify a database name, `mysqlshow` displays a list of the tables within the database:

```
C:\> bin\mysqlshow mysql
Database: mysql
+---------------------------+
|          Tables           |
+---------------------------+
| columns_priv              |
| db                        |
| event                     |
| func                      |
| general_log               |
| help_category             |
| help_keyword              |
| help_relation             |
| help_topic                |
| innodb_index_stats        |
| innodb_table_stats        |
```

```
| ndb_binlog_index          |
| plugin                    |
| proc                      |
| procs_priv                |
| proxies_priv              |
| servers                   |
| slave_master_info         |
| slave_relay_log_info      |
| slave_worker_info         |
| slow_log                  |
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
| user                      |
+---------------------------+
```

Use the `mysql` program to select information from a table in the `mysql` database:

```
C:\> bin\mysql -e "SELECT User, Host, plugin FROM mysql.user" mysql
+------+-----------+-----------------------+
| User | Host      | plugin                |
+------+-----------+-----------------------+
| root | localhost | mysql_native_password |
+------+-----------+-----------------------+
```

For more information about `mysql` and `mysqlshow`, see Section 4.5.1, “`mysql` — The MySQL Command-Line Client”, and Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”.

2.3.7 Windows Platform Restrictions

The following restrictions apply to use of MySQL on the Windows platform:

- **Process memory**

  On Windows 32-bit platforms, it is not possible by default to use more than 2GB of RAM within a single process, including MySQL. This is because the physical address limit on Windows 32-bit is 4GB and the default setting within Windows is to split the virtual address space between kernel (2GB) and user/applications (2GB).

  Some versions of Windows have a boot time setting to enable larger applications by reducing the kernel application. Alternatively, to use more than 2GB, use a 64-bit version of Windows.

- **File system aliases**

  When using `MyISAM` tables, you cannot use aliases within Windows link to the data files on another volume and then link back to the main MySQL `datadir` location.

  This facility is often used to move the data and index files to a RAID or other fast solution, while retaining the main `.frm` files in the default data directory configured with the `datadir` option.

- **Limited number of ports**

  Windows systems have about 4,000 ports available for client connections, and after a connection on a port closes, it takes two to four minutes before the port can be reused. In situations where clients connect to and disconnect from the server at a high rate, it is possible for all available ports to be used up before closed ports become available again. If this happens, the MySQL server appears to be unresponsive even though it is running. Ports may be used by other applications running on the machine as well, in which case the number of ports available to MySQL is lower.

  For more information about this problem, see http://support.microsoft.com/default.aspx?scid=kb;en-us;196271.

- **`DATA DIRECTORY` and `INDEX DIRECTORY`**

  The `DATA DIRECTORY` clause of the `CREATE TABLE` statement is supported on Windows for `InnoDB` tables only, as described in Section 14.6.1.2, “Creating Tables Externally”. For `MyISAM` and other storage engines, the `DATA DIRECTORY` and `INDEX DIRECTORY` clauses for `CREATE TABLE` are ignored on Windows and any other platforms with a nonfunctional `realpath()` call.

- **`DROP DATABASE`**

  You cannot drop a database that is in use by another session.

- **Case-insensitive names**

  File names are not case-sensitive on Windows, so MySQL database and table names are also not case-sensitive on Windows. The only restriction is that database and table names must be specified using the same case throughout a given statement. See Section 9.2.3, “Identifier Case Sensitivity”.

- **Directory and file names**

  On Windows, MySQL Server supports only directory and file names that are compatible with the current ANSI code pages. For example, the following Japanese directory name does not work in the Western locale (code page 1252):

  ```
  datadir="C:/私たちのプロジェクトのデータ"
  ```

  The same limitation applies to directory and file names referred to in SQL statements, such as the data file path name in `LOAD DATA`.

- **The `\` path name separator character**

  Path name components in Windows are separated by the `\` character, which is also the escape character in MySQL. If you are using `LOAD DATA` or `SELECT ... INTO OUTFILE`, use Unix-style file names with `/` characters:

  ```
  mysql> LOAD DATA INFILE 'C:/tmp/skr.txt' INTO TABLE skr;
  mysql> SELECT * INTO OUTFILE 'C:/tmp/skr.txt' FROM skr;
  ```

  Alternatively, you must double the `\` character:

  ```
  mysql> LOAD DATA INFILE 'C:\\tmp\\skr.txt' INTO TABLE skr;
  mysql> SELECT * INTO OUTFILE 'C:\\tmp\\skr.txt' FROM skr;
  ```

- **Problems with pipes**

  Pipes do not work reliably from the Windows command-line prompt. If the pipe includes the character `^Z` / `CHAR(24)`, Windows thinks that it has encountered end-of-file and aborts the program.

  This is mainly a problem when you try to apply a binary log as follows:

  ```
  C:\> mysqlbinlog binary_log_file | mysql --user=root
  ```

  If you have a problem applying the log and suspect that it is because of a `^Z` / `CHAR(24)` character, you can use the following workaround:

  ```
  C:\> mysqlbinlog binary_log_file --result-file=/tmp/bin.sql
  C:\> mysql --user=root --execute "source /tmp/bin.sql"
  ```

  The latter command also can be used to reliably read any SQL file that may contain binary data.

2.4 Installing MySQL on OS X

For a list of OS X versions that the MySQL server supports, see https://www.mysql.com/support/supportedplatforms/database.html.

MySQL for OS X is available in a number of different forms:

- Native Package Installer, which uses the native OS X installer (DMG) to walk you through the installation of MySQL. For more information, see Section 2.4.2, “Installing MySQL on OS X Using Native Packages”. You can use the package installer with OS X. The user you use to perform the installation must have administrator privileges.
- Compressed TAR archive, which uses a file packaged using the Unix `tar` and `gzip` commands. To use this method, you must open a `Terminal` window. You do not need administrator privileges using this method, as you can install the MySQL server anywhere using this method. For more information on using this method, you can use the generic instructions for using a tarball, Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”.

  In addition to the core installation, the Package Installer also includes Section 2.4.3, “Installing a MySQL Launch Daemon” and Section 2.4.4, “Installing and Using the MySQL Preference Pane”, both of which simplify the management of your installation.

For additional information on using MySQL on OS X, see Section 2.4.1, “General Notes on Installing MySQL on OS X”.

2.4.1 General Notes on Installing MySQL on OS X

You should keep the following issues and notes in mind:

- As of MySQL server 5.6.26, the DMG bundles a launchd daemon instead of the deprecated startup item. Startup items do not function as of OS X 10.10 (Yosemite), so using launchd is preferred. The available MySQL preference pane under OS X **System Preferences** was also updated to use launchd.
- You may need (or want) to create a specific `mysql` user to own the MySQL directory and data. You can do this through the `Directory Utility`, and the `mysql` user should already exist. For use in single user mode, an entry for `_mysql` (note the underscore prefix) should already exist within the system `/etc/passwd` file.
- Because the MySQL package installer installs the MySQL contents into a version and platform specific directory, you can use this to upgrade and migrate your database between versions. You must either copy the `data` directory from the old version to the new version, or alternatively specify an alternative `datadir` value to set location of the data directory. By default, the MySQL directories are installed under `/usr/local/`.
- You might want to add aliases to your shell's resource file to make it easier to access commonly used programs such as `mysql` and `mysqladmin` from the command line. The syntax for `bash` is:

  ```
  alias mysql=/usr/local/mysql/bin/mysql
  alias mysqladmin=/usr/local/mysql/bin/mysqladmin
  ```

  For `tcsh`, use:

  ```
  alias mysql /usr/local/mysql/bin/mysql
  alias mysqladmin /usr/local/mysql/bin/mysqladmin
  ```

  Even better, add `/usr/local/mysql/bin` to your `PATH` environment variable. You can do this by modifying the appropriate startup file for your shell. For more information, see Section 4.2.1, “Invoking MySQL Programs”.
- After you have copied over the MySQL database files from the previous installation and have successfully started the new server, you should consider removing the old installation files to save disk space. Additionally, you should also remove older versions of the Package Receipt directories located in `/Library/Receipts/mysql-VERSION.pkg`.
- Prior to OS X 10.7, MySQL server was bundled with OS X Server.

2.4.2 Installing MySQL on OS X Using Native Packages

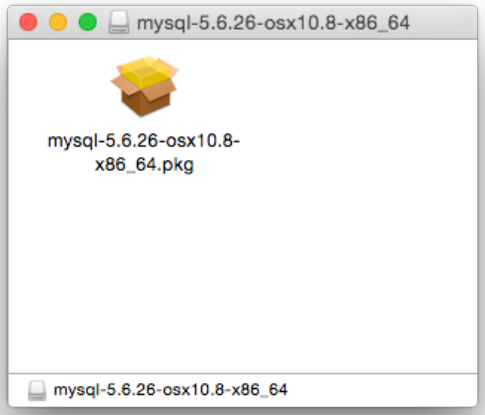
Note

Before proceeding with the installation, be sure to stop all running MySQL server instances by using either the MySQL Manager Application (on OS X Server), the preference pane, or `mysqladmin shutdown` on the command line.

To install MySQL using the package installer:

1. Download the disk image (`.dmg`) file (the community version is available here) that contains the MySQL package installer. Double-click the file to mount the disk image and see its contents.

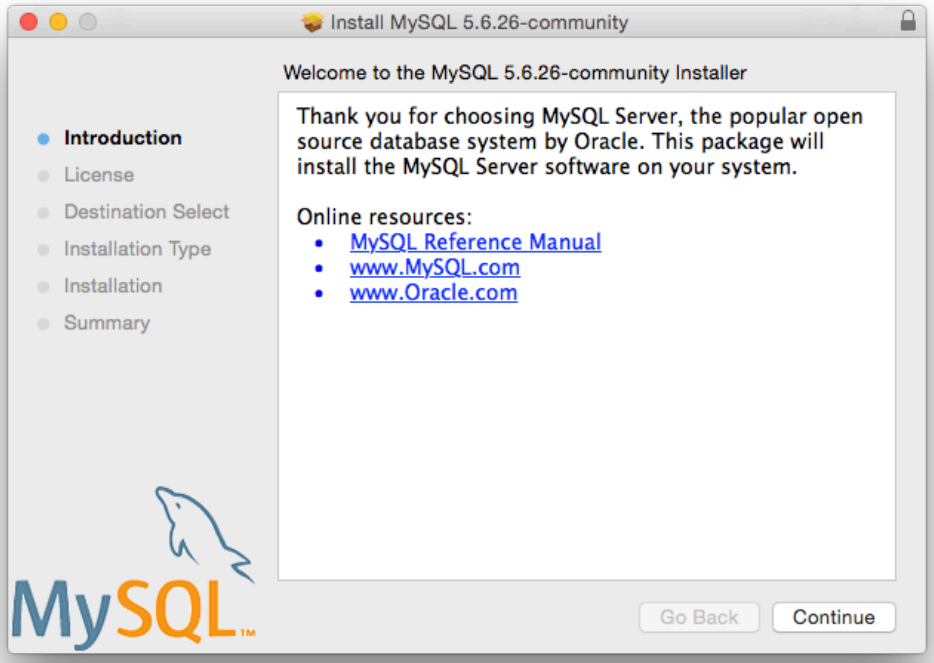
   Figure 2.13 MySQL Package Installer: DMG Contents

2. Double-click the MySQL installer package. It is named according to the MySQL version and the OS X version you have chosen. For example, if you have downloaded the package for MySQL 5.6.51 and OS X 10.8, double-click `mysql-5.6.51-osx-10.8-x86_64.pkg`.

3. You are presented with the opening installer dialog. Click **Continue** to begin installation.

Figure 2.14 MySQL Package Installer: Introduction



4. If you have downloaded the community version of MySQL, you are shown a copy of the relevant GNU General Public License. Click **Continue** and then **Agree** to continue.

5. From the **Installation Type** page you can either click **Install** to execute the installation wizard using all defaults, click **Customize** to alter which components to install (MySQL server, Preference Pane, Launchd Support -- all enabled by default).

Note

Although the **Change Install Location** option is visible, the installation location cannot be changed.

Figure 2.15 MySQL Package Installer: Installation Type

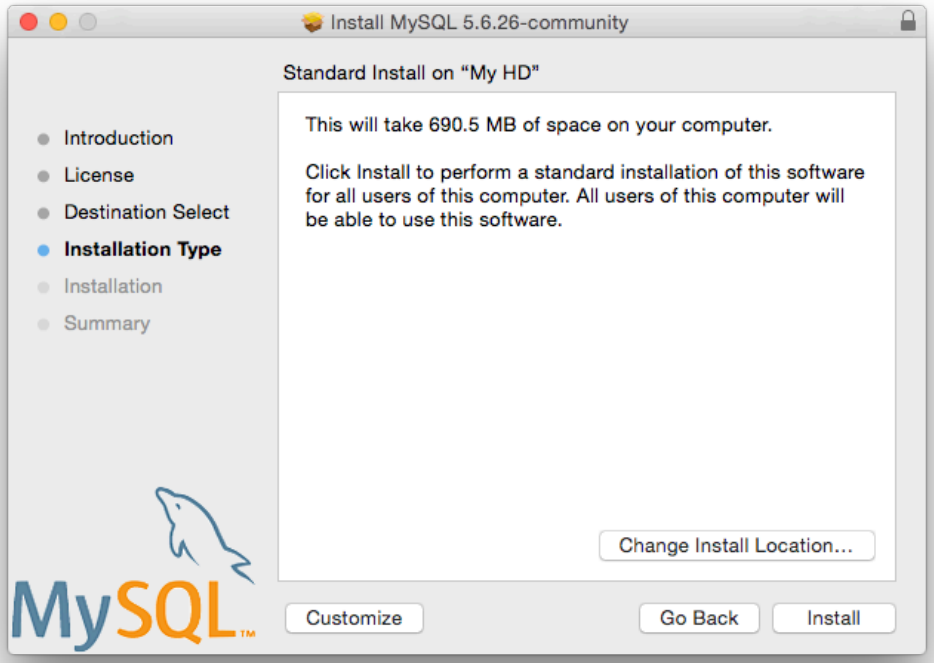
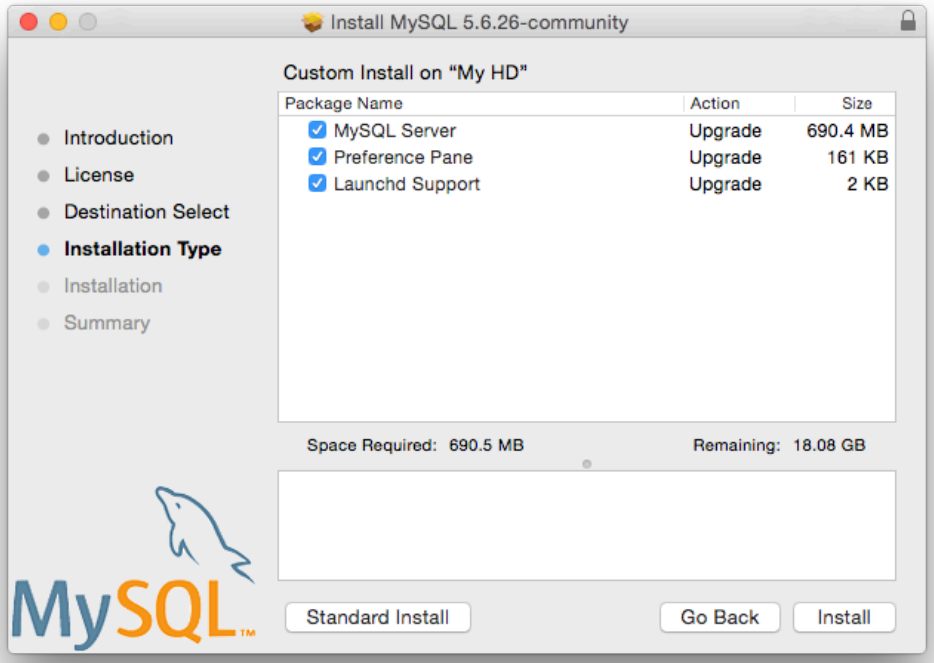


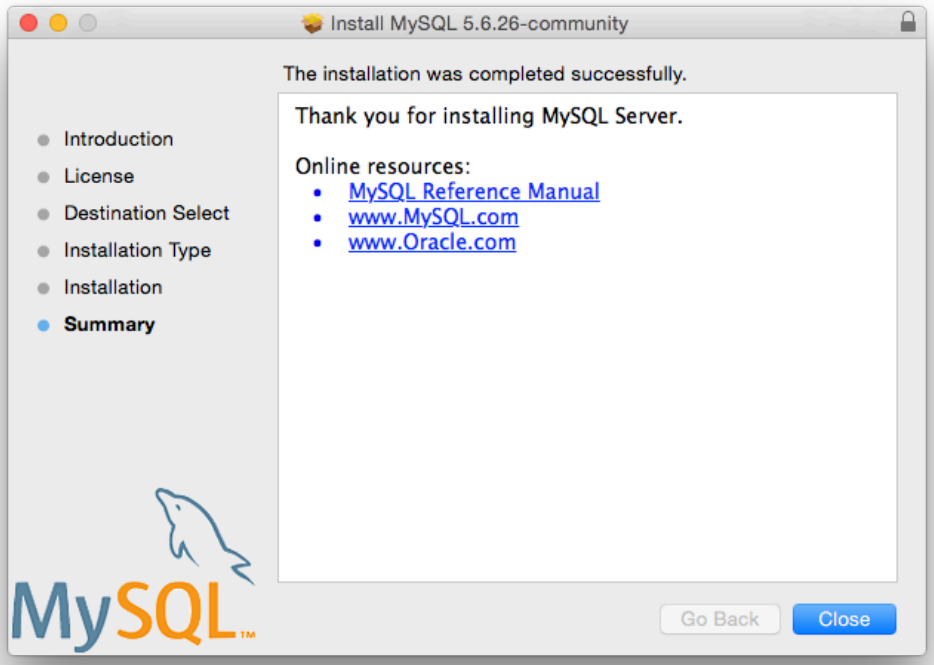
Figure 2.16 MySQL Package Installer: Customize



6. Click **Install** to begin the installation process.

7. Once the installation has been completed successfully, you are shown an **Install Succeeded** message with a short summary. Now, **Close** the wizard and begin using the MySQL server.

Figure 2.17 MySQL Package Installer: Summary



MySQL server is now installed, but it is not loaded (or started) by default. Use either launchctl from the command line, or start MySQL by clicking "Start" using the MySQL preference pane. For additional information, see Section 2.4.3, “Installing a MySQL Launch Daemon”, and Section 2.4.4, “Installing and Using the MySQL Preference Pane”. Use the MySQL Preference Pane or launchd to configure MySQL to automatically start at bootup.

When installing using the package installer, the files are installed into a directory within `/usr/local` matching the name of the installation version and platform. For example, the installer file `mysql-5.6.51-osx10.8-x86_64.dmg` installs MySQL into `/usr/local/mysql-5.6.51-osx10.8-x86_64/` . The following table shows the layout of the installation directory.

Table 2.7 MySQL Installation Layout on OS X

Directory	Contents of Directory
`bin`, `scripts`	`mysqld` server, client and utility programs
`data`	Log files, databases
`docs`	Helper documents, like the Release Notes and build information
`include`	Include (header) files
`lib`	Libraries
`man`	Unix manual pages
`mysql-test`	MySQL test suite
`share`	Miscellaneous support files, including error messages, sample configuration files, SQL for database installation

Directory	Contents of Directory
`sql-bench`	Benchmarks
`support-files`	Scripts and sample configuration files
`/tmp/mysql.sock`	Location of the MySQL Unix socket

During the package installer process, a symbolic link from `/usr/local/mysql` to the version/platform specific directory created during installation is created automatically.

2.4.3 Installing a MySQL Launch Daemon

OS X uses launch daemons to automatically start, stop, and manage processes and applications such as MySQL.

Note

Before MySQL 5.6.26, the OS X builds installed startup items instead of launchd daemons. However, startup items do not function as of OS X 10.10 (Yosemite). The OS X builds now install launchd daemons.

By default, the installation package (DMG) on OS X installs a launchd file named `/Library/LaunchDaemons/com.oracle.oss.mysql.mysqld.plist` that contains a plist definition similar to:

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE plist PUBLIC "-//Apple Computer//DTD PLIST 1.0//EN" "http://www.apple.com/DTDs/PropertyList-1
<plist version="1.0">
<dict>
    <key>Label</key>             <string>com.oracle.oss.mysql.mysqld</string>
    <key>ProcessType</key>       <string>Interactive</string>
    <key>Disabled</key>          <false/>
    <key>RunAtLoad</key>         <true/>
    <key>KeepAlive</key>         <true/>
    <key>SessionCreate</key>     <true/>
    <key>LaunchOnlyOnce</key>    <false/>
    <key>UserName</key>          <string>_mysql</string>
    <key>GroupName</key>         <string>_mysql</string>
    <key>ExitTimeOut</key>       <integer>600</integer>
    <key>Program</key>           <string>/usr/local/mysql/bin/mysqld</string>
    <key>ProgramArguments</key>
        <array>
            <string>/usr/local/mysql/bin/mysqld</string>
            <string>--user=_mysql</string>
            <string>--basedir=/usr/local/mysql</string>
            <string>--datadir=/usr/local/mysql/data</string>
            <string>--plugin-dir=/usr/local/mysql/lib/plugin</string>
            <string>--log-error=/usr/local/mysql/data/mysqld.local.err</string>
            <string>--pid-file=/usr/local/mysql/data/mysqld.local.pid</string>
        </array>
    <key>WorkingDirectory</key>  <string>/usr/local/mysql</string>
</dict>
</plist>
```

Note

Some users report that adding a plist DOCTYPE declaration causes the launchd operation to fail, despite it passing the lint check. We suspect it's a copy-n-paste error. The md5 checksum of a file containing the above snippet is 60d7963a0bb2994b69b8b9c123db09df.

To enable the launchd service, you can either:

- Click **Start MySQL Server** from the MySQL preference pane.

Figure 2.18 MySQL Preference Pane: Location

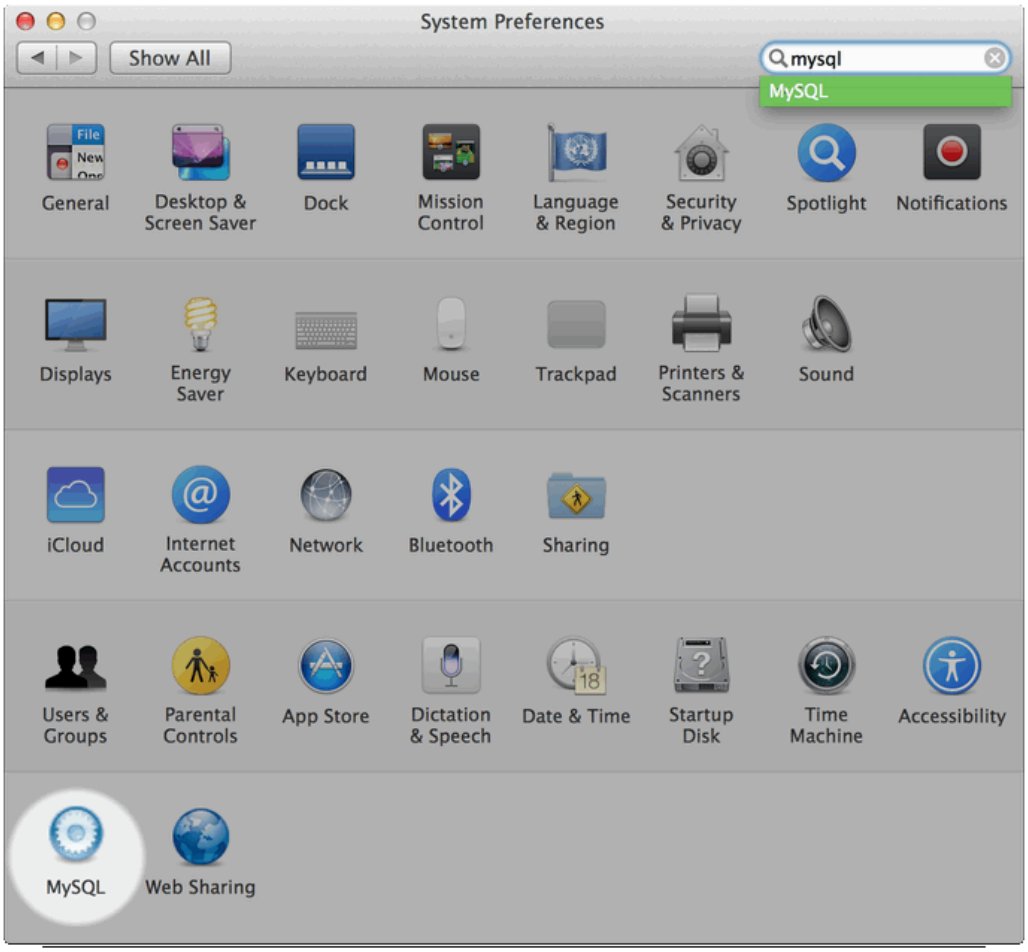


Figure 2.19 MySQL Preference Pane: Usage

- Or, manually load the launchd file.

```
shell> cd /Library/LaunchDaemons
shell> sudo launchctl load -F com.oracle.oss.mysql.mysqld.plist
```

- To configure MySQL to automatically start at bootup, you can:

```
shell> sudo launchctl load -w com.oracle.oss.mysql.mysqld.plist
```

Note

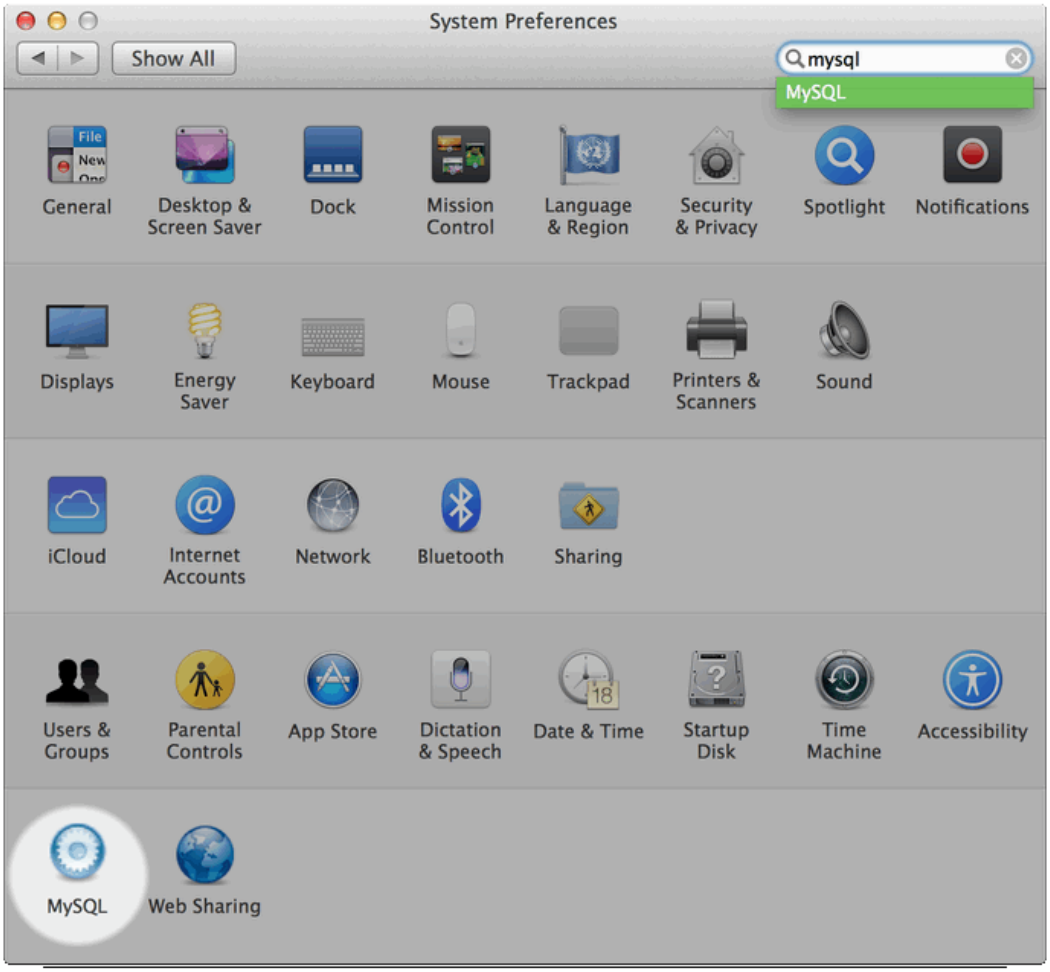
When upgrading MySQL server, the launchd installation process removes the old startup items that were installed with MySQL server 5.6.25 and earlier.

2.4.4 Installing and Using the MySQL Preference Pane

The MySQL Installation Package includes a MySQL preference pane that enables you to start, stop, and control automated startup during boot of your MySQL installation.

This preference pane is installed by default, and is listed under your system's *System Preferences* window.

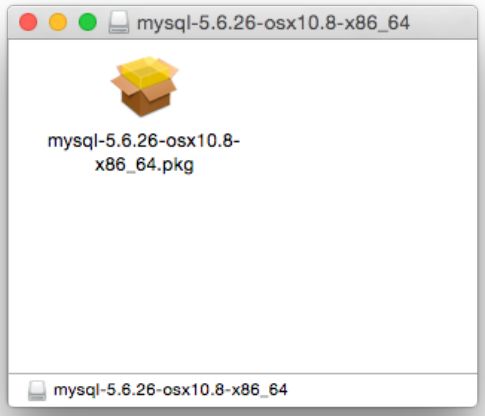
Figure 2.20 MySQL Preference Pane: Location



To install the MySQL Preference Pane:

1. Download the disk image (`.dmg`) file (the community version is available here) that contains the MySQL package installer. Double-click the file to mount the disk image and see its contents.

Figure 2.21 MySQL Package Installer: DMG Contents



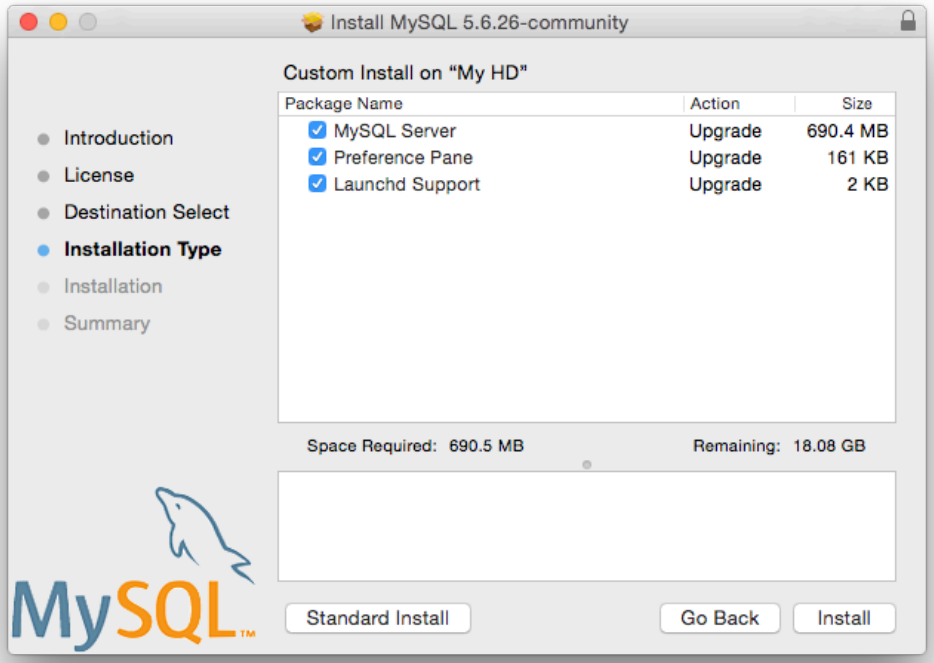
Note

Before MySQL 5.6.26, OS X packages included the deprecated startup items instead of launchd daemons, and the preference pane managed that instead of launchd.

2. Go through the process of installing the MySQL server, as described in the documentation at Section 2.4.2, “Installing MySQL on OS X Using Native Packages”.

3. Click **Customize** at the **Installation Type** step. The "Preference Pane" option is listed there and enabled by default; make sure it is not deselected.

Figure 2.22 MySQL Installer on OS X: Customize



4. Complete the MySQL server installation process.

Note

The MySQL preference pane only starts and stops MySQL installation installed from the MySQL package installation that have been installed in the default location.

Once the MySQL preference pane has been installed, you can control your MySQL server instance using the preference pane. To use the preference pane, open the **System Preferences...** from the Apple menu. Select the MySQL preference pane by clicking the MySQL icon within the preference panes list.

Figure 2.23 MySQL Preference Pane: Location

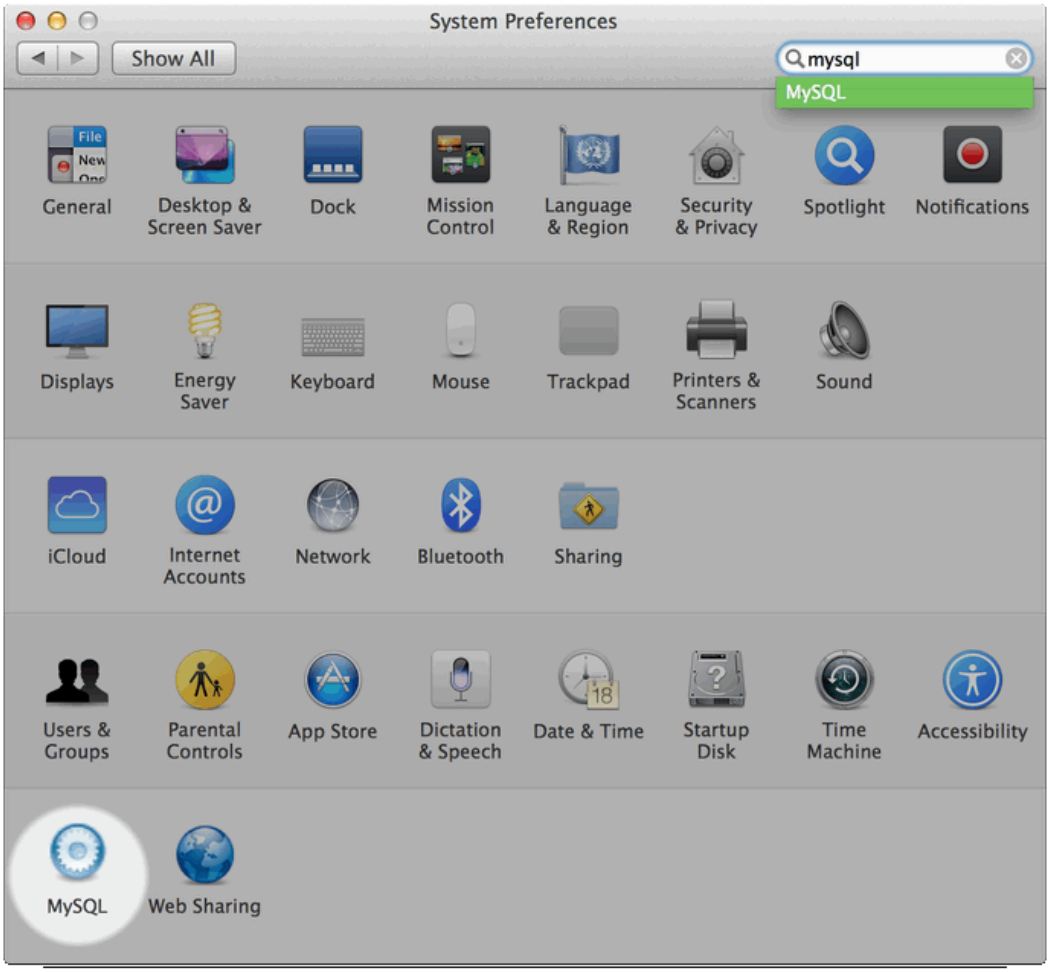
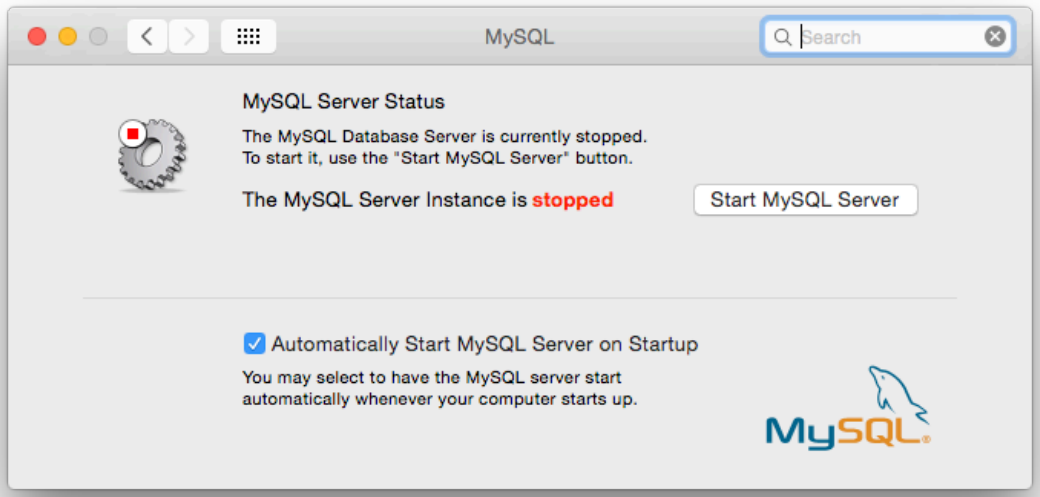


Figure 2.24 MySQL Preference Pane: Usage



The MySQL Preference Pane shows the current status of the MySQL server, showing **stopped** (in red) if the server is not running and **running** (in green) if the server has already been started. The preference pane also shows the current setting for whether the MySQL server has been set to start automatically.

- **To start the MySQL server using the preference pane:**

  Click **Start MySQL Server**. You may be prompted for the username and password of a user with administrator privileges to start the MySQL server.

- **To stop the MySQL server using the preference pane:**

  Click **Stop MySQL Server**. You may be prompted for the username and password of a user with administrator privileges to stop the MySQL server.

- **To automatically start the MySQL server when the system boots:**

  Check the check box next to **Automatically Start MySQL Server on Startup**.

- **To disable automatic MySQL server startup when the system boots:**

  Uncheck the check box next to **Automatically Start MySQL Server on Startup**.

You can close the `System Preferences...` window once you have completed your settings.

2.5 Installing MySQL on Linux

Linux supports a number of different solutions for installing MySQL. We recommend that you use one of the distributions from Oracle, for which several methods for installation are available:

Table 2.8 Linux Installation Methods and Information

Type	Setup Method	Additional Information
Apt	Enable the MySQL Apt repository	Documentation
Yum	Enable the MySQL Yum repository	Documentation

Type	Setup Method	Additional Information
Zypper	Enable the MySQL SLES repository	Documentation
RPM	Download a specific package	Documentation
DEB	Download a specific package	Documentation
Generic	Download a generic package	Documentation
Source	Compile from source	Documentation
Docker	Use Docker Hub	Documentation
Oracle Unbreakable Linux Network	Use ULN channels	Documentation

As an alternative, you can use the package manager on your system to automatically download and install MySQL with packages from the native software repositories of your Linux distribution. These native packages are often several versions behind the currently available release. You are also normally unable to install development milestone releases (DMRs), as these are not usually made available in the native repositories. For more information on using the native package installers, see Section 2.5.7, “Installing MySQL on Linux from the Native Software Repositories”.

Note

For many Linux installations, you may want to set up MySQL to be started automatically when your machine starts. Many of the native package installations perform this operation for you, but for source, binary and RPM solutions you may need to set this up separately. The required script, `mysql.server`, can be found in the `support-files` directory under the MySQL installation directory or in a MySQL source tree. You can install it as `/etc/init.d/mysql` for automatic MySQL startup and shutdown. See Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”.

2.5.1 Installing MySQL on Linux Using the MySQL Yum Repository

The MySQL Yum repository for Oracle Linux, Red Hat Enterprise Linux, and CentOS provides RPM packages for installing the MySQL server, client, MySQL Workbench, MySQL Utilities, MySQL Router, MySQL Shell, Connector/ODBC, Connector/Python and so on (not all packages are available for all the distributions; see Installing Additional MySQL Products and Components with Yum for details).

Before You Start

As a popular, open-source software, MySQL, in its original or re-packaged form, is widely installed on many systems from various sources, including different software download sites, software repositories, and so on. The following instructions assume that MySQL is not already installed on your system using a third-party-distributed RPM package; if that is not the case, follow the instructions given in Section 2.11.5, “Upgrading MySQL with the MySQL Yum Repository” or Section 2.5.2, “Replacing a Third-Party Distribution of MySQL Using the MySQL Yum Repository”.

Steps for a Fresh Installation of MySQL

Follow the steps below to install the latest GA release of MySQL (from the MySQL 5.7 series currently) with the MySQL Yum repository:

Adding the MySQL Yum Repository

First, add the MySQL Yum repository to your system's repository list. This is a one-time operation, which can be performed by installing an RPM provided by MySQL. Follow these steps:

a. Go to the Download MySQL Yum Repository page (https://dev.mysql.com/downloads/repo/yum/) in the MySQL Developer Zone.

b. Select and download the release package for your platform.

c. Install the downloaded release package with the following command, replacing *platform-and-version-specific-package-name* with the name of the downloaded RPM package:

```
shell> sudo yum localinstall platform-and-version-specific-package-name.rpm
```

For an EL6-based system, the command is in the form of:

```
shell> sudo yum localinstall mysql57-community-release-el6-{version-number}.noarch.rpm
```

For an EL7-based system:

```
shell> sudo yum localinstall mysql57-community-release-el7-{version-number}.noarch.rpm
```

The installation command adds the MySQL Yum repository to your system's repository list and downloads the GnuPG key to check the integrity of the software packages. See Section 2.1.4.2, "Signature Checking Using GnuPG" for details on GnuPG key checking.

You can check that the MySQL Yum repository has been successfully added by the following command:

```
shell> yum repolist enabled | grep "mysql.*-community.*"
```

Note

Once the MySQL Yum repository is enabled on your system, any system-wide update by the `yum update` command upgrades MySQL packages on your system and also replace any native third-party packages, if Yum finds replacements for them in the MySQL Yum repository; see Section 2.11.5, "Upgrading MySQL with the MySQL Yum Repository" and, for a discussion on some possible effects of that on your system, see Upgrading the Shared Client Libraries.

2. Selecting a Release Series

When using the MySQL Yum repository, the latest GA series (currently MySQL 5.7) is selected for installation by default. If this is what you want, you can skip to the next step, Installing MySQL.

Within the MySQL Yum repository, different release series of the MySQL Community Server are hosted in different subrepositories. The subrepository for the latest GA series (currently MySQL 5.7) is enabled by default, and the subrepositories for all other series (for example, the MySQL 5.6 series) are disabled by default. Use this command to see all the subrepositories in the MySQL Yum repository, and see which of them are enabled or disabled:

```
shell> yum repolist all | grep mysql
```

To install the latest release from the latest GA series, no configuration is needed. To install the latest release from a specific series other than the latest GA series, disable the subrepository for the latest GA series and enable the subrepository for the specific series before running the installation command. If your platform supports `yum-config-manager`, you can do that by issuing these commands, which disable the subrepository for the 5.7 series and enable the one for the 5.6 series:

```
shell> sudo yum-config-manager --disable mysql57-community
shell> sudo yum-config-manager --enable mysql56-community
```

Besides using `yum-config-manager` command, you can also select a release series by editing manually the `/etc/yum.repos.d/mysql-community.repo` file. This is a typical entry for a release series' subrepository in the file:

```
[mysql57-community]
```

```
name=MySQL 5.7 Community Server
baseurl=http://repo.mysql.com/yum/mysql-5.7-community/el/6/$basearch/
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-mysql
```

Find the entry for the subrepository you want to configure, and edit the `enabled` option. Specify `enabled=0` to disable a subrepository, or `enabled=1` to enable a subrepository. For example, to install MySQL 5.6, make sure you have `enabled=0` for the above subrepository entry for MySQL 5.7, and have `enabled=1` for the entry for the 5.6 series:

```
# Enable to use MySQL 5.6
[mysql56-community]
name=MySQL 5.6 Community Server
baseurl=http://repo.mysql.com/yum/mysql-5.6-community/el/6/$basearch/
enabled=1
gpgcheck=1
gpgkey=file:///etc/pki/rpm-gpg/RPM-GPG-KEY-mysql
```

You should only enable subrepository for one release series at any time. When subrepositories for more than one release series are enabled, the latest series is used by Yum.

Verify that the correct subrepositories have been enabled and disabled by running the following command and checking its output:

```
shell> yum repolist enabled | grep mysql
```

Disabling the Default MySQL Module

(EL8 systems only) EL8-based systems such as RHEL8 and Oracle Linux 8 include a MySQL module that is enabled by default. Unless this module is disabled, it masks packages provided by MySQL repositories. To disable the included module and make the MySQL repository packages visible, use the following command (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
shell> sudo yum module disable mysql
```

Installing MySQL

Install MySQL by the following command:

```
shell> sudo yum install mysql-community-server
```

This installs the package for MySQL server (`mysql-community-server`) and also packages for the components required to run the server, including packages for the client (`mysql-community-client`), the common error messages and character sets for client and server (`mysql-community-common`), and the shared client libraries (`mysql-community-libs`).

Starting the MySQL Server

Start the MySQL server with the following command:

```
shell> sudo service mysqld start
```

This is a sample output of the above command:

```
Starting mysqld:[ OK ]
```

You can check the status of the MySQL server with the following command:

```
shell> sudo service mysqld status
```

This is a sample output of the above command:

```
mysqld (pid 3066) is running.
```

Securing the MySQL Installation

6.

The program `mysql_secure_installation` allows you to perform important operations like setting the root password, removing anonymous users, and so on. Always run it to secure your MySQL installation:

```
shell> mysql_secure_installation
```

It is important to remember the root password you set. See Section 4.4.5, "`mysql_secure_installation` — Improve MySQL Installation Security" for details.

For more information on the postinstallation procedures, see Section 2.10, "Postinstallation Setup and Testing".

Note

Compatibility Information for EL7-based platforms: The following RPM packages from the native software repositories of the platforms are incompatible with the package from the MySQL Yum repository that installs the MySQL server. Once you have installed MySQL using the MySQL Yum repository, you cannot install these packages (and vice versa).

- akonadi-mysql

Installing Additional MySQL Products and Components with Yum

You can use Yum to install and manage individual components of MySQL. Some of these components are hosted in sub-repositories of the MySQL Yum repository: for example, the MySQL Connectors are to be found in the MySQL Connectors Community sub-repository, and the MySQL Workbench in MySQL Tools Community. You can use the following command to list the packages for all the MySQL components available for your platform from the MySQL Yum repository:

```
shell> sudo yum --disablerepo=\* --enablerepo='mysql*-community*' list available
```

Install any packages of your choice with the following command, replacing *`package-name`* with name of the package:

```
shell> sudo yum install package-name
```

For example, to install MySQL Workbench:

```
shell> sudo yum install mysql-workbench-community
```

To install the shared client libraries:

```
shell> sudo yum install mysql-community-libs
```

Updating MySQL with Yum

Besides installation, you can also perform updates for MySQL products and components using the MySQL Yum repository. See Section 2.11.5, "Upgrading MySQL with the MySQL Yum Repository" for details.

2.5.2 Replacing a Third-Party Distribution of MySQL Using the MySQL Yum Repository

For supported Yum-based platforms (see Section 2.5.1, "Installing MySQL on Linux Using the MySQL Yum Repository", for a list), you can replace a third-party distribution of MySQL with the latest GA release (from the MySQL 5.7 series currently) from the MySQL Yum repository. According to how your third-party distribution of MySQL was installed, there are different steps to follow:

Replacing a Native Third-Party Distribution of MySQL

If you have installed a third-party distribution of MySQL from a native software repository (that is, a software repository provided by your own Linux distribution), follow these steps:

Backing Up Your Database

1. To avoid loss of data, always back up your database before trying to replace your MySQL installation using the MySQL Yum repository. See Chapter 7, *Backup and Recovery*, on how to back up your database.

Adding the MySQL Yum Repository

2. Add the MySQL Yum repository to your system's repository list by following the instructions given in Adding the MySQL Yum Repository.

Replacing the Native Third-Party Distribution by a Yum Update

3. By design, the MySQL Yum repository replaces your native third-party MySQL with the latest GA release (from the MySQL 5.7 series currently) from the MySQL Yum repository when you perform a `yum update` command on the system, or a `yum update mysql-server`.

After updating MySQL using the Yum repository, applications compiled with older versions of the shared client libraries should continue to work. However, *if you want to recompile applications and dynamically link them with the updated libraries*, see Upgrading the Shared Client Libraries, for some special considerations.

Replacing a Nonnative Third-Party Distribution of MySQL

If you have installed a third-party distribution of MySQL from a nonnative software repository (that is, a software repository not provided by your own Linux distribution), follow these steps:

Backing Up Your Database

1. To avoid loss of data, always back up your database before trying to replace your MySQL installation using the MySQL Yum repository. See Chapter 7, *Backup and Recovery*, on how to back up your database.

Stopping Yum from Receiving MySQL Packages from Third-Party, Nonnative Repositories

2. Before you can use the MySQL Yum repository for installing MySQL, you must stop your system from receiving MySQL packages from any third-party, nonnative Yum repositories.

For example, if you have installed MariaDB using their own software repository, get a list of the installed MariaDB packages using the following command:

```
shell> yum list installed mariadb\*
```

This is a sample output for the command:

```
MariaDB-common.i686                    10.0.4-1                        @mariadb
MariaDB-compat.i686                    10.0.4-1                        @mariadb
MariaDB-server.i686                    10.0.4-1                        @mariadb
```

From the command output, we can identify the installed packages (`MariaDB-common`, `MariaDB-compat`, and `MariaDB-server`) and the source of them (a nonnative software repository named `mariadb`).

As another example, if you have installed Percona using their own software repository, get a list of the installed Percona packages using the following command:

```
shell> yum list installed Percona\*
```

This is a sample output for the command:

```
Percona-Server-client-55.i686     5.5.39-rel36.0.el6          @percona-release-i386
Percona-Server-server-55.i686     5.5.39-rel36.0.el6          @percona-release-i386
Percona-Server-shared-55.i686     5.5.39-rel36.0.el6          @percona-release-i386
percona-release.noarch            0.1-3                       @/percona-release-0.1-3.noarch
```

From the command output, we can identify the installed packages (`Percona-Server-client`, `Percona-Server-server`, `Percona-Server-shared`, and `percona-release.noarch`) and the source of them (a nonnative software repository named `percona-release`).

If you are not sure which third-party MySQL fork you have installed, this command should reveal it and list the RPM packages installed for it, as well as the third-party repository that supplies the packages:

```
shell> yum --disablerepo=\* provides mysql\*
```

The next step is to stop Yum from receiving packages from the nonnative repository. If the `yum-config-manager` utility is supported on your platform, you can, for example, use this command for stopping delivery from MariaDB:

```
shell> sudo yum-config-manager --disable mariadb
```

And use this command for stopping delivery from Percona:

```
shell> sudo yum-config-manager --disable percona-release
```

You can perform the same task by removing the entry for the software repository existing in one of the repository files under the `/etc/yum.repos.d/` directory. This is how the entry typically looks like for MariaDB:

```
[mariadb] name = MariaDB
 baseurl = [base URL for repository]
 gpgkey = [URL for GPG key]
 gpgcheck =1
```

The entry is usually found in the file `/etc/yum.repos.d/MariaDB.repo` for MariaDB—delete the file, or remove entry from it (or from the file in which you find the entry).

Note

This step is not necessary for an installation that was configured with a Yum repository release package (like Percona) if you are going to remove the release package (`percona-release.noarch` for Percona), as shown in the uninstall command for Percona in Step 3 below.

Uninstalling the Nonnative Third-Party MySQL Distribution of MySQL

The nonnative third-party MySQL distribution must first be uninstalled before you can use the MySQL Yum repository to install MySQL. For the MariaDB packages found in Step 2 above, uninstall them with the following command:

```
shell> sudo yum remove MariaDB-common MariaDB-compat MariaDB-server
```

For the Percona packages we found in Step 2 above:

```
shell> sudo yum remove Percona-Server-client-55 Percona-Server-server-55 \
  Percona-Server-shared-55.i686 percona-release
```

4. Installing MySQL with the MySQL Yum Repository

Then, install MySQL with the MySQL Yum repository by following the instructions given in Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository”: .

Important

- If you have chosen to replace your third-party MySQL distribution with a newer version of MySQL from the MySQL Yum repository, remember to run `mysql_upgrade` after the server starts, to check and possibly resolve any incompatibilities between the old data and the upgraded software. `mysql_upgrade` also performs other functions; see Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables” for details.
- *For EL7-based platforms:* See Compatibility Information for EL7-based platforms [134].

2.5.3 Installing MySQL on Linux Using the MySQL APT Repository

The MySQL APT repository provides `deb` packages for installing and managing the MySQL server, client, and other components on Debian and Ubuntu platforms.

Instructions for using the MySQL APT Repository are available in A Quick Guide to Using the MySQL APT Repository.

2.5.4 Installing MySQL on Linux Using the MySQL SLES Repository

The MySQL SLES repository provides RPM packages for installing and managing the MySQL server, client, and other components on SUSE Enterprise Linux Server.

Instructions for using the MySQL SLES repository are available in A Quick Guide to Using the MySQL SLES Repository.

Note

The MySQL SLES repository is now in development release. We encourage you to try it and provide us with feedback. Please report any bugs or inconsistencies you observe to our Bugs Database.

2.5.5 Installing MySQL on Linux Using RPM Packages from Oracle

The recommended way to install MySQL on RPM-based Linux distributions is by using the RPM packages provided by Oracle. There are two sources for obtaining them, for the Community Edition of MySQL:

- From the MySQL software repositories:
  - The MySQL Yum repository (see Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository” for details).
  - The MySQL SLES repository (see Section 2.5.4, “Installing MySQL on Linux Using the MySQL SLES Repository” for details).
- From the MySQL Downloads page in the MySQL Developer Zone.

Note

RPM distributions of MySQL are also provided by other vendors. Be aware that they may differ from those built by Oracle in features, capabilities, and conventions (including communication setup), and that the installation

> instructions in this manual do not necessarily apply to them. The vendor's instructions should be consulted instead.
>
> If you have such a third-party distribution of MySQL running on your system and now want to migrate to Oracle's distribution using the RPM packages downloaded from the MySQL Developer Zone, see Compatibility with RPM Packages from Other Vendors below. The preferred method of migration, however, is to use the MySQL Yum repository or MySQL SLES repository.

There are two kinds of RPM packages for installing MySQL 5.6 :

- The older kind: Their package names started with `MYSQL-` . They are available from the MySQL Downloads page in the MySQL Developer Zone. The instructions given in this section are for using these packages.
- The newer kind: Their package names started with `mysql-community-` or `mysql-commercial-`. They are available from the MySQL Yum repository and MySQL SLES repository. If, instead of configuring your system to install these RPM directly from the MySQL repositories (which is recommended), you are downloading the packages from the repositories and then installing them manually in separate steps, use the installation commands given for the MySQL 5.7 RPMs in Installing MySQL on Linux Using RPM Packages from Oracle, but consult this section for information like installation layout, server initialization, root password, and so on.

RPM packages for MySQL are listed in the following tables:

Table 2.9 RPM Packages for MySQL Community Edition

Package Name	Summary
`MySQL-server`	Database server and related tools
`MySQL-client`	MySQL client applications and tools
`MySQL-devel`	Development header files and libraries for MySQL database client applications
`MySQL-shared`	Shared libraries for MySQL database client applications
`MySQL-shared-compat`	Shared compatibility libraries for previous MySQL installations
`MySQL-embedded`	MySQL embedded library
`MySQL-test`	Test suite for the MySQL server

Dependency relationships exist among some of the packages. If you plan to install many of the packages, you may wish to download the RPM bundle `tar` file instead, which contains all the RPM packages listed above, so that you need not download them separately.

The full names for the RPMs have the following syntax:

```
packagename-version-distribution-arch.rpm
```

The *`distribution`* and *`arch`* values indicate the Linux distribution and the processor type for which the package was built. See the table below for lists of the distribution identifiers:

Table 2.10 MySQL Linux RPM Package Distribution Identifiers

distribution Value	Intended Use
el6, el7	Red Hat Enterprise Linux/Oracle Linux/CentOS 5, 6, or 7
sles11, sles12	SUSE Linux Enterprise Server 11 or 12
linux_glibc2.5	Distribution independent; run on any RPM-based Linux distribution

To see all files in an RPM package (for example, MySQL-server), use the following command:

```
shell> rpm -qpl MySQL-server-version-distribution-arch.rpm
```

In most cases, you need to install the MySQL-server and MySQL-client to get a functional, standard MySQL installation. To perform such a standard, minimal installation, go to the folder that contains all those packages (and, preferably, no other RPM packages with similar names), and issue the following command (replace yum with zypper for SLES systems):

```
shell> yum install MySQL-{server,client}-*
```

While it is much preferable to use a high-level package management tool like yum to install the packages, users who prefer direct rpm commands can replace the yum install command with the rpm -Uvh command; however, using rpm -Uvh instead makes the installation process more prone to failure, due to potential dependency issues the installation process might run into.

To install only the client programs, you can skip installing the MySQL-server package; issue the following command (replace yum with zypper for SLES systems):

```
shell> yum install MySQL-client-*
```

A standard installation of MySQL using the RPM packages result in files and resources created under the system directories, shown in the following table.

Table 2.11 MySQL Installation Layout for Linux RPM Packages from the MySQL Developer Zone

Files or Resources	Location
Client programs and scripts	/usr/bin
mysqld server	/usr/sbin
Data directory	/var/lib/mysql
Error log file	For RHEL, Oracle Linux, or CentOS: /var/lib/mysql/*host_name*.err For SLES: /var/log/mysql/mysqld.log
System V init script	/etc/init.d/mysql
Systemd service	mysql
Pid file	/var/lib/mysql/*host_name*.pid
Unix manual pages	/usr/share/man
Include (header) files	/usr/include/mysql
Libraries	/usr/lib/mysql
Socket	/var/lib/mysql/mysql.sock
Miscellaneous support files (for example, error messages, and character set files)	/usr/share/mysql

The installation also creates a user named mysql and a group named mysql on the system.

MySQL is not automatically started at the end of the installation process. Use the following command to start MySQL:

```
shell> service mysql start
```

At the initial start up of the server, the server is initialized if the data directory of the server is empty. `mysql_install_db` is invoked with the `--random-passwords` option, which assigns a random password to the MySQL `root` accounts and sets the "password expired" flag for those accounts. It is necessary after installation to start the server, connect as `root` using the initial random password, and assign a new `root` password. Until this is done, `root` cannot do anything else. This must be done for each `root` account you intend to use. To change the password, you can use the `SET PASSWORD` statement (for example, with the `mysql` client). You can also use `mysqladmin` or `mysql_secure_installation`. For additional details (including where to find the assigned random `root` password), see Section 4.4.3, "`mysql_install_db` — Initialize MySQL Data Directory". (Install operations using RPMs for Unbreakable Linux Network are unaffected because they do not run `mysql_install_db`.)

During an upgrade installation using RPM packages, if the MySQL server is running when the upgrade occurs then the MySQL server is stopped, the upgrade occurs, and the MySQL server is restarted. One exception: if the edition also changes during an upgrade (such as community to commercial, or vice-versa), then MySQL server is not restarted.

If something goes wrong during installation, you might find debug information in the error log file `/var/lib/mysql/host_name.err`.

Compatibility with RPM Packages from Other Vendors. If you have installed packages for MySQL from your Linux distribution's local software repository, it is much preferable to install the new, directly-downloaded packages from Oracle using the package management system of your platform (`yum` or `zypper`), as described above. The command replaces old packages with new ones to ensure compatibility of old applications with the new installation; for example, the old `MySQL-shared` package is replaced with the `MySQL-shared-compat` package, which provides a replacement-compatible client library for applications that were using your older MySQL installation. If there was an older version of `MySQL-shared-compat` on the system, it also gets replaced.

If you have installed third-party packages for MySQL that are NOT from your Linux distribution's local software repository (for example, packages directly downloaded from a vendor other than Oracle), you should uninstall all those packages before installing the new, directly-downloaded packages from Oracle. This is because conflicts may arise between those vendor's RPM packages and Oracle's: for example, a vendor's convention about which files belong with the server and which belong with the client library may differ from that used for Oracle packages. Attempts to install an Oracle RPM may then result in messages saying that files in the RPM to be installed conflict with files from an installed package.

Debug Package. A special variant of MySQL Server compiled with the debug package has been included in the server RPM packages. It performs debugging and memory allocation checks and produces a trace file when the server is running. To use that debug version, start MySQL with `/usr/sbin/mysqld-debug`, instead of starting it as a service or with `/usr/sbin/mysqld`. See Section 5.8.3, "The DBUG Package" for the debug options you can use.

Note

The default plugin directory for debug builds changed from `/usr/lib64/mysql/plugin` to `/usr/lib64/mysql/plugin/debug` in 5.6.39. Previously, it was necessary to change `plugin_dir` to `/usr/lib64/mysql/plugin/debug` for debug builds.

Rebuilding RPMs from source SRPMs. Source code SRPM packages for MySQL are available for download. They can be used as-is to rebuild the MySQL RPMs with the standard `rpmbuild` tool chain.

Important

RPMs for NDB Cluster. Standard MySQL server RPMs built by MySQL do not provide support for the `NDBCLUSTER` storage engine. For more information

about installing NDB Cluster from RPMs, see Section 18.2, “NDB Cluster Installation”.

2.5.6 Installing MySQL on Linux Using Debian Packages from Oracle

Oracle provides Debian packages for installing MySQL on Debian or Debian-like Linux systems. The packages are available through two different channels:

- The MySQL APT Repository, supporting Debian and Ubuntu platforms. For details, see Section 2.5.3, “Installing MySQL on Linux Using the MySQL APT Repository”.
- The MySQL Developer Zone's Download Area. For details, see Section 2.1.3, “How to Get MySQL”. The following are some information on the Debian packages available there and the instructions for installing them:
  - You may also need to install the `libaio` library if it is not already present on your system:

    ```
    shell> sudo apt-get install libaio1
    ```

  - Various Debian packages are provided in the MySQL Developer Zone for installing different components of MySQL. The preferred method is to use the tarball bundle, which contains the packages needed for a basic setup of MySQL. The tarball bundles have names in the format of `mysql-server_`*`MVER`*`-`*`DVER`*`_CPU.deb-bundle.tar`. *`MVER`* is the MySQL version and *`DVER`* is the Linux distribution version. The *`CPU`* value indicates the processor type or family for which the package is built, as shown in the following table:

    Table 2.12 MySQL Debian and Ubuntu Installation Package CPU Identifiers

`CPU` Value	Intended Processor Type or Family
`i386`	Pentium processor or better, 32 bit
`amd64`	64-bit x86 processor

  - After downloading the tarball, unpack it with the following command:

    ```
    shell> tar -xvf mysql-server_MVER-DVER_CPU.deb-bundle.tar
    ```

  - In general, install the `deb` packages unpacked from the tarball with the command (see explanations below for the extra steps required for installing the server package):

    ```
    shell> sudo dpkg -i package-name.deb
    ```

    There are four packages to install:

    - The database common files (install this package before the other ones):

      ```
      shell> sudo dpkg -i mysql-common_MVER-DVER_CPU.deb
      ```

    - The MySQL server:

      Install first the package for the database common files (see the last bullet), and then pre-configure your server installation by the following command:

      ```
      shell> sudo dpkg-preconfigure mysql-community-server_MVER-DVER_CPU.deb
      ```

      You are asked to provide a password for the `root` user for your MySQL installation. You might also be asked other questions regarding the installation.

      Important

      Make sure you remember the root password you set. Users who want to set a password later can leave the **password** field blank in the dialogue box and just press **OK**. However, it is very important that you set the

password soon using the program mysql_secure_installation, as people can gain anonymous access to your MySQL server until you have secured the database's root account with a password.

Next, install the server package with the following command:

```
shell> sudo dpkg -i mysql-community-server_MVER-DVER_CPU.deb
```

- The MySQL client:

```
shell> sudo dpkg -i mysql-community-client_MVER-DVER_CPU.deb
```

- The MySQL shared client library:

```
shell> sudo dpkg -i libmysqlclient18_MVER-DVER_CPU.deb
```

Here are where the files are installed on the system:

- All configuration files (like `my.cnf`) are under `/etc`
- All binaries, libraries, headers, etc., are under `/usr`
- The data directory is under `/var`

Note

Debian distributions of MySQL are also provided by other vendors. Be aware that they may differ from those built by Oracle in features, capabilities, and conventions (including communication setup), and that the instructions in this manual do not necessarily apply to installing them. The vendor's instructions should be consulted instead.

2.5.7 Installing MySQL on Linux from the Native Software Repositories

Many Linux distributions include a version of the MySQL server, client tools, and development components in their native software repositories and can be installed with the platforms' standard package management systems. This section provides basic instructions for installing MySQL using those package management systems.

Important

Native packages are often several versions behind the currently available release. You are also normally unable to install development milestone releases (DMRs), as these are not usually made available in the native repositories. Before proceeding, we recommend that you check out the other installation options described in Section 2.5, “Installing MySQL on Linux”.

Distribution specific instructions are shown below:

- **Red Hat Linux, Fedora, CentOS**

Note

For a number of Linux distributions, you can install MySQL using the MySQL Yum repository instead of the platform's native software repository. See Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository” for details.

For Red Hat and similar distributions, the MySQL distribution is divided into a number of separate packages, `mysql` for the client tools, `mysql-server` for the server and associated tools, and `mysql-libs` for the libraries. The libraries are required if you want to provide connectivity from different languages and environments such as Perl, Python and others.

To install, use the `yum` command to specify the packages that you want to install. For example:

```
root-shell> yum install mysql mysql-server mysql-libs mysql-server
Loaded plugins: presto, refresh-packagekit
Setting up Install Process
Resolving Dependencies
--> Running transaction check
---> Package mysql.x86_64 0:5.1.48-2.fc13 set to be updated
---> Package mysql-libs.x86_64 0:5.1.48-2.fc13 set to be updated
---> Package mysql-server.x86_64 0:5.1.48-2.fc13 set to be updated
--> Processing Dependency: perl-DBD-MySQL for package: mysql-server-5.1.48-2.fc13.x86_64
--> Running transaction check
---> Package perl-DBD-MySQL.x86_64 0:4.017-1.fc13 set to be updated
--> Finished Dependency Resolution

Dependencies Resolved

================================================================================
 Package                Arch          Version                Repository      Size
================================================================================
Installing:
 mysql                  x86_64        5.1.48-2.fc13          updates        889 k
 mysql-libs             x86_64        5.1.48-2.fc13          updates        1.2 M
 mysql-server           x86_64        5.1.48-2.fc13          updates        8.1 M
Installing for dependencies:
 perl-DBD-MySQL         x86_64        4.017-1.fc13           updates        136 k

Transaction Summary
================================================================================
Install       4 Package(s)
Upgrade       0 Package(s)

Total download size: 10 M
Installed size: 30 M
Is this ok [y/N]: y
Downloading Packages:
Setting up and reading Presto delta metadata
Processing delta metadata
Package(s) data still to download: 10 M
(1/4): mysql-5.1.48-2.fc13.x86_64.rpm                    | 889 kB     00:04
(2/4): mysql-libs-5.1.48-2.fc13.x86_64.rpm               | 1.2 MB     00:06
(3/4): mysql-server-5.1.48-2.fc13.x86_64.rpm             | 8.1 MB     00:40
(4/4): perl-DBD-MySQL-4.017-1.fc13.x86_64.rpm            | 136 kB     00:00
--------------------------------------------------------------------------------
Total                                           201 kB/s |  10 MB     00:52
Running rpm_check_debug
Running Transaction Test
Transaction Test Succeeded
Running Transaction
  Installing     : mysql-libs-5.1.48-2.fc13.x86_64                         1/4
  Installing     : mysql-5.1.48-2.fc13.x86_64                              2/4
  Installing     : perl-DBD-MySQL-4.017-1.fc13.x86_64                      3/4
  Installing     : mysql-server-5.1.48-2.fc13.x86_64                       4/4

Installed:
  mysql.x86_64 0:5.1.48-2.fc13            mysql-libs.x86_64 0:5.1.48-2.fc13
  mysql-server.x86_64 0:5.1.48-2.fc13

Dependency Installed:
  perl-DBD-MySQL.x86_64 0:4.017-1.fc13
```

```
Complete!
```

MySQL and the MySQL server should now be installed. A sample configuration file is installed into `/etc/my.cnf`. An `init` script, to start and stop the server, is installed into `/etc/init.d/mysqld`. To start the MySQL server use `service`:

```
root-shell> service mysqld start
```

To enable the server to be started and stopped automatically during boot, use `chkconfig`:

```
root-shell> chkconfig --levels 235 mysqld on
```

Which enables the MySQL server to be started (and stopped) automatically at the specified the run levels.

The database tables are automatically created for you, if they do not already exist. You should, however, run `mysql_secure_installation` to set the root passwords on your server.

- **Debian, Ubuntu, Kubuntu**

Note

For Debian, Ubuntu, and Kubuntu, MySQL can be installed using the MySQL APT Repository instead of the platform's native software repository. See Section 2.5.3, “Installing MySQL on Linux Using the MySQL APT Repository” for details.

On Debian and related distributions, there are two packages for MySQL in their software repositories, `mysql-client` and `mysql-server`, for the client and server components respectively. You should specify an explicit version, for example `mysql-client-5.1`, to ensure that you install the version of MySQL that you want.

To download and install, including any dependencies, use the `apt-get` command, specifying the packages that you want to install.

Note

Before installing, make sure that you update your `apt-get` index files to ensure you are downloading the latest available version.

A sample installation of the MySQL packages might look like this (some sections trimmed for clarity):

```
root-shell> apt-get install mysql-client-5.1 mysql-server-5.1
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  linux-headers-2.6.28-11 linux-headers-2.6.28-11-generic
Use 'apt-get autoremove' to remove them.
The following extra packages will be installed:
  bsd-mailx libdbd-mysql-perl libdbi-perl libhtml-template-perl
  libmysqlclient15off libmysqlclient16 libnet-daemon-perl libplrpc-perl mailx
  mysql-common postfix
Suggested packages:
  dbishell libipc-sharedcache-perl tinyca procmail postfix-mysql postfix-pgsql
  postfix-ldap postfix-pcre sasl2-bin resolvconf postfix-cdb
The following NEW packages will be installed
  bsd-mailx libdbd-mysql-perl libdbi-perl libhtml-template-perl
  libmysqlclient15off libmysqlclient16 libnet-daemon-perl libplrpc-perl mailx
  mysql-client-5.1 mysql-common mysql-server-5.1 postfix
0 upgraded, 13 newly installed, 0 to remove and 182 not upgraded.
Need to get 1907kB/25.3MB of archives.
After this operation, 59.5MB of additional disk space will be used.
Do you want to continue [Y/n]? Y
Get: 1 http://gb.archive.ubuntu.com jaunty-updates/main mysql-common 5.1.30really5.0.75-0ubuntu10.5 [63.6
Get: 2 http://gb.archive.ubuntu.com jaunty-updates/main libmysqlclient15off 5.1.30really5.0.75-0ubuntu10.
```

```
Fetched 1907kB in 9s (205kB/s)
Preconfiguring packages ...
Selecting previously deselected package mysql-common.
(Reading database ... 121260 files and directories currently installed.)
...
Processing 1 added doc-base file(s)...
Registering documents with scrollkeeper...
Setting up libnet-daemon-perl (0.43-1) ...
Setting up libplrpc-perl (0.2020-1) ...
Setting up libdbi-perl (1.607-1) ...
Setting up libmysqlclient15off (5.1.30really5.0.75-0ubuntu10.5) ...

Setting up libdbd-mysql-perl (4.008-1) ...
Setting up libmysqlclient16 (5.1.31-1ubuntu2) ...

Setting up mysql-client-5.1 (5.1.31-1ubuntu2) ...

Setting up mysql-server-5.1 (5.1.31-1ubuntu2) ...
 * Stopping MySQL database server mysqld
   ...done.
100825 11:46:15  InnoDB: Started; log sequence number 0 46409
100825 11:46:15  InnoDB: Starting shutdown...
100825 11:46:17  InnoDB: Shutdown completed; log sequence number 0 46409
100825 11:46:17 [Warning] Forcing shutdown of 1 plugins
 * Starting MySQL database server mysqld
   ...done.
 * Checking for corrupt, not cleanly closed and upgrade needing tables.
...
Processing triggers for libc6 ...
ldconfig deferred processing now taking place
```

Note

The `apt-get` command installs a number of packages, including the MySQL server, in order to provide the typical tools and application environment. This can mean that you install a large number of packages in addition to the main MySQL package.

During installation, the initial database is created, and you are prompted for the MySQL `root` password (and confirmation). A configuration file is created in `/etc/mysql/my.cnf`. An `init` script is created in `/etc/init.d/mysql`.

The server is already started. You can manually start and stop the server using:

```
root-shell> service mysql [start|stop]
```

The service is automatically added to run levels 2, 3 and 4,, with stop scripts in the single, shutdown and restart levels.

- **Gentoo Linux**

  As a source-based distribution, installing MySQL on Gentoo involves downloading the source, patching the Gentoo specifics, and then compiling the MySQL server and installing it. This process is handled automatically by the `emerge` command.

  The MySQL server and client tools are provided within a single package, `dev-db/mysql`. You can obtain a list of the versions available to install by looking at the portage directory for the package:

```
root-shell> ls /usr/portage/dev-db/mysql/mysql-5.6*
mysql-5.6.27.ebuild
mysql-5.6.27-r1.ebuild
```

```
mysql-5.6.28.ebuild
```

To install a specific MySQL version, you must specify the entire atom. For example:

```
root-shell> emerge =dev-db/mysql-5.6.27-r1
```

After installation, you should initialize the data directory and set the password for the MySQL *`root`* user (see Section 2.10.1, “Initializing the Data Directory”). Alternatively, use the configuration interface to perform those tasks:

```
root-shell> emerge --config =dev-db/mysql-5.6.27-r1
```

During installation, a sample configuration file is created for you in `/etc/mysql/my.cnf`, and an init script is created in `/etc/init.d/mysql`.

To enable MySQL to start automatically at the normal (default) run levels, use this command:

```
root-shell> rc-update add mysql default
```

2.5.8 Deploying MySQL on Linux with Docker

The Docker deployment framework supports easy installation and configuration of MySQL Server. This section explains how to use a MySQL Server Docker image.

You need to have Docker installed on your system before you can use a MySQL Server Docker image. See Install Docker for instructions.

Important

You need to either run `docker` commands with `sudo`, or create a `docker` usergroup, and then add to it any users who want to run `docker` commands. See details here. Because Docker containers are always run with root privileges, you should understand the Docker daemon attack surface and properly mitigate the related risks.

The instructions for using the MySQL Docker container are divided into two sections.

2.5.8.1 Basic Steps for MySQL Server Deployment with Docker

Warning

The MySQL Docker images maintained by the MySQL team are built specifically for Linux platforms. Other platforms are not supported, and users using these MySQL Docker images on them are doing so at their own risk. See the discussion here for some known limitations for running these containers on non-Linux operating systems.

- Downloading a MySQL Server Docker Image
- Starting a MySQL Server Instance
- Connecting to MySQL Server from within the Container
- Container Shell Access
- Stopping and Deleting a MySQL Container
- More Topics on Deploying MySQL Server with Docker

Downloading a MySQL Server Docker Image

Downloading the server image in a separate step is not strictly necessary; however, performing this step before you create your Docker container ensures your local image is up to date. To download the MySQL Community Server image, run this command:

```
docker pull mysql/mysql-server:tag
```

The *tag* is the label for the image version you want to pull (for example, `5.5`, `5.6`, `5.7`, `8.0`, or `latest`). If **:tag** is omitted, the `latest` label is used, and the image for the latest GA version of MySQL Community Server is downloaded. Refer to the list of tags for available versions on the mysql/mysql-server page in the Docker Hub.

You can list downloaded Docker images with this command:

```
shell> docker images
REPOSITORY           TAG                 IMAGE ID            CREATED             SIZE
mysql/mysql-server   latest              3157d7f55f8d        4 weeks ago         241MB
```

Starting a MySQL Server Instance

Start a new Docker container for the MySQL Server with this command:

```
docker run --name=mysql1 -d mysql/mysql-server:tag
```

The `--name` option, for supplying a custom name for your server container (`mysql1` in the example), is optional; if no container name is supplied, a random one is generated. If the Docker image of the specified name and tag has not been downloaded by an earlier `docker pull` or `docker run` command, the image is now downloaded. After download completes, initialization for the container begins, and the container appears in the list of running containers when you run the `docker ps` command; for example:

```
shell> docker ps
CONTAINER ID   IMAGE                 COMMAND                  CREATED             STATUS
a24888f0d6f4   mysql/mysql-server   "/entrypoint.sh my..."   14 seconds ago      Up 13 seconds (health:
```

The container initialization might take some time. When the server is ready for use, the `STATUS` of the container in the output of the `docker ps` command changes from `(health: starting)` to `(healthy)`.

The `-d` option used in the `docker run` command above makes the container run in the background. Use this command to monitor the output from the container:

```
docker logs mysql1
```

Once initialization is finished, the command's output is going to contain the random password generated for the root user; check the password with, for example, this command:

```
shell> docker logs mysql1 2>&1 | grep GENERATED
GENERATED ROOT PASSWORD: Axegh3kAJyDLaRuBemecis&EShOs
```

Connecting to MySQL Server from within the Container

Once the server is ready, you can run the `mysql` client within the MySQL Server container you just started, and connect it to the MySQL Server. Use the `docker exec -it` command to start a `mysql` client inside the Docker container you have started, like the following:

```
docker exec -it mysql1 mysql -uroot -p
```

When asked, enter the generated root password (see the last step in Starting a MySQL Server Instance above on how to find the password). Because the `MYSQL_ONETIME_PASSWORD` option is true by default, after you have connected a `mysql` client to the server, you must reset the server root password by issuing this statement:

```
mysql> ALTER USER 'root'@'localhost' IDENTIFIED BY 'newpassword';
```

Substitute *newpassword* with the password of your choice. Once the password is reset, the server is ready for use.

Container Shell Access

To have shell access to your MySQL Server container, use the `docker exec -it` command to start a bash shell inside the container:

```
shell> docker exec -it mysql1 bash
bash-4.2#
```

You can then run Linux commands inside the container. For example, to view contents in the server's data directory inside the container, use this command:

```
bash-4.2# ls /var/lib/mysql
auto.cnf    ca.pem      client-key.pem  ib_logfile0  ibdata1  mysql       mysql.sock.lock    private_key.pe
ca-key.pem  client-cert.pem  ib_buffer_pool  ib_logfile1  ibtmp1   mysql.sock  performance_schema  public_k
```

Stopping and Deleting a MySQL Container

To stop the MySQL Server container we have created, use this command:

```
docker stop mysql1
```

`docker stop` sends a SIGTERM signal to the `mysqld` process, so that the server is shut down gracefully.

Also notice that when the main process of a container (`mysqld` in the case of a MySQL Server container) is stopped, the Docker container stops automatically.

To start the MySQL Server container again:

```
docker start mysql1
```

To stop and start again the MySQL Server container with a single command:

```
docker restart mysql1
```

To delete the MySQL container, stop it first, and then use the `docker rm` command:

```
docker stop mysql1
```

```
docker rm mysql1
```

If you want the Docker volume for the server's data directory to be deleted at the same time, add the `-v` option to the `docker rm` command.

More Topics on Deploying MySQL Server with Docker

For more topics on deploying MySQL Server with Docker like server configuration, persisting data and configuration, server error log, and container environment variables, see Section 2.5.8.2, “More Topics on Deploying MySQL Server with Docker”.

2.5.8.2 More Topics on Deploying MySQL Server with Docker

- The Optimized MySQL Installation for Docker
- Configuring the MySQL Server
- Persisting Data and Configuration Changes
- Running Additional Initialization Scripts
- Connect to MySQL from an Application in Another Docker Container
- Server Error Log
- Known Issues

- Docker Environment Variables

The Optimized MySQL Installation for Docker

Docker images for MySQL are optimized for code size, which means they only include crucial components that are expected to be relevant for the majority of users who run MySQL instances in Docker containers. A MySQL Docker installation is different from a common, non-Docker installation in the following aspects:

- Included binaries are limited to:
  - /usr/bin/my_print_defaults
  - /usr/bin/mysql
  - /usr/bin/mysql_config
  - /usr/bin/mysql_install_db
  - /usr/bin/mysql_tzinfo_to_sql
  - /usr/bin/mysql_upgrade
  - /usr/bin/mysqladmin
  - /usr/bin/mysqlcheck
  - /usr/bin/mysqldump
  - /usr/sbin/mysqld
- All binaries are stripped; they contain no debug information.

Configuring the MySQL Server

When you start the MySQL Docker container, you can pass configuration options to the server through the `docker run` command; for example, for the MySQL Server:

```
docker run --name mysql1 -d mysql/mysql-server --character-set-server=utf8mb4 --collation-server=utf8mb
```

The command starts your MySQL Server with `utf8mb4` as the default character set and `utf8mb4_col` as the default collation for your databases.

Another way to configure the MySQL Server is to prepare a configuration file and mount it at the location of the server configuration file inside the container. See Persisting Data and Configuration Changes for details.

Persisting Data and Configuration Changes

Docker containers are in principle ephemeral, and any data or configuration are expected to be lost if the container is deleted or corrupted (see discussions here). Docker volumes, however, provides a mechanism to persist data created inside a Docker container. At its initialization, the MySQL Server container creates a Docker volume for the server data directory. The JSON output for running the `docker inspect` command on the container has a `Mount` key, whose value provides information on the data directory volume:

```
shell> docker inspect mysql1
...
 "Mounts": [
           {
               "Type": "volume",
               "Name": "4f2d463cfc4bdd4baebcb098c97d7da3337195ed2c6572bc0b89f7e845d27652",
               "Source": "/var/lib/docker/volumes/4f2d463cfc4bdd4baebcb098c97d7da3337195ed2c6572bc0b89
               "Destination": "/var/lib/mysql",
```

```
                "Driver": "local",
                "Mode": "",
                "RW": true,
                "Propagation": ""
            }
        ],
...
```

The output shows that the source folder `/var/lib/docker/volumes/4f2d463cfc4bdd4baebcb098c97d7da3337195ed2c6572bc0b89f7e845d27652/_data`, in which data is persisted on the host, has been mounted at `/var/lib/mysql`, the server data directory inside the container.

Another way to preserve data is to bind-mount a host directory using the `--mount` option when creating the container. The same technique can be used to persist the configuration of the server. The following command creates a MySQL Server container and bind-mounts both the data directory and the server configuration file:

```
docker run --name=mysql1 \
--mount type=bind,src=/path-on-host-machine/my.cnf,dst=/etc/my.cnf \
--mount type=bind,src=/path-on-host-machine/datadir,dst=/var/lib/mysql \
-d mysql/mysql-server:tag
```

The command mounts *`path-on-host-machine/my.cnf`* at `/etc/my.cnf` (the server configuration file inside the container), and *`path-on-host-machine/datadir`* at `/var/lib/mysql` (the data directory inside the container). The following conditions must be met for the bind-mounting to work:

- The configuration file *`path-on-host-machine/my.cnf`* must already exist, and it must contain the specification for starting the server using the user `mysql`:

  ```
  [mysqld]
  user=mysql
  ```

  You can also include other server configuration options in the file.

- The data directory *`path-on-host-machine/datadir`* must already exist. For server initialization to happen, the directory must be empty. You can also mount a directory prepopulated with data and start the server with it; however, you must make sure you start the Docker container with the same configuration as the server that created the data, and any host files or directories required are mounted when starting the container.

Running Additional Initialization Scripts

If there are any `.sh` or `.sql` scripts you want to run on the database immediately after it has been created, you can put them into a host directory and then mount the directory at `/docker-entrypoint-initdb.d/` inside the container. For example, for a MySQL Server container:

```
docker run --name=mysql1 \
--mount type=bind,src=/path-on-host-machine/scripts/,dst=/docker-entrypoint-initdb.d/ \
-d mysql/mysql-server:tag
```

Connect to MySQL from an Application in Another Docker Container

By setting up a Docker network, you can allow multiple Docker containers to communicate with each other, so that a client application in another Docker container can access the MySQL Server in the server container. First, create a Docker network:

```
docker network create my-custom-net
```

Then, when you are creating and starting the server and the client containers, use the `--network` option to put them on network you created. For example:

```
docker run --name=mysql1 --network=my-custom-net -d mysql/mysql-server
```

```
docker run --name=myapp1 --network=my-custom-net -d myapp
```

The `myapp1` container can then connect to the `mysql1` container with the `mysql1` hostname and vice versa, as Docker automatically sets up a DNS for the given container names. In the following example, we run the `mysql` client from inside the `myapp1` container to connect to host `mysql1` in its own container:

```
docker exec -it myapp1 mysql --host=mysql1 --user=myuser --password
```

For other networking techniques for containers, see the Docker container networking section in the Docker Documentation.

Server Error Log

When the MySQL Server is first started with your server container, a server error log is NOT generated if either of the following conditions is true:

- A server configuration file from the host has been mounted, but the file does not contain the system variable `log_error` (see Persisting Data and Configuration Changes on bind-mounting a server configuration file).
- A server configuration file from the host has not been mounted, but the Docker environment variable `MYSQL_LOG_CONSOLE` is `true` (the variable's default state for MySQL 5.6 server containers is `false`). The MySQL Server's error log is then redirected to `stderr`, so that the error log goes into the Docker container's log and is viewable using the `docker logs` *`mysqld-container`* command.

To make MySQL Server generate an error log when either of the two conditions is true, use the `--log-error` option to configure the server to generate the error log at a specific location inside the container. To persist the error log, mount a host file at the location of the error log inside the container as explained in Persisting Data and Configuration Changes. However, you must make sure your MySQL Server inside its container has write access to the mounted host file.

Known Issues

- When using the server system variable `audit_log_file` to configure the audit log file name, use the `loose` option modifier with it, or Docker will be unable to start the server.

Docker Environment Variables

When you create a MySQL Server container, you can configure the MySQL instance by using the `--env` option (`-e` in short) and specifying one or more of the following environment variables.

Notes

- None of the variables below has any effect if the data directory you mount is not empty, as no server initialization is going to be attempted then (see Persisting Data and Configuration Changes for more details). Any pre-existing contents in the folder, including any old server settings, are not modified during the container startup.
- The boolean variables including `MYSQL_RANDOM_ROOT_PASSWORD`, `MYSQL_ONETIME_PASSWORD`, `MYSQL_ALLOW_EMPTY_PASSWORD`, and `MYSQL_LOG_CONSOLE` are made true by setting them with any strings of nonzero lengths. Therefore, setting them to, for example, “0”, “false”, or “no” does not make them false, but actually makes them true. This is a known issue of the MySQL Server containers.

- `MYSQL_RANDOM_ROOT_PASSWORD`: When this variable is true (which is its default state, unless `MYSQL_ROOT_PASSWORD` or `MYSQL_ALLOW_EMPTY_PASSWORD` is set to true), a random password for the server's root user is generated when the Docker container is started. The password is printed to `stdout` of the container and can be found by looking at the container’s log (see Starting a MySQL Server Instance).

- `MYSQL_ONETIME_PASSWORD`: When the variable is true (which is its default state, unless `MYSQL_ROOT_PASSWORD` is set or `MYSQL_ALLOW_EMPTY_PASSWORD` is set to true), the root user's password is set as expired and must be changed before MySQL can be used normally.

- `MYSQL_DATABASE`: This variable allows you to specify the name of a database to be created on image startup. If a user name and a password are supplied with `MYSQL_USER` and `MYSQL_PASSWORD`, the user is created and granted superuser access to this database (corresponding to `GRANT ALL`). The specified database is created by a CREATE DATABASE IF NOT EXIST statement, so that the variable has no effect if the database already exists.

- `MYSQL_USER`, `MYSQL_PASSWORD`: These variables are used in conjunction to create a user and set that user's password, and the user is granted superuser permissions for the database specified by the `MYSQL_DATABASE` variable. Both `MYSQL_USER` and `MYSQL_PASSWORD` are required for a user to be created—if any of the two variables is not set, the other is ignored. If both variables are set but `MYSQL_DATABASE` is not, the user is created without any privileges.

Note

There is no need to use this mechanism to create the root superuser, which is created by default with the password set by either one of the mechanisms discussed in the descriptions for `MYSQL_ROOT_PASSWORD` and `MYSQL_RANDOM_ROOT_PASSWORD`, unless `MYSQL_ALLOW_EMPTY_PASSWORD` is true.

- `MYSQL_ROOT_HOST`: By default, MySQL creates the `'root'@'localhost'` account. This account can only be connected to from inside the container as described in Connecting to MySQL Server from within the Container. To allow root connections from other hosts, set this environment variable. For example, the value `172.17.0.1`, which is the default Docker gateway IP, allows connections from the host machine that runs the container. The option accepts only one entry, but wildcards are allowed (for example, `MYSQL_ROOT_HOST=172.*.*.*` or `MYSQL_ROOT_HOST=%`).

- `MYSQL_LOG_CONSOLE`: When the variable is true (the variable's default state for MySQL 5.6 server containers is `false`), the MySQL Server's error log is redirected to `stderr`, so that the error log goes into the Docker container's log and is viewable using the `docker logs` *`mysqld-container`* command.

Note

The variable has no effect if a server configuration file from the host has been mounted (see Persisting Data and Configuration Changes on bind-mounting a configuration file).

- `MYSQL_ROOT_PASSWORD`: This variable specifies a password that is set for the MySQL root account.

Warning

Setting the MySQL root user password on the command line is insecure. As an alternative to specifying the password explicitly, you can set the variable with a container file path for a password file, and then mount a file from your host that contains the password at the container file path. This is still not very secure, as the location of the password file is still exposed. It is preferable to use the default settings of `MYSQL_RANDOM_ROOT_PASSWORD=true` and `MYSQL_ONETIME_PASSWORD=true` being both true.

- `MYSQL_ALLOW_EMPTY_PASSWORD`. Set it to true to allow the container to be started with a blank password for the root user.

Warning

Setting this variable to true is insecure, because it is going to leave your MySQL instance completely unprotected, allowing anyone

to gain complete superuser access. It is preferable to use the default settings of `MYSQL_RANDOM_ROOT_PASSWORD=true` and `MYSQL_ONETIME_PASSWORD=true` being both true.

2.5.8.3 Deploying MySQL on Windows and Other Non-Linux Platforms with Docker

Warning

The MySQL Docker images provided by Oracle are built specifically for Linux platforms. Other platforms are not supported, and users running the MySQL Docker images from Oracle on them are doing so at their own risk. This section discusses some known issues for the images when used on non-Linux platforms.

Known Issues for using the MySQL Server Docker images from Oracle on Windows include:

- If you are bind-mounting on the container's MySQL data directory (see Persisting Data and Configuration Changes for details), you have to set the location of the server socket file with the `--socket` option to somewhere outside of the MySQL data directory; otherwise, the server fails to start. This is because the way Docker for Windows handles file mounting does not allow a host file from being bind-mounted on the socket file.

2.5.9 Installing MySQL on Linux with Juju

The Juju deployment framework supports easy installation and configuration of MySQL servers. For instructions, see https://jujucharms.com/mysql/.

2.6 Installing MySQL Using Unbreakable Linux Network (ULN)

Linux supports a number of different solutions for installing MySQL, covered in Section 2.5, “Installing MySQL on Linux”. One of the methods, covered in this section, is installing from Oracle's Unbreakable Linux Network (ULN). You can find information about Oracle Linux and ULN under http://linux.oracle.com/.

To use ULN, you need to obtain a ULN login and register the machine used for installation with ULN. This is described in detail in the ULN FAQ. The page also describes how to install and update packages. The MySQL packages are in the “MySQL for Oracle Linux 6” and “MySQL for Oracle Linux 7” channels for your system architecture on ULN.

Note

At the time of this writing, ULN provides MySQL 5.6 for Oracle Linux 6 and Oracle Linux 7.

Once MySQL has been installed using ULN, you can find information on starting and stopping the server, and more, in this section, particularly under Section 2.5.5, “Installing MySQL on Linux Using RPM Packages from Oracle”.

If you're updating an existing MySQL installation to an installation using ULN, the recommended procedure is to export your data using `mysqldump`, remove the existing installation, install MySQL from ULN, and load the exported data into your freshly installed MySQL.

If the existing MySQL installation you're upgrading from is from a previous release series (prior to MySQL 5.6), make sure to read the section on upgrading MySQL, Section 2.11, “Upgrading MySQL”.

2.7 Installing MySQL on Solaris

Note

MySQL 5.6 supports Solaris 10 (Update 11 and later), and Solaris 11 (Update 3 and later).

MySQL on Solaris is available in a number of different formats.

- For information on installing using the native Solaris PKG format, see Section 2.7.1, “Installing MySQL on Solaris Using a Solaris PKG”.
- To use a standard `tar` binary installation, use the notes provided in Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”. Check the notes and hints at the end of this section for Solaris specific notes that you may need before or after installation.

To obtain a binary MySQL distribution for Solaris in tarball or PKG format, https://dev.mysql.com/downloads/mysql/5.6.html.

Additional notes to be aware of when installing and using MySQL on Solaris:

- If you want to use MySQL with the `mysql` user and group, use the `groupadd` and `useradd` commands:

```
groupadd mysql
useradd -g mysql -s /bin/false mysql
```

- If you install MySQL using a binary tarball distribution on Solaris, because the Solaris `tar` cannot handle long file names, use GNU `tar` (`gtar`) to unpack the distribution. If you do not have GNU `tar` on your system, install it with the following command:

```
pkg install archiver/gnu-tar
```

- You should mount any file systems on which you intend to store `InnoDB` files with the `forcedirectio` option. (By default mounting is done without this option.) Failing to do so causes a significant drop in performance when using the `InnoDB` storage engine on this platform.
- If you would like MySQL to start automatically, you can copy `support-files/mysql.server` to `/etc/init.d` and create a symbolic link to it named `/etc/rc3.d/S99mysql.server`.
- If too many processes try to connect very rapidly to `mysqld`, you should see this error in the MySQL log:

```
Error in accept: Protocol error
```

  You might try starting the server with the `--back_log=50` option as a workaround for this.

- To configure the generation of core files on Solaris you should use the `coreadm` command. Because of the security implications of generating a core on a `setuid()` application, by default, Solaris does not support core files on `setuid()` programs. However, you can modify this behavior using `coreadm`. If you enable `setuid()` core files for the current user, they are generated using mode 600, and are owned by the superuser.

2.7.1 Installing MySQL on Solaris Using a Solaris PKG

You can install MySQL on Solaris using a binary package using the native Solaris PKG format instead of the binary tarball distribution.

To use this package, download the corresponding `mysql-VERSION-solaris10-PLATFORM.pkg.gz` file, then uncompress it. For example:

```
shell> gunzip mysql-5.6.51-solaris10-x86_64.pkg.gz
```

To install a new package, use `pkgadd` and follow the onscreen prompts. You must have root privileges to perform this operation:

```
shell> pkgadd -d mysql-5.6.51-solaris10-x86_64.pkg

The following packages are available:
```

```
  1  mysql     MySQL Community Server (GPL)
               (i86pc) 5.6.51

Select package(s) you wish to process (or 'all' to process
all packages). (default: all) [?,??,q]:
```

The PKG installer installs all of the files and tools needed, and then initializes your database if one does not exist. To complete the installation, you should set the root password for MySQL as provided in the instructions at the end of the installation. Alternatively, you can run the `mysql_secure_installation` script that comes with the installation.

By default, the PKG package installs MySQL under the root path `/opt/mysql`. You can change only the installation root path when using `pkgadd`, which can be used to install MySQL in a different Solaris zone. If you need to install in a specific directory, use a binary `tar` file distribution.

The `pkg` installer copies a suitable startup script for MySQL into `/etc/init.d/mysql`. To enable MySQL to startup and shutdown automatically, you should create a link between this file and the init script directories. For example, to ensure safe startup and shutdown of MySQL you could use the following commands to add the right links:

```
shell> ln /etc/init.d/mysql /etc/rc3.d/S91mysql
shell> ln /etc/init.d/mysql /etc/rc0.d/K02mysql
```

To remove MySQL, the installed package name is `mysql`. You can use this in combination with the `pkgrm` command to remove the installation.

To upgrade when using the Solaris package file format, you must remove the existing installation before installing the updated package. Removal of the package does not delete the existing database information, only the server, binaries and support files. The typical upgrade sequence is therefore:

```
shell> mysqladmin shutdown
shell> pkgrm mysql
shell> pkgadd -d mysql-5.6.51-solaris10-x86_64.pkg
shell> mysqld_safe &
shell> mysql_upgrade
```

You should check the notes in Section 2.11, “Upgrading MySQL” before performing any upgrade.

2.8 Installing MySQL on FreeBSD

This section provides information about installing MySQL on variants of FreeBSD Unix.

You can install MySQL on FreeBSD by using the binary distribution provided by Oracle. For more information, see Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”.

The easiest (and preferred) way to install MySQL is to use the `mysql-server` and `mysql-client` ports available at http://www.freebsd.org/. Using these ports gives you the following benefits:

- A working MySQL with all optimizations enabled that are known to work on your version of FreeBSD.
- Automatic configuration and build.
- Startup scripts installed in `/usr/local/etc/rc.d`.
- The ability to use `pkg_info -L` to see which files are installed.
- The ability to use `pkg_delete` to remove MySQL if you no longer want it on your machine.

The MySQL build process requires GNU make (`gmake`) to work. If GNU `make` is not available, you must install it first before compiling MySQL.

To install using the ports system:

```
# cd /usr/ports/databases/mysql56-server
# make
...
# cd /usr/ports/databases/mysql56-client
# make
...
```

The standard port installation places the server into `/usr/local/libexec/mysqld`, with the startup script for the MySQL server placed in `/usr/local/etc/rc.d/mysql-server`.

Some additional notes on the BSD implementation:

- To remove MySQL after installation using the ports system:

```
# cd /usr/ports/databases/mysql56-server
# make deinstall
...
# cd /usr/ports/databases/mysql56-client
# make deinstall
...
```

- If you get problems with the current date in MySQL, setting the `TZ` variable should help. See Section 4.9, “Environment Variables”.

2.9 Installing MySQL from Source

Building MySQL from the source code enables you to customize build parameters, compiler optimizations, and installation location. For a list of systems on which MySQL is known to run, see https://www.mysql.com/support/supportedplatforms/database.html.

Before you proceed with an installation from source, check whether Oracle produces a precompiled binary distribution for your platform and whether it works for you. We put a great deal of effort into ensuring that our binaries are built with the best possible options for optimal performance. Instructions for installing binary distributions are available in Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”.

If you are interested in building MySQL from a source distribution using build options the same as or similar to those use by Oracle to produce binary distributions on your platform, obtain a binary distribution, unpack it, and look in the `docs/INFO_BIN` file, which contains information about how that MySQL distribution was configured and compiled.

Warning

Building MySQL with nonstandard options may lead to reduced functionality, performance, or security.

2.9.1 Source Installation Methods

There are two methods for installing MySQL from source:

- Use a standard MySQL source distribution. To obtain a standard distribution, see Section 2.1.3, “How to Get MySQL”. For instructions on building from a standard distribution, see Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”.

  Standard distributions are available as compressed `tar` files, Zip archives, or RPM packages. Distribution files have names of the form `mysql-`*`VERSION`*`.tar.gz`, `mysql-`*`VERSION`*`.zip`, or `mysql-`*`VERSION`*`.rpm`, where *`VERSION`* is a number like `5.6.51`. File names for source distributions can be distinguished from those for precompiled binary distributions in that source distribution names are generic and include no platform name, whereas binary distribution names include a platform name indicating the type of system for which the distribution is intended (for example, `pc-linux-i686` or `winx64`).

- Use a MySQL development tree. For information on building from one of the development trees, see Section 2.9.5, “Installing MySQL Using a Development Source Tree”.

2.9.2 Source Installation Prerequisites

Installation of MySQL from source requires several development tools. Some of these tools are needed no matter whether you use a standard source distribution or a development source tree. Other tool requirements depend on which installation method you use.

To install MySQL from source, the following system requirements must be satisfied, regardless of installation method:

- `CMake`, which is used as the build framework on all platforms. `CMake` can be downloaded from http://www.cmake.org.
- A good `make` program. Although some platforms come with their own `make` implementations, it is highly recommended that you use GNU `make` 3.75 or higher. It may already be available on your system as `gmake`. GNU `make` is available from http://www.gnu.org/software/make/.
- A working ANSI C++ compiler. GCC 4.2.1 or later, Sun Studio 12 or later, Visual Studio 2010 or later, and many current vendor-supplied compilers are known to work.
- An SSL library is required for support of encrypted connections, entropy for random number generation, and other encryption-related operations. To specify the library explicitly, use the `WITH_SSL` option when you invoke `CMake`. For additional information, see Section 2.9.6, “Configuring SSL Library Support”.
- The ncurses library.
- Sufficient free memory. If you encounter problems such as “internal compiler error” when compiling large source files, it may be that you have too little memory. If compiling on a virtual machine, try increasing the memory allocation.
- Perl is needed if you intend to run test scripts. Most Unix-like systems include Perl. On Windows, you can use a version such as ActiveState Perl.

To install MySQL from a standard source distribution, one of the following tools is required to unpack the distribution file:

- For a `.tar.gz` compressed `tar` file: GNU `gunzip` to uncompress the distribution and a reasonable `tar` to unpack it. If your `tar` program supports the `z` option, it can both uncompress and unpack the file.

  GNU `tar` is known to work. The standard `tar` provided with some operating systems is not able to unpack the long file names in the MySQL distribution. You should download and install GNU `tar`, or if available, use a preinstalled version of GNU tar. Usually this is available as `gnutar`, `gtar`, or as `tar` within a GNU or Free Software directory, such as `/usr/sfw/bin` or `/usr/local/bin`. GNU `tar` is available from http://www.gnu.org/software/tar/.
- For a `.zip` Zip archive: `WinZip` or another tool that can read `.zip` files.
- For an `.rpm` RPM package: The `rpmbuild` program used to build the distribution unpacks it.

To install MySQL from a development source tree, the following additional tools are required:

- The Git revision control system is required to obtain the development source code. The GitHub Help provides instructions for downloading and installing Git on different platforms. MySQL officially joined GitHub in September, 2014. For more information about MySQL's move to GitHub, refer to the announcement on the MySQL Release Engineering blog: MySQL on GitHub
- `bison` 2.1 or higher, available from http://www.gnu.org/software/bison/. (Version 1 is no longer supported.) Use the latest version of `bison` where possible; if you experience problems, upgrade to a later version, rather than revert to an earlier one.

`bison` is available from http://www.gnu.org/software/bison/. `bison` for Windows can be downloaded from http://gnuwin32.sourceforge.net/packages/bison.htm. Download the package labeled "Complete package, excluding sources". On Windows, the default location for `bison` is the `C:\Program Files\GnuWin32` directory. Some utilities may fail to find `bison` because of the space in the directory name. Also, Visual Studio may simply hang if there are spaces in the path. You can resolve these problems by installing into a directory that does not contain a space (for example `C:\GnuWin32`).

- On Solaris Express, `m4` must be installed in addition to `bison`. `m4` is available from http://www.gnu.org/software/m4/.

Note

If you have to install any programs, modify your `PATH` environment variable to include any directories in which the programs are located. See Section 4.2.7, "Setting Environment Variables".

If you run into problems and need to file a bug report, please use the instructions in Section 1.6, "How to Report Bugs or Problems".

2.9.3 MySQL Layout for Source Installation

By default, when you install MySQL after compiling it from source, the installation step installs files under `/usr/local/mysql`. The component locations under the installation directory are the same as for binary distributions. See Table 2.3, "MySQL Installation Layout for Generic Unix/Linux Binary Package", and Section 2.3.1, "MySQL Installation Layout on Microsoft Windows". To configure installation locations different from the defaults, use the options described at Section 2.9.7, "MySQL Source-Configuration Options".

2.9.4 Installing MySQL Using a Standard Source Distribution

To install MySQL from a standard source distribution:

1. Verify that your system satisfies the tool requirements listed at Section 2.9.2, "Source Installation Prerequisites".
2. Obtain a distribution file using the instructions in Section 2.1.3, "How to Get MySQL".
3. Configure, build, and install the distribution using the instructions in this section.
4. Perform postinstallation procedures using the instructions in Section 2.10, "Postinstallation Setup and Testing".

MySQL uses `CMake` as the build framework on all platforms. The instructions given here should enable you to produce a working installation. For additional information on using `CMake` to build MySQL, see How to Build MySQL Server with CMake.

If you start from a source RPM, use the following command to make a binary RPM that you can install. If you do not have `rpmbuild`, use `rpm` instead.

```
shell> rpmbuild --rebuild --clean MySQL-VERSION.src.rpm
```

The result is one or more binary RPM packages that you install as indicated in Section 2.5.5, "Installing MySQL on Linux Using RPM Packages from Oracle".

The sequence for installation from a compressed `tar` file or Zip archive source distribution is similar to the process for installing from a generic binary distribution (see Section 2.2, "Installing MySQL on Unix/Linux Using Generic Binaries"), except that it is used on all platforms and includes steps to configure

and compile the distribution. For example, with a compressed `tar` file source distribution on Unix, the basic installation command sequence looks like this:

```
# Preconfiguration setup
shell> groupadd mysql
shell> useradd -r -g mysql -s /bin/false mysql
# Beginning of source-build specific instructions
shell> tar zxvf mysql-VERSION.tar.gz
shell> cd mysql-VERSION
shell> mkdir bld
shell> cd bld
shell> cmake ..
shell> make
shell> make install
# End of source-build specific instructions
# Postinstallation setup
shell> cd /usr/local/mysql
shell> scripts/mysql_install_db --user=mysql
shell> bin/mysqld_safe --user=mysql &
# Next command is optional
shell> cp support-files/mysql.server /etc/init.d/mysql.server
```

`mysql_install_db` creates a default option file named `my.cnf` in the base installation directory. This file is created from a template included in the distribution package named `my-default.cnf`. For more information, see Section 5.1.2.2, “Using a Sample Default Server Configuration File”.

A more detailed version of the source-build specific instructions is shown following.

Note

The procedure shown here does not set up any passwords for MySQL accounts. After following the procedure, proceed to Section 2.10, “Postinstallation Setup and Testing”, for postinstallation setup and testing.

- Perform Preconfiguration Setup
- Obtain and Unpack the Distribution
- Configure the Distribution
- Build the Distribution
- Install the Distribution
- Perform Postinstallation Setup

Perform Preconfiguration Setup

On Unix, set up the `mysql` user and group that are used to run and execute the MySQL server, and own the database directory. For details, see Create a mysql User and Group. Then perform the following steps as the `mysql` user, except as noted.

Obtain and Unpack the Distribution

Pick the directory under which you want to unpack the distribution and change location into it.

Obtain a distribution file using the instructions in Section 2.1.3, “How to Get MySQL”.

Unpack the distribution into the current directory:

- To unpack a compressed `tar` file, `tar` can uncompress and unpack the distribution if it has `z` option support:

```
shell> tar zxvf mysql-VERSION.tar.gz
```

If your `tar` does not have `z` option support, use `gunzip` to unpack the distribution and `tar` to unpack it:

```
shell> gunzip < mysql-VERSION.tar.gz | tar xvf -
```

Alternatively, `CMake` can uncompress and unpack the distribution:

```
shell> cmake -E tar zxvf mysql-VERSION.tar.gz
```

- To unpack a Zip archive, use `WinZip` or another tool that can read `.zip` files.

Unpacking the distribution file creates a directory named `mysql-VERSION`.

Configure the Distribution

Change location into the top-level directory of the unpacked distribution:

```
shell> cd mysql-VERSION
```

Build outside of the source tree to keep the tree clean. If the top-level source directory is named `mysql-src` under your current working directory, you can build in a directory named `bld` at the same level. Create the directory and go there:

```
shell> mkdir bld
shell> cd bld
```

Configure the build directory. The minimum configuration command includes no options to override configuration defaults:

```
shell> cmake ../mysql-src
```

The build directory needs not be outside the source tree. For example, you can build in a directory named `bld` under the top-level source tree. To do this, starting with `mysql-src` as your current working directory, create the directory `bld` and then go there:

```
shell> mkdir bld
shell> cd bld
```

Configure the build directory. The minimum configuration command includes no options to override configuration defaults:

```
shell> cmake ..
```

If you have multiple source trees at the same level (for example, to build multiple versions of MySQL), the second strategy can be advantageous. The first strategy places all build directories at the same level, which requires that you choose a unique name for each. With the second strategy, you can use the same name for the build directory within each source tree. The following instructions assume this second strategy.

On Windows, specify the development environment. For example, the following commands configure MySQL for 32-bit or 64-bit builds, respectively:

```
shell> cmake .. -G "Visual Studio 12 2013"
shell> cmake .. -G "Visual Studio 12 2013 Win64"
```

On macOS, to use the Xcode IDE:

```
shell> cmake .. -G Xcode
```

When you run `cmake`, you might want to add options to the command line. Here are some examples:

- `-DBUILD_CONFIG=mysql_release`: Configure the source with the same build options used by Oracle to produce binary distributions for official MySQL releases.

- `-DCMAKE_INSTALL_PREFIX=dir_name`: Configure the distribution for installation under a particular location.
- `-DCPACK_MONOLITHIC_INSTALL=1`: Cause `make package` to generate a single installation file rather than multiple files.
- `-DWITH_DEBUG=1`: Build the distribution with debugging support.

For a more extensive list of options, see Section 2.9.7, “MySQL Source-Configuration Options”.

To list the configuration options, use one of the following commands:

```
shell> cmake .. -L   # overview
shell> cmake .. -LH  # overview with help text
shell> cmake .. -LAH # all params with help text
shell> ccmake ..     # interactive display
```

If `CMake` fails, you might need to reconfigure by running it again with different options. If you do reconfigure, take note of the following:

- If `CMake` is run after it has previously been run, it may use information that was gathered during its previous invocation. This information is stored in `CMakeCache.txt`. When `CMake` starts, it looks for that file and reads its contents if it exists, on the assumption that the information is still correct. That assumption is invalid when you reconfigure.
- Each time you run `CMake`, you must run `make` again to recompile. However, you may want to remove old object files from previous builds first because they were compiled using different configuration options.

To prevent old object files or configuration information from being used, run these commands in the build direcotry on Unix before re-running `CMake`:

```
shell> make clean
shell> rm CMakeCache.txt
```

Or, on Windows:

```
shell> devenv MySQL.sln /clean
shell> del CMakeCache.txt
```

Before asking on the MySQL Community Slack, check the files in the `CMakeFiles` directory for useful information about the failure. To file a bug report, please use the instructions in Section 1.6, “How to Report Bugs or Problems”.

Build the Distribution

On Unix:

```
shell> make
shell> make VERBOSE=1
```

The second command sets `VERBOSE` to show the commands for each compiled source.

Use `gmake` instead on systems where you are using GNU `make` and it has been installed as `gmake`.

On Windows:

```
shell> devenv MySQL.sln /build RelWithDebInfo
```

If you have gotten to the compilation stage, but the distribution does not build, see Section 2.9.8, “Dealing with Problems Compiling MySQL”, for help. If that does not solve the problem, please enter it into our bugs database using the instructions given in Section 1.6, “How to Report Bugs or Problems”.

If you have installed the latest versions of the required tools, and they crash trying to process our configuration files, please report that also. However, if you get a `command not found` error or a similar problem for required tools, do not report it. Instead, make sure that all the required tools are installed and that your `PATH` variable is set correctly so that your shell can find them.

Install the Distribution

On Unix:

```
shell> make install
```

This installs the files under the configured installation directory (by default, `/usr/local/mysql`). You might need to run the command as `root`.

To install in a specific directory, add a `DESTDIR` parameter to the command line:

```
shell> make install DESTDIR="/opt/mysql"
```

Alternatively, generate installation package files that you can install where you like:

```
shell> make package
```

This operation produces one or more `.tar.gz` files that can be installed like generic binary distribution packages. See Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”. If you run `CMake` with `-DCPACK_MONOLITHIC_INSTALL=1`, the operation produces a single file. Otherwise, it produces multiple files.

On Windows, generate the data directory, then create a `.zip` archive installation package:

```
shell> devenv MySQL.sln /build RelWithDebInfo /project initial_database
shell> devenv MySQL.sln /build RelWithDebInfo /project package
```

You can install the resulting `.zip` archive where you like. See Section 2.3.4, “Installing MySQL on Microsoft Windows Using a `noinstall` ZIP Archive”.

Perform Postinstallation Setup

The remainder of the installation process involves setting up the configuration file, creating the core databases, and starting the MySQL server. For instructions, see Section 2.10, “Postinstallation Setup and Testing”.

Note

The accounts that are listed in the MySQL grant tables initially have no passwords. After starting the server, you should set up passwords for them using the instructions in Section 2.10, “Postinstallation Setup and Testing”.

2.9.5 Installing MySQL Using a Development Source Tree

This section describes how to install MySQL from the latest development source code, which is hosted on GitHub. To obtain the MySQL Server source code from this repository hosting service, you can set up a local MySQL Git repository.

On GitHub, MySQL Server and other MySQL projects are found on the MySQL page. The MySQL Server project is a single repository that contains branches for several MySQL series.

MySQL officially joined GitHub in September, 2014. For more information about MySQL's move to GitHub, refer to the announcement on the MySQL Release Engineering blog: MySQL on GitHub

- Prerequisites for Installing from Development Source

- Setting Up a MySQL Git Repository

Prerequisites for Installing from Development Source

To install MySQL from a development source tree, your system must satisfy the tool requirements listed at Section 2.9.2, “Source Installation Prerequisites”.

Setting Up a MySQL Git Repository

To set up a MySQL Git repository on your machine:

1. Clone the MySQL Git repository to your machine. The following command clones the MySQL Git repository to a directory named `mysql-server`. The initial download may take some time to complete, depending on the speed of your connection.

```
~$ git clone https://github.com/mysql/mysql-server.git
Cloning into 'mysql-server'...
remote: Counting objects: 1035465, done.
remote: Total 1035465 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (1035465/1035465), 437.48 MiB | 5.10 MiB/s, done.
Resolving deltas: 100% (855607/855607), done.
Checking connectivity... done.
Checking out files: 100% (21902/21902), done.
```

2. When the clone operation completes, the contents of your local MySQL Git repository appear similar to the following:

```
~$ cd mysql-server
~/mysql-server$ ls
client             extra               mysys             storage
cmake              include             packaging         strings
CMakeLists.txt     INSTALL             plugin            support-files
components         libbinlogevents     README            testclients
config.h.cmake     libbinlogstandalone router            unittest
configure.cmake    libmysql            run_doxygen.cmake utilities
Docs               libservices         scripts           VERSION
Doxyfile-ignored   LICENSE             share             vio
Doxyfile.in        man                 sql               win
doxygen_resources  mysql-test          sql-common
```

3. Use the `git branch -r` command to view the remote tracking branches for the MySQL repository.

```
~/mysql-server$ git branch -r
  origin/5.5
  origin/5.6
  origin/5.7
  origin/8.0
  origin/HEAD -> origin/8.0
  origin/cluster-7.2
  origin/cluster-7.3
  origin/cluster-7.4
  origin/cluster-7.5
  origin/cluster-7.6
```

4. To view the branches that are checked out in your local repository, issue the `git branch` command. When you clone the MySQL Git repository, the latest MySQL GA branch is checked out automatically. The asterisk identifies the active branch.

```
~/mysql-server$ git branch
* 8.0
```

5. To check out an earlier MySQL branch, run the `git checkout` command, specifying the branch name. For example, to check out the MySQL 5.6 branch:

```
~/mysql-server$ git checkout 5.6
Branch 5.6 set up to track remote branch 5.6 from origin.
Switched to a new branch '5.6'
```

6. To obtain changes made after your initial setup of the MySQL Git repository, switch to the branch you want to update and issue the `git pull` command:

   ```
   ~/mysql-server$ git checkout 5.6
   ~/mysql-server$ git pull
   ```

   To examine the commit history, use the `git log` option:

   ```
   ~/mysql-server$ git log
   ```

   You can also browse commit history and source code on the GitHub MySQL site.

   If you see changes or code that you have a question about, ask on the MySQL Community Slack. For information about contributing a patch, see Contributing to MySQL Server.

7. After you have cloned the MySQL Git repository and have checked out the branch you want to build, you can build MySQL Server from the source code. Instructions are provided in Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”, except that you skip the part about obtaining and unpacking the distribution.

   Be careful about installing a build from a distribution source tree on a production machine. The installation command may overwrite your live release installation. If you already have MySQL installed and do not want to overwrite it, run `CMake` with values for the `CMAKE_INSTALL_PREFIX`, `MYSQL_TCP_PORT`, and `MYSQL_UNIX_ADDR` options different from those used by your production server. For additional information about preventing multiple servers from interfering with each other, see Section 5.7, “Running Multiple MySQL Instances on One Machine”.

   Play hard with your new installation. For example, try to make new features crash. Start by running `make test`. See The MySQL Test Suite.

2.9.6 Configuring SSL Library Support

An SSL library is required for support of encrypted connections, entropy for random number generation, and other encryption-related operations. Your system must support either OpenSSL or yaSSL:

- MySQL Enterprise Edition binary distributions are compiled using OpenSSL. It is not possible to use yaSSL with MySQL Enterprise Edition.
- MySQL Community Edition binary distributions are compiled using yaSSL.
- MySQL Community Edition source distributions can be compiled using either OpenSSL or yaSSL.

Note

It is possible to compile MySQL using yaSSL as an alternative to OpenSSL only prior to MySQL 5.6.46. As of MySQL 5.6.46, support for yaSSL is removed and all MySQL builds use OpenSSL.

If you compile MySQL from a source distribution, `CMake` configures the distribution to use the installed OpenSSL library by default.

To compile using OpenSSL, use this procedure:

1. Ensure that OpenSSL 1.0.1 or higher is installed on your system. If the installed OpenSSL version is lower than 1.0.1, `CMake` produces an error at MySQL configuration time. If it is necessary to obtain OpenSSL, visit http://www.openssl.org.

2. The `WITH_SSL` `CMake` option determines which SSL library to use for compiling MySQL (see Section 2.9.7, “MySQL Source-Configuration Options”). The default is `-DWITH_SSL=system`, which uses OpenSSL. To make this explicit, specify that option on the `CMake` command line. For example:

```
cmake . -DWITH_SSL=system
```

That command configures the distribution to use the installed OpenSSL library. Alternatively, to explicitly specify the path name to the OpenSSL installation, use the following syntax. This can be useful if you have multiple versions of OpenSSL installed, to prevent `CMake` from choosing the wrong one:

```
cmake . -DWITH_SSL=path_name
```

3. Compile and install the distribution.

To check whether a `mysqld` server supports encrypted connections, examine the value of the `have_ssl` system variable:

```
mysql> SHOW VARIABLES LIKE 'have_ssl';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| have_ssl      | YES   |
+---------------+-------+
```

If the value is `YES`, the server supports encrypted connections. If the value is `DISABLED`, the server is capable of supporting encrypted connections but was not started with the appropriate `--ssl-xxx` options to enable encrypted connections to be used; see Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

To determine whether a server was compiled using OpenSSL or yaSSL, check the existence of any of the system or status variables that are present only for OpenSSL. See Section 6.3.4, “SSL Library-Dependent Capabilities”.

2.9.7 MySQL Source-Configuration Options

The `CMake` program provides a great deal of control over how you configure a MySQL source distribution. Typically, you do this using options on the `CMake` command line. For information about options supported by `CMake`, run either of these commands in the top-level source directory:

```
cmake . -LH
ccmake .
```

You can also affect `CMake` using certain environment variables. See Section 4.9, “Environment Variables”.

For boolean options, the value may be specified as 1 or `ON` to enable the option, or as 0 or `OFF` to disable the option.

Many options configure compile-time defaults that can be overridden at server startup. For example, the `CMAKE_INSTALL_PREFIX`, `MYSQL_TCP_PORT`, and `MYSQL_UNIX_ADDR` options that configure the default installation base directory location, TCP/IP port number, and Unix socket file can be changed at server startup with the `--basedir`, `--port`, and `--socket` options for `mysqld`. Where applicable, configuration option descriptions indicate the corresponding `mysqld` startup option.

The following sections provide more information about `CMake` options.

- CMake Option Reference
- General Options
- Installation Layout Options
- Storage Engine Options
- Feature Options
- Compiler Flags

- CMake Options for Compiling NDB Cluster

CMake Option Reference

The following table shows the available `CMake` options. In the `Default` column, `PREFIX` stands for the value of the `CMAKE_INSTALL_PREFIX` option, which specifies the installation base directory. This value is used as the parent location for several of the installation subdirectories.

Table 2.13 MySQL Source-Configuration Option Reference (CMake)

Formats	Description	Default	Introduced	Removed
`BUILD_CONFIG`	Use same build options as official releases			
`CMAKE_BUILD_TYPE`	Type of build to produce	`RelWithDebInfo`		
`CMAKE_CXX_FLAGS`	Flags for C++ Compiler			
`CMAKE_C_FLAGS`	Flags for C Compiler			
`CMAKE_INSTALL_PREFIX`	Installation base directory	`/usr/local/mysql`		
`COMPILATION_COMMENT`	Comment about compilation environment			
`CPACK_MONOLITHIC_INSTALL`	Whether package build produces single file	`OFF`		
`DEFAULT_CHARSET`	The default server character set	`latin1`		
`DEFAULT_COLLATION`	The default server collation	`latin1_swedish_ci`		
`ENABLED_LOCAL_INFILE`	Whether to enable LOCAL for LOAD DATA	`OFF`		
`ENABLED_PROFILING`	Whether to enable query profiling code	`ON`		
`ENABLE_DEBUG_SYNC`	Whether to enable Debug Sync support	`ON`		5.6.36
`ENABLE_DOWNLOADS`	Whether to download optional files	`OFF`		
`ENABLE_DTRACE`	Whether to include DTrace support			
`ENABLE_GCOV`	Whether to include gcov support			
`ENABLE_GPROF`	Enable gprof (optimized Linux builds only)	`OFF`		
`IGNORE_AIO_CHECK`	With -DBUILD_CONFIG=mysql_release, ignore libaio check	`OFF`		
`INNODB_PAGE_ATOMIC_REF_COUNT`	Enable or disable atomic page reference counting	`ON`	5.6.16	
`INSTALL_BINDIR`	User executables directory	`PREFIX/bin`		
`INSTALL_DOCDIR`	Documentation directory	`PREFIX/docs`		
`INSTALL_DOCREADMEDIR`	README file directory	`PREFIX`		
`INSTALL_INCLUDEDIR`	Header file directory	`PREFIX/include`		
`INSTALL_INFODIR`	Info file directory	`PREFIX/docs`		

Formats	Description	Default	Introduced	Removed
INSTALL_LAYOUT	Select predefined installation layout	STANDALONE		
INSTALL_LIBDIR	Library file directory	PREFIX/lib		
INSTALL_MANDIR	Manual page directory	PREFIX/man		
INSTALL_MYSQLSHAREDIR	Shared data directory	PREFIX/share		
INSTALL_MYSQLTESTDIR	mysql-test directory	PREFIX/mysql-test		
INSTALL_PLUGINDIR	Plugin directory	PREFIX/lib/plugin		
INSTALL_SBINDIR	Server executable directory	PREFIX/bin		
INSTALL_SCRIPTDIR	Scripts directory	PREFIX/scripts		
INSTALL_SECURE_FILE_PRIVDIR	secure_file_priv default value	platform specific	5.6.34	
INSTALL_SECURE_FILE_PRIVDIR_EMBEDDED	secure_file_priv default value for libmysqld		5.6.34	
INSTALL_SHAREDIR	aclocal/mysql.m4 installation directory	PREFIX/share		
INSTALL_SQLBENCHDIR	sql-bench directory	PREFIX		
INSTALL_SUPPORTFILESDIR	Extra support files directory	PREFIX/support-files		
MEMCACHED_HOME	Path to memcached	[none]		
MYSQL_DATADIR	Data directory			
MYSQL_MAINTAINER_MODE	Whether to enable MySQL maintainer-specific development environment	OFF		
MYSQL_PROJECT_NAME	Windows/OS X project name	MySQL		
MYSQL_TCP_PORT	TCP/IP port number	3306		
MYSQL_UNIX_ADDR	Unix socket file	/tmp/mysql.sock		
ODBC_INCLUDES	ODBC includes directory			
ODBC_LIB_DIR	ODBC library directory			
OPTIMIZER_TRACE	Whether to support optimizer tracing			
REPRODUCIBLE_BUILD	Take extra care to create a build result independent of build location and time		5.6.37	
SUNPRO_CXX_LIBRARY	Client link library on Solaris 10+		5.6.20	
SYSCONFDIR	Option file directory			
TMPDIR	tmpdir default value		5.6.16	
WITHOUT_xxx_STORAGE_ENGINE	Exclude storage engine xxx from build			
WITH_ASAN	Enable AddressSanitizer	OFF	5.6.15	
WITH_BUNDLED_LIBEVENT	Use bundled libevent when building ndbmemcache	ON		

Formats	Description	Default	Introduced	Removed
WITH_BUNDLED_MEMCACHED	Use bundled memcached when building ndbmemcache	ON		
WITH_CLASSPATH	Classpath to use when building MySQL Cluster Connector for Java. Default is an empty string.			
WITH_DEBUG	Whether to include debugging support	OFF		
WITH_DEFAULT_COMPILER_OPTIONS	Whether to use default compiler options	ON		
WITH_DEFAULT_FEATURE_SET	Whether to use default feature set	ON		
WITH_EDITLINE	Which libedit/editline library to use	bundled	5.6.12	
WITH_EMBEDDED_SERVER	Whether to build embedded server	OFF		
WITH_EMBEDDED_SHARED_LIBRARY	Whether to build a shared embedded server library	OFF	5.6.17	
WITH_ERROR_INSERT	Enable error injection in the NDB storage engine. Should not be used for building binaries intended for production.	OFF		
WITH_EXTRA_CHARSETS	Which extra character sets to include	all		
WITH_GMOCK	Path to googlemock distribution			
WITH_INNODB_MEMCACHED	Whether to generate memcached shared libraries.	OFF		
WITH_LIBEDIT	Use bundled libedit library	ON		5.6.12
WITH_LIBEVENT	Which libevent library to use	bundled		
WITH_LIBWRAP	Whether to include libwrap (TCP wrappers) support	OFF		
WITH_NDBCLUSTER	Build the NDB storage engine; alias for WITH_NDBCLUSTER_STORAGE_ENGINE	ON		
WITH_NDBCLUSTER_STORAGE_ENGINE	Build the NDB storage engine	ON		
WITH_NDBMTD	Build multithreaded data node.	ON		
WITH_NDB_BINLOG	Enable binary logging by default by mysqld.	ON		
WITH_NDB_DEBUG	Produce a debug build for testing or troubleshooting.	OFF		
WITH_NDB_JAVA	Enable building of Java and ClusterJ support. Enabled by default. Supported in MySQL Cluster only.	ON		

Formats	Description	Default	Introduced	Removed
WITH_NDB_PORT	Default port used by a management server built with this option. If this option was not used to build it, the management server's default port is 1186.	[none]		
WITH_NDB_TEST	Include NDB API test programs.	OFF		
WITH_NUMA	Set NUMA memory allocation policy		5.6.27	
WITH_SSL	Type of SSL support	system		
WITH_SYMVER16	Whether libmysqlclient.so.18 contains both symver 16 and 18 symbols.	OFF	5.6.31	
WITH_UNIT_TESTS	Compile MySQL with unit tests	ON		
WITH_UNIXODBC	Enable unixODBC support	OFF		
WITH_VALGRIND	Whether to compile in Valgrind header files	OFF		
WITH_ZLIB	Type of zlib support	bundled		
WITH_xxx_STORAGE_ENGINE	Compile storage engine xxx statically into server			

General Options

- -DBUILD_CONFIG=mysql_release

  This option configures a source distribution with the same build options used by Oracle to produce binary distributions for official MySQL releases.

- -DCMAKE_BUILD_TYPE=*type*

  The type of build to produce:

  - RelWithDebInfo: Enable optimizations and generate debugging information. This is the default MySQL build type.
  - Debug: Disable optimizations and generate debugging information. This build type is also used if the WITH_DEBUG option is enabled. That is, -DWITH_DEBUG=1 has the same effect as -DCMAKE_BUILD_TYPE=Debug.

- -DCPACK_MONOLITHIC_INSTALL=*bool*

  This option affects whether the make package operation produces multiple installation package files or a single file. If disabled, the operation produces multiple installation package files, which may be useful if you want to install only a subset of a full MySQL installation. If enabled, it produces a single file for installing everything.

Installation Layout Options

The CMAKE_INSTALL_PREFIX option indicates the base installation directory. Other options with names of the form INSTALL_*xxx* that indicate component locations are interpreted relative to the prefix and their values are relative pathnames. Their values should not include the prefix.

- -DCMAKE_INSTALL_PREFIX=*dir_name*

The installation base directory.

This value can be set at server startup with the `--basedir` option.

- `-DINSTALL_BINDIR=dir_name`

  Where to install user programs.

- `-DINSTALL_DOCDIR=dir_name`

  Where to install documentation.

- `-DINSTALL_DOCREADMEDIR=dir_name`

  Where to install `README` files.

- `-DINSTALL_INCLUDEDIR=dir_name`

  Where to install header files.

- `-DINSTALL_INFODIR=dir_name`

  Where to install Info files.

- `-DINSTALL_LAYOUT=name`

  Select a predefined installation layout:

  - `STANDALONE`: Same layout as used for `.tar.gz` and `.zip` packages. This is the default.
  - `RPM`: Layout similar to RPM packages.
  - `SVR4`: Solaris package layout.
  - `DEB`: DEB package layout (experimental).

  You can select a predefined layout but modify individual component installation locations by specifying other options. For example:

  ```
  cmake . -DINSTALL_LAYOUT=SVR4 -DMYSQL_DATADIR=/var/mysql/data
  ```

- `-DINSTALL_LIBDIR=dir_name`

  Where to install library files.

- `-DINSTALL_MANDIR=dir_name`

  Where to install manual pages.

- `-DINSTALL_MYSQLSHAREDIR=dir_name`

  Where to install shared data files.

- `-DINSTALL_MYSQLTESTDIR=dir_name`

  Where to install the `mysql-test` directory. As of MySQL 5.6.12, to suppress installation of this directory, explicitly set the option to the empty value (`-DINSTALL_MYSQLTESTDIR=`).

- `-DINSTALL_PLUGINDIR=dir_name`

  The location of the plugin directory.

  This value can be set at server startup with the `--plugin_dir` option.

- `-DINSTALL_SBINDIR=dir_name`

  Where to install the `mysqld` server.

- `-DINSTALL_SCRIPTDIR=dir_name`

  Where to install `mysql_install_db`.

- `-DINSTALL_SECURE_FILE_PRIVDIR=dir_name`

  The default value for the `secure_file_priv` system variable. The default value is platform specific and depends on the value of the `INSTALL_LAYOUT` `CMake` option; see the description of the `secure_file_priv` system variable in Section 5.1.7, “Server System Variables”.

  This option was added in MySQL 5.6.34. To set the value for the `libmysqld` embedded server, use `INSTALL_SECURE_FILE_PRIV_EMBEDDEDDIR`.

- `-DINSTALL_SECURE_FILE_PRIV_EMBEDDEDDIR=dir_name`

  The default value for the `secure_file_priv` system variable, for the `libmysqld` embedded server. This option was added in MySQL 5.6.34.

- `-DINSTALL_SHAREDIR=dir_name`

  Where to install `aclocal/mysql.m4`.

- `-DINSTALL_SQLBENCHDIR=dir_name`

  Where to install the `sql-bench` directory. To suppress installation of this directory, explicitly set the option to the empty value (`-DINSTALL_SQLBENCHDIR=`).

- `-DINSTALL_SUPPORTFILESDIR=dir_name`

  Where to install extra support files.

- `-DMYSQL_DATADIR=dir_name`

  The location of the MySQL data directory.

  This value can be set at server startup with the `--datadir` option.

- `-DODBC_INCLUDES=dir_name`

  The location of the ODBC includes directory, and may be used while configuring Connector/ODBC.

- `-DODBC_LIB_DIR=dir_name`

  The location of the ODBC library directory, and may be used while configuring Connector/ODBC.

- `-DSYSCONFDIR=dir_name`

  The default `my.cnf` option file directory.

  This location cannot be set at server startup, but you can start the server with a given option file using the `--defaults-file=file_name` option, where *`file_name`* is the full path name to the file.

- `-DTMPDIR=dir_name`

  The default location to use for the `tmpdir` system variable. If unspecified, the value defaults to `P_tmpdir` in `<stdio.h>`. This option was added in MySQL 5.6.16.

Storage Engine Options

Storage engines are built as plugins. You can build a plugin as a static module (compiled into the server) or a dynamic module (built as a dynamic library that must be installed into the server using the `INSTALL PLUGIN` statement or the `--plugin-load` option before it can be used). Some plugins might not support static or dynamic building.

The `InnoDB`, `MyISAM`, `MERGE`, `MEMORY`, and `CSV` engines are mandatory (always compiled into the server) and need not be installed explicitly.

To compile a storage engine statically into the server, use `-DWITH_`*`engine`*`_STORAGE_ENGINE=1`. Some permissible *`engine`* values are `ARCHIVE`, `BLACKHOLE`, `EXAMPLE`, `FEDERATED`, `NDB` or `NDBCLUSTER` (`NDB`), `PARTITION` (partitioning support), and `PERFSCHEMA` (Performance Schema). Examples:

```
-DWITH_ARCHIVE_STORAGE_ENGINE=1
-DWITH_BLACKHOLE_STORAGE_ENGINE=1
-DWITH_PERFSCHEMA_STORAGE_ENGINE=1
```

Note

`WITH_NDBCLUSTER_STORAGE_ENGINE` is supported only when building NDB Cluster using the NDB Cluster sources. It cannot be used to enable clustering support in other MySQL source trees or distributions. In NDB Cluster source distributions, it is enabled by default. See Section 18.2.1.4, “Building NDB Cluster from Source on Linux”, and Section 18.2.2.2, “Compiling and Installing NDB Cluster from Source on Windows”, for more information.

To exclude a storage engine from the build, use `-DWITHOUT_`*`engine`*`_STORAGE_ENGINE=1`. Examples:

```
-DWITHOUT_EXAMPLE_STORAGE_ENGINE=1
-DWITHOUT_FEDERATED_STORAGE_ENGINE=1
-DWITHOUT_PARTITION_STORAGE_ENGINE=1
```

If neither `-DWITH_`*`engine`*`_STORAGE_ENGINE` nor `-DWITHOUT_`*`engine`*`_STORAGE_ENGINE` are specified for a given storage engine, the engine is built as a shared module, or excluded if it cannot be built as a shared module.

Feature Options

- `-DCOMPILATION_COMMENT=`*`string`*

  A descriptive comment about the compilation environment.

- `-DDEFAULT_CHARSET=`*`charset_name`*

  The server character set. By default, MySQL uses the `latin1` (cp1252 West European) character set.

  `charset_name` may be one of `binary`, `armscii8`, `ascii`, `big5`, `cp1250`, `cp1251`, `cp1256`, `cp1257`, `cp850`, `cp852`, `cp866`, `cp932`, `dec8`, `eucjpms`, `euckr`, `gb2312`, `gbk`, `geostd8`, `greek`, `hebrew`, `hp8`, `keybcs2`, `koi8r`, `koi8u`, `latin1`, `latin2`, `latin5`, `latin7`, `macce`, `macroman`, `sjis`, `swe7`, `tis620`, `ucs2`, `ujis`, `utf8`, `utf8mb4`, `utf16`, `utf16le`, `utf32`. The permissible character sets are listed in the `cmake/character_sets.cmake` file as the value of `CHARSETS_AVAILABLE`.

  This value can be set at server startup with the `--character_set_server` option.

- `-DDEFAULT_COLLATION=`*`collation_name`*

  The server collation. By default, MySQL uses `latin1_swedish_ci`. Use the `SHOW COLLATION` statement to determine which collations are available for each character set.

This value can be set at server startup with the `--collation_server` option.

- `-DENABLE_DEBUG_SYNC=bool`

  Note

  As of MySQL 5.6.36, `ENABLE_DEBUG_SYNC` is removed and enabling `WITH_DEBUG` enables Debug Sync.

  Whether to compile the Debug Sync facility into the server. This facility is used for testing and debugging. This option is enabled by default, but has no effect unless MySQL is configured with debugging enabled. If debugging is enabled and you want to disable Debug Sync, use `-DENABLE_DEBUG_SYNC=0`.

  When compiled in, Debug Sync is disabled by default at runtime. To enable it, start `mysqld` with the `--debug-sync-timeout=N` option, where *`N`* is a timeout value greater than 0. (The default value is 0, which disables Debug Sync.) *`N`* becomes the default timeout for individual synchronization points.

  For a description of the Debug Sync facility and how to use synchronization points, see MySQL Internals: Test Synchronization.

- `-DENABLE_DOWNLOADS=bool`

  Whether to download optional files. For example, with this option enabled, `CMake` downloads the Google Test distribution that is used by the test suite to run unit tests.

- `-DENABLE_DTRACE=bool`

  Whether to include support for DTrace probes. For information about DTrace, wee Section 5.8.4, “Tracing mysqld Using DTrace”

- `-DENABLE_GCOV=bool`

  Whether to include gcov support (Linux only).

- `-DENABLE_GPROF=bool`

  Whether to enable `gprof` (optimized Linux builds only).

- `-DENABLED_LOCAL_INFILE=bool`

  This option controls the compiled-in default `LOCAL` capability for the MySQL client library. Clients that make no explicit arrangements therefore have `LOCAL` capability disabled or enabled according to the `ENABLED_LOCAL_INFILE` setting specified at MySQL build time.

  By default, the client library in MySQL binary distributions is compiled with `ENABLED_LOCAL_INFILE` enabled. If you compile MySQL from source, configure it with `ENABLED_LOCAL_INFILE` disabled or enabled based on whether clients that make no explicit arrangements should have `LOCAL` capability disabled or enabled, respectively.

  `ENABLED_LOCAL_INFILE` controls the default for client-side `LOCAL` capability. For the server, the `local_infile` system variable controls server-side `LOCAL` capability. To explicitly cause the server to refuse or permit `LOAD DATA LOCAL` statements (regardless of how client programs and libraries are configured at build time or runtime), start `mysqld` with `local_infile` disabled or enabled, respectively. `local_infile` can also be set at runtime. See Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”.

- `-DENABLED_PROFILING=bool`

  Whether to enable query profiling code (for the `SHOW PROFILE` and `SHOW PROFILES` statements).

- `-DIGNORE_AIO_CHECK=bool`

If the `-DBUILD_CONFIG=mysql_release` option is given on Linux, the `libaio` library must be linked in by default. If you do not have `libaio` or do not want to install it, you can suppress the check for it by specifying `-DIGNORE_AIO_CHECK=1`.

- `-DINNODB_PAGE_ATOMIC_REF_COUNT=bool`

  Whether to enable or disable atomic page reference counting. Fetching and releasing pages from the buffer pool and tracking the page state are expensive and complex operations. Using a page mutex to track these operations does not scale well. With `INNODB_PAGE_ATOMIC_REF_COUNT=ON` (default), fetch and release is tracked using atomics where available. For platforms that do not support atomics, set `INNODB_PAGE_ATOMIC_REF_COUNT=OFF` to disable atomic page reference counting.

  When atomic page reference counting is enabled (default), "`[Note] InnoDB: Using atomics to ref count buffer pool pages`" is printed to the error log at server startup. If atomic page reference counting is disabled, "`[Note] InnoDB: Using mutexes to ref count buffer pool pages`" is printed instead.

  `INNODB_PAGE_ATOMIC_REF_COUNT` was introduced with the fix for MySQL Bug #68079. The option is removed in MySQL 5.7.5. Support for atomics is required to build MySQL as of MySQL 5.7.5, which makes the option obsolete.

- `-DMYSQL_MAINTAINER_MODE=bool`

  Whether to enable a MySQL maintainer-specific development environment. If enabled, this option causes compiler warnings to become errors. It may also cause some minor changes in generated code, to initialize some variables to 0.

- `-DMYSQL_PROJECT_NAME=name`

  For Windows or macOS, the project name to incorporate into the project file name.

- `-DMYSQL_TCP_PORT=port_num`

  The port number on which the server listens for TCP/IP connections. The default is 3306.

  This value can be set at server startup with the `--port` option.

- `-DMYSQL_UNIX_ADDR=file_name`

  The Unix socket file path on which the server listens for socket connections. This must be an absolute path name. The default is `/tmp/mysql.sock`.

  This value can be set at server startup with the `--socket` option.

- `-DOPTIMIZER_TRACE=bool`

  Whether to support optimizer tracing. See MySQL Internals: Tracing the Optimizer.

- `-DREPRODUCIBLE_BUILD=bool`

  For builds on Linux systems, this option controls whether to take extra care to create a build result independent of build location and time.

  This option was added in MySQL 5.6.37.

- `-DWITH_ASAN=bool`

  Whether to enable AddressSanitizer, for compilers that support it. The default is off. This option was added in MySQL 5.6.15.

- `-DWITH_DEBUG=bool`

Whether to include debugging support.

Configuring MySQL with debugging support enables you to use the `--debug="d,parser_debug"` option when you start the server. This causes the Bison parser that is used to process SQL statements to dump a parser trace to the server's standard error output. Typically, this output is written to the error log.

As of MySQL 5.6.36, enabling `WITH_DEBUG` also enables Debug Sync. For a description of the Debug Sync facility and how to use synchronization points, see MySQL Internals: Test Synchronization.

- `-DWITH_DEFAULT_FEATURE_SET=bool`

  Whether to use the flags from `cmake/build_configurations/feature_set.cmake`.

- `-DWITH_EDITLINE=value`

  Which `libedit`/`editline` library to use. The permitted values are `bundled` (the default) and `system`.

  `WITH_EDITLINE` was added in MySQL 5.6.12. It replaces `WITH_LIBEDIT`, which has been removed.

- `-DWITH_EMBEDDED_SERVER=bool`

  Whether to build the `libmysqld` embedded server library.

- `-DWITH_EMBEDDED_SHARED_LIBRARY=bool`

  Whether to build a shared `libmysqld` embedded server library. This option was added in MySQL 5.6.17.

- `-DWITH_EXTRA_CHARSETS=name`

  Which extra character sets to include:

  - `all`: All character sets. This is the default.
  - `complex`: Complex character sets.
  - `none`: No extra character sets.

- `-DWITH_GMOCK=path_name`

  The path to the googlemock distribution, for use with Google Test-based unit tests. The option value is the path to the distribution Zip file. Alternatively, set the `WITH_GMOCK` environment variable to the path name. It is also possible to use `-DENABLE_DOWNLOADS=1` so that `CMake` downloads the distribution from GitHub.

  If you build MySQL without the Google Test-based unit tests (by configuring wihout `WITH_GMOCK`), `CMake` displays a message indicating how to download it.

- `-DWITH_INNODB_MEMCACHED=bool`

  Whether to generate memcached shared libraries (`libmemcached.so` and `innodb_engine.so`).

- `-DWITH_LIBEVENT=string`

  Which `libevent` library to use. Permitted values are `bundled` (default), `system`, and `yes`. If you specify `system` or `yes`, the system `libevent` library is used if present. If the system library is not found, the bundled `libevent` library is used. The `libevent` library is required by `InnoDB` memcached.

- `-DWITH_LIBEDIT=bool`

  Whether to use the `libedit` library bundled with the distribution.

  `WITH_LIBEDIT` was removed in MySQL 5.6.12. Use `WITH_EDITLINE` instead.

- `-DWITH_LIBWRAP=bool`

  Whether to include `libwrap` (TCP wrappers) support.

- `-DWITH_NUMA=bool`

  Explicitly set the NUMA memory allocation policy. `CMake` sets the default `WITH_NUMA` value based on whether the current platform has `NUMA` support. For platforms without NUMA support, `CMake` behaves as follows:

  - With no NUMA option (the normal case), `CMake` continues normally, producing only this warning: NUMA library missing or required version not available
  - With `-DWITH_NUMA=ON`, `CMake` aborts with this error: NUMA library missing or required version not available

  This option was added in MySQL 5.6.27.

- `-DWITH_SSL={ssl_type|path_name}`

  For support of encrypted connections, entropy for random number generation, and other encryption-related operations, MySQL must be built using an SSL library. This option specifies which SSL library to use.

  - *`ssl_type`* can be one of the following values:
    - `no`: No SSL support. This is the default before MySQL 5.6.6. As of 5.6.6, this is no longer a permitted value and the default is `bundled`.
    - `yes`: Use the system OpenSSL library if present, else the library bundled with the distribution.
    - `bundled`: Use the SSL library bundled with the distribution. This is the default from MySQL 5.6.6 through 5.6.45. As of 5.6.46, this is no longer a permitted value and the default is `system`.
    - `system`: Use the system OpenSSL library. This is the default as of MySQL 5.6.46.
  - *`path_name`*, permitted for MySQL 5.6.7 and after, is the path name to the OpenSSL installation to use. This can be preferable to using the *`ssl_type`* value of `system` because it can prevent CMake from detecting and using an older or incorrect OpenSSL version installed on the system. (Another permitted way to do the same thing is to set `WITH_SSL` to `system` and set the `CMAKE_PREFIX_PATH` option to *`path_name`*.)

  For additional information about configuring the SSL library, see Section 2.9.6, “Configuring SSL Library Support”.

- `-DWITH_SYMVER16=bool`

  If enabled, this option causes the `libmysqlclient` client library to contain extra symbols to be compatible with `libmysqlclient` on RHEL/OEL 5, 6, and 7; and Fedora releases. All symbols present in `libmysqlclient.so.16` are tagged with symver 16 in `libmsqlclient.so.18`, making those symbols have both symver 16 and 18. The default is `OFF`.

  This option was added in MySQL 5.6.31.

- `-DWITH_UNIT_TESTS={ON|OFF}`

  If enabled, compile MySQL with unit tests. The default is ON unless the server is not being compiled.

- `-DWITH_UNIXODBC=1`

  Enables unixODBC support, for Connector/ODBC.

- `-DWITH_VALGRIND=bool`

  Whether to compile in the Valgrind header files, which exposes the Valgrind API to MySQL code. The default is `OFF`.

  To generate a Valgrind-aware debug build, `-DWITH_VALGRIND=1` normally is combined with `-DWITH_DEBUG=1`. See Building Debug Configurations.

- `-DWITH_ZLIB=zlib_type`

  Some features require that the server be built with compression library support, such as the `COMPRESS()` and `UNCOMPRESS()` functions, and compression of the client/server protocol. The `WITH_ZLIB` indicates the source of `zlib` support:

  - `bundled`: Use the `zlib` library bundled with the distribution. This is the default.
  - `system`: Use the system `zlib` library.

Compiler Flags

- `-DCMAKE_C_FLAGS="flags"`

  Flags for the C Compiler.

- `-DCMAKE_CXX_FLAGS="flags"`

  Flags for the C++ Compiler.

- `-DWITH_DEFAULT_COMPILER_OPTIONS=bool`

  Whether to use the flags from `cmake/build_configurations/compiler_options.cmake`.

  Note

  All optimization flags were carefully chosen and tested by the MySQL build team. Overriding them can lead to unexpected results and is done at your own risk.

- `-DSUNPRO_CXX_LIBRARY="lib_name"`

  Enable linking against `libCstd` instead of `stlport4` on Solaris 10 or later. This works only for client code because the server depends on C++98.

  This option was added in MySQL 5.6.20.

To specify your own C and C++ compiler flags, for flags that do not affect optimization, use the `CMAKE_C_FLAGS` and `CMAKE_CXX_FLAGS` CMake options.

When providing your own compiler flags, you might want to specify `CMAKE_BUILD_TYPE` as well.

For example, to create a 32-bit release build on a 64-bit Linux machine, do this:

```
mkdir bld
cd bld
cmake .. -DCMAKE_C_FLAGS=-m32 \
  -DCMAKE_CXX_FLAGS=-m32 \
  -DCMAKE_BUILD_TYPE=RelWithDebInfo
```

If you set flags that affect optimization (`-O`*`number`*), you must set the `CMAKE_C_FLAGS_`*`build_type`* and/or `CMAKE_CXX_FLAGS_`*`build_type`* options, where *`build_type`* corresponds to the `CMAKE_BUILD_TYPE` value. To specify a different optimization for the default build type (`RelWithDebInfo`) set the `CMAKE_C_FLAGS_RELWITHDEBINFO` and `CMAKE_CXX_FLAGS_RELWITHDEBINFO` options. For example, to compile on Linux with `-O3` and with debug symbols, do this:

```
cmake .. -DCMAKE_C_FLAGS_RELWITHDEBINFO="-O3 -g" \
  -DCMAKE_CXX_FLAGS_RELWITHDEBINFO="-O3 -g"
```

CMake Options for Compiling NDB Cluster

The following options are for use when building NDB Cluster with the NDB Cluster sources; they are not currently supported when using sources from the MySQL 5.6 Server tree.

- `-DMEMCACHED_HOME=`*`dir_name`*

  Perform the build using the memcached (version 1.6 or later) installed in the system directory indicated by *`dir_name`*. Files from this installation that are used in the build include the memcached binary, header files, and libraries, as well as the `memcached_utilities` library and the header file `engine_testapp.h`.

  You must leave this option unset when building `ndbmemcache` using the bundled memcached sources (`WITH_BUNDLED_MEMCACHED` option); in other words, the bundled sources are used by default).

  This option was added in MySQL NDB Cluster 7.2.2.

  While additional `CMake` options—such as for SASL authorization and for providing `dtrace` support—are available for use when compiling `memcached` from external sources, these options are currently not enabled for the `memcached` sources bundled with NDB Cluster.

- `-DWITH_BUNDLED_LIBEVENT={ON|OFF}`

  Use the `libevent` included in the NDB Cluster sources when building NDB Cluster with ndbmemcached support (MySQL NDB Cluster 7.2.2 and later). Enabled by default. OFF causes the system's libevent to be used instead.

- `-DWITH_BUNDLED_MEMCACHED={ON|OFF}`

  Build the memcached sources included in the NDB Cluster source tree (MySQL NDB Cluster 7.2.3 and later), then use the resulting memcached server when building the ndbmemcache engine. In this case, `make install` places the `memcached` binary in the installation `bin` directory, and the ndbmemcache engine shared library file `ndb_engine.so` in the installation `lib` directory.

  This option is ON by default.

- `-DWITH_CLASSPATH=`*`path`*

  Sets the classpath for building NDB Cluster Connector for Java. The default is empty. In MySQL NDB Cluster 7.2.9 and later, this option is ignored if `-DWITH_NDB_JAVA=OFF` is used.

- `-DWITH_ERROR_INSERT={ON|OFF}`

  Enables error injection in the `NDB` kernel. For testing only; not intended for use in building production binaries. The default is `OFF`.

- `-DWITH_NDBCLUSTER_STORAGE_ENGINE={ON|OFF}`

  Build and link in support for the `NDB` (`NDBCLUSTER`) storage engine in `mysqld`. The default is `ON`.

- `-DWITH_NDBCLUSTER={ON|OFF}`

This is an alias for `WITH_NDBCLUSTER_STORAGE_ENGINE`.

- `-DWITH_NDBMTD={ON|OFF}`

  Build the multithreaded data node executable `ndbmtd`. The default is `ON`.

- `-DWITH_NDB_BINLOG={ON|OFF}`

  Enable binary logging by default in the `mysqld` built using this option. ON by default.

- `-DWITH_NDB_DEBUG={ON|OFF}`

  Enable building the debug versions of the NDB Cluster binaries. OFF by default.

- `-DWITH_NDB_JAVA={ON|OFF}`

  Enable building NDB Cluster with Java support, including `ClusterJ`.

  This option was added in MySQL NDB Cluster 7.2.9, and is ON by default. If you do not wish to compile NDB Cluster with Java support, you must disable it explicitly by specifying `-DWITH_NDB_JAVA=OFF` when running `CMake`. Otherwise, if Java cannot be found, configuration of the build fails.

- `-DWITH_NDB_PORT=`*`port`*

  Causes the NDB Cluster management server (`ndb_mgmd`) that is built to use this *`port`* by default. If this option is unset, the resulting management server tries to use port 1186 by default.

- `-DWITH_NDB_TEST={ON|OFF}`

  If enabled, include a set of NDB API test programs. The default is OFF.

2.9.8 Dealing with Problems Compiling MySQL

The solution to many problems involves reconfiguring. If you do reconfigure, take note of the following:

- If `CMake` is run after it has previously been run, it may use information that was gathered during its previous invocation. This information is stored in `CMakeCache.txt`. When `CMake` starts, it looks for that file and reads its contents if it exists, on the assumption that the information is still correct. That assumption is invalid when you reconfigure.
- Each time you run `CMake`, you must run `make` again to recompile. However, you may want to remove old object files from previous builds first because they were compiled using different configuration options.

To prevent old object files or configuration information from being used, run the following commands before re-running `CMake`:

On Unix:

```
shell> make clean
shell> rm CMakeCache.txt
```

On Windows:

```
shell> devenv MySQL.sln /clean
shell> del CMakeCache.txt
```

If you build outside of the source tree, remove and recreate your build directory before re-running `CMake`. For instructions on building outside of the source tree, see How to Build MySQL Server with CMake.

On some systems, warnings may occur due to differences in system include files. The following list describes other problems that have been found to occur most often when compiling MySQL:

- To define which C and C++ compilers to use, you can define the `CC` and `CXX` environment variables. For example:

  ```
  shell> CC=gcc
  shell> CXX=g++
  shell> export CC CXX
  ```

  To specify your own C and C++ compiler flags, use the `CMAKE_C_FLAGS` and `CMAKE_CXX_FLAGS` CMake options. See Compiler Flags.

  To see what flags you might need to specify, invoke `mysql_config` with the `--cflags` and `--cxxflags` options.

- To see what commands are executed during the compile stage, after using `CMake` to configure MySQL, run `make VERBOSE=1` rather than just `make`.

- If compilation fails, check whether the `MYSQL_MAINTAINER_MODE` option is enabled. This mode causes compiler warnings to become errors, so disabling it may enable compilation to proceed.

- If your compile fails with errors such as any of the following, you must upgrade your version of `make` to GNU `make`:

  ```
  make: Fatal error in reader: Makefile, line 18:
  Badly formed macro assignment
  ```

  Or:

  ```
  make: file `Makefile' line 18: Must be a separator (:
  ```

  Or:

  ```
  pthread.h: No such file or directory
  ```

  Solaris and FreeBSD are known to have troublesome `make` programs.

  GNU `make` 3.75 is known to work.

- The `sql_yacc.cc` file is generated from `sql_yacc.yy`. Normally, the build process does not need to create `sql_yacc.cc` because MySQL comes with a pregenerated copy. However, if you do need to re-create it, you might encounter this error:

  ```
  "sql_yacc.yy", line xxx fatal: default action causes potential...
  ```

  This is a sign that your version of `yacc` is deficient. You probably need to install a recent version of `bison` (the GNU version of `yacc`) and use that instead.

  Versions of `bison` older than 1.75 may report this error:

  ```
  sql_yacc.yy:#####: fatal error: maximum table size (32767) exceeded
  ```

  The maximum table size is not actually exceeded; the error is caused by bugs in older versions of `bison`.

For information about acquiring or updating tools, see the system requirements in Section 2.9, “Installing MySQL from Source”.

2.9.9 MySQL Configuration and Third-Party Tools

Third-party tools that need to determine the MySQL version from the MySQL source can read the `VERSION` file in the top-level source directory. The file lists the pieces of the version separately. For example, if the version is MySQL 5.6.4-m7, the file looks like this:

```
MYSQL_VERSION_MAJOR=5
```

```
MYSQL_VERSION_MINOR=6
MYSQL_VERSION_PATCH=4
MYSQL_VERSION_EXTRA=-m7
```

If the source is not for a General Availablility (GA) release, the `MYSQL_VERSION_EXTRA` value is nonempty. In the preceding example, the value shown corresponds to Milestone 7.

`MYSQL_VERSION_EXTRA` is also nonempty for NDB Cluster releases (including GA releases of NDB Cluster), as shown here:

```
MYSQL_VERSION_MAJOR=5
MYSQL_VERSION_MINOR=6
MYSQL_VERSION_PATCH=50
MYSQL_VERSION_EXTRA=-ndb-7.4.31
```

To construct a five-digit number from the version components, use this formula:

```
MYSQL_VERSION_MAJOR*10000 + MYSQL_VERSION_MINOR*100 + MYSQL_VERSION_PATCH
```

2.10 Postinstallation Setup and Testing

This section discusses tasks that you should perform after installing MySQL:

- If necessary, initialize the data directory and create the MySQL grant tables. For some MySQL installation methods, data directory initialization may be done for you automatically:
  - Installation on Windows
  - Installation on Linux using a server RPM or Debian distribution from Oracle.
  - Installation using the native packaging system on many platforms, including Debian Linux, Ubuntu Linux, Gentoo Linux, and others.
  - Installation on macOS using a DMG distribution.

  For other platforms and installation types, you must initialize the data directory manually. These include installation from generic binary and source distributions on Unix and Unix-like system, and installation from a ZIP Archive package on Windows. For instructions, see Section 2.10.1, “Initializing the Data Directory”.
- For instructions, see Section 2.10.2, “Starting the Server”, and Section 2.10.3, “Testing the Server”.
- Assign passwords to any initial accounts in the grant tables, if that was not already done during data directory initialization. Passwords prevent unauthorized access to the MySQL server. You may also wish to restrict access to test databases. For instructions, see Section 2.10.4, “Securing the Initial MySQL Accounts”.
- Optionally, arrange for the server to start and stop automatically when your system starts and stops. For instructions, see Section 2.10.5, “Starting and Stopping MySQL Automatically”.
- Optionally, populate time zone tables to enable recognition of named time zones. For instructions, see Section 5.1.13, “MySQL Server Time Zone Support”.

When you are ready to create additional user accounts, you can find information on the MySQL access control system and account management in Section 6.2, “Access Control and Account Management”.

2.10.1 Initializing the Data Directory

After MySQL is installed, the data directory must be initialized, including the tables in the `mysql` system database:

- For some MySQL installation methods, data directory initialization is automatic, as described in Section 2.10, “Postinstallation Setup and Testing”.

- For other installation methods, you must initialize the data directory manually. These include installation from generic binary and source distributions on Unix and Unix-like systems, and installation from a ZIP Archive package on Windows.

This section describes how to initialize the data directory manually for MySQL installation methods for which data directory initialization is not automatic. For some suggested commands that enable testing whether the server is accessible and working properly, see Section 2.10.3, “Testing the Server”.

In the examples shown here, the server is intended to run under the user ID of the `mysql` login account. This assumes that such an account exists. Either create the account if it does not exist (see Create a mysql User and Group), or substitute the name of a different existing login account that you plan to use for running the server.

1. Change location to the top-level directory of your MySQL installation, which is typically `/usr/local/mysql` (adjust the path name for your system as necessary):

   ```
   cd /usr/local/mysql
   ```

   You can find several files and subdirectories inside the directory, including the `bin` and `scripts` subdirectories, which contain the server as well as client and utility programs.

2. Initialize the data directory, including the `mysql` database containing the initial MySQL grant tables that determine how users are permitted to connect to the server. For example:

   ```
   scripts/mysql_install_db --user=mysql
   ```

   Typically, data directory initialization need be done only after you first install MySQL. (For upgrades to an existing installation, perform the upgrade procedure instead; see Section 2.11, “Upgrading MySQL”.) However, the command that initializes the data directory does not overwrite any existing privilege tables, so it is safe to run in any circumstances.

   It is important to make sure that the database directories and files are owned by the `mysql` login account so that the server has read and write access to them when you run it later. To ensure this if you run `mysql_install_db` as `root`, include the `--user` option as shown.

   The `mysql_install_db` command initializes the server's data directory. Under the data directory, it creates directories for the `mysql` database that holds the grant tables and the `test` database that you can use to test MySQL. The program also creates privilege table entries for the initial account or accounts. `test_`. For a complete listing and description of the grant tables, see Section 6.2, “Access Control and Account Management”.

   It might be necessary to specify other options such as `--basedir` or `--datadir` if `mysql_install_db` cannot identify the correct locations for the installation directory or data directory. For example (enter the command on a single line):

   ```
   scripts/mysql_install_db --user=mysql
     --basedir=/opt/mysql/mysql
     --datadir=/opt/mysql/mysql/data
   ```

   For a more secure installation, invoke `mysql_install_db` with the `--random-passwords` option. This causes it to assign a random password to the MySQL `root` accounts, set the “password expired” flag for those accounts, and remove the anonymous-user MySQL accounts. For additional details, see Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”. (Install operations using RPMs for Unbreakable Linux Network are unaffected because they do not use `mysql_install_db`.)

   If you do not want to have the `test` database, you can remove it after starting the server, using the instructions in Section 2.10.4, “Securing the Initial MySQL Accounts”.

   If you have trouble with `mysql_install_db` at this point, see Section 2.10.1.1, “Problems Running mysql_install_db”.

3. In the absence of any option files, the server starts with its default settings. (See Section 5.1.2, “Server Configuration Defaults”.) To specify options that the MySQL server should use at startup, put them in an option file such as `/etc/my.cnf` or `/etc/mysql/my.cnf`. (See Section 4.2.2.2, “Using Option Files”.) For example, you can use an option file to set the `secure_file_priv` system variable.

4. To arrange for MySQL to start without manual intervention at system boot time, see Section 2.10.5, “Starting and Stopping MySQL Automatically”.

5. Data directory initialization creates time zone tables in the `mysql` database but does not populate them. To do so, use the instructions in Section 5.1.13, “MySQL Server Time Zone Support”.

2.10.1.1 Problems Running mysql_install_db

The purpose of the `mysql_install_db` program is to initialize the data directory, including the tables in the `mysql` system database. It does not overwrite existing MySQL privilege tables, and it does not affect any other data.

To re-create your privilege tables, first stop the `mysqld` server if it is running. Then rename the `mysql` directory under the data directory to save it, and run `mysql_install_db`. Suppose that your current directory is the MySQL installation directory and that `mysql_install_db` is located in the `bin` directory and the data directory is named `data`. To rename the `mysql` database and re-run `mysql_install_db`, use these commands.

```
mv data/mysql data/mysql.old
scripts/mysql_install_db --user=mysql
```

When you run `mysql_install_db`, you might encounter the following problems:

- **`mysql_install_db` fails to install the grant tables**

  You may find that `mysql_install_db` fails to install the grant tables and terminates after displaying the following messages:

  ```
  Starting mysqld daemon with databases from XXXXXX
  mysqld ended
  ```

  In this case, you should examine the error log file very carefully. The log should be located in the directory `XXXXXX` named by the error message and should indicate why `mysqld` did not start. If you do not understand what happened, include the log when you post a bug report. See Section 1.6, “How to Report Bugs or Problems”.

- **There is a `mysqld` process running**

  This indicates that the server is running, in which case the grant tables have probably been created already. If so, there is no need to run `mysql_install_db` at all because it needs to be run only once, when you first install MySQL.

- **Installing a second `mysqld` server does not work when one server is running**

  This can happen when you have an existing MySQL installation, but want to put a new installation in a different location. For example, you might have a production installation, but you want to create a second installation for testing purposes. Generally the problem that occurs when you try to run a second server is that it tries to use a network interface that is in use by the first server. In this case, you should see one of the following error messages:

  ```
  Can't start server: Bind on TCP/IP port:
  Address already in use
  Can't start server: Bind on unix socket...
  ```

  For instructions on setting up multiple servers, see Section 5.7, “Running Multiple MySQL Instances on One Machine”.

- **You do not have write access to the `/tmp` directory**

  If you do not have write access to create temporary files or a Unix socket file in the default location (the `/tmp` directory) or the `TMPDIR` environment variable, if it has been set, an error occurs when you run `mysql_install_db` or the `mysqld` server.

  You can specify different locations for the temporary directory and Unix socket file by executing these commands prior to starting `mysql_install_db` or `mysqld`, where *`some_tmp_dir`* is the full path name to some directory for which you have write permission:

  ```
  TMPDIR=/some_tmp_dir/
  MYSQL_UNIX_PORT=/some_tmp_dir/mysql.sock
  export TMPDIR MYSQL_UNIX_PORT
  ```

  Then you should be able to run `mysql_install_db` and start the server with these commands:

  ```
  scripts/mysql_install_db --user=mysql
  bin/mysqld_safe --user=mysql &
  ```

  If `mysql_install_db` is located in the `scripts` directory, modify the first command to `scripts/mysql_install_db`.

  See Section B.3.3.6, “How to Protect or Change the MySQL Unix Socket File”, and Section 4.9, “Environment Variables”.

There are some alternatives to running the `mysql_install_db` program provided in the MySQL distribution:

- If you want the initial privileges to differ from the standard defaults, use account-management statements such as `CREATE USER`, `GRANT`, and `REVOKE` to change the privileges *after* the grant tables have been set up. In other words, run `mysql_install_db`, and then use `mysql -u root mysql` to connect to the server as the MySQL `root` user so that you can issue the necessary statements. (See Section 13.7.1, “Account Management Statements”.)

  To install MySQL on several machines with the same privileges, put the `CREATE USER`, `GRANT`, and `REVOKE` statements in a file and execute the file as a script using `mysql` after running `mysql_install_db`. For example:

  ```
  scripts/mysql_install_db --user=mysql
  bin/mysql -u root < your_script_file
  ```

  This enables you to avoid issuing the statements manually on each machine.

- It is possible to re-create the grant tables completely after they have previously been created. You might want to do this if you are just learning how to use `CREATE USER`, `GRANT`, and `REVOKE` and have made so many modifications after running `mysql_install_db` that you want to wipe out the tables and start over.

  To re-create the grant tables, stop the server if it is running and remove the `mysql` database directory. Then run `mysql_install_db` again.

2.10.2 Starting the Server

This section describes how start the server on Unix and Unix-like systems. (For Windows, see Section 2.3.4.4, “Starting the Server for the First Time”.) For some suggested commands that you can use to test whether the server is accessible and working properly, see Section 2.10.3, “Testing the Server”.

Start the MySQL server like this:

```
shell> bin/mysqld_safe --user=mysql &
```

It is important that the MySQL server be run using an unprivileged (non-`root`) login account. To ensure this if you run `mysqld_safe` as `root`, include the `--user` option as shown. Otherwise, execute the program while logged in as `mysql`, in which case you can omit the `--user` option from the command.

For further instructions for running MySQL as an unprivileged user, see Section 6.1.5, "How to Run MySQL as a Normal User".

If the command fails immediately and prints `mysqld ended`, look for information in the error log (which by default is the *`host_name`*`.err` file in the data directory).

If the server is unable to access the data directory it starts or read the grant tables in the `mysql` database, it writes a message to its error log. Such problems can occur if you neglected to create the grant tables by initializing the data directory before proceeding to this step, or if you ran the command that initializes the data directory without the `--user` option. Remove the `data` directory and run the command with the `--user` option.

If you have other problems starting the server, see Section 2.10.2.1, "Troubleshooting Problems Starting the MySQL Server". For more information about `mysqld_safe`, see Section 4.3.2, "`mysqld_safe` — MySQL Server Startup Script".

2.10.2.1 Troubleshooting Problems Starting the MySQL Server

This section provides troubleshooting suggestions for problems starting the server. For additional suggestions for Windows systems, see Section 2.3.5, "Troubleshooting a Microsoft Windows MySQL Server Installation".

If you have problems starting the server, here are some things to try:

- Check the error log to see why the server does not start. Log files are located in the data directory (typically `C:\Program Files\MySQL\MySQL Server 5.6\data` on Windows, `/usr/local/mysql/data` for a Unix/Linux binary distribution, and `/usr/local/var` for a Unix/Linux source distribution). Look in the data directory for files with names of the form *`host_name`*`.err` and *`host_name`*`.log`, where *`host_name`* is the name of your server host. Then examine the last few lines of these files. Use `tail` to display them:

  ```
  shell> tail host_name.err
  shell> tail host_name.log
  ```

- Specify any special options needed by the storage engines you are using. You can create a `my.cnf` file and specify startup options for the engines that you plan to use. If you are going to use storage engines that support transactional tables (`InnoDB`, `NDB`), be sure that you have them configured the way you want before starting the server. If you are using `InnoDB` tables, see Section 14.8, "InnoDB Configuration" for guidelines and Section 14.14, "InnoDB Startup Options and System Variables" for option syntax.

  Although storage engines use default values for options that you omit, Oracle recommends that you review the available options and specify explicit values for any options whose defaults are not appropriate for your installation.

- Make sure that the server knows where to find the data directory. The `mysqld` server uses this directory as its current directory. This is where it expects to find databases and where it expects to write log files. The server also writes the pid (process ID) file in the data directory.

  The default data directory location is hardcoded when the server is compiled. To determine what the default path settings are, invoke `mysqld` with the `--verbose` and `--help` options. If the data directory is located somewhere else on your system, specify that location with the `--datadir` option to `mysqld` or `mysqld_safe`, on the command line or in an option file. Otherwise, the server does not work properly. As an alternative to the `--datadir` option, you can specify `mysqld` the location of the base directory under which MySQL is installed with the `--basedir`, and `mysqld` looks for the `data` directory there.

To check the effect of specifying path options, invoke `mysqld` with those options followed by the `--verbose` and `--help` options. For example, if you change location to the directory where `mysqld` is installed and then run the following command, it shows the effect of starting the server with a base directory of `/usr/local`:

```
shell> ./mysqld --basedir=/usr/local --verbose --help
```

You can specify other options such as `--datadir` as well, but `--verbose` and `--help` must be the last options.

Once you determine the path settings you want, start the server without `--verbose` and `--help`.

If `mysqld` is currently running, you can find out what path settings it is using by executing this command:

```
shell> mysqladmin variables
```

Or:

```
shell> mysqladmin -h host_name variables
```

`host_name` is the name of the MySQL server host.

- Make sure that the server can access the data directory. The ownership and permissions of the data directory and its contents must allow the server to read and modify them.

  If you get `Errcode 13` (which means `Permission denied`) when starting `mysqld`, this means that the privileges of the data directory or its contents do not permit server access. In this case, you change the permissions for the involved files and directories so that the server has the right to use them. You can also start the server as `root`, but this raises security issues and should be avoided.

  Change location to the data directory and check the ownership of the data directory and its contents to make sure the server has access. For example, if the data directory is `/usr/local/mysql/var`, use this command:

  ```
  shell> ls -la /usr/local/mysql/var
  ```

  If the data directory or its files or subdirectories are not owned by the login account that you use for running the server, change their ownership to that account. If the account is named `mysql`, use these commands:

  ```
  shell> chown -R mysql /usr/local/mysql/var
  shell> chgrp -R mysql /usr/local/mysql/var
  ```

  Even with correct ownership, MySQL might fail to start up if there is other security software running on your system that manages application access to various parts of the file system. In this case, reconfigure that software to enable `mysqld` to access the directories it uses during normal operation.

- Verify that the network interfaces the server wants to use are available.

  If either of the following errors occur, it means that some other program (perhaps another `mysqld` server) is using the TCP/IP port or Unix socket file that `mysqld` is trying to use:

  ```
  Can't start server: Bind on TCP/IP port: Address already in use
  Can't start server: Bind on unix socket...
  ```

  Use `ps` to determine whether you have another `mysqld` server running. If so, shut down the server before starting `mysqld` again. (If another server is running, and you really want to run multiple

servers, you can find information about how to do so in Section 5.7, “Running Multiple MySQL Instances on One Machine”.)

If no other server is running, execute the command `telnet your_host_name tcp_ip_port_number`. (The default MySQL port number is 3306.) Then press Enter a couple of times. If you do not get an error message like `telnet: Unable to connect to remote host: Connection refused`, some other program is using the TCP/IP port that `mysqld` is trying to use. Track down what program this is and disable it, or tell `mysqld` to listen to a different port with the `--port` option. In this case, specify the same non-default port number for client programs when connecting to the server using TCP/IP.

Another reason the port might be inaccessible is that you have a firewall running that blocks connections to it. If so, modify the firewall settings to permit access to the port.

If the server starts but you cannot connect to it, make sure that you have an entry in `/etc/hosts` that looks like this:

```
127.0.0.1       localhost
```

- If you cannot get `mysqld` to start, try to make a trace file to find the problem by using the `--debug` option. See Section 5.8.3, “The DBUG Package”.

2.10.3 Testing the Server

After the data directory is initialized and you have started the server, perform some simple tests to make sure that it works satisfactorily. This section assumes that your current location is the MySQL installation directory and that it has a `bin` subdirectory containing the MySQL programs used here. If that is not true, adjust the command path names accordingly.

Alternatively, add the `bin` directory to your `PATH` environment variable setting. That enables your shell (command interpreter) to find MySQL programs properly, so that you can run a program by typing only its name, not its path name. See Section 4.2.7, “Setting Environment Variables”.

Use `mysqladmin` to verify that the server is running. The following commands provide simple tests to check whether the server is up and responding to connections:

```
shell> bin/mysqladmin version
shell> bin/mysqladmin variables
```

If you cannot connect to the server, specify a `-u root` option to connect as `root`. If you have assigned a password for the `root` account already, you'll also need to specify `-p` on the command line and enter the password when prompted. For example:

```
shell> bin/mysqladmin -u root -p version
Enter password: (enter root password here)
```

The output from `mysqladmin version` varies slightly depending on your platform and version of MySQL, but should be similar to that shown here:

```
shell> bin/mysqladmin version
mysqladmin  Ver 14.12 Distrib 5.6.51, for pc-linux-gnu on i686
...

Server version          5.6.51
Protocol version        10
Connection              Localhost via UNIX socket
UNIX socket             /var/lib/mysql/mysql.sock
Uptime:                 14 days 5 hours 5 min 21 sec

Threads: 1  Questions: 366  Slow queries: 0
Opens: 0  Flush tables: 1  Open tables: 19
Queries per second avg: 0.000
```

To see what else you can do with `mysqladmin`, invoke it with the `--help` option.

Verify that you can shut down the server (include a `-p` option if the `root` account has a password already):

```
shell> bin/mysqladmin -u root shutdown
```

Verify that you can start the server again. Do this by using `mysqld_safe` or by invoking `mysqld` directly. For example:

```
shell> bin/mysqld_safe --user=mysql &
```

If `mysqld_safe` fails, see Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”.

Run some simple tests to verify that you can retrieve information from the server. The output should be similar to that shown here.

Use `mysqlshow` to see what databases exist:

```
shell> bin/mysqlshow
+--------------------+
|     Databases      |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| test               |
+--------------------+
```

The list of installed databases may vary, but always includes at least `mysql` and `information_schema`.

If you specify a database name, `mysqlshow` displays a list of the tables within the database:

```
shell> bin/mysqlshow mysql
Database: mysql
+---------------------------+
|          Tables           |
+---------------------------+
| columns_priv              |
| db                        |
| event                     |
| func                      |
| general_log               |
| help_category             |
| help_keyword              |
| help_relation             |
| help_topic                |
| innodb_index_stats        |
| innodb_table_stats        |
| ndb_binlog_index          |
| plugin                    |
| proc                      |
| procs_priv                |
| proxies_priv              |
| servers                   |
| slave_master_info         |
| slave_relay_log_info      |
| slave_worker_info         |
| slow_log                  |
| tables_priv               |
| time_zone                 |
| time_zone_leap_second     |
| time_zone_name            |
| time_zone_transition      |
| time_zone_transition_type |
| user                      |
+---------------------------+
```

Use the `mysql` program to select information from a table in the `mysql` database:

```
shell> bin/mysql -e "SELECT User, Host, plugin FROM mysql.user" mysql
+------+-----------+-----------------------+
| User | Host      | plugin                |
+------+-----------+-----------------------+
| root | localhost | mysql_native_password |
+------+-----------+-----------------------+
```

At this point, your server is running and you can access it. To tighten security if you have not yet assigned passwords to the initial account or accounts, follow the instructions in Section 2.10.4, “Securing the Initial MySQL Accounts”.

For more information about `mysql`, `mysqladmin`, and `mysqlshow`, see Section 4.5.1, “`mysql` — The MySQL Command-Line Client”, Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”, and Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”.

2.10.4 Securing the Initial MySQL Accounts

The MySQL installation process involves initializing the data directory, including the grant tables in the `mysql` system database that define MySQL accounts. For details, see Section 2.10.1, “Initializing the Data Directory”.

This section describes how to assign passwords to the initial accounts created during the MySQL installation procedure, if you have not already done so.

The `mysql.user` grant table defines the initial MySQL user accounts and their access privileges:

- Some accounts have the user name `root`. These are superuser accounts that have all privileges and can do anything. If these `root` accounts have empty passwords, anyone can connect to the MySQL server as `root` *without a password* and be granted all privileges.

  - On Windows, `root` accounts are created that permit connections from the local host only. Connections can be made by specifying the host name `localhost`, the IP address `127.0.0.1`, or the IPv6 address `::1`. If the user selects the **Enable root access from remote machines** option during installation, the Windows installer creates another `root` account that permits connections from any host.

  - On Unix, each `root` account permits connections from the local host. Connections can be made by specifying the host name `localhost`, the IP address `127.0.0.1`, the IPv6 address `::1`, or the actual host name or IP address.

  An attempt to connect to the host `127.0.0.1` normally resolves to the `localhost` account. However, this fails if the server is run with `skip_name_resolve` enabled, so the `127.0.0.1` account is useful in that case. The `::1` account is used for IPv6 connections.

- If accounts for anonymous users were created, these have an empty user name. The anonymous accounts have no password, so anyone can use them to connect to the MySQL server.

  - On Windows, there is one anonymous account that permits connections from the local host. Connections can be made by specifying a host name of `localhost`.

  - On Unix, each anonymous account permits connections from the local host. Connections can be made by specifying a host name of `localhost` for one of the accounts, or the actual host name or IP address for the other.

  - The `'root'@'localhost'` account also has a row in the `mysql.proxies_priv` table that enables granting the `PROXY` privilege for `''@''`, that is, for all users and all hosts. This enables `root` to set up proxy users, as well as to delegate to other accounts the authority to set up proxy users. See Section 6.2.12, “Proxy Users”.

To display which accounts exist in the `mysql.user` system table and check whether their passwords are empty, use the following statement:

```
mysql> SELECT User, Host, Password FROM mysql.user;
+------+-----------------------+----------+
| User | Host                  | Password |
+------+-----------------------+----------+
| root | localhost             |          |
| root | myhost.example.com    |          |
| root | 127.0.0.1             |          |
| root | ::1                   |          |
|      | localhost             |          |
|      | myhost.example.com    |          |
+------+-----------------------+----------+
```

This output indicates that there are several `root` and anonymous-user accounts, none of which have passwords. The output might differ on your system, but the presence of accounts with empty passwords means that your MySQL installation is unprotected until you do something about it:

- Assign a password to each MySQL `root` account that does not have one.
- To prevent clients from connecting as anonymous users without a password, either assign a password to each anonymous account or remove the accounts.

In addition, the `mysql.db` table contains rows that permit all accounts to access the `test` database and other databases with names that start with `test_`. This is true even for accounts that otherwise have no special privileges such as the default anonymous accounts. This is convenient for testing but inadvisable on production servers. Administrators who want database access restricted only to accounts that have permissions granted explicitly for that purpose should remove these `mysql.db` table rows.

The following instructions describe how to set up passwords for the initial MySQL accounts, first for the `root` accounts, then for the anonymous accounts. The instructions also cover how to remove anonymous accounts, should you prefer not to permit anonymous access at all, and describe how to remove permissive access to test databases. Replace *`new_password`* in the examples with the password that you want to use. Replace *`host_name`* with the name of the server host. You can determine this name from the output of the preceding `SELECT` statement. For the output shown, *`host_name`* is `myhost.example.com`.

You need not remove anonymous entries in the `mysql.proxies_priv` table, which are used to support proxy users. See Section 6.2.12, “Proxy Users”.

Note

For additional information about setting passwords, see Section 6.2.9, “Assigning Account Passwords”. If you forget your `root` password after setting it, see Section B.3.3.2, “How to Reset the Root Password”.

To set up additional accounts, see Section 6.2.7, “Adding Accounts, Assigning Privileges, and Dropping Accounts”.

You might want to defer setting the passwords until later, to avoid the need to specify them while you perform additional setup or testing. However, be sure to set them before using your installation for production purposes.

Note

Alternative means for performing the process described in this section:

- On Windows, you can perform the process during installation with MySQL Installer (see Section 2.3.3, “MySQL Installer for Windows”).
- On all platforms, the MySQL distribution includes `mysql_secure_installation`, a command-line utility that automates much of the process of securing a MySQL installation.

- On all platforms, MySQL Workbench is available and offers the ability to manage user accounts (see Chapter 25, *MySQL Workbench*).

- Assigning root Account Passwords
- Assigning Anonymous Account Passwords
- Removing Anonymous Accounts
- Securing Test Databases

Assigning root Account Passwords

A `root` account password can be set several ways. The following discussion demonstrates three methods:

- Use the `SET PASSWORD` statement
- Use the `UPDATE` statement
- Use the `mysqladmin` command-line client program

To assign passwords using `SET PASSWORD`, connect to the server as `root` and issue a `SET PASSWORD` statement for each `root` account listed in the `mysql.user` system table.

For Windows, do this:

```
shell> mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('new_password');
mysql> SET PASSWORD FOR 'root'@'127.0.0.1' = PASSWORD('new_password');
mysql> SET PASSWORD FOR 'root'@'::1' = PASSWORD('new_password');
mysql> SET PASSWORD FOR 'root'@'%' = PASSWORD('new_password');
```

The last statement is unnecessary if the `mysql.user` table has no `root` account with a host value of `%`.

For Unix, do this:

```
shell> mysql -u root
mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('new_password');
mysql> SET PASSWORD FOR 'root'@'127.0.0.1' = PASSWORD('new_password');
mysql> SET PASSWORD FOR 'root'@'::1' = PASSWORD('new_password');
mysql> SET PASSWORD FOR 'root'@'host_name' = PASSWORD('new_password');
```

You can also use a single statement that assigns a password to all `root` accounts by using `UPDATE` to modify the `mysql.user` table directly. This method works on any platform:

```
shell> mysql -u root
mysql> UPDATE mysql.user SET Password = PASSWORD('new_password')
    ->     WHERE User = 'root';
mysql> FLUSH PRIVILEGES;
```

The `FLUSH` statement causes the server to reread the grant tables. Without it, the password change remains unnoticed by the server until you restart it.

To assign passwords to the `root` accounts using `mysqladmin`, execute the following commands:

```
shell> mysqladmin -u root password "new_password"
shell> mysqladmin -u root -h host_name password "new_password"
```

Those commands apply both to Windows and to Unix. The double quotation marks around the password are not always necessary, but you should use them if the password contains spaces or other characters that are special to your command interpreter.

The `mysqladmin` method of setting the `root` account passwords does not work for the `'root'@'127.0.0.1'` or `'root'@'::1'` account. Use the `SET PASSWORD` method shown earlier.

After the `root` passwords have been set, you must supply the appropriate password whenever you connect as `root` to the server. For example, to shut down the server with `mysqladmin`, use this command:

```
shell> mysqladmin -u root -p shutdown
Enter password: (enter root password here)
```

The `mysql` commands in the following instructions include a `-p` option based on the assumption that you have assigned the `root` account passwords using the preceding instructions and must specify that password when connecting to the server.

Assigning Anonymous Account Passwords

To assign passwords to the anonymous accounts, connect to the server as `root`, then use either `SET PASSWORD` or `UPDATE`.

To use `SET PASSWORD` on Windows, do this:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> SET PASSWORD FOR ''@'localhost' = PASSWORD('new_password');
```

To use `SET PASSWORD` on Unix, do this:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> SET PASSWORD FOR ''@'localhost' = PASSWORD('new_password');
mysql> SET PASSWORD FOR ''@'host_name' = PASSWORD('new_password');
```

To set the anonymous-user account passwords with a single `UPDATE` statement, do this (on any platform):

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> UPDATE mysql.user SET Password = PASSWORD('new_password')
    ->     WHERE User = '';
mysql> FLUSH PRIVILEGES;
```

The `FLUSH` statement causes the server to reread the grant tables. Without it, the password change remains unnoticed by the server until you restart it.

Removing Anonymous Accounts

If you prefer to remove any anonymous accounts rather than assigning them passwords, do so as follows on Windows:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> DROP USER ''@'localhost';
```

On Unix, remove the anonymous accounts like this:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> DROP USER ''@'localhost';
mysql> DROP USER ''@'host_name';
```

Securing Test Databases

By default, the `mysql.db` table contains rows that permit access by any user to the `test` database and other databases with names that start with `test_`. (These rows have an empty `User` column

value, which for access-checking purposes matches any user name.) This means that such databases can be used even by accounts that otherwise possess no privileges. If you want to remove any-user access to test databases, do so as follows:

```
shell> mysql -u root -p
Enter password: (enter root password here)
mysql> DELETE FROM mysql.db WHERE Db LIKE 'test%';
mysql> FLUSH PRIVILEGES;
```

The `FLUSH` statement causes the server to reread the grant tables. Without it, the privilege change remains unnoticed by the server until you restart it.

With the preceding change, only users who have global database privileges or privileges granted explicitly for the `test` database can use it. However, if you prefer that the database not exist at all, drop it:

```
mysql> DROP DATABASE test;
```

2.10.5 Starting and Stopping MySQL Automatically

This section discusses methods for starting and stopping the MySQL server.

Generally, you start the `mysqld` server in one of these ways:

- Invoke `mysqld` directly. This works on any platform.
- On Windows, you can set up a MySQL service that runs automatically when Windows starts. See Section 2.3.4.7, “Starting MySQL as a Windows Service”.
- On Unix and Unix-like systems, you can invoke `mysqld_safe`, which tries to determine the proper options for `mysqld` and then runs it with those options. See Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.
- On systems that use System V-style run directories (that is, `/etc/init.d` and run-level specific directories), invoke `mysql.server`. This script is used primarily at system startup and shutdown. It usually is installed under the name `mysql`. The `mysql.server` script starts the server by invoking `mysqld_safe`. See Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”.
- On macOS, install a launchd daemon to enable automatic MySQL startup at system startup. The daemon starts the server by invoking `mysqld_safe`. For details, see Section 2.4.3, “Installing a MySQL Launch Daemon”. A MySQL Preference Pane also provides control for starting and stopping MySQL through the System Preferences. See Section 2.4.4, “Installing and Using the MySQL Preference Pane”.
- On Solaris, use the service management framework (SMF) system to initiate and control MySQL startup.

The `mysqld_safe` and `mysql.server` scripts, Solaris SMF, and the macOS Startup Item (or MySQL Preference Pane) can be used to start the server manually, or automatically at system startup time. `mysql.server` and the Startup Item also can be used to stop the server.

The following table shows which option groups the server and startup scripts read from option files.

Table 2.14 MySQL Startup Scripts and Supported Server Option Groups

Script	Option Groups
`mysqld`	`[mysqld]`, `[server]`, `[mysqld-major_version]`
`mysqld_safe`	`[mysqld]`, `[server]`, `[mysqld_safe]`
`mysql.server`	`[mysqld]`, `[mysql.server]`, `[server]`

`[mysqld-major_version]` means that groups with names like `[mysqld-5.5]` and `[mysqld-5.6]` are read by servers having versions 5.5.x, 5.6.x, and so forth. This feature can be used to specify options that can be read only by servers within a given release series.

For backward compatibility, `mysql.server` also reads the `[mysql_server]` group and `mysqld_safe` also reads the `[safe_mysqld]` group. However, you should update your option files to use the `[mysql.server]` and `[mysqld_safe]` groups instead.

For more information on MySQL configuration files and their structure and contents, see Section 4.2.2.2, “Using Option Files”.

2.11 Upgrading MySQL

This section describes the steps to upgrade a MySQL installation.

Upgrading is a common procedure, as you pick up bug fixes within the same MySQL release series or significant features between major MySQL releases. You perform this procedure first on some test systems to make sure everything works smoothly, and then on the production systems.

Note

In the following discussion, MySQL commands that must be run using a MySQL account with administrative privileges include `-u root` on the command line to specify the MySQL `root` user. Commands that require a password for `root` also include a `-p` option. Because `-p` is followed by no option value, such commands prompt for the password. Type the password when prompted and press Enter.

SQL statements can be executed using the `mysql` command-line client (connect as `root` to ensure that you have the necessary privileges).

2.11.1 Before You Begin

Review the information in this section before upgrading. Perform any recommended actions.

- Protect your data by creating a backup. The backup should include the `mysql` system database, which contains the MySQL system tables. See Section 7.2, “Database Backup Methods”.
- Review Section 2.11.2, “Upgrade Paths” to ensure that your intended upgrade path is supported.
- Review Section 2.11.3, “Changes in MySQL 5.6” for changes that you should be aware of before upgrading. Some changes may require action.
- Review Section 1.3, “What Is New in MySQL 5.6” for deprecated and removed features. An upgrade may require changes with respect to those features if you use any of them.
- Review Section 1.4, “Server and Status Variables and Options Added, Deprecated, or Removed in MySQL 5.6”. If you use deprecated or removed variables, an upgrade may require configuration changes.
- Review the Release Notes for information about fixes, changes, and new features.
- If you use replication, review Section 17.4.3, “Upgrading a Replication Setup”.
- Upgrade procedures vary by platform and how the initial installation was performed. Use the procedure that applies to your current MySQL installation:
  - For binary and package-based installations on non-Windows platforms, refer to Section 2.11.4, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”.
  - For installations on an Enterprise Linux platform or Fedora using the MySQL Yum Repository, refer to Section 2.11.5, “Upgrading MySQL with the MySQL Yum Repository”.

  - For installations on Ubuntu using the MySQL APT repository, refer to Section 2.11.6, “Upgrading MySQL with the MySQL APT Repository”.
  - For installations on SLES using the MySQL SLES repository, refer to Section 2.11.7, “Upgrading MySQL with the MySQL SLES Repository”.
  - For installations on Windows, refer to Section 2.11.8, “Upgrading MySQL on Windows”.
- If your MySQL installation contains a large amount of data that might take a long time to convert after an in-place upgrade, it may be useful to create a test instance for assessing the conversions that are required and the work involved to perform them. To create a test instance, make a copy of your MySQL instance that contains the `mysql` database and other databases without the data. Run the upgrade procedure on the test instance to assess the work involved to perform the actual data conversion.
- Rebuilding and reinstalling MySQL language interfaces is recommended when you install or upgrade to a new release of MySQL. This applies to MySQL interfaces such as PHP `mysql` extensions and the Perl `DBD::mysql` module.

2.11.2 Upgrade Paths

- Upgrade is only supported between General Availability (GA) releases.
- Upgrade from MySQL 5.5 to 5.6 is supported. Upgrading to the latest release is recommended before upgrading to the next version. For example, upgrade to the latest MySQL 5.5 release before upgrading to MySQL 5.6.
- Upgrade that skips versions is not supported. For example, upgrading directly from MySQL 5.1 to 5.6 is not supported.
- Upgrade within a release series is supported. For example, upgrading from MySQL 5.6.*x* to 5.6.*y* is supported. Skipping a release is also supported. For example, upgrading from MySQL 5.6.*x* to 5.6.*z* is supported.

2.11.3 Changes in MySQL 5.6

Before upgrading to MySQL 5.6, review the changes described in this section to identify those that apply to your current MySQL installation and applications. Perform any recommended actions.

Changes marked as **Incompatible change** are incompatibilities with earlier versions of MySQL, and may require your attention *before upgrading*. Our aim is to avoid these changes, but occasionally they are necessary to correct problems that would be worse than an incompatibility between releases. If an upgrade issue applicable to your installation involves an incompatibility, follow the instructions given in the description. Sometimes this involves dumping and reloading tables, or use of a statement such as `CHECK TABLE` or `REPAIR TABLE`.

For dump and reload instructions, see Section 2.11.10, “Rebuilding or Repairing Tables or Indexes”. Any procedure that involves `REPAIR TABLE` with the `USE_FRM` option *must* be done before upgrading. Use of this statement with a version of MySQL different from the one used to create the table (that is, using it after upgrading) may damage the table. See Section 13.7.2.5, “REPAIR TABLE Statement”.

Note

Beginning with MySQL 5.6.6, several MySQL Server parameters have defaults that differ from previous releases. See the notes regarding these changes under Configuration Changes, particularly regarding overriding them to preserve backward compatibility if that is a concern.

- Configuration Changes
- Server Changes

- InnoDB Changes
- SQL Changes

Configuration Changes

- Beginning with MySQL 5.6.6, several MySQL Server parameters have defaults that differ from previous releases. The motivation for these changes is to provide better out-of-box performance and to reduce the need for the database administrator to change settings manually. These changes are subject to possible revision in future releases as we gain feedback.

  In some cases, a parameter has a different static default value. In other cases, the server autosizes a parameter at startup using a formula based on other related parameters or server host configuration, rather than using a static value. For example, the setting for `back_log` now is its previous default of 50, adjusted up by an amount proportional to the value of `max_connections`. The idea behind autosizing is that when the server has information available to make a decision about a parameter setting likely to be better than a fixed default, it does so.

  The following table summarizes changes to defaults. Any of these can be overridden by specifying an explicit value at server startup.

Parameter	Old Default	New Default
`back_log`	50	Autosized using `max_connections`
`binlog_checksum`	`NONE`	`CRC32`
`--binlog-row-event-max-size`	1024	8192
`flush_time`	1800 (on Windows)	0
`innodb_autoextend_increment`	8	64
`innodb_buffer_pool_instances`	1	8 (platform dependent)
`innodb_checksum_algorithm`	`INNODB`	`CRC32 (changed back to INNODB in MySQL 5.6.7)`
`innodb_concurrency_tickets`	500	5000
`innodb_file_per_table`	`0`	`1`
`innodb_old_blocks_time`	0	1000
`innodb_open_files`	300	Autosized using `innodb_file_per_table`, `table_open_cache`
`innodb_stats_on_metadata`	`ON`	`OFF`
`join_buffer_size`	128KB	256KB
`max_allowed_packet`	1MB	4MB
`max_connect_errors`	10	100
`sync_master_info`	0	10000
`sync_relay_log`	0	10000
`sync_relay_log_info`	0	10000

  With regard to compatibility with previous releases, the most important changes are:

  - `innodb_file_per_table` is enabled (previously disabled).
  - `innodb_checksum_algorithm` is `CRC32` (previously `INNODB` and changed back to `INNODB` in MySQL 5.6.7).

- `binlog_checksum` is `CRC32` (previously `NONE`).

Therefore, if you are upgrading an existing MySQL installation, have not already changed the values of these parameters from their previous defaults, and backward compatibility is a concern, you may want to explicitly set these parameters to their previous defaults. For example, put these lines in the server option file:

```
[mysqld]
innodb_file_per_table=0
innodb_checksum_algorithm=INNODB
binlog_checksum=NONE
```

Those settings preserve compatibility as follows:

- With the new default of `innodb_file_per_table` enabled, `ALTER TABLE` operations following an upgrade move `InnoDB` tables that are in the system tablespace to individual `.ibd` files. Using `innodb_file_per_table=0` prevents this from happening.
- Setting `innodb_checksum_algorithm=INNODB` permits binary downgrades after upgrading to this release. With a setting of `CRC32`, InnoDB would use checksumming that older MySQL versions cannot use.
- With `binlog_checksum=NONE`, the server can be used as a replication source without causing failure of older replicas that do not understand binary log checksums.

- As of MySQL 5.6.5, pre-4.1 passwords and the `mysql_old_password` authentication plugin are deprecated. Passwords stored in the older hash format used before MySQL 4.1 are less secure than passwords that use the native password hashing method and should be avoided. To prevent connections using accounts that have pre-4.1 password hashes, the `secure_auth` system variable is now enabled by default. (To permit connections for accounts that have such password hashes, start the server with `--secure_auth=0`.)

  DBAs are advised to convert accounts that use the `mysql_old_password` authentication plugin to use `mysql_native_password` instead. For account upgrade instructions, see Section 6.4.1.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”.

  In some early development versions of MySQL 5.6 (5.6.6 to 5.6.10), the server could create accounts with a mismatched password hash and authentication plugin. For example, if the default authentication plugin is `mysql_native_password`, this sequence of statements results in an account with a plugin of `mysql_native_password` but a pre-4.1 password hash (the format used by `mysql_old_password`):

  ```
  SET old_passwords = 1;
  CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'password';
  ```

  The mismatch produces symptoms such as being unable to connect to the MySQL server and being unable to use `SET PASSWORD` with `OLD_PASSWORD()` or with `old_passwords=1`.

  As of MySQL 5.6.11, this mismatch no longer occurs. Instead, the server produces an error:

  ```
  mysql> SET old_passwords = 1;
  mysql> CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'password';
  ERROR 1827 (HY000): The password hash doesn't have the expected
  format. Check if the correct password algorithm is being used with
  ```

```
the PASSWORD() function.
```

To deal with an account affected by a mismatch, the DBA can modify either the `plugin` or `Password` column in the account's `mysql.user` system table row to be consistent with the other column:

- Set `old_passwords` to 0, then assign a new password to the account using `SET PASSWORD` and `PASSWORD()`. This sets the `Password` column to have a 4.1 password hash, consistent with the `mysql_native_password` plugin. This is the preferred method of fixing the account.
- Alternatively, the DBA can change the plugin to `mysql_old_password` to make the plugin match the password hash format, then flush the privileges. This is not recommended because the `mysql_old_password` plugin and pre-4.1 password hashing are deprecated; expect support for them to be removed in a future version of MySQL.

Server Changes

- **Incompatible change**: It is possible for a column `DEFAULT` value to be valid for the `sql_mode` value at table-creation time but invalid for the `sql_mode` value when rows are inserted or updated. Example:

```
SET sql_mode = '';
CREATE TABLE t (d DATE DEFAULT 0);
SET sql_mode = 'NO_ZERO_DATE,STRICT_ALL_TABLES';
INSERT INTO t (d) VALUES(DEFAULT);
```

In this case, 0 should be accepted for the `CREATE TABLE` but rejected for the `INSERT`. However, the server did not evaluate `DEFAULT` values used for inserts or updates against the current `sql_mode`. In the example, the `INSERT` succeeds and inserts `'0000-00-00'` into the `DATE` column.

As of MySQL 5.6.13, the server applies the proper `sql_mode` checks to generate a warning or error at insert or update time.

A resulting incompatibility for replication if you use statement-based logging (`binlog_format=STATEMENT`) is that if a replica is upgraded, a source which has not been upgraded executes the preceding example without error, whereas the `INSERT` fails on the replica and replication stops.

To deal with this, stop all new statements on the source and wait until the replicas catch up. Then upgrade the replicas followed by the source. Alternatively, if you cannot stop new statements, temporarily change to row-based logging on the source (`binlog_format=ROW`) and wait until all replicas have processed all binary logs produced up to the point of this change. Then upgrade the replicas followed by the source and change the source back to statement-based logging.

- **Incompatible change**: MySQL 5.6.11 and later supports `CREATE TABLE ... [SUB]PARTITION BY ALGORITHM=`*`n`*` [LINEAR] KEY (...)`, which can be used to create a table whose `KEY` partitioning is compatible with a MySQL 5.1 server (*`n`*=1). (Bug #14521864, Bug #66462) This syntax is not accepted by MySQL 5.6.10 and earlier, although it is supported in MySQL 5.5 beginning with MySQL 5.5.31. `mysqldump` in MySQL 5.5.31 and later MySQL 5.5 releases includes the `ALGORITHM` option when dumping tables using this option, but surrounds it with conditional comments, like this:

```
CREATE TABLE t1 (a INT)
/*!50100 PARTITION BY KEY */ /*!50531 ALGORITHM = 1 */ /*!50100 ()
      PARTITIONS 3 */
```

When importing a dump containing such `CREATE TABLE` statements into a MySQL 5.6.10 or earlier MySQL 5.6 server, the versioned comment is not ignored, which causes a syntax error. Therefore, prior to importing such a dump file, you must either change the comments so that the MySQL 5.6 server ignores them (by removing the string `!50531` or replacing it with `!50611`, wherever it occurs), or remove them.

This is not an issue with dump files made using MySQL 5.6.11 or later, where the `ALGORITHM` option is written using `/*!50611 ... */`.

- **Incompatible change**: For `TIME`, `DATETIME`, and `TIMESTAMP` columns, the storage required for tables created before MySQL 5.6.4 differs from storage required for tables created in 5.6.4 and later. This is due to a change in 5.6.4 that permits these temporal types to have a fractional part. This change can affect the output of statements that depend on the row format, such as `CHECKSUM TABLE`. After upgrading from MySQL 5.5 to MySQL 5.6.4 or later, it is recommended that you also upgrade from MySQL 5.5 to MySQL 5.6 `TIME`, `DATETIME`, and `TIMESTAMP` types. `ALTER TABLE` currently allows the creation of tables containing temporal columns in both MySQL 5.5 and MySQL 5.6.4 (or later) binary format but this makes it more difficult to recreate tables in cases where `.frm` files are not available. Additionally, as of MySQL 5.6.4, the aforementioned temporal types are more space efficient. For more information about changes to temporal types in MySQL 5.6.4, see Date and Time Type Storage Requirements.

  As of MySQL 5.6.16, `ALTER TABLE` upgrades old temporal columns to 5.6 format for `ADD COLUMN`, `CHANGE COLUMN`, `MODIFY COLUMN`, `ADD INDEX`, and `FORCE` operations. Hence, the following statement upgrades a table containing columns in the old format:

  ```
  ALTER TABLE tbl_name FORCE;
  ```

  This conversion cannot be done using the `INPLACE` algorithm because the table must be rebuilt, so specifying `ALGORITHM=INPLACE` in these cases results in an error. Specify `ALGORITHM=COPY` if necessary.

  When `ALTER TABLE` does produce a temporal-format conversion, it generates a message that can be displayed with `SHOW WARNINGS`: `TIME/TIMESTAMP/DATETIME columns of old format have been upgraded to the new format`.

  When upgrading to MySQL 5.6.4 or later, be aware that `CHECK TABLE ... FOR UPGRADE` does not report temporal columns that use the pre-MySQL 5.6.4 format (Bug #73008, Bug #18985579). In MySQL 5.6.24, two new system variables, `avoid_temporal_upgrade` and `show_old_temporals`, were added to provide control over temporal column upgrades (Bug #72997, Bug #18985760).

- Due to the temporal type changes described in the previous incompatible change item above, importing pre-MySQL 5.6.4 tables (using `ALTER TABLE ... IMPORT TABLESPACE`) that contain `DATETIME` and `TIMESTAMP` types into MySQL 5.6.4 (or later) fails. Importing a MySQL 5.5 table with these temporal types into MySQL 5.6.4 (or later) is the mostly likely scenario for this problem to occur.

  The following procedures describe workarounds that use the original pre-MySQL 5.6.4 `.frm` file to recreate a table with a row structure that is compatible with 5.6.4 (or later). The procedures involve changing the original pre-MySQL 5.6.4 `.frm` file to use the `Memory` storage engine instead of `InnoDB`, copying the `.frm` file to the data directory of the destination instance, and using `ALTER TABLE` to change the table's storage engine type back to `InnoDB`. Use the first procedure if your tables do not have foreign keys. Use the second procedure, which has additional steps, if your table includes foreign keys.

  If the table does not have foreign keys:

  1. Copy the table's original `.frm` file to the data directory on the server where you want to import the tablespace.

2. Modify the table's `.frm` file to use the `Memory` storage engine instead of the `InnoDB` storage engine. This modification requires changing 7 bytes in the `.frm` file that define the table's storage engine type. Using a hexidecimal editing tool:

   - Change the byte at offset position 0003, which is the `legacy_db_type`, from `0c` (for `InnoDB`) to `06` (for `Memory`), as shown below:

     ```
     00000000  fe 01 09 06 03 00 00 10  01 00 00 30 00 00 10 00
     ```

   - The remaining 6 bytes do not have a fixed offset. Search the `.frm` file for "`InnoDB`" to locate the line with the other 6 bytes. The line appears as shown below:

     ```
     00001010  ff 00 00 00 00 00 00 06  00 49 6e 6e 6f 44 42 00  |.........InnoDB.|
     ```

   - Modify the bytes so that the line appears as follows:

     ```
     00001010  ff 00 00 00 00 00 00 06 00 4d 45 4d 4f 52 59 00
     ```

3. Run `ALTER TABLE ... ENGINE=INNODB` to add the table definition to the `InnoDB` data dictionary. This creates the `InnoDB` table with the temporal data types in the new format. For the `ALTER TABLE` operation to complete successfully, the `.frm` file must correspond to the tablespace.

4. Import the table using `ALTER TABLE ... IMPORT TABLESPACE`.

If table has foreign keys:

1. Recreate the tables with foreign keys using table definitions from `SHOW CREATE TABLE` output. The incorrect temporal column formats do not matter at this point.

2. Dump all foreign key definitions to a text file by selecting the foreign key information from `INFORMATION_SCHEMA.TABLE_CONSTRAINTS` and `INFORMATION_SCHEMA.KEY_COLUMN_USAGE`.

3. Drop all tables and complete the table import process described in steps 1 to 4 in the procedure described above for tables without foreign keys.

4. After the import operation is complete, add the foreign keys from foreign key definitions that you saved to a text file.

- **Incompatible change**: As of MySQL 5.6, the full-text stopword file is loaded and searched using `latin1` if `character_set_server` is `ucs2`, `utf16`, `utf16le`, or `utf32`. If any table was created with `FULLTEXT` indexes while the server character set was `ucs2`, `utf16`, `utf16le`, or `utf32`, repair it using this statement:

  ```
  REPAIR TABLE tbl_name QUICK;
  ```

- **Incompatible change**: In MySQL 5.6.20, the patch for Bug #69477 limits the size of redo log `BLOB` writes to 10% of the redo log file size. As a result of this new limit, `innodb_log_file_size` should be set to a value greater than 10 times the largest `BLOB` data size found in the rows of your tables. No action is required if your `innodb_log_file_size` setting is already 10 times the largest `BLOB` data size or your tables contain no `BLOB` data.

  In MySQL 5.6.22, the redo log `BLOB` write limit is relaxed to 10% of the *total redo log size* (`innodb_log_file_size` * `innodb_log_files_in_group`). (Bug #19498877)

InnoDB Changes

As of MySQL 5.6.42, the zlib library version bundled with MySQL was raised from version 1.2.3 to version 1.2.11.

The zlib `compressBound()` function in zlib 1.2.11 returns a slightly higher estimate of the buffer size required to compress a given length of bytes than it did in zlib version 1.2.3. The `compressBound()` function is called by `InnoDB` functions that determine the maximum row size permitted when creating compressed `InnoDB` tables or inserting rows into compressed `InnoDB` tables. As a result, `CREATE TABLE ... ROW_FORMAT=COMPRESSED` or `INSERT` operations with row sizes very close to the maximum row size that were successful in earlier releases could now fail.

If you have compressed `InnoDB` tables with large rows, it is recommended that you test compressed table `CREATE TABLE` statements on a MySQL 5.6 test instance prior to upgrading.

SQL Changes

- Some keywords may be reserved in MySQL 5.6 that were not reserved in MySQL 5.5. See Section 9.3, “Keywords and Reserved Words”. This can cause words previously used as identifiers to become illegal. To fix affected statements, use identifier quoting. See Section 9.2, “Schema Object Names”.

- The `YEAR(2)` data type has certain issues that you should consider before choosing to use it. As of MySQL 5.6.6, `YEAR(2)` is deprecated: `YEAR(2)` columns in existing tables are treated as before, but `YEAR(2)` in new or altered tables is converted to `YEAR(4)`. For more information, see Section 11.2.5, “2-Digit YEAR(2) Limitations and Migrating to 4-Digit YEAR”.

- As of MySQL 5.6.6, it is explicitly disallowed to assign the value `DEFAULT` to stored procedure or function parameters or stored program local variables (for example with a `SET *var_name* = DEFAULT` statement). This was not previously supported, or documented as permitted, but is flagged as an incompatible change in case existing code inadvertently used this construct. It remains permissible to assign `DEFAULT` to system variables, as before, but assigning `DEFAULT` to parameters or local variables now results in a syntax error.

  After an upgrade to MySQL 5.6.6 or later, existing stored programs that use this construct produce a syntax error when invoked. If a `mysqldump` file from 5.6.5 or earlier is loaded into 5.6.6 or later, the load operation fails and affected stored program definitions must be changed.

- In MySQL, the `TIMESTAMP` data type differs in nonstandard ways from other data types:

  - `TIMESTAMP` columns not explicitly declared with the `NULL` attribute are assigned the `NOT NULL` attribute. (Columns of other data types, if not explicitly declared as `NOT NULL`, permit `NULL` values.) Setting such a column to `NULL` sets it to the current timestamp.

  - The first `TIMESTAMP` column in a table, if not declared with the `NULL` attribute or an explicit `DEFAULT` or `ON UPDATE` clause, is automatically assigned the `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` attributes.

  - `TIMESTAMP` columns following the first one, if not declared with the `NULL` attribute or an explicit `DEFAULT` clause, are automatically assigned `DEFAULT '0000-00-00 00:00:00'` (the “zero” timestamp). For inserted rows that specify no explicit value for such a column, the column is assigned `'0000-00-00 00:00:00'` and no warning occurs.

  Those nonstandard behaviors remain the default for `TIMESTAMP` but as of MySQL 5.6.6 are deprecated and this warning appears at startup:

  ```
  [Warning] TIMESTAMP with implicit DEFAULT value is deprecated.
  Please use --explicit_defaults_for_timestamp server option (see
  documentation for more details).
  ```

  As indicated by the warning, to turn off the nonstandard behaviors, enable the new `explicit_defaults_for_timestamp` system variable at server startup. With this variable enabled, the server handles `TIMESTAMP` as follows instead:

  - `TIMESTAMP` columns not explicitly declared as `NOT NULL` permit `NULL` values. Setting such a column to `NULL` sets it to `NULL`, not the current timestamp.

- No `TIMESTAMP` column is assigned the `DEFAULT CURRENT_TIMESTAMP` or `ON UPDATE CURRENT_TIMESTAMP` attributes automatically. Those attributes must be explicitly specified.
- `TIMESTAMP` columns declared as `NOT NULL` and without an explicit `DEFAULT` clause are treated as having no default value. For inserted rows that specify no explicit value for such a column, the result depends on the SQL mode. If strict SQL mode is enabled, an error occurs. If strict SQL mode is not enabled, the column is assigned the implicit default of `'0000-00-00 00:00:00'` and a warning occurs. This is similar to how MySQL treats other temporal types such as `DATETIME`.

To upgrade servers used for replication, upgrade the replicas first, then the source. Replication between the source and its replicas should work provided that all use the same value of `explicit_defaults_for_timestamp`:

1. Bring down the replicas, upgrade them, configure them with the desired value of `explicit_defaults_for_timestamp`, and bring them back up.

   The replicas recognize from the format of the binary logs received from the source that the source is older (predates the introduction of `explicit_defaults_for_timestamp`) and that operations on `TIMESTAMP` columns coming from the source use the old `TIMESTAMP` behavior.

2. Bring down the source, upgrade it, configure it with the same `explicit_defaults_for_timestamp` value used on the replicas, and bring it back up.

2.11.4 Upgrading MySQL Binary or Package-based Installations on Unix/Linux

This section describes how to upgrade MySQL binary and package-based installations on Unix/Linux. In-place and logical upgrade methods are described.

Note

A logical upgrade is recommended when upgrading from a previous version. For example, use this method when upgrading from 5.5 to 5.6.

- In-Place Upgrade
- Logical Upgrade

In-Place Upgrade

An in-place upgrade involves shutting down the old MySQL server, replacing the old MySQL binaries or packages with the new ones, restarting MySQL on the existing data directory, and upgrading any remaining parts of the existing installation that require upgrading.

Note

If you upgrade an installation originally produced by installing multiple RPM packages, upgrade all the packages, not just some. For example, if you previously installed the server and client RPMs, do not upgrade just the server RPM.

To perform an in-place upgrade:

1. If you use XA transactions with `InnoDB`, run `XA RECOVER` before upgrading to check for uncommitted XA transactions. If results are returned, either commit or rollback the XA transactions by issuing an `XA COMMIT` or `XA ROLLBACK` statement.
2. If you use `InnoDB`, configure MySQL to perform a slow shutdown by setting `innodb_fast_shutdown` to `0`. For example:

```
mysql -u root -p --execute="SET GLOBAL innodb_fast_shutdown=0"
```

With a slow shutdown, `InnoDB` performs a full purge and change buffer merge before shutting down, which ensures that data files are fully prepared in case of file format differences between releases.

3. Shut down the old MySQL server. For example:

```
mysqladmin -u root -p shutdown
```

4. Upgrade the MySQL binary installation or packages. If upgrading a binary installation, unpack the new MySQL binary distribution package. See Obtain and Unpack the Distribution. For package-based installations, install the new packages.

5. Start the MySQL 5.6 server, using the existing data directory. For example:

```
mysqld_safe --user=mysql --datadir=/path/to/existing-datadir &
```

6. Run `mysql_upgrade`. For example:

```
mysql_upgrade -u root -p
```

`mysql_upgrade` examines all tables in all databases for incompatibilities with the current version of MySQL. `mysql_upgrade` also upgrades the `mysql` system database so that you can take advantage of new privileges or capabilities.

Note

`mysql_upgrade` does not upgrade the contents of the time zone tables or help tables. For upgrade instructions, see Section 5.1.13, “MySQL Server Time Zone Support”, and Section 5.1.14, “Server-Side Help Support”.

7. Shut down and restart the MySQL server to ensure that any changes made to the system tables take effect. For example:

```
mysqladmin -u root -p shutdown
mysqld_safe --user=mysql --datadir=/path/to/existing-datadir &
```

Logical Upgrade

A logical upgrade involves exporting SQL from the old MySQL instance using a backup or export utility such as `mysqldump`, installing the new MySQL server, and applying the SQL to your new MySQL instance.

To perform a logical upgrade:

1. Review the information in Section 2.11.1, “Before You Begin”.

2. Export your existing data from the previous MySQL installation:

```
mysqldump -u root -p
  --add-drop-table --routines --events
  --all-databases --force > data-for-upgrade.sql
```

Note

Use the `--routines` and `--events` options with `mysqldump` (as shown above) if your databases include stored programs. The `--all-databases` option includes all databases in the dump, including the `mysql` database that holds the system tables.

3. Shut down the old MySQL server. For example:

```
mysqladmin -u root -p shutdown
```

4. Install MySQL 5.6. For installation instructions, see Chapter 2, *Installing and Upgrading MySQL*.

5. Initialize a new data directory, as described at Section 2.10.1, “Initializing the Data Directory”. For example:

```
scripts/mysql_install_db --user=mysql --datadir=/path/to/5.6-datadir
```

6. Start the MySQL 5.6 server, using the new data directory. For example:

```
mysqld_safe --user=mysql --datadir=/path/to/5.6-datadir &
```

7. Load the previously created dump file into the new MySQL server. For example:

```
mysql -u root -p --force < data-for-upgrade.sql
```

Note

It is not recommended to load a dump file when GTIDs are enabled on the server (`gtid_mode=ON`), if your dump file includes system tables. `mysqldump` issues DML instructions for the system tables which use the non-transactional MyISAM storage engine, and this combination is not permitted when GTIDs are enabled. Also be aware that loading a dump file from a server with GTIDs enabled, into another server with GTIDs enabled, causes different transaction identifiers to be generated.

8. Run `mysql_upgrade`. For example:

```
mysql_upgrade -u root -p
```

`mysql_upgrade` examines all tables in all databases for incompatibilities with the current version of MySQL. `mysql_upgrade` also upgrades the `mysql` system database so that you can take advantage of new privileges or capabilities.

Note

`mysql_upgrade` does not upgrade the contents of the time zone tables or help tables. For upgrade instructions, see Section 5.1.13, “MySQL Server Time Zone Support”, and Section 5.1.14, “Server-Side Help Support”.

9. Shut down and restart the MySQL server to ensure that any changes made to the system tables take effect. For example:

```
mysqladmin -u root -p shutdown
mysqld_safe --user=mysql --datadir=/path/to/5.6-datadir &
```

2.11.5 Upgrading MySQL with the MySQL Yum Repository

For supported Yum-based platforms (see Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository”, for a list), you can perform an in-place upgrade for MySQL (that is, replacing the old version and then running the new version using the old data files) with the MySQL Yum repository.

Notes

- Before performing any update to MySQL, follow carefully the instructions in Section 2.11, “Upgrading MySQL”. Among other instructions discussed there, it is especially important to back up your database before the update.
- The following instructions assume you have installed MySQL with the MySQL Yum repository or with an RPM package directly downloaded from MySQL Developer Zone's MySQL Download page; if that is not the case, following the instructions in Section 2.5.2, “Replacing a Third-Party Distribution of MySQL Using the MySQL Yum Repository”.

1. Selecting a Target Series

By default, the MySQL Yum repository updates MySQL to the latest version in the release series you have chosen during installation (see Selecting a Release Series for details), which means, for example, a 5.6.x installation is *not* be updated to a 5.7.x release automatically. To update to another release series, you need to first disable the subrepository for the series that has been selected (by default, or by yourself) and enable the subrepository for your target series. To do that, see the general instructions given in Selecting a Release Series. For upgrading from MySQL 5.6 to 5.7, perform the *reverse* of the steps illustrated in Selecting a Release Series, disabling the subrepository for the MySQL 5.6 series and enabling that for the MySQL 5.7 series.

As a general rule, to upgrade from one release series to another, go to the next series rather than skipping a series. For example, if you are currently running MySQL 5.5 and wish to upgrade to 5.7, upgrade to MySQL 5.6 first before upgrading to 5.7.

Important

For important information about upgrading from MySQL 5.6 to 5.7, see Upgrading from MySQL 5.6 to 5.7.

2. Upgrading MySQL

Upgrade MySQL and its components by the following command, for platforms that are not dnf-enabled:

```
sudo yum update mysql-server
```

For platforms that are dnf-enabled:

```
sudo dnf upgrade mysql-server
```

Alternatively, you can update MySQL by telling Yum to update everything on your system, which might take considerably more time. For platforms that are not dnf-enabled:

```
sudo yum update
```

For platforms that are dnf-enabled:

```
sudo dnf upgrade
```

3. Restarting MySQL

The MySQL server always restarts after an update by Yum. Once the server restarts, run `mysql_upgrade` to check and possibly resolve any incompatibilities between the old data and the upgraded software. `mysql_upgrade` also performs other functions; see Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables” for details.

You can also update only a specific component. Use the following command to list all the installed packages for the MySQL components (for dnf-enabled systems, replace `yum` in the command with `dnf`):

```
sudo yum list installed | grep "^mysql"
```

After identifying the package name of the component of your choice, update the package with the following command, replacing *`package-name`* with the name of the package. For platforms that are not dnf-enabled:

```
sudo yum update package-name
```

For dnf-enabled platforms:

```
sudo dnf upgrade package-name
```

Upgrading the Shared Client Libraries

After updating MySQL using the Yum repository, applications compiled with older versions of the shared client libraries should continue to work.

If you recompile applications and dynamically link them with the updated libraries: As typical with new versions of shared libraries where there are differences or additions in symbol versioning between the newer and older libraries (for example, between the newer, standard 5.6 shared client libraries and some older—prior or variant—versions of the shared libraries shipped natively by the Linux distributions' software repositories, or from some other sources), any applications compiled using the updated, newer shared libraries require those updated libraries on systems where the applications are deployed. And, as expected, if those libraries are not in place, the applications requiring the shared libraries fail. So, be sure to deploy the packages for the shared libraries from MySQL on those systems. To do this, add the MySQL Yum repository to the systems (see Adding the MySQL Yum Repository) and install the latest shared libraries using the instructions given in Installing Additional MySQL Products and Components with Yum.

2.11.6 Upgrading MySQL with the MySQL APT Repository

On Debian and Ubuntu platforms, to perform an in-place upgrade of MySQL and its components, use the MySQL APT repository. See Upgrading MySQL with the MySQL APT Repository in A Quick Guide to Using the MySQL APT Repository.

2.11.7 Upgrading MySQL with the MySQL SLES Repository

On the SUSE Linux Enterprise Server (SLES) platform, to perform an in-place upgrade of MySQL and its components, use the MySQL SLES repository. See Upgrading MySQL with the MySQL SLES Repository in A Quick Guide to Using the MySQL SLES Repository.

2.11.8 Upgrading MySQL on Windows

There are two approaches for upgrading MySQL on Windows:

- Using MySQL Installer
- Using the Windows ZIP archive distribution

The approach you select depends on how the existing installation was performed. Before proceeding, review Section 2.11, “Upgrading MySQL” for additional information on upgrading MySQL that is not specific to Windows.

Note

Whichever approach you choose, always back up your current MySQL installation before performing an upgrade. See Section 7.2, “Database Backup Methods”.

Upgrades between milestone releases (or from a milestone release to a GA release) are not supported. Significant development changes take place in milestone releases and you may encounter compatibility issues or problems starting the server. For instructions on how to perform a logical upgrade with a milestone release, see Logical Upgrade.

Note

MySQL Installer does not support upgrades between *Community* releases and *Commercial* releases. If you require this type of upgrade, perform it using the ZIP archive approach.

Upgrading MySQL with MySQL Installer

Performing an upgrade with MySQL Installer is the best approach when the current server installation was performed with it and the upgrade is within the current release series. MySQL Installer does

not support upgrades between release series, such as from 5.5 to 5.6, and it does not provide an upgrade indicator to prompt you to upgrade. For instructions on upgrading between release series, see Upgrading MySQL Using the Windows ZIP Distribution.

To perform an upgrade using MySQL Installer:

1. Start MySQL Installer.
2. From the dashboard, click **Catalog** to download the latest changes to the catalog. The installed server can be upgraded only if the dashboard displays an arrow next to the version number of the server.
3. Click **Upgrade**. All products that have a newer version now appear in a list.

Note

MySQL Installer deselects the server upgrade option for milestone releases (Pre-Release) in the same release series. In addition, it displays a warning to indicate that the upgrade is not supported, identifies the risks of continuing, and provides a summary of the steps to perform a logical upgrade manually. You can reselect server upgrade and proceed at your own risk.

4. Deselect all but the MySQL server product, unless you intend to upgrade other products at this time, and click **Next**.
5. Click **Execute** to start the download. When the download finishes, click **Next** to begin the upgrade operation.
6. Configure the server.

Upgrading MySQL Using the Windows ZIP Distribution

To perform an upgrade using the Windows ZIP archive distribution:

1. Download the latest Windows ZIP Archive distribution of MySQL from https://dev.mysql.com/downloads/.
2. If the server is running, stop it. If the server is installed as a service, stop the service with the following command from the command prompt:

```
C:\> SC STOP mysqld_service_name
```

Alternatively, use `NET STOP mysqld_service_name`.

If you are not running the MySQL server as a service, use `mysqladmin` to stop it. For example, before upgrading from MySQL 5.5 to 5.6, use `mysqladmin` from MySQL 5.5 as follows:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.5\bin\mysqladmin" -u root shutdown
```

Note

If the MySQL `root` user account has a password, invoke `mysqladmin` with the `-p` option and enter the password when prompted.

3. Extract the ZIP archive. You may either overwrite your existing MySQL installation (usually located at `C:\mysql`), or install it into a different directory, such as `C:\mysql5`. Overwriting the existing installation is recommended. However, for upgrades (as opposed to installing for the first time), you must remove the data directory from your existing MySQL installation to avoid replacing your current data files. To do so, follow these steps:

   a. Unzip the ZIP archive in some location other than your current MySQL installation.

b. Remove the data directory.

c. Move the data directory from the current MySQL installation to the location of the just-removed data directory

d. Remove the current MySQL installation

e. Move the unzipped installation to the location of the just-removed installation

4. Restart the server. For example, use the `SC START mysqld_service_name` or `NET START mysqld_service_name` command if you run MySQL as a service, or invoke `mysqld` directly otherwise.

5. As Administrator, run `mysql_upgrade` to check your tables, attempt to repair them if necessary, and update your grant tables if they have changed so that you can take advantage of any new capabilities. See Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”.

6. If you encounter errors, see Section 2.3.5, “Troubleshooting a Microsoft Windows MySQL Server Installation”.

2.11.9 Upgrade Troubleshooting

- If problems occur, such as that the new `mysqld` server does not start or that you cannot connect without a password, verify that you do not have an old `my.cnf` file from your previous installation. You can check this with the `--print-defaults` option (for example, `mysqld --print-defaults`). If this command displays anything other than the program name, you have an active `my.cnf` file that affects server or client operation.

- If, after an upgrade, you experience problems with compiled client programs, such as `Commands out of sync` or unexpected core dumps, you probably have used old header or library files when compiling your programs. In this case, check the date for your `mysql.h` file and `libmysqlclient.a` library to verify that they are from the new MySQL distribution. If not, recompile your programs with the new headers and libraries. Recompilation might also be necessary for programs compiled against the shared client library if the library major version number has changed (for example, from `libmysqlclient.so.15` to `libmysqlclient.so.16`).

- If you have created a user-defined function (UDF) with a given name and upgrade MySQL to a version that implements a new built-in function with the same name, the UDF becomes inaccessible. To correct this, use `DROP FUNCTION` to drop the UDF, and then use `CREATE FUNCTION` to re-create the UDF with a different nonconflicting name. The same is true if the new version of MySQL implements a built-in function with the same name as an existing stored function. See Section 9.2.5, “Function Name Parsing and Resolution”, for the rules describing how the server interprets references to different kinds of functions.

2.11.10 Rebuilding or Repairing Tables or Indexes

This section describes how to rebuild or repair tables or indexes, which may be necessitated by:

- Changes to how MySQL handles data types or character sets. For example, an error in a collation might have been corrected, necessitating a table rebuild to update the indexes for character columns that use the collation.

- Required table repairs or upgrades reported by `CHECK TABLE`, `mysqlcheck`, or `mysql_upgrade`.

Methods for rebuilding a table include:

- Dump and Reload Method

- ALTER TABLE Method

- REPAIR TABLE Method

Dump and Reload Method

If you are rebuilding tables because a different version of MySQL cannot handle them after a binary (in-place) upgrade or downgrade, you must use the dump-and-reload method. Dump the tables *before* upgrading or downgrading using your original version of MySQL. Then reload the tables *after* upgrading or downgrading.

If you use the dump-and-reload method of rebuilding tables only for the purpose of rebuilding indexes, you can perform the dump either before or after upgrading or downgrading. Reloading still must be done afterward.

If you need to rebuild an `InnoDB` table because a `CHECK TABLE` operation indicates that a table upgrade is required, use `mysqldump` to create a dump file and `mysql` to reload the file. If the `CHECK TABLE` operation indicates that there is a corruption or causes `InnoDB` to fail, refer to Section 14.21.2, “Forcing InnoDB Recovery” for information about using the `innodb_force_recovery` option to restart `InnoDB`. To understand the type of problem that `CHECK TABLE` may be encountering, refer to the `InnoDB` notes in Section 13.7.2.2, “CHECK TABLE Statement”.

To rebuild a table by dumping and reloading it, use `mysqldump` to create a dump file and `mysql` to reload the file:

```
mysqldump db_name t1 > dump.sql
mysql db_name < dump.sql
```

To rebuild all the tables in a single database, specify the database name without any following table name:

```
mysqldump db_name > dump.sql
mysql db_name < dump.sql
```

To rebuild all tables in all databases, use the `--all-databases` option:

```
mysqldump --all-databases > dump.sql
mysql < dump.sql
```

ALTER TABLE Method

To rebuild a table with `ALTER TABLE`, use a “null” alteration; that is, an `ALTER TABLE` statement that “changes” the table to use the storage engine that it already has. For example, if `t1` is an `InnoDB` table, use this statement:

```
ALTER TABLE t1 ENGINE = InnoDB;
```

If you are not sure which storage engine to specify in the `ALTER TABLE` statement, use `SHOW CREATE TABLE` to display the table definition.

REPAIR TABLE Method

The `REPAIR TABLE` method is only applicable to `MyISAM`, `ARCHIVE`, and `CSV` tables.

You can use `REPAIR TABLE` if the table checking operation indicates that there is a corruption or that an upgrade is required. For example, to repair a `MyISAM` table, use this statement:

```
REPAIR TABLE t1;
```

`mysqlcheck --repair` provides command-line access to the `REPAIR TABLE` statement. This can be a more convenient means of repairing tables because you can use the `--databases` or `--all-databases` option to repair all tables in specific databases or all databases, respectively:

```
mysqlcheck --repair --databases db_name ...
mysqlcheck --repair --all-databases
```

2.11.11 Copying MySQL Databases to Another Machine

In cases where you need to transfer databases between different architectures, you can use `mysqldump` to create a file containing SQL statements. You can then transfer the file to the other machine and feed it as input to the `mysql` client.

Note

You can copy the `.frm`, `.MYI`, and `.MYD` files for `MyISAM` tables between different architectures that support the same floating-point format. (MySQL takes care of any byte-swapping issues.) See Section 15.2, “The MyISAM Storage Engine”.

Use `mysqldump --help` to see what options are available.

The easiest (although not the fastest) way to move a database between two machines is to run the following commands on the machine on which the database is located:

```
mysqladmin -h 'other_hostname' create db_name
mysqldump db_name | mysql -h 'other_hostname' db_name
```

If you want to copy a database from a remote machine over a slow network, you can use these commands:

```
mysqladmin create db_name
mysqldump -h 'other_hostname' --compress db_name | mysql db_name
```

You can also store the dump in a file, transfer the file to the target machine, and then load the file into the database there. For example, you can dump a database to a compressed file on the source machine like this:

```
mysqldump --quick db_name | gzip > db_name.gz
```

Transfer the file containing the database contents to the target machine and run these commands there:

```
mysqladmin create db_name
gunzip < db_name.gz | mysql db_name
```

You can also use `mysqldump` and `mysqlimport` to transfer the database. For large tables, this is much faster than simply using `mysqldump`. In the following commands, *`DUMPDIR`* represents the full path name of the directory you use to store the output from `mysqldump`.

First, create the directory for the output files and dump the database:

```
mkdir DUMPDIR
mysqldump --tab=DUMPDIR db_name
```

Then transfer the files in the *`DUMPDIR`* directory to some corresponding directory on the target machine and load the files into MySQL there:

```
mysqladmin create db_name           # create database
cat DUMPDIR/*.sql | mysql db_name   # create tables in database
mysqlimport db_name DUMPDIR/*.txt   # load data into tables
```

Do not forget to copy the `mysql` database because that is where the grant tables are stored. You might have to run commands as the MySQL `root` user on the new machine until you have the `mysql` database in place.

After you import the `mysql` database on the new machine, execute `mysqladmin flush-privileges` so that the server reloads the grant table information.

2.12 Downgrading MySQL

This section describes the steps to downgrade a MySQL installation.

Downgrading is a less common operation than upgrade. Downgrading is typically performed because of a compatibility or performance issue that occurs on a of some compatibility or performance issue that occurs on a production system, and was not uncovered during initial upgrade verification on the test systems. As with the upgrade procedure Section 2.11, “Upgrading MySQL”), perform and verify the downgrade procedure on some test systems first, before using it on a production system.

Note

In the following discussion, MySQL commands that must be run using a MySQL account with administrative privileges include `-u root` on the command line to specify the MySQL `root` user. Commands that require a password for `root` also include a `-p` option. Because `-p` is followed by no option value, such commands prompt for the password. Type the password when prompted and press Enter.

SQL statements can be executed using the `mysql` command-line client (connect as `root` to ensure that you have the necessary privileges).

2.12.1 Before You Begin

Review the information in this section before downgrading. Perform any recommended actions.

- Protect your data by taking a backup. The backup should include the `mysql` database, which contains the MySQL system tables. See Section 7.2, “Database Backup Methods”.
- Review Section 2.12.2, “Downgrade Paths” to ensure that your intended downgrade path is supported.
- Review Section 2.12.3, “Downgrade Notes” for items that may require action before downgrading.

Note

The downgrade procedures described in the following sections assume you are downgrading with data files created or modified by the newer MySQL version. However, if you did not modify your data after upgrading, downgrading using backups taken *before* upgrading to the new MySQL version is recommended. Many of the changes described in Section 2.12.3, “Downgrade Notes” that require action are not applicable when downgrading using backups taken *before* upgrading to the new MySQL version.

- Use of new features, new configuration options, or new configuration option values that are not supported by a previous release may cause downgrade errors or failures. Before downgrading, reverse changes resulting from the use of new features and remove configuration settings that are not supported by the release you are downgrading to.

2.12.2 Downgrade Paths

- Downgrade is only supported between General Availability (GA) releases.
- Downgrade from MySQL 5.6 to 5.5 is supported using the *logical downgrade* method.
- Downgrade that skips versions is not supported. For example, downgrading directly from MySQL 5.6 to 5.1 is not supported.

- Downgrade within a release series is supported. For example, downgrading from MySQL 5.6.*z* to 5.6.*y* is supported. Skipping a release is also supported. For example, downgrading from MySQL 5.6.*z* to 5.6.*x* is supported.

2.12.3 Downgrade Notes

Before downgrading from MySQL 5.6, review the information in this section. Some items may require action before downgrading.

System Tables

- The `mysql.user` system table in MySQL 5.6 has a `password_expired` column. The `mysql.user` table in MySQL 5.5 does not. This means that an account with an expired password in MySQL 5.6 works normally in MySQL 5.5.
- The `mysql.host` table was removed in MySQL 5.6.7. When downgrading to a previous release, startup on the downgraded server fails with an error if the `mysql.host` table is not present. You can recreate the table manually or restore it from a backup taken prior to upgrading to MySQL 5.6.7 or higher. To recreate the table manually, retrieve the table definition from a pre-MySQL 5.6.7 instance using `SHOW CREATE TABLE`, or see Bug #73634.

Data Types

- For `TIME`, `DATETIME`, and `TIMESTAMP` columns, the storage required for tables created before MySQL 5.6.4 differs from storage required for tables created in 5.6.4 and later. This is due to a change in 5.6.4 that permits these temporal types to have a fractional part. To downgrade to a version older than 5.6.4, dump affected tables with `mysqldump` before downgrading, and reload the tables after downgrading.

  The following query identifies tables and columns that may be affected by this problem. Some of them are system tables in the `mysql` database (such as `columns_priv` and `proxies_priv`). This means that `mysql` is one of the databases you must dump and reload, or server startup may fail after downgrading.

```
SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, DATA_TYPE
FROM INFORMATION_SCHEMA.COLUMNS
WHERE DATA_TYPE IN ('TIME','DATETIME','TIMESTAMP')
ORDER BY TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME;
```

InnoDB

- `InnoDB` search indexes (with a type of `FULLTEXT`), introduced in MySQL 5.6.4, are not compatible with earlier versions of MySQL, including earlier releases in the 5.6 series. Drop such indexes before performing a downgrade.

  `InnoDB` tables with `FULLTEXT` indexes can be identified using an `INFORMATION_SCHEMA` query. For example:

```
SELECT a.NAME AS Table_name, b.NAME AS Index_name
  FROM INFORMATION_SCHEMA.INNODB_SYS_TABLES a,
       INFORMATION_SCHEMA.INNODB_SYS_INDEXES b
  WHERE a.TABLE_ID = b.TABLE_ID
        AND b.TYPE = 32;
```

- `InnoDB` small page sizes specified by the `innodb_page_size` configuration option, introduced in MySQL 5.6.4, are not compatible with earlier versions of MySQL, including earlier releases in the 5.6 series. Dump all `InnoDB` tables in instances that use a smaller `InnoDB` page size, drop the tables, and re-create and reload them after the downgrade.
- Tables created using persistent statistics table options (`STATS_PERSISTENT`, `STATS_AUTO_RECALC`, and `STATS_SAMPLE_PAGES`) introduced in MySQL 5.6.6, are not compatible with earlier releases (Bug #70778). Remove the options from table definitions prior to

downgrading. For information about these options, see Section 14.8.11.1, “Configuring Persistent Optimizer Statistics Parameters”.

- The `innodb_log_file_size` default and maximum values were increased in MySQL 5.6. Before downgrading, ensure that the configured log file size is compatible with the previous release.
- In MySQL 5.6.3, the length limit for index prefix keys is increased from 767 bytes to 3072 bytes, for `InnoDB` tables using `ROW_FORMAT=DYNAMIC` or `ROW_FORMAT=COMPRESSED`. See Section 14.22, “InnoDB Limits” for details. This change is also backported to MySQL 5.5.14. If you downgrade from one of these releases or higher, to an earlier release with a lower length limit, the index prefix keys could be truncated at 767 bytes or the downgrade could fail. This issue could only occur if the configuration option `innodb_large_prefix` was enabled on the server being downgraded.

Replication

- As of MySQL 5.6, the `relay-log.info` file contains a line count and a replication delay value, so the file format differs from that in older versions. See Section 17.2.2.2, “Replication Metadata Repositories”. If you downgrade a replica server to a version older than MySQL 5.6, the older server does not read the file correctly. To address this, modify the file in a text editor to delete the initial line containing the number of lines.
- Beginning with MySQL 5.6.6, the MySQL Server employs Version 2 binary log events when writing the binary log. Binary logs written using Version 2 log events cannot by read by earlier versions of MySQL Server. To generate a binary log that is written using Version 1 log events readable by older servers, start the MySQL 5.6.6 or later server using `--log-bin-use-v1-row-events=1`, which forces the server to employ Version 1 events when writing the binary log.
- The MySQL 5.6.5 release introduced *global transaction identifiers* (GTIDs) for MySQL Replication. If you enabled GTIDs in MySQL 5.6 and want to downgrade to a MySQL release that does not support GTIDs, you must disable GTIDs before downgrading (see Section 17.1.3.5, “Disabling GTID Transactions”).

2.12.4 Downgrading Binary and Package-based Installations on Unix/Linux

This section describes how to downgrade MySQL binary and package-based installations on Unix/Linux. In-place and logical downgrade methods are described.

- In-Place Downgrade
- Logical Downgrade

In-Place Downgrade

In-place downgrade involves shutting down the new MySQL version, replacing the new MySQL binaries or packages with the old ones, and restarting the old MySQL version on the existing data directory.

In-place downgrade is supported for downgrades between GA releases within the same release series.

In-place downgrade is not supported for MySQL APT, SLES, and Yum repository installations.

To perform an in-place downgrade:

1. Review the information in Section 2.12.1, “Before You Begin”.
2. If you use XA transactions with `InnoDB`, run `XA RECOVER` before downgrading to check for uncommitted XA transactions. If results are returned, either commit or rollback the XA transactions by issuing an `XA COMMIT` or `XA ROLLBACK` statement.
3. If you use `InnoDB`, configure MySQL to perform a slow shutdown by setting `innodb_fast_shutdown` to `0`. For example:

```
mysql -u root -p --execute="SET GLOBAL innodb_fast_shutdown=0"
```

With a slow shutdown, `InnoDB` performs a full purge and change buffer merge before shutting down, which ensures that data files are fully prepared in case of file format differences between releases.

4. Shut down the newer MySQL server. For example:

```
mysqladmin -u root -p shutdown
```

5. After the slow shutdown, remove the `InnoDB` redo log files (the `ib_logfile*` files) from the `data` directory to avoid downgrade issues related to redo log file format changes that may have occurred between releases.

```
rm ib_logfile*
```

6. Downgrade the MySQL binaries or packages in-place by replacing the newer binaries or packages with the older ones.

7. Start the older (downgraded) MySQL server, using the existing data directory. For example:

```
mysqld_safe --user=mysql --datadir=/path/to/existing-datadir
```

8. Run `mysql_upgrade`. For example:

```
mysql_upgrade -u root -p
```

9. Shut down and restart the MySQL server to ensure that any changes made to the system tables take effect. For example:

```
mysqladmin -u root -p shutdown
mysqld_safe --user=mysql --datadir=/path/to/existing-datadir
```

Logical Downgrade

Logical downgrade involves using `mysqldump` to dump all tables from the new MySQL version, and then loading the dump file into the old MySQL version.

Logical downgrades are supported for downgrades between releases within the same release series and for downgrades to the previous release level. Only downgrades between General Availability (GA) releases are supported. Before proceeding, review Section 2.12.1, “Before You Begin”.

Note

For MySQL APT, SLES, and Yum repository installations, only downgrades to the previous release level are supported. Where the instructions call for initializing an older instance, use the package management utility to remove MySQL 5.6 packages and install MySQL 5.5 packages.

To perform a logical downgrade:

1. Review the information in Section 2.12.1, “Before You Begin”.

2. Dump all databases. For example:

```
mysqldump -u root -p
  --add-drop-table --routines --events
  --all-databases --force > data-for-downgrade.sql
```

3. Shut down the newer MySQL server. For example:

```
mysqladmin -u root -p shutdown
```

4. Initialize an older MySQL instance, with a new data directory. For example:

```
scripts/mysql_install_db --user=mysql
```

5. Start the older MySQL server, using the new data directory. For example:

```
mysqld_safe --user=mysql --datadir=/path/to/new-datadir
```

6. Load the dump file into the older MySQL server. For example:

```
mysql -u root -p --force < data-for-upgrade.sql
```

7. Run `mysql_upgrade`. For example:

```
mysql_upgrade -u root -p
```

8. Shut down and restart the MySQL server to ensure that any changes made to the system tables take effect. For example:

```
mysqladmin -u root -p shutdown
mysqld_safe --user=mysql --datadir=/path/to/new-datadir
```

2.12.5 Downgrade Troubleshooting

If you downgrade from one release series to another, there may be incompatibilities in table storage formats. In this case, use `mysqldump` to dump your tables before downgrading. After downgrading, reload the dump file using `mysql` or `mysqlimport` to re-create your tables. For examples, see Section 2.11.11, “Copying MySQL Databases to Another Machine”.

A typical symptom of a downward-incompatible table format change when you downgrade is that you cannot open tables. In that case, use the following procedure:

1. Stop the older MySQL server that you are downgrading to.
2. Restart the newer MySQL server you are downgrading from.
3. Dump any tables that were inaccessible to the older server by using `mysqldump` to create a dump file.
4. Stop the newer MySQL server and restart the older one.
5. Reload the dump file into the older server. Your tables should be accessible.

2.13 Perl Installation Notes

The Perl `DBI` module provides a generic interface for database access. You can write a `DBI` script that works with many different database engines without change. To use `DBI`, you must install the `DBI` module, as well as a DataBase Driver (DBD) module for each type of database server you want to access. For MySQL, this driver is the `DBD::mysql` module.

Perl, and the `DBD::MySQL` module for `DBI` must be installed if you want to run the MySQL benchmark scripts; see Section 8.13.2, “The MySQL Benchmark Suite”.

Note

Perl support is not included with MySQL distributions. You can obtain the necessary modules from http://search.cpan.org for Unix, or by using the ActiveState `ppm` program on Windows. The following sections describe how to do this.

The `DBI`/`DBD` interface requires Perl 5.6.0, and 5.6.1 or later is preferred. DBI *does not work* if you have an older version of Perl. You should use `DBD::mysql` 4.009 or higher. Although earlier versions are available, they do not support the full functionality of MySQL 5.6.

2.13.1 Installing Perl on Unix

MySQL Perl support requires that you have installed MySQL client programming support (libraries and header files). Most installation methods install the necessary files. If you install MySQL from RPM files on Linux, be sure to install the developer RPM as well. The client programs are in the client RPM, but client programming support is in the developer RPM.

The files you need for Perl support can be obtained from the CPAN (Comprehensive Perl Archive Network) at http://search.cpan.org.

The easiest way to install Perl modules on Unix is to use the `CPAN` module. For example:

```
shell> perl -MCPAN -e shell
cpan> install DBI
cpan> install DBD::mysql
```

The `DBD::mysql` installation runs a number of tests. These tests attempt to connect to the local MySQL server using the default user name and password. (The default user name is your login name on Unix, and `ODBC` on Windows. The default password is “no password.”) If you cannot connect to the server with those values (for example, if your account has a password), the tests fail. You can use `force install DBD::mysql` to ignore the failed tests.

`DBI` requires the `Data::Dumper` module. It may be installed; if not, you should install it before installing `DBI`.

It is also possible to download the module distributions in the form of compressed `tar` archives and build the modules manually. For example, to unpack and build a DBI distribution, use a procedure such as this:

1. Unpack the distribution into the current directory:

   ```
   shell> gunzip < DBI-VERSION.tar.gz | tar xvf -
   ```

   This command creates a directory named `DBI-VERSION`.

2. Change location into the top-level directory of the unpacked distribution:

   ```
   shell> cd DBI-VERSION
   ```

3. Build the distribution and compile everything:

   ```
   shell> perl Makefile.PL
   shell> make
   shell> make test
   shell> make install
   ```

The `make test` command is important because it verifies that the module is working. Note that when you run that command during the `DBD::mysql` installation to exercise the interface code, the MySQL server must be running or the test fails.

It is a good idea to rebuild and reinstall the `DBD::mysql` distribution whenever you install a new release of MySQL. This ensures that the latest versions of the MySQL client libraries are installed correctly.

If you do not have access rights to install Perl modules in the system directory or if you want to install local Perl modules, the following reference may be useful: http://learn.perl.org/faq/perlfaq8.html#How-do-I-keep-my-own-module-library-directory-

2.13.2 Installing ActiveState Perl on Windows

On Windows, you should do the following to install the MySQL `DBD` module with ActiveState Perl:

1. Get ActiveState Perl from http://www.activestate.com/Products/ActivePerl/ and install it.

2. Open a console window.
3. If necessary, set the `HTTP_proxy` variable. For example, you might try a setting like this:

```
C:\> set HTTP_proxy=my.proxy.com:3128
```

4. Start the PPM program:

```
C:\> C:\perl\bin\ppm.pl
```

5. If you have not previously done so, install `DBI`:

```
ppm> install DBI
```

6. If this succeeds, run the following command:

```
ppm> install DBD-mysql
```

This procedure should work with ActiveState Perl 5.6 or higher.

If you cannot get the procedure to work, you should install the ODBC driver instead and connect to the MySQL server through ODBC:

```
use DBI;
$dbh= DBI->connect("DBI:ODBC:$dsn",$user,$password) ||
  die "Got error $DBI::errstr when connecting to $dsn\n";
```

2.13.3 Problems Using the Perl DBI/DBD Interface

If Perl reports that it cannot find the `../mysql/mysql.so` module, the problem is probably that Perl cannot locate the `libmysqlclient.so` shared library. You should be able to fix this problem by one of the following methods:

- Copy `libmysqlclient.so` to the directory where your other shared libraries are located (probably `/usr/lib` or `/lib`).
- Modify the `-L` options used to compile `DBD::mysql` to reflect the actual location of `libmysqlclient.so`.
- On Linux, you can add the path name of the directory where `libmysqlclient.so` is located to the `/etc/ld.so.conf` file.
- Add the path name of the directory where `libmysqlclient.so` is located to the `LD_RUN_PATH` environment variable. Some systems use `LD_LIBRARY_PATH` instead.

You may also need to modify the `-L` options if there are other libraries that the linker fails to find. For example, if the linker cannot find `libc` because it is in `/lib` and the link command specifies `-L/usr/lib`, change the `-L` option to `-L/lib` or add `-L/lib` to the existing link command.

If you get the following errors from `DBD::mysql`, you are probably using `gcc` (or using an old binary compiled with `gcc`):

```
/usr/bin/perl: can't resolve symbol '__moddi3'
/usr/bin/perl: can't resolve symbol '__divdi3'
```

Add `-L/usr/lib/gcc-lib/... -lgcc` to the link command when the `mysql.so` library gets built (check the output from `make` for `mysql.so` when you compile the Perl client). The `-L` option should specify the path name of the directory where `libgcc.a` is located on your system.

Another cause of this problem may be that Perl and MySQL are not both compiled with `gcc`. In this case, you can solve the mismatch by compiling both with `gcc`.

Chapter 3 Tutorial

Table of Contents

3.1 Connecting to and Disconnecting from the Server 219
3.2 Entering Queries 220
3.3 Creating and Using a Database 223
3.3.1 Creating and Selecting a Database 224
3.3.2 Creating a Table 225
3.3.3 Loading Data into a Table 226
3.3.4 Retrieving Information from a Table 227
3.4 Getting Information About Databases and Tables 240
3.5 Using mysql in Batch Mode 241
3.6 Examples of Common Queries 242
3.6.1 The Maximum Value for a Column 242
3.6.2 The Row Holding the Maximum of a Certain Column 243
3.6.3 Maximum of Column per Group 243
3.6.4 The Rows Holding the Group-wise Maximum of a Certain Column 243
3.6.5 Using User-Defined Variables 244
3.6.6 Using Foreign Keys 244
3.6.7 Searching on Two Keys 246
3.6.8 Calculating Visits Per Day 246
3.6.9 Using AUTO_INCREMENT 247
3.7 Using MySQL with Apache 249

This chapter provides a tutorial introduction to MySQL by showing how to use the `mysql` client program to create and use a simple database. `mysql` (sometimes referred to as the “terminal monitor” or just “monitor”) is an interactive program that enables you to connect to a MySQL server, run queries, and view the results. `mysql` may also be used in batch mode: you place your queries in a file beforehand, then tell `mysql` to execute the contents of the file. Both ways of using `mysql` are covered here.

To see a list of options provided by `mysql`, invoke it with the `--help` option:

```
shell> mysql --help
```

This chapter assumes that `mysql` is installed on your machine and that a MySQL server is available to which you can connect. If this is not true, contact your MySQL administrator. (If *you* are the administrator, you need to consult the relevant portions of this manual, such as Chapter 5, *MySQL Server Administration*.)

This chapter describes the entire process of setting up and using a database. If you are interested only in accessing an existing database, you may want to skip the sections that describe how to create the database and the tables it contains.

Because this chapter is tutorial in nature, many details are necessarily omitted. Consult the relevant sections of the manual for more information on the topics covered here.

3.1 Connecting to and Disconnecting from the Server

To connect to the server, you usually need to provide a MySQL user name when you invoke `mysql` and, most likely, a password. If the server runs on a machine other than the one where you log in, you also need to specify a host name. Contact your administrator to find out what connection parameters you should use to connect (that is, what host, user name, and password to use). Once you know the proper parameters, you should be able to connect like this:

```
shell> mysql -h host -u user -p
```

```
Enter password: ********
```

host and *user* represent the host name where your MySQL server is running and the user name of your MySQL account. Substitute appropriate values for your setup. The `********` represents your password; enter it when `mysql` displays the `Enter password:` prompt.

If that works, you should see some introductory information followed by a `mysql>` prompt:

```
shell> mysql -h host -u user -p
Enter password: ********
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 25338 to server version: 5.6.51-standard

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

The `mysql>` prompt tells you that `mysql` is ready for you to enter SQL statements.

If you are logging in on the same machine that MySQL is running on, you can omit the host, and simply use the following:

```
shell> mysql -u user -p
```

If, when you attempt to log in, you get an error message such as `ERROR 2002 (HY000): Can't connect to local MySQL server through socket '/tmp/mysql.sock' (2)`, it means that the MySQL server daemon (Unix) or service (Windows) is not running. Consult the administrator or see the section of Chapter 2, *Installing and Upgrading MySQL* that is appropriate to your operating system.

For help with other problems often encountered when trying to log in, see Section B.3.2, “Common Errors When Using MySQL Programs”.

Some MySQL installations permit users to connect as the anonymous (unnamed) user to the server running on the local host. If this is the case on your machine, you should be able to connect to that server by invoking `mysql` without any options:

```
shell> mysql
```

After you have connected successfully, you can disconnect any time by typing `QUIT` (or `\q`) at the `mysql>` prompt:

```
mysql> QUIT
Bye
```

On Unix, you can also disconnect by pressing Control+D.

Most examples in the following sections assume that you are connected to the server. They indicate this by the `mysql>` prompt.

3.2 Entering Queries

Make sure that you are connected to the server, as discussed in the previous section. Doing so does not in itself select any database to work with, but that is okay. At this point, it is more important to find out a little about how to issue queries than to jump right in creating tables, loading data into them, and retrieving data from them. This section describes the basic principles of entering queries, using several queries you can try out to familiarize yourself with how `mysql` works.

Here is a simple query that asks the server to tell you its version number and the current date. Type it in as shown here following the `mysql>` prompt and press Enter:

```
mysql> SELECT VERSION(), CURRENT_DATE;
+--------------+--------------+
```

```
| VERSION()    | CURRENT_DATE |
+--------------+--------------+
| 5.6.1-m4-log | 2010-08-06   |
+--------------+--------------+
1 row in set (0.01 sec)
mysql>
```

This query illustrates several things about `mysql`:

- A query normally consists of an SQL statement followed by a semicolon. (There are some exceptions where a semicolon may be omitted. `QUIT`, mentioned earlier, is one of them. We'll get to others later.)
- When you issue a query, `mysql` sends it to the server for execution and displays the results, then prints another `mysql>` prompt to indicate that it is ready for another query.
- `mysql` displays query output in tabular form (rows and columns). The first row contains labels for the columns. The rows following are the query results. Normally, column labels are the names of the columns you fetch from database tables. If you're retrieving the value of an expression rather than a table column (as in the example just shown), `mysql` labels the column using the expression itself.
- `mysql` shows how many rows were returned and how long the query took to execute, which gives you a rough idea of server performance. These values are imprecise because they represent wall clock time (not CPU or machine time), and because they are affected by factors such as server load and network latency. (For brevity, the “rows in set” line is sometimes not shown in the remaining examples in this chapter.)

Keywords may be entered in any lettercase. The following queries are equivalent:

```
mysql> SELECT VERSION(), CURRENT_DATE;
mysql> select version(), current_date;
mysql> SeLeCt vErSiOn(), current_DATE;
```

Here is another query. It demonstrates that you can use `mysql` as a simple calculator:

```
mysql> SELECT SIN(PI()/4), (4+1)*5;
+------------------+---------+
| SIN(PI()/4)      | (4+1)*5 |
+------------------+---------+
| 0.70710678118655 |      25 |
+------------------+---------+
1 row in set (0.02 sec)
```

The queries shown thus far have been relatively short, single-line statements. You can even enter multiple statements on a single line. Just end each one with a semicolon:

```
mysql> SELECT VERSION(); SELECT NOW();
+--------------+
| VERSION()    |
+--------------+
| 5.6.1-m4-log |
+--------------+
1 row in set (0.00 sec)
+---------------------+
| NOW()               |
+---------------------+
| 2010-08-06 12:17:13 |
+---------------------+
1 row in set (0.00 sec)
```

A query need not be given all on a single line, so lengthy queries that require several lines are not a problem. `mysql` determines where your statement ends by looking for the terminating semicolon, not by looking for the end of the input line. (In other words, `mysql` accepts free-format input: it collects input lines but does not execute them until it sees the semicolon.)

Here is a simple multiple-line statement:

```
mysql> SELECT
    -> USER()
    -> ,
    -> CURRENT_DATE;
+---------------+--------------+
| USER()        | CURRENT_DATE |
+---------------+--------------+
| jon@localhost | 2010-08-06   |
+---------------+--------------+
```

In this example, notice how the prompt changes from `mysql>` to `->` after you enter the first line of a multiple-line query. This is how `mysql` indicates that it has not yet seen a complete statement and is waiting for the rest. The prompt is your friend, because it provides valuable feedback. If you use that feedback, you can always be aware of what `mysql` is waiting for.

If you decide you do not want to execute a query that you are in the process of entering, cancel it by typing `\c`:

```
mysql> SELECT
    -> USER()
    -> \c
mysql>
```

Here, too, notice the prompt. It switches back to `mysql>` after you type `\c`, providing feedback to indicate that `mysql` is ready for a new query.

The following table shows each of the prompts you may see and summarizes what they mean about the state that `mysql` is in.

Prompt	Meaning
`mysql>`	Ready for new query
`->`	Waiting for next line of multiple-line query
`'>`	Waiting for next line, waiting for completion of a string that began with a single quote (`'`)
`">`	Waiting for next line, waiting for completion of a string that began with a double quote (`"`)
`` `> ``	Waiting for next line, waiting for completion of an identifier that began with a backtick (`` ` ``)
`/*>`	Waiting for next line, waiting for completion of a comment that began with `/*`

Multiple-line statements commonly occur by accident when you intend to issue a query on a single line, but forget the terminating semicolon. In this case, `mysql` waits for more input:

```
mysql> SELECT USER()
    ->
```

If this happens to you (you think you've entered a statement but the only response is a `->` prompt), most likely `mysql` is waiting for the semicolon. If you don't notice what the prompt is telling you, you might sit there for a while before realizing what you need to do. Enter a semicolon to complete the statement, and `mysql` executes it:

```
mysql> SELECT USER()
    -> ;
+---------------+
| USER()        |
+---------------+
| jon@localhost |
+---------------+
```

The `'>` and `">` prompts occur during string collection (another way of saying that MySQL is waiting for completion of a string). In MySQL, you can write strings surrounded by either `'` or `"` characters (for example, `'hello'` or `"goodbye"`), and `mysql` lets you enter strings that span multiple lines. When

you see a `'>` or `">` prompt, it means that you have entered a line containing a string that begins with a `'` or `"` quote character, but have not yet entered the matching quote that terminates the string. This often indicates that you have inadvertently left out a quote character. For example:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
    '>
```

If you enter this `SELECT` statement, then press **Enter** and wait for the result, nothing happens. Instead of wondering why this query takes so long, notice the clue provided by the `'>` prompt. It tells you that `mysql` expects to see the rest of an unterminated string. (Do you see the error in the statement? The string `'Smith` is missing the second single quotation mark.)

At this point, what do you do? The simplest thing is to cancel the query. However, you cannot just type `\c` in this case, because `mysql` interprets it as part of the string that it is collecting. Instead, enter the closing quote character (so `mysql` knows you've finished the string), then type `\c`:

```
mysql> SELECT * FROM my_table WHERE name = 'Smith AND age < 30;
    '> '\c
mysql>
```

The prompt changes back to `mysql>`, indicating that `mysql` is ready for a new query.

The `` `> `` prompt is similar to the `'>` and `">` prompts, but indicates that you have begun but not completed a backtick-quoted identifier.

It is important to know what the `'>`, `">`, and `` `> `` prompts signify, because if you mistakenly enter an unterminated string, any further lines you type appear to be ignored by `mysql`—including a line containing `QUIT`. This can be quite confusing, especially if you do not know that you need to supply the terminating quote before you can cancel the current query.

Note

Multiline statements from this point on are written without the secondary (`->` or other) prompts, to make it easier to copy and paste the statements to try for yourself.

3.3 Creating and Using a Database

Once you know how to enter SQL statements, you are ready to access a database.

Suppose that you have several pets in your home (your menagerie) and you would like to keep track of various types of information about them. You can do so by creating tables to hold your data and loading them with the desired information. Then you can answer different sorts of questions about your animals by retrieving data from the tables. This section shows you how to perform the following operations:

- Create a database
- Create a table
- Load data into the table
- Retrieve data from the table in various ways
- Use multiple tables

The menagerie database is simple (deliberately), but it is not difficult to think of real-world situations in which a similar type of database might be used. For example, a database like this could be used by a farmer to keep track of livestock, or by a veterinarian to keep track of patient records. A menagerie distribution containing some of the queries and sample data used in the following sections can be obtained from the MySQL website. It is available in both compressed `tar` file and Zip formats at https://dev.mysql.com/doc/.

Use the SHOW statement to find out what databases currently exist on the server:

```
mysql> SHOW DATABASES;
+----------+
| Database |
+----------+
| mysql    |
| test     |
| tmp      |
+----------+
```

The mysql database describes user access privileges. The test database often is available as a workspace for users to try things out.

The list of databases displayed by the statement may be different on your machine; SHOW DATABASES does not show databases that you have no privileges for if you do not have the SHOW DATABASES privilege. See Section 13.7.5.15, “SHOW DATABASES Statement”.

If the test database exists, try to access it:

```
mysql> USE test
Database changed
```

USE, like QUIT, does not require a semicolon. (You can terminate such statements with a semicolon if you like; it does no harm.) The USE statement is special in another way, too: it must be given on a single line.

You can use the test database (if you have access to it) for the examples that follow, but anything you create in that database can be removed by anyone else with access to it. For this reason, you should probably ask your MySQL administrator for permission to use a database of your own. Suppose that you want to call yours menagerie. The administrator needs to execute a statement like this:

```
mysql> GRANT ALL ON menagerie.* TO 'your_mysql_name'@'your_client_host';
```

where your_mysql_name is the MySQL user name assigned to you and your_client_host is the host from which you connect to the server.

3.3.1 Creating and Selecting a Database

If the administrator creates your database for you when setting up your permissions, you can begin using it. Otherwise, you need to create it yourself:

```
mysql> CREATE DATABASE menagerie;
```

Under Unix, database names are case-sensitive (unlike SQL keywords), so you must always refer to your database as menagerie, not as Menagerie, MENAGERIE, or some other variant. This is also true for table names. (Under Windows, this restriction does not apply, although you must refer to databases and tables using the same lettercase throughout a given query. However, for a variety of reasons, the recommended best practice is always to use the same lettercase that was used when the database was created.)

Note

If you get an error such as ERROR 1044 (42000): Access denied for user 'micah'@'localhost' to database 'menagerie' when attempting to create a database, this means that your user account does not have the necessary privileges to do so. Discuss this with the administrator or see Section 6.2, “Access Control and Account Management”.

Creating a database does not select it for use; you must do that explicitly. To make menagerie the current database, use this statement:

```
mysql> USE menagerie
Database changed
```

Your database needs to be created only once, but you must select it for use each time you begin a `mysql` session. You can do this by issuing a `USE` statement as shown in the example. Alternatively, you can select the database on the command line when you invoke `mysql`. Just specify its name after any connection parameters that you might need to provide. For example:

```
shell> mysql -h host -u user -p menagerie
Enter password: ********
```

Important

`menagerie` in the command just shown is **not** your password. If you want to supply your password on the command line after the `-p` option, you must do so with no intervening space (for example, as `-ppassword`, not as `-p password`). However, putting your password on the command line is not recommended, because doing so exposes it to snooping by other users logged in on your machine.

Note

You can see at any time which database is currently selected using `SELECT DATABASE()`.

3.3.2 Creating a Table

Creating the database is the easy part, but at this point it is empty, as `SHOW TABLES` tells you:

```
mysql> SHOW TABLES;
Empty set (0.00 sec)
```

The harder part is deciding what the structure of your database should be: what tables you need and what columns should be in each of them.

You want a table that contains a record for each of your pets. This can be called the `pet` table, and it should contain, as a bare minimum, each animal's name. Because the name by itself is not very interesting, the table should contain other information. For example, if more than one person in your family keeps pets, you might want to list each animal's owner. You might also want to record some basic descriptive information such as species and sex.

How about age? That might be of interest, but it is not a good thing to store in a database. Age changes as time passes, which means you'd have to update your records often. Instead, it is better to store a fixed value such as date of birth. Then, whenever you need age, you can calculate it as the difference between the current date and the birth date. MySQL provides functions for doing date arithmetic, so this is not difficult. Storing birth date rather than age has other advantages, too:

- You can use the database for tasks such as generating reminders for upcoming pet birthdays. (If you think this type of query is somewhat silly, note that it is the same question you might ask in the context of a business database to identify clients to whom you need to send out birthday greetings in the current week or month, for that computer-assisted personal touch.)
- You can calculate age in relation to dates other than the current date. For example, if you store death date in the database, you can easily calculate how old a pet was when it died.

You can probably think of other types of information that would be useful in the `pet` table, but the ones identified so far are sufficient: name, owner, species, sex, birth, and death.

Use a `CREATE TABLE` statement to specify the layout of your table:

```
mysql> CREATE TABLE pet (name VARCHAR(20), owner VARCHAR(20),
       species VARCHAR(20), sex CHAR(1), birth DATE, death DATE);
```

`VARCHAR` is a good choice for the `name`, `owner`, and `species` columns because the column values vary in length. The lengths in those column definitions need not all be the same, and need not be `20`.

You can normally pick any length from `1` to `65535`, whatever seems most reasonable to you. If you make a poor choice and it turns out later that you need a longer field, MySQL provides an `ALTER TABLE` statement.

Several types of values can be chosen to represent sex in animal records, such as `'m'` and `'f'`, or perhaps `'male'` and `'female'`. It is simplest to use the single characters `'m'` and `'f'`.

The use of the `DATE` data type for the `birth` and `death` columns is a fairly obvious choice.

Once you have created a table, `SHOW TABLES` should produce some output:

```
mysql> SHOW TABLES;
+----------------------+
| Tables in menagerie |
+----------------------+
| pet                  |
+----------------------+
```

To verify that your table was created the way you expected, use a `DESCRIBE` statement:

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

You can use `DESCRIBE` any time, for example, if you forget the names of the columns in your table or what types they have.

For more information about MySQL data types, see Chapter 11, *Data Types*.

3.3.3 Loading Data into a Table

After creating your table, you need to populate it. The `LOAD DATA` and `INSERT` statements are useful for this.

Suppose that your pet records can be described as shown here. (Observe that MySQL expects dates in `'YYYY-MM-DD'` format; this may differ from what you are used to.)

name	owner	species	sex	birth	death
Fluffy	Harold	cat	f	1993-02-04	
Claws	Gwen	cat	m	1994-03-17	
Buffy	Harold	dog	f	1989-05-13	
Fang	Benny	dog	m	1990-08-27	
Bowser	Diane	dog	m	1979-08-31	1995-07-29
Chirpy	Gwen	bird	f	1998-09-11	
Whistler	Gwen	bird		1997-12-09	
Slim	Benny	snake	m	1996-04-29	

Because you are beginning with an empty table, an easy way to populate it is to create a text file containing a row for each of your animals, then load the contents of the file into the table with a single statement.

You could create a text file `pet.txt` containing one record per line, with values separated by tabs, and given in the order in which the columns were listed in the `CREATE TABLE` statement. For missing

values (such as unknown sexes or death dates for animals that are still living), you can use NULL values. To represent these in your text file, use \N (backslash, capital-N). For example, the record for Whistler the bird would look like this (where the whitespace between values is a single tab character):

```
Whistler        Gwen    bird    \N      1997-12-09      \N
```

To load the text file pet.txt into the pet table, use this statement:

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet;
```

If you created the file on Windows with an editor that uses \r\n as a line terminator, you should use this statement instead:

```
mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet
       LINES TERMINATED BY '\r\n';
```

(On an Apple machine running macOS, you would likely want to use LINES TERMINATED BY '\r'.)

You can specify the column value separator and end of line marker explicitly in the LOAD DATA statement if you wish, but the defaults are tab and linefeed. These are sufficient for the statement to read the file pet.txt properly.

If the statement fails, it is likely that your MySQL installation does not have local file capability enabled by default. See Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”, for information on how to change this.

When you want to add new records one at a time, the INSERT statement is useful. In its simplest form, you supply values for each column, in the order in which the columns were listed in the CREATE TABLE statement. Suppose that Diane gets a new hamster named “Puffball.” You could add a new record using an INSERT statement like this:

```
mysql> INSERT INTO pet
       VALUES ('Puffball','Diane','hamster','f','1999-03-30',NULL);
```

String and date values are specified as quoted strings here. Also, with INSERT, you can insert NULL directly to represent a missing value. You do not use \N like you do with LOAD DATA.

From this example, you should be able to see that there would be a lot more typing involved to load your records initially using several INSERT statements rather than a single LOAD DATA statement.

3.3.4 Retrieving Information from a Table

The SELECT statement is used to pull information from a table. The general form of the statement is:

```
SELECT what_to_select
FROM which_table
WHERE conditions_to_satisfy;
```

what_to_select indicates what you want to see. This can be a list of columns, or * to indicate “all columns.” *which_table* indicates the table from which you want to retrieve data. The WHERE clause is optional. If it is present, *conditions_to_satisfy* specifies one or more conditions that rows must satisfy to qualify for retrieval.

3.3.4.1 Selecting All Data

The simplest form of SELECT retrieves everything from a table:

```
mysql> SELECT * FROM pet;
+----------+--------+---------+------+------------+------------+
| name     | owner  | species | sex  | birth      | death      |
+----------+--------+---------+------+------------+------------+
| Fluffy   | Harold | cat     | f    | 1993-02-04 | NULL       |
| Claws    | Gwen   | cat     | m    | 1994-03-17 | NULL       |
| Buffy    | Harold | dog     | f    | 1989-05-13 | NULL       |
| Fang     | Benny  | dog     | m    | 1990-08-27 | NULL       |
```

```
| Bowser   | Diane  | dog     | m    | 1979-08-31 | 1995-07-29 |
| Chirpy   | Gwen   | bird    | f    | 1998-09-11 | NULL       |
| Whistler | Gwen   | bird    | NULL | 1997-12-09 | NULL       |
| Slim     | Benny  | snake   | m    | 1996-04-29 | NULL       |
| Puffball | Diane  | hamster | f    | 1999-03-30 | NULL       |
+----------+--------+---------+------+------------+------------+
```

This form of `SELECT` uses `*`, which is shorthand for “select all columns.” This is useful if you want to review your entire table, for example, after you've just loaded it with your initial data set. For example, you may happen to think that the birth date for Bowser doesn't seem quite right. Consulting your original pedigree papers, you find that the correct birth year should be 1989, not 1979.

There are at least two ways to fix this:

- Edit the file `pet.txt` to correct the error, then empty the table and reload it using `DELETE` and `LOAD DATA`:

  ```
  mysql> DELETE FROM pet;
  mysql> LOAD DATA LOCAL INFILE '/path/pet.txt' INTO TABLE pet;
  ```

  However, if you do this, you must also re-enter the record for Puffball.

- Fix only the erroneous record with an `UPDATE` statement:

  ```
  mysql> UPDATE pet SET birth = '1989-08-31' WHERE name = 'Bowser';
  ```

  The `UPDATE` changes only the record in question and does not require you to reload the table.

3.3.4.2 Selecting Particular Rows

As shown in the preceding section, it is easy to retrieve an entire table. Just omit the `WHERE` clause from the `SELECT` statement. But typically you don't want to see the entire table, particularly when it becomes large. Instead, you're usually more interested in answering a particular question, in which case you specify some constraints on the information you want. Let's look at some selection queries in terms of questions about your pets that they answer.

You can select only particular rows from your table. For example, if you want to verify the change that you made to Bowser's birth date, select Bowser's record like this:

```
mysql> SELECT * FROM pet WHERE name = 'Bowser';
+--------+-------+---------+------+------------+------------+
| name   | owner | species | sex  | birth      | death      |
+--------+-------+---------+------+------------+------------+
| Bowser | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+-------+---------+------+------------+------------+
```

The output confirms that the year is correctly recorded as 1989, not 1979.

String comparisons normally are case-insensitive, so you can specify the name as `'bowser'`, `'BOWSER'`, and so forth. The query result is the same.

You can specify conditions on any column, not just `name`. For example, if you want to know which animals were born during or after 1998, test the `birth` column:

```
mysql> SELECT * FROM pet WHERE birth >= '1998-1-1';
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Chirpy   | Gwen  | bird    | f    | 1998-09-11 | NULL  |
| Puffball | Diane | hamster | f    | 1999-03-30 | NULL  |
+----------+-------+---------+------+------------+-------+
```

You can combine conditions, for example, to locate female dogs:

```
mysql> SELECT * FROM pet WHERE species = 'dog' AND sex = 'f';
+-------+--------+---------+------+------------+-------+
```

```
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

The preceding query uses the `AND` logical operator. There is also an `OR` operator:

```
mysql> SELECT * FROM pet WHERE species = 'snake' OR species = 'bird';
+----------+-------+---------+------+------------+-------+
| name     | owner | species | sex  | birth      | death |
+----------+-------+---------+------+------------+-------+
| Chirpy   | Gwen  | bird    | f    | 1998-09-11 | NULL  |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL  |
| Slim     | Benny | snake   | m    | 1996-04-29 | NULL  |
+----------+-------+---------+------+------------+-------+
```

`AND` and `OR` may be intermixed, although `AND` has higher precedence than `OR`. If you use both operators, it is a good idea to use parentheses to indicate explicitly how conditions should be grouped:

```
mysql> SELECT * FROM pet WHERE (species = 'cat' AND sex = 'm')
       OR (species = 'dog' AND sex = 'f');
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

3.3.4.3 Selecting Particular Columns

If you do not want to see entire rows from your table, just name the columns in which you are interested, separated by commas. For example, if you want to know when your animals were born, select the `name` and `birth` columns:

```
mysql> SELECT name, birth FROM pet;
+----------+------------+
| name     | birth      |
+----------+------------+
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Buffy    | 1989-05-13 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Puffball | 1999-03-30 |
+----------+------------+
```

To find out who owns pets, use this query:

```
mysql> SELECT owner FROM pet;
+--------+
| owner  |
+--------+
| Harold |
| Gwen   |
| Harold |
| Benny  |
| Diane  |
| Gwen   |
| Gwen   |
| Benny  |
| Diane  |
+--------+
```

Notice that the query simply retrieves the `owner` column from each record, and some of them appear more than once. To minimize the output, retrieve each unique output record just once by adding the keyword `DISTINCT`:

```
mysql> SELECT DISTINCT owner FROM pet;
+--------+
| owner  |
+--------+
| Benny  |
| Diane  |
| Gwen   |
| Harold |
+--------+
```

You can use a `WHERE` clause to combine row selection with column selection. For example, to get birth dates for dogs and cats only, use this query:

```
mysql> SELECT name, species, birth FROM pet
       WHERE species = 'dog' OR species = 'cat';
+--------+---------+------------+
| name   | species | birth      |
+--------+---------+------------+
| Fluffy | cat     | 1993-02-04 |
| Claws  | cat     | 1994-03-17 |
| Buffy  | dog     | 1989-05-13 |
| Fang   | dog     | 1990-08-27 |
| Bowser | dog     | 1989-08-31 |
+--------+---------+------------+
```

3.3.4.4 Sorting Rows

You may have noticed in the preceding examples that the result rows are displayed in no particular order. It is often easier to examine query output when the rows are sorted in some meaningful way. To sort a result, use an `ORDER BY` clause.

Here are animal birthdays, sorted by date:

```
mysql> SELECT name, birth FROM pet ORDER BY birth;
+----------+------------+
| name     | birth      |
+----------+------------+
| Buffy    | 1989-05-13 |
| Bowser   | 1989-08-31 |
| Fang     | 1990-08-27 |
| Fluffy   | 1993-02-04 |
| Claws    | 1994-03-17 |
| Slim     | 1996-04-29 |
| Whistler | 1997-12-09 |
| Chirpy   | 1998-09-11 |
| Puffball | 1999-03-30 |
+----------+------------+
```

On character type columns, sorting—like all other comparison operations—is normally performed in a case-insensitive fashion. This means that the order is undefined for columns that are identical except for their case. You can force a case-sensitive sort for a column by using `BINARY` like so: `ORDER BY BINARY` *`col_name`*.

The default sort order is ascending, with smallest values first. To sort in reverse (descending) order, add the `DESC` keyword to the name of the column you are sorting by:

```
mysql> SELECT name, birth FROM pet ORDER BY birth DESC;
+----------+------------+
| name     | birth      |
+----------+------------+
| Puffball | 1999-03-30 |
| Chirpy   | 1998-09-11 |
| Whistler | 1997-12-09 |
| Slim     | 1996-04-29 |
| Claws    | 1994-03-17 |
| Fluffy   | 1993-02-04 |
| Fang     | 1990-08-27 |
| Bowser   | 1989-08-31 |
| Buffy    | 1989-05-13 |
```

```
+----------+------------+
```

You can sort on multiple columns, and you can sort different columns in different directions. For example, to sort by type of animal in ascending order, then by birth date within animal type in descending order (youngest animals first), use the following query:

```
mysql> SELECT name, species, birth FROM pet
       ORDER BY species, birth DESC;
+----------+---------+------------+
| name     | species | birth      |
+----------+---------+------------+
| Chirpy   | bird    | 1998-09-11 |
| Whistler | bird    | 1997-12-09 |
| Claws    | cat     | 1994-03-17 |
| Fluffy   | cat     | 1993-02-04 |
| Fang     | dog     | 1990-08-27 |
| Bowser   | dog     | 1989-08-31 |
| Buffy    | dog     | 1989-05-13 |
| Puffball | hamster | 1999-03-30 |
| Slim     | snake   | 1996-04-29 |
+----------+---------+------------+
```

The `DESC` keyword applies only to the column name immediately preceding it (`birth`); it does not affect the `species` column sort order.

3.3.4.5 Date Calculations

MySQL provides several functions that you can use to perform calculations on dates, for example, to calculate ages or extract parts of dates.

To determine how many years old each of your pets is, use the `TIMESTAMPDIFF()` function. Its arguments are the unit in which you want the result expressed, and the two dates for which to take the difference. The following query shows, for each pet, the birth date, the current date, and the age in years. An *alias* (`age`) is used to make the final output column label more meaningful.

```
mysql> SELECT name, birth, CURDATE(),
       TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age
       FROM pet;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
+----------+------------+------------+------+
```

The query works, but the result could be scanned more easily if the rows were presented in some order. This can be done by adding an `ORDER BY name` clause to sort the output by name:

```
mysql> SELECT name, birth, CURDATE(),
       TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age
       FROM pet ORDER BY name;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
```

```
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
+----------+------------+------------+------+
```

To sort the output by `age` rather than `name`, just use a different `ORDER BY` clause:

```
mysql> SELECT name, birth, CURDATE(),
       TIMESTAMPDIFF(YEAR,birth,CURDATE()) AS age
       FROM pet ORDER BY age;
+----------+------------+------------+------+
| name     | birth      | CURDATE()  | age  |
+----------+------------+------------+------+
| Chirpy   | 1998-09-11 | 2003-08-19 |    4 |
| Puffball | 1999-03-30 | 2003-08-19 |    4 |
| Whistler | 1997-12-09 | 2003-08-19 |    5 |
| Slim     | 1996-04-29 | 2003-08-19 |    7 |
| Claws    | 1994-03-17 | 2003-08-19 |    9 |
| Fluffy   | 1993-02-04 | 2003-08-19 |   10 |
| Fang     | 1990-08-27 | 2003-08-19 |   12 |
| Bowser   | 1989-08-31 | 2003-08-19 |   13 |
| Buffy    | 1989-05-13 | 2003-08-19 |   14 |
+----------+------------+------------+------+
```

A similar query can be used to determine age at death for animals that have died. You determine which animals these are by checking whether the `death` value is `NULL`. Then, for those with non-`NULL` values, compute the difference between the `death` and `birth` values:

```
mysql> SELECT name, birth, death,
       TIMESTAMPDIFF(YEAR,birth,death) AS age
       FROM pet WHERE death IS NOT NULL ORDER BY age;
+--------+------------+------------+------+
| name   | birth      | death      | age  |
+--------+------------+------------+------+
| Bowser | 1989-08-31 | 1995-07-29 |    5 |
+--------+------------+------------+------+
```

The query uses `death IS NOT NULL` rather than `death <> NULL` because `NULL` is a special value that cannot be compared using the usual comparison operators. This is discussed later. See Section 3.3.4.6, “Working with NULL Values”.

What if you want to know which animals have birthdays next month? For this type of calculation, year and day are irrelevant; you simply want to extract the month part of the `birth` column. MySQL provides several functions for extracting parts of dates, such as `YEAR()`, `MONTH()`, and `DAYOFMONTH()`. `MONTH()` is the appropriate function here. To see how it works, run a simple query that displays the value of both `birth` and `MONTH(birth)`:

```
mysql> SELECT name, birth, MONTH(birth) FROM pet;
+----------+------------+--------------+
| name     | birth      | MONTH(birth) |
+----------+------------+--------------+
| Fluffy   | 1993-02-04 |            2 |
| Claws    | 1994-03-17 |            3 |
| Buffy    | 1989-05-13 |            5 |
| Fang     | 1990-08-27 |            8 |
| Bowser   | 1989-08-31 |            8 |
| Chirpy   | 1998-09-11 |            9 |
| Whistler | 1997-12-09 |           12 |
| Slim     | 1996-04-29 |            4 |
| Puffball | 1999-03-30 |            3 |
+----------+------------+--------------+
```

Finding animals with birthdays in the upcoming month is also simple. Suppose that the current month is April. Then the month value is `4` and you can look for animals born in May (month `5`) like this:

```
mysql> SELECT name, birth FROM pet WHERE MONTH(birth) = 5;
+-------+------------+
| name  | birth      |
+-------+------------+
| Buffy | 1989-05-13 |
```

```
+-------+------------+
```

There is a small complication if the current month is December. You cannot merely add one to the month number (`12`) and look for animals born in month `13`, because there is no such month. Instead, you look for animals born in January (month `1`).

You can write the query so that it works no matter what the current month is, so that you do not have to use the number for a particular month. `DATE_ADD()` enables you to add a time interval to a given date. If you add a month to the value of `CURDATE()`, then extract the month part with `MONTH()`, the result produces the month in which to look for birthdays:

```
mysql> SELECT name, birth FROM pet
       WHERE MONTH(birth) = MONTH(DATE_ADD(CURDATE(),INTERVAL 1 MONTH));
```

A different way to accomplish the same task is to add `1` to get the next month after the current one after using the modulo function (`MOD`) to wrap the month value to `0` if it is currently `12`:

```
mysql> SELECT name, birth FROM pet
       WHERE MONTH(birth) = MOD(MONTH(CURDATE()), 12) + 1;
```

`MONTH()` returns a number between `1` and `12`. And `MOD(something,12)` returns a number between `0` and `11`. So the addition has to be after the `MOD()`, otherwise we would go from November (`11`) to January (`1`).

If a calculation uses invalid dates, the calculation fails and produces warnings:

```
mysql> SELECT '2018-10-31' + INTERVAL 1 DAY;
+-------------------------------+
| '2018-10-31' + INTERVAL 1 DAY |
+-------------------------------+
| 2018-11-01                    |
+-------------------------------+
mysql> SELECT '2018-10-32' + INTERVAL 1 DAY;
+-------------------------------+
| '2018-10-32' + INTERVAL 1 DAY |
+-------------------------------+
| NULL                          |
+-------------------------------+
mysql> SHOW WARNINGS;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1292 | Incorrect datetime value: '2018-10-32' |
+---------+------+----------------------------------------+
```

3.3.4.6 Working with NULL Values

The `NULL` value can be surprising until you get used to it. Conceptually, `NULL` means “a missing unknown value” and it is treated somewhat differently from other values.

To test for `NULL`, use the `IS NULL` and `IS NOT NULL` operators, as shown here:

```
mysql> SELECT 1 IS NULL, 1 IS NOT NULL;
+-----------+---------------+
| 1 IS NULL | 1 IS NOT NULL |
+-----------+---------------+
|         0 |             1 |
+-----------+---------------+
```

You cannot use arithmetic comparison operators such as `=`, `<`, or `<>` to test for `NULL`. To demonstrate this for yourself, try the following query:

```
mysql> SELECT 1 = NULL, 1 <> NULL, 1 < NULL, 1 > NULL;
+----------+-----------+----------+----------+
| 1 = NULL | 1 <> NULL | 1 < NULL | 1 > NULL |
+----------+-----------+----------+----------+
|     NULL |      NULL |     NULL |     NULL |
```

```
+----------+-----------+----------+----------+
```

Because the result of any arithmetic comparison with `NULL` is also `NULL`, you cannot obtain any meaningful results from such comparisons.

In MySQL, `0` or `NULL` means false and anything else means true. The default truth value from a boolean operation is `1`.

This special treatment of `NULL` is why, in the previous section, it was necessary to determine which animals are no longer alive using `death IS NOT NULL` instead of `death <> NULL`.

Two `NULL` values are regarded as equal in a `GROUP BY`.

When doing an `ORDER BY`, `NULL` values are presented first if you do `ORDER BY ... ASC` and last if you do `ORDER BY ... DESC`.

A common error when working with `NULL` is to assume that it is not possible to insert a zero or an empty string into a column defined as `NOT NULL`, but this is not the case. These are in fact values, whereas `NULL` means "not having a value." You can test this easily enough by using `IS [NOT] NULL` as shown:

```
mysql> SELECT 0 IS NULL, 0 IS NOT NULL, '' IS NULL, '' IS NOT NULL;
+-----------+---------------+------------+----------------+
| 0 IS NULL | 0 IS NOT NULL | '' IS NULL | '' IS NOT NULL |
+-----------+---------------+------------+----------------+
|         0 |             1 |          0 |              1 |
+-----------+---------------+------------+----------------+
```

Thus it is entirely possible to insert a zero or empty string into a `NOT NULL` column, as these are in fact `NOT NULL`. See Section B.3.4.3, "Problems with NULL Values".

3.3.4.7 Pattern Matching

MySQL provides standard SQL pattern matching as well as a form of pattern matching based on extended regular expressions similar to those used by Unix utilities such as `vi`, `grep`, and `sed`.

SQL pattern matching enables you to use `_` to match any single character and `%` to match an arbitrary number of characters (including zero characters). In MySQL, SQL patterns are case-insensitive by default. Some examples are shown here. Do not use `=` or `<>` when you use SQL patterns. Use the `LIKE` or `NOT LIKE` comparison operators instead.

To find names beginning with `b`:

```
mysql> SELECT * FROM pet WHERE name LIKE 'b%';
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

To find names ending with `fy`:

```
mysql> SELECT * FROM pet WHERE name LIKE '%fy';
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

To find names containing a `w`:

```
mysql> SELECT * FROM pet WHERE name LIKE '%w%';
+----------+-------+---------+------+------------+------------+
```

```
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

To find names containing exactly five characters, use five instances of the `_` pattern character:

```
mysql> SELECT * FROM pet WHERE name LIKE '_____';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

The other type of pattern matching provided by MySQL uses extended regular expressions. When you test for a match for this type of pattern, use the `REGEXP` and `NOT REGEXP` operators (or `RLIKE` and `NOT RLIKE`, which are synonyms).

The following list describes some characteristics of extended regular expressions:

- `.` matches any single character.
- A character class `[...]` matches any character within the brackets. For example, `[abc]` matches `a`, `b`, or `c`. To name a range of characters, use a dash. `[a-z]` matches any letter, whereas `[0-9]` matches any digit.
- `*` matches zero or more instances of the thing preceding it. For example, `x*` matches any number of `x` characters, `[0-9]*` matches any number of digits, and `.*` matches any number of anything.
- A regular expression pattern match succeeds if the pattern matches anywhere in the value being tested. (This differs from a `LIKE` pattern match, which succeeds only if the pattern matches the entire value.)
- To anchor a pattern so that it must match the beginning or end of the value being tested, use `^` at the beginning or `$` at the end of the pattern.

To demonstrate how extended regular expressions work, the `LIKE` queries shown previously are rewritten here to use `REGEXP`.

To find names beginning with `b`, use `^` to match the beginning of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^b';
+--------+--------+---------+------+------------+------------+
| name   | owner  | species | sex  | birth      | death      |
+--------+--------+---------+------+------------+------------+
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL       |
| Bowser | Diane  | dog     | m    | 1989-08-31 | 1995-07-29 |
+--------+--------+---------+------+------------+------------+
```

To force a `REGEXP` comparison to be case-sensitive, use the `BINARY` keyword to make one of the strings a binary string. This query matches only lowercase `b` at the beginning of a name:

```
SELECT * FROM pet WHERE name REGEXP BINARY '^b';
```

To find names ending with `fy`, use `$` to match the end of the name:

```
mysql> SELECT * FROM pet WHERE name REGEXP 'fy$';
+--------+--------+---------+------+------------+-------+
| name   | owner  | species | sex  | birth      | death |
+--------+--------+---------+------+------------+-------+
| Fluffy | Harold | cat     | f    | 1993-02-04 | NULL  |
| Buffy  | Harold | dog     | f    | 1989-05-13 | NULL  |
+--------+--------+---------+------+------------+-------+
```

To find names containing a `w`, use this query:

```
mysql> SELECT * FROM pet WHERE name REGEXP 'w';
+----------+-------+---------+------+------------+------------+
| name     | owner | species | sex  | birth      | death      |
+----------+-------+---------+------+------------+------------+
| Claws    | Gwen  | cat     | m    | 1994-03-17 | NULL       |
| Bowser   | Diane | dog     | m    | 1989-08-31 | 1995-07-29 |
| Whistler | Gwen  | bird    | NULL | 1997-12-09 | NULL       |
+----------+-------+---------+------+------------+------------+
```

Because a regular expression pattern matches if it occurs anywhere in the value, it is not necessary in the previous query to put a wildcard on either side of the pattern to get it to match the entire value as would be true with an SQL pattern.

To find names containing exactly five characters, use `^` and `$` to match the beginning and end of the name, and five instances of `.` in between:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^.....$';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

You could also write the previous query using the `{`*n*`}` (“repeat-*n*-times”) operator:

```
mysql> SELECT * FROM pet WHERE name REGEXP '^.{5}$';
+-------+--------+---------+------+------------+-------+
| name  | owner  | species | sex  | birth      | death |
+-------+--------+---------+------+------------+-------+
| Claws | Gwen   | cat     | m    | 1994-03-17 | NULL  |
| Buffy | Harold | dog     | f    | 1989-05-13 | NULL  |
+-------+--------+---------+------+------------+-------+
```

For more information about the syntax for regular expressions, see Section 12.8.2, “Regular Expressions”.

3.3.4.8 Counting Rows

Databases are often used to answer the question, “How often does a certain type of data occur in a table?” For example, you might want to know how many pets you have, or how many pets each owner has, or you might want to perform various kinds of census operations on your animals.

Counting the total number of animals you have is the same question as “How many rows are in the `pet` table?” because there is one record per pet. `COUNT(*)` counts the number of rows, so the query to count your animals looks like this:

```
mysql> SELECT COUNT(*) FROM pet;
+----------+
| COUNT(*) |
+----------+
|        9 |
+----------+
```

Earlier, you retrieved the names of the people who owned pets. You can use `COUNT()` if you want to find out how many pets each owner has:

```
mysql> SELECT owner, COUNT(*) FROM pet GROUP BY owner;
+--------+----------+
| owner  | COUNT(*) |
+--------+----------+
| Benny  |        2 |
| Diane  |        2 |
| Gwen   |        3 |
| Harold |        2 |
```

```
+--------+----------+
```

The preceding query uses `GROUP BY` to group all records for each `owner`. The use of `COUNT()` in conjunction with `GROUP BY` is useful for characterizing your data under various groupings. The following examples show different ways to perform animal census operations.

Number of animals per species:

```
mysql> SELECT species, COUNT(*) FROM pet GROUP BY species;
+---------+----------+
| species | COUNT(*) |
+---------+----------+
| bird    |        2 |
| cat     |        2 |
| dog     |        3 |
| hamster |        1 |
| snake   |        1 |
+---------+----------+
```

Number of animals per sex:

```
mysql> SELECT sex, COUNT(*) FROM pet GROUP BY sex;
+------+----------+
| sex  | COUNT(*) |
+------+----------+
| NULL |        1 |
| f    |        4 |
| m    |        4 |
+------+----------+
```

(In this output, `NULL` indicates that the sex is unknown.)

Number of animals per combination of species and sex:

```
mysql> SELECT species, sex, COUNT(*) FROM pet GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| bird    | NULL |        1 |
| bird    | f    |        1 |
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
| hamster | f    |        1 |
| snake   | m    |        1 |
+---------+------+----------+
```

You need not retrieve an entire table when you use `COUNT()`. For example, the previous query, when performed just on dogs and cats, looks like this:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
       WHERE species = 'dog' OR species = 'cat'
       GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
+---------+------+----------+
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
+---------+------+----------+
```

Or, if you wanted the number of animals per sex only for animals whose sex is known:

```
mysql> SELECT species, sex, COUNT(*) FROM pet
       WHERE sex IS NOT NULL
       GROUP BY species, sex;
+---------+------+----------+
| species | sex  | COUNT(*) |
```

```
+---------+------+----------+
| bird    | f    |        1 |
| cat     | f    |        1 |
| cat     | m    |        1 |
| dog     | f    |        1 |
| dog     | m    |        2 |
| hamster | f    |        1 |
| snake   | m    |        1 |
+---------+------+----------+
```

If you name columns to select in addition to the `COUNT()` value, a `GROUP BY` clause should be present that names those same columns. Otherwise, the following occurs:

- If the `ONLY_FULL_GROUP_BY` SQL mode is enabled, an error occurs:

```
mysql> SET sql_mode = 'ONLY_FULL_GROUP_BY';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT owner, COUNT(*) FROM pet;
ERROR 1140 (42000): Mixing of GROUP columns (MIN(),MAX(),COUNT()...)
with no GROUP columns is illegal if there is no GROUP BY clause
```

- If `ONLY_FULL_GROUP_BY` is not enabled, the query is processed by treating all rows as a single group, but the value selected for each named column is nondeterministic. The server is free to select the value from any row:

```
mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT owner, COUNT(*) FROM pet;
+--------+----------+
| owner  | COUNT(*) |
+--------+----------+
| Harold |        8 |
+--------+----------+
1 row in set (0.00 sec)
```

See also Section 12.19.3, “MySQL Handling of GROUP BY”. See Section 12.19.1, “Aggregate Function Descriptions” for information about `COUNT(expr)` behavior and related optimizations.

3.3.4.9 Using More Than one Table

The `pet` table keeps track of which pets you have. If you want to record other information about them, such as events in their lives like visits to the vet or when litters are born, you need another table. What should this table look like? It needs to contain the following information:

- The pet name so that you know which animal each event pertains to.
- A date so that you know when the event occurred.
- A field to describe the event.
- An event type field, if you want to be able to categorize events.

Given these considerations, the `CREATE TABLE` statement for the `event` table might look like this:

```
mysql> CREATE TABLE event (name VARCHAR(20), date DATE,
       type VARCHAR(15), remark VARCHAR(255));
```

As with the `pet` table, it is easiest to load the initial records by creating a tab-delimited text file containing the following information.

name	date	type	remark
Fluffy	1995-05-15	litter	4 kittens, 3 female, 1 male
Buffy	1993-06-23	litter	5 puppies, 2 female, 3 male

name	date	type	remark
Buffy	1994-06-19	litter	3 puppies, 3 female
Chirpy	1999-03-21	vet	needed beak straightened
Slim	1997-08-03	vet	broken rib
Bowser	1991-10-12	kennel	
Fang	1991-10-12	kennel	
Fang	1998-08-28	birthday	Gave him a new chew toy
Claws	1998-03-17	birthday	Gave him a new flea collar
Whistler	1998-12-09	birthday	First birthday

Load the records like this:

```
mysql> LOAD DATA LOCAL INFILE 'event.txt' INTO TABLE event;
```

Based on what you have learned from the queries that you have run on the `pet` table, you should be able to perform retrievals on the records in the `event` table; the principles are the same. But when is the `event` table by itself insufficient to answer questions you might ask?

Suppose that you want to find out the ages at which each pet had its litters. We saw earlier how to calculate ages from two dates. The litter date of the mother is in the `event` table, but to calculate her age on that date you need her birth date, which is stored in the `pet` table. This means the query requires both tables:

```
mysql> SELECT pet.name,
       TIMESTAMPDIFF(YEAR,birth,date) AS age,
       remark
       FROM pet INNER JOIN event
         ON pet.name = event.name
       WHERE event.type = 'litter';
+--------+------+-----------------------------+
| name   | age  | remark                      |
+--------+------+-----------------------------+
| Fluffy |    2 | 4 kittens, 3 female, 1 male |
| Buffy  |    4 | 5 puppies, 2 female, 3 male |
| Buffy  |    5 | 3 puppies, 3 female         |
+--------+------+-----------------------------+
```

There are several things to note about this query:

- The `FROM` clause joins two tables because the query needs to pull information from both of them.
- When combining (joining) information from multiple tables, you need to specify how records in one table can be matched to records in the other. This is easy because they both have a `name` column. The query uses an `ON` clause to match up records in the two tables based on the `name` values.

  The query uses an `INNER JOIN` to combine the tables. An `INNER JOIN` permits rows from either table to appear in the result if and only if both tables meet the conditions specified in the `ON` clause. In this example, the `ON` clause specifies that the `name` column in the `pet` table must match the `name` column in the `event` table. If a name appears in one table but not the other, the row does not appear in the result because the condition in the `ON` clause fails.
- Because the `name` column occurs in both tables, you must be specific about which table you mean when referring to the column. This is done by prepending the table name to the column name.

You need not have two different tables to perform a join. Sometimes it is useful to join a table to itself, if you want to compare records in a table to other records in that same table. For example, to find breeding pairs among your pets, you can join the `pet` table with itself to produce candidate pairs of live males and females of like species:

```
mysql> SELECT p1.name, p1.sex, p2.name, p2.sex, p1.species
```

```
       FROM pet AS p1 INNER JOIN pet AS p2
         ON p1.species = p2.species
         AND p1.sex = 'f' AND p1.death IS NULL
         AND p2.sex = 'm' AND p2.death IS NULL;
+--------+------+-------+------+---------+
| name   | sex  | name  | sex  | species |
+--------+------+-------+------+---------+
| Fluffy | f    | Claws | m    | cat     |
| Buffy  | f    | Fang  | m    | dog     |
+--------+------+-------+------+---------+
```

In this query, we specify aliases for the table name to refer to the columns and keep straight which instance of the table each column reference is associated with.

3.4 Getting Information About Databases and Tables

What if you forget the name of a database or table, or what the structure of a given table is (for example, what its columns are called)? MySQL addresses this problem through several statements that provide information about the databases and tables it supports.

You have previously seen `SHOW DATABASES`, which lists the databases managed by the server. To find out which database is currently selected, use the `DATABASE()` function:

```
mysql> SELECT DATABASE();
+------------+
| DATABASE() |
+------------+
| menagerie  |
+------------+
```

If you have not yet selected any database, the result is `NULL`.

To find out what tables the default database contains (for example, when you are not sure about the name of a table), use this statement:

```
mysql> SHOW TABLES;
+----------------------+
| Tables_in_menagerie |
+----------------------+
| event                |
| pet                  |
+----------------------+
```

The name of the column in the output produced by this statement is always `Tables_in_`*`db_name`*, where *`db_name`* is the name of the database. See Section 13.7.5.38, “SHOW TABLES Statement”, for more information.

If you want to find out about the structure of a table, the `DESCRIBE` statement is useful; it displays information about each of a table's columns:

```
mysql> DESCRIBE pet;
+---------+-------------+------+-----+---------+-------+
| Field   | Type        | Null | Key | Default | Extra |
+---------+-------------+------+-----+---------+-------+
| name    | varchar(20) | YES  |     | NULL    |       |
| owner   | varchar(20) | YES  |     | NULL    |       |
| species | varchar(20) | YES  |     | NULL    |       |
| sex     | char(1)     | YES  |     | NULL    |       |
| birth   | date        | YES  |     | NULL    |       |
| death   | date        | YES  |     | NULL    |       |
+---------+-------------+------+-----+---------+-------+
```

`Field` indicates the column name, `Type` is the data type for the column, `NULL` indicates whether the column can contain `NULL` values, `Key` indicates whether the column is indexed, and `Default` specifies the column's default value. `Extra` displays special information about columns: If a column was created with the `AUTO_INCREMENT` option, the value is `auto_increment` rather than empty.

`DESC` is a short form of `DESCRIBE`. See Section 13.8.1, “DESCRIBE Statement”, for more information.

You can obtain the `CREATE TABLE` statement necessary to create an existing table using the `SHOW CREATE TABLE` statement. See Section 13.7.5.12, “SHOW CREATE TABLE Statement”.

If you have indexes on a table, `SHOW INDEX FROM tbl_name` produces information about them. See Section 13.7.5.23, “SHOW INDEX Statement”, for more about this statement.

3.5 Using mysql in Batch Mode

In the previous sections, you used `mysql` interactively to enter statements and view the results. You can also run `mysql` in batch mode. To do this, put the statements you want to run in a file, then tell `mysql` to read its input from the file:

```
shell> mysql < batch-file
```

If you are running `mysql` under Windows and have some special characters in the file that cause problems, you can do this:

```
C:\> mysql -e "source batch-file"
```

If you need to specify connection parameters on the command line, the command might look like this:

```
shell> mysql -h host -u user -p < batch-file
Enter password: ********
```

When you use `mysql` this way, you are creating a script file, then executing the script.

If you want the script to continue even if some of the statements in it produce errors, you should use the `--force` command-line option.

Why use a script? Here are a few reasons:

- If you run a query repeatedly (say, every day or every week), making it a script enables you to avoid retyping it each time you execute it.
- You can generate new queries from existing ones that are similar by copying and editing script files.
- Batch mode can also be useful while you're developing a query, particularly for multiple-line statements or multiple-statement sequences. If you make a mistake, you don't have to retype everything. Just edit your script to correct the error, then tell `mysql` to execute it again.
- If you have a query that produces a lot of output, you can run the output through a pager rather than watching it scroll off the top of your screen:

  ```
  shell> mysql < batch-file | more
  ```

- You can catch the output in a file for further processing:

  ```
  shell> mysql < batch-file > mysql.out
  ```

- You can distribute your script to other people so that they can also run the statements.
- Some situations do not allow for interactive use, for example, when you run a query from a `cron` job. In this case, you must use batch mode.

The default output format is different (more concise) when you run `mysql` in batch mode than when you use it interactively. For example, the output of `SELECT DISTINCT species FROM pet` looks like this when `mysql` is run interactively:

```
+---------+
| species |
+---------+
| bird    |
```

```
| cat     |
| dog     |
| hamster |
| snake   |
+---------+
```

In batch mode, the output looks like this instead:

```
species
bird
cat
dog
hamster
snake
```

If you want to get the interactive output format in batch mode, use `mysql -t`. To echo to the output the statements that are executed, use `mysql -v`.

You can also use scripts from the `mysql` prompt by using the `source` command or `\.` command:

```
mysql> source filename;
mysql> \. filename
```

See Section 4.5.1.5, “Executing SQL Statements from a Text File”, for more information.

3.6 Examples of Common Queries

Here are examples of how to solve some common problems with MySQL.

Some of the examples use the table `shop` to hold the price of each article (item number) for certain traders (dealers). Supposing that each trader has a single fixed price per article, then (`article`, `dealer`) is a primary key for the records.

Start the command-line tool `mysql` and select a database:

```
shell> mysql your-database-name
```

To create and populate the example table, use these statements:

```
CREATE TABLE shop (
    article INT UNSIGNED  DEFAULT '0000' NOT NULL,
    dealer  CHAR(20)      DEFAULT ''     NOT NULL,
    price   DECIMAL(16,2) DEFAULT '0.00' NOT NULL,
    PRIMARY KEY(article, dealer));
INSERT INTO shop VALUES
    (1,'A',3.45),(1,'B',3.99),(2,'A',10.99),(3,'B',1.45),
    (3,'C',1.69),(3,'D',1.25),(4,'D',19.95);
```

After issuing the statements, the table should have the following contents:

```
SELECT * FROM shop ORDER BY article;

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|       1 | A      |  3.45 |
|       1 | B      |  3.99 |
|       2 | A      | 10.99 |
|       3 | B      |  1.45 |
|       3 | C      |  1.69 |
|       3 | D      |  1.25 |
|       4 | D      | 19.95 |
+---------+--------+-------+
```

3.6.1 The Maximum Value for a Column

“What is the highest item number?”

```
SELECT MAX(article) AS article FROM shop;

+---------+
| article |
+---------+
|       4 |
+---------+
```

3.6.2 The Row Holding the Maximum of a Certain Column

Task: Find the number, dealer, and price of the most expensive article.

This is easily done with a subquery:

```
SELECT article, dealer, price
FROM   shop
WHERE  price=(SELECT MAX(price) FROM shop);

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

Other solutions are to use a `LEFT JOIN` or to sort all rows descending by price and get only the first row using the MySQL-specific `LIMIT` clause:

```
SELECT s1.article, s1.dealer, s1.price
FROM shop s1
LEFT JOIN shop s2 ON s1.price < s2.price
WHERE s2.article IS NULL;

SELECT article, dealer, price
FROM shop
ORDER BY price DESC
LIMIT 1;
```

Note

If there were several most expensive articles, each with a price of 19.95, the `LIMIT` solution would show only one of them.

3.6.3 Maximum of Column per Group

Task: Find the highest price per article.

```
SELECT article, MAX(price) AS price
FROM   shop
GROUP BY article
ORDER BY article;

+---------+-------+
| article | price |
+---------+-------+
|    0001 |  3.99 |
|    0002 | 10.99 |
|    0003 |  1.69 |
|    0004 | 19.95 |
+---------+-------+
```

3.6.4 The Rows Holding the Group-wise Maximum of a Certain Column

Task: For each article, find the dealer or dealers with the most expensive price.

This problem can be solved with a subquery like this one:

```
SELECT article, dealer, price
```

```
FROM   shop s1
WHERE  price=(SELECT MAX(s2.price)
              FROM shop s2
              WHERE s1.article = s2.article)
ORDER BY article;

+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0001 | B      |  3.99 |
|    0002 | A      | 10.99 |
|    0003 | C      |  1.69 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

The preceding example uses a correlated subquery, which can be inefficient (see Section 13.2.10.7, “Correlated Subqueries”). Other possibilities for solving the problem are to use an uncorrelated subquery in the `FROM` clause or a `LEFT JOIN`.

Uncorrelated subquery:

```
SELECT s1.article, dealer, s1.price
FROM shop s1
JOIN (
  SELECT article, MAX(price) AS price
  FROM shop
  GROUP BY article) AS s2
  ON s1.article = s2.article AND s1.price = s2.price
ORDER BY article;
```

`LEFT JOIN`:

```
SELECT s1.article, s1.dealer, s1.price
FROM shop s1
LEFT JOIN shop s2 ON s1.article = s2.article AND s1.price < s2.price
WHERE s2.article IS NULL
ORDER BY s1.article;
```

The `LEFT JOIN` works on the basis that when `s1.price` is at its maximum value, there is no `s2.price` with a greater value and thus the corresponding `s2.article` value is `NULL`. See Section 13.2.9.2, “JOIN Clause”.

3.6.5 Using User-Defined Variables

You can employ MySQL user variables to remember results without having to store them in temporary variables in the client. (See Section 9.4, “User-Defined Variables”.)

For example, to find the articles with the highest and lowest price you can do this:

```
mysql> SELECT @min_price:=MIN(price),@max_price:=MAX(price) FROM shop;
mysql> SELECT * FROM shop WHERE price=@min_price OR price=@max_price;
+---------+--------+-------+
| article | dealer | price |
+---------+--------+-------+
|    0003 | D      |  1.25 |
|    0004 | D      | 19.95 |
+---------+--------+-------+
```

Note

It is also possible to store the name of a database object such as a table or a column in a user variable and then to use this variable in an SQL statement; however, this requires the use of a prepared statement. See Section 13.5, “Prepared Statements”, for more information.

3.6.6 Using Foreign Keys

In MySQL, `InnoDB` tables support checking of foreign key constraints. See Chapter 14, *The InnoDB Storage Engine*, and Section 1.7.2.3, “FOREIGN KEY Constraint Differences”.

A foreign key constraint is not required merely to join two tables. For storage engines other than `InnoDB`, it is possible when defining a column to use a `REFERENCES tbl_name(col_name)` clause, which has no actual effect, and *serves only as a memo or comment to you that the column which you are currently defining is intended to refer to a column in another table*. It is extremely important to realize when using this syntax that:

- MySQL does not perform any sort of check to make sure that *`col_name`* actually exists in *`tbl_name`* (or even that *`tbl_name`* itself exists).
- MySQL does not perform any sort of action on *`tbl_name`* such as deleting rows in response to actions taken on rows in the table which you are defining; in other words, this syntax induces no `ON DELETE` or `ON UPDATE` behavior whatsoever. (Although you can write an `ON DELETE` or `ON UPDATE` clause as part of the `REFERENCES` clause, it is also ignored.)
- This syntax creates a *column*; it does **not** create any sort of index or key.

You can use a column so created as a join column, as shown here:

```
CREATE TABLE person (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    name CHAR(60) NOT NULL,
    PRIMARY KEY (id)
);

CREATE TABLE shirt (
    id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
    style ENUM('t-shirt', 'polo', 'dress') NOT NULL,
    color ENUM('red', 'blue', 'orange', 'white', 'black') NOT NULL,
    owner SMALLINT UNSIGNED NOT NULL REFERENCES person(id),
    PRIMARY KEY (id)
);

INSERT INTO person VALUES (NULL, 'Antonio Paz');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'polo', 'blue', @last),
(NULL, 'dress', 'white', @last),
(NULL, 't-shirt', 'blue', @last);

INSERT INTO person VALUES (NULL, 'Lilliana Angelovska');

SELECT @last := LAST_INSERT_ID();

INSERT INTO shirt VALUES
(NULL, 'dress', 'orange', @last),
(NULL, 'polo', 'red', @last),
(NULL, 'dress', 'blue', @last),
(NULL, 't-shirt', 'white', @last);

SELECT * FROM person;
+----+---------------------+
| id | name                |
+----+---------------------+
|  1 | Antonio Paz         |
|  2 | Lilliana Angelovska |
+----+---------------------+

SELECT * FROM shirt;
+----+---------+--------+-------+
| id | style   | color  | owner |
+----+---------+--------+-------+
|  1 | polo    | blue   |     1 |
|  2 | dress   | white  |     1 |
```

```
|  3 | t-shirt | blue   |     1 |
|  4 | dress   | orange |     2 |
|  5 | polo    | red    |     2 |
|  6 | dress   | blue   |     2 |
|  7 | t-shirt | white  |     2 |
+----+---------+--------+-------+


SELECT s.* FROM person p INNER JOIN shirt s
   ON s.owner = p.id
 WHERE p.name LIKE 'Lilliana%'
   AND s.color <> 'white';

+----+-------+--------+-------+
| id | style | color  | owner |
+----+-------+--------+-------+
|  4 | dress | orange |     2 |
|  5 | polo  | red    |     2 |
|  6 | dress | blue   |     2 |
+----+-------+--------+-------+
```

When used in this fashion, the `REFERENCES` clause is not displayed in the output of `SHOW CREATE TABLE` or `DESCRIBE`:

```
SHOW CREATE TABLE shirt\G
*************************** 1. row ***************************
Table: shirt
Create Table: CREATE TABLE `shirt` (
`id` smallint(5) unsigned NOT NULL auto_increment,
`style` enum('t-shirt','polo','dress') NOT NULL,
`color` enum('red','blue','orange','white','black') NOT NULL,
`owner` smallint(5) unsigned NOT NULL,
PRIMARY KEY  (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

The use of `REFERENCES` in this way as a comment or “reminder” in a column definition works with `MyISAM` tables.

3.6.7 Searching on Two Keys

An `OR` using a single key is well optimized, as is the handling of `AND`.

The one tricky case is that of searching on two different keys combined with `OR`:

```
SELECT field1_index, field2_index FROM test_table
WHERE field1_index = '1' OR  field2_index = '1'
```

This case is optimized. See Section 8.2.1.3, “Index Merge Optimization”.

You can also solve the problem efficiently by using a `UNION` that combines the output of two separate `SELECT` statements. See Section 13.2.9.3, “UNION Clause”.

Each `SELECT` searches only one key and can be optimized:

```
SELECT field1_index, field2_index
    FROM test_table WHERE field1_index = '1'
UNION
SELECT field1_index, field2_index
    FROM test_table WHERE field2_index = '1';
```

3.6.8 Calculating Visits Per Day

The following example shows how you can use the bit group functions to calculate the number of days per month a user has visited a Web page.

```
CREATE TABLE t1 (year YEAR, month INT UNSIGNED,
             day INT UNSIGNED);
INSERT INTO t1 VALUES(2000,1,1),(2000,1,20),(2000,1,30),(2000,2,2),
```

```
            (2000,2,23),(2000,2,23);
```

The example table contains year-month-day values representing visits by users to the page. To determine how many different days in each month these visits occur, use this query:

```
SELECT year,month,BIT_COUNT(BIT_OR(1<<day)) AS days FROM t1
       GROUP BY year,month;
```

Which returns:

```
+------+-------+------+
| year | month | days |
+------+-------+------+
| 2000 |     1 |    3 |
| 2000 |     2 |    2 |
+------+-------+------+
```

The query calculates how many different days appear in the table for each year/month combination, with automatic removal of duplicate entries.

3.6.9 Using AUTO_INCREMENT

The `AUTO_INCREMENT` attribute can be used to generate a unique identity for new rows:

```
CREATE TABLE animals (
     id MEDIUMINT NOT NULL AUTO_INCREMENT,
     name CHAR(30) NOT NULL,
     PRIMARY KEY (id)
);

INSERT INTO animals (name) VALUES
    ('dog'),('cat'),('penguin'),
    ('lax'),('whale'),('ostrich');

SELECT * FROM animals;
```

Which returns:

```
+----+---------+
| id | name    |
+----+---------+
|  1 | dog     |
|  2 | cat     |
|  3 | penguin |
|  4 | lax     |
|  5 | whale   |
|  6 | ostrich |
+----+---------+
```

No value was specified for the `AUTO_INCREMENT` column, so MySQL assigned sequence numbers automatically. You can also explicitly assign 0 to the column to generate sequence numbers, unless the `NO_AUTO_VALUE_ON_ZERO` SQL mode is enabled. For example:

```
INSERT INTO animals (id,name) VALUES(0,'groundhog');
```

If the column is declared `NOT NULL`, it is also possible to assign `NULL` to the column to generate sequence numbers. For example:

```
INSERT INTO animals (id,name) VALUES(NULL,'squirrel');
```

When you insert any other value into an `AUTO_INCREMENT` column, the column is set to that value and the sequence is reset so that the next automatically generated value follows sequentially from the largest column value. For example:

```
INSERT INTO animals (id,name) VALUES(100,'rabbit');
INSERT INTO animals (id,name) VALUES(NULL,'mouse');
SELECT * FROM animals;
+-----+-----------+
```

```
| id  | name      |
+-----+-----------+
|   1 | dog       |
|   2 | cat       |
|   3 | penguin   |
|   4 | lax       |
|   5 | whale     |
|   6 | ostrich   |
|   7 | groundhog |
|   8 | squirrel  |
| 100 | rabbit    |
| 101 | mouse     |
+-----+-----------+
```

Updating an existing `AUTO_INCREMENT` column value in an `InnoDB` table does not reset the `AUTO_INCREMENT` sequence as it does for `MyISAM` and `NDB` tables.

You can retrieve the most recent automatically generated `AUTO_INCREMENT` value with the `LAST_INSERT_ID()` SQL function or the `mysql_insert_id()` C API function. These functions are connection-specific, so their return values are not affected by another connection which is also performing inserts.

Use the smallest integer data type for the `AUTO_INCREMENT` column that is large enough to hold the maximum sequence value you need. When the column reaches the upper limit of the data type, the next attempt to generate a sequence number fails. Use the `UNSIGNED` attribute if possible to allow a greater range. For example, if you use `TINYINT`, the maximum permissible sequence number is 127. For `TINYINT UNSIGNED`, the maximum is 255. See Section 11.1.2, “Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT” for the ranges of all the integer types.

Note

For a multiple-row insert, `LAST_INSERT_ID()` and `mysql_insert_id()` actually return the `AUTO_INCREMENT` key from the *first* of the inserted rows. This enables multiple-row inserts to be reproduced correctly on other servers in a replication setup.

To start with an `AUTO_INCREMENT` value other than 1, set that value with `CREATE TABLE` or `ALTER TABLE`, like this:

```
mysql> ALTER TABLE tbl AUTO_INCREMENT = 100;
```

InnoDB Notes

For information about `AUTO_INCREMENT` usage specific to `InnoDB`, see Section 14.6.1.6, “AUTO_INCREMENT Handling in InnoDB”.

MyISAM Notes

- For `MyISAM` tables, you can specify `AUTO_INCREMENT` on a secondary column in a multiple-column index. In this case, the generated value for the `AUTO_INCREMENT` column is calculated as `MAX(auto_increment_column) + 1 WHERE prefix=given-prefix`. This is useful when you want to put data into ordered groups.

```
CREATE TABLE animals (
    grp ENUM('fish','mammal','bird') NOT NULL,
    id MEDIUMINT NOT NULL AUTO_INCREMENT,
    name CHAR(30) NOT NULL,
    PRIMARY KEY (grp,id)
) ENGINE=MyISAM;

INSERT INTO animals (grp,name) VALUES
    ('mammal','dog'),('mammal','cat'),
    ('bird','penguin'),('fish','lax'),('mammal','whale'),
    ('bird','ostrich');
```

```
SELECT * FROM animals ORDER BY grp,id;
```

Which returns:

```
+--------+----+---------+
| grp    | id | name    |
+--------+----+---------+
| fish   |  1 | lax     |
| mammal |  1 | dog     |
| mammal |  2 | cat     |
| mammal |  3 | whale   |
| bird   |  1 | penguin |
| bird   |  2 | ostrich |
+--------+----+---------+
```

In this case (when the `AUTO_INCREMENT` column is part of a multiple-column index), `AUTO_INCREMENT` values are reused if you delete the row with the biggest `AUTO_INCREMENT` value in any group. This happens even for `MyISAM` tables, for which `AUTO_INCREMENT` values normally are not reused.

- If the `AUTO_INCREMENT` column is part of multiple indexes, MySQL generates sequence values using the index that begins with the `AUTO_INCREMENT` column, if there is one. For example, if the `animals` table contained indexes `PRIMARY KEY (grp, id)` and `INDEX (id)`, MySQL would ignore the `PRIMARY KEY` for generating sequence values. As a result, the table would contain a single sequence, not a sequence per `grp` value.

Further Reading

More information about `AUTO_INCREMENT` is available here:

- How to assign the `AUTO_INCREMENT` attribute to a column: Section 13.1.17, “CREATE TABLE Statement”, and Section 13.1.7, “ALTER TABLE Statement”.
- How `AUTO_INCREMENT` behaves depending on the `NO_AUTO_VALUE_ON_ZERO` SQL mode: Section 5.1.10, “Server SQL Modes”.
- How to use the `LAST_INSERT_ID()` function to find the row that contains the most recent `AUTO_INCREMENT` value: Section 12.16, “Information Functions”.
- Setting the `AUTO_INCREMENT` value to be used: Section 5.1.7, “Server System Variables”.
- Section 14.6.1.6, “AUTO_INCREMENT Handling in InnoDB”
- `AUTO_INCREMENT` and replication: Section 17.4.1.1, “Replication and AUTO_INCREMENT”.
- Server-system variables related to `AUTO_INCREMENT` (`auto_increment_increment` and `auto_increment_offset`) that can be used for replication: Section 5.1.7, “Server System Variables”.

3.7 Using MySQL with Apache

There are programs that let you authenticate your users from a MySQL database and also let you write your log files into a MySQL table.

You can change the Apache logging format to be easily readable by MySQL by putting the following into the Apache configuration file:

```
LogFormat \
        "\"%h\",%{%Y%m%d%H%M%S}t,%>s,\"%b\",\"%{Content-Type}o\",  \
        \"%U\",\"%{Referer}i\",\"%{User-Agent}i\""
```

To load a log file in that format into MySQL, you can use a statement something like this:

```
LOAD DATA INFILE '/local/access_log' INTO TABLE tbl_name
```

```
FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'
```

The named table should be created to have columns that correspond to those that the `LogFormat` line writes to the log file.

Chapter 4 MySQL Programs

Table of Contents

4.1 Overview of MySQL Programs 252
4.2 Using MySQL Programs 256
4.2.1 Invoking MySQL Programs 256
4.2.2 Specifying Program Options 257
4.2.3 Command Options for Connecting to the Server 270
4.2.4 Connecting to the MySQL Server Using Command Options 276
4.2.5 Connection Transport Protocols 278
4.2.6 Connection Compression Control 280
4.2.7 Setting Environment Variables 280
4.3 Server and Server-Startup Programs 281
4.3.1 `mysqld` — The MySQL Server 281
4.3.2 `mysqld_safe` — MySQL Server Startup Script 282
4.3.3 `mysql.server` — MySQL Server Startup Script 287
4.3.4 `mysqld_multi` — Manage Multiple MySQL Servers 289
4.4 Installation-Related Programs 292
4.4.1 `comp_err` — Compile MySQL Error Message File 292
4.4.2 `mysqlbug` — Generate Bug Report 293
4.4.3 `mysql_install_db` — Initialize MySQL Data Directory 293
4.4.4 `mysql_plugin` — Configure MySQL Server Plugins 297
4.4.5 `mysql_secure_installation` — Improve MySQL Installation Security 299
4.4.6 `mysql_tzinfo_to_sql` — Load the Time Zone Tables 300
4.4.7 `mysql_upgrade` — Check and Upgrade MySQL Tables 300
4.5 Client Programs 306
4.5.1 `mysql` — The MySQL Command-Line Client 307
4.5.2 `mysqladmin` — A MySQL Server Administration Program 331
4.5.3 `mysqlcheck` — A Table Maintenance Program 340
4.5.4 `mysqldump` — A Database Backup Program 348
4.5.5 `mysqlimport` — A Data Import Program 368
4.5.6 `mysqlshow` — Display Database, Table, and Column Information 375
4.5.7 `mysqlslap` — A Load Emulation Client 380
4.6 Administrative and Utility Programs 389
4.6.1 `innochecksum` — Offline InnoDB File Checksum Utility 389
4.6.2 `myisam_ftdump` — Display Full-Text Index information 390
4.6.3 `myisamchk` — MyISAM Table-Maintenance Utility 391
4.6.4 `myisamlog` — Display MyISAM Log File Contents 407
4.6.5 `myisampack` — Generate Compressed, Read-Only MyISAM Tables 408
4.6.6 `mysql_config_editor` — MySQL Configuration Utility 414
4.6.7 `mysqlaccess` — Client for Checking Access Privileges 420
4.6.8 `mysqlbinlog` — Utility for Processing Binary Log Files 423
4.6.9 `mysqldumpslow` — Summarize Slow Query Log Files 442
4.6.10 `mysqlhotcopy` — A Database Backup Program 444
4.6.11 `mysql_convert_table_format` — Convert Tables to Use a Given Storage Engine 447
4.6.12 `mysql_find_rows` — Extract SQL Statements from Files 448
4.6.13 `mysql_fix_extensions` — Normalize Table File Name Extensions 448
4.6.14 `mysql_setpermission` — Interactively Set Permissions in Grant Tables 449
4.6.15 `mysql_waitpid` — Kill Process and Wait for Its Termination 450
4.6.16 `mysql_zap` — Kill Processes That Match a Pattern 450
4.7 Program Development Utilities 451
4.7.1 `msql2mysql` — Convert mSQL Programs for Use with MySQL 451
4.7.2 `mysql_config` — Display Options for Compiling Clients 451
4.7.3 `my_print_defaults` — Display Options from Option Files 453

4.7.4 resolve_stack_dump — Resolve Numeric Stack Trace Dump to Symbols 454
4.8 Miscellaneous Programs 455
4.8.1 perror — Display MySQL Error Message Information 455
4.8.2 replace — A String-Replacement Utility 455
4.8.3 resolveip — Resolve Host name to IP Address or Vice Versa 456
4.9 Environment Variables 457
4.10 Unix Signal Handling in MySQL 458

This chapter provides a brief overview of the MySQL command-line programs provided by Oracle Corporation. It also discusses the general syntax for specifying options when you run these programs. Most programs have options that are specific to their own operation, but the option syntax is similar for all of them. Finally, the chapter provides more detailed descriptions of individual programs, including which options they recognize.

4.1 Overview of MySQL Programs

There are many different programs in a MySQL installation. This section provides a brief overview of them. Later sections provide a more detailed description of each one, with the exception of NDB Cluster programs. Each program's description indicates its invocation syntax and the options that it supports. Section 18.4, “NDB Cluster Programs”, describes programs specific to NDB Cluster.

Most MySQL distributions include all of these programs, except for those programs that are platform-specific. (For example, the server startup scripts are not used on Windows.) The exception is that RPM distributions are more specialized. There is one RPM for the server, another for client programs, and so forth. If you appear to be missing one or more programs, see Chapter 2, *Installing and Upgrading MySQL*, for information on types of distributions and what they contain. It may be that you have a distribution that does not include all programs and you need to install an additional package.

Each MySQL program takes many different options. Most programs provide a `--help` option that you can use to get a description of the program's different options. For example, try `mysql --help`.

You can override default option values for MySQL programs by specifying options on the command line or in an option file. See Section 4.2, “Using MySQL Programs”, for general information on invoking programs and specifying program options.

The MySQL server, `mysqld`, is the main program that does most of the work in a MySQL installation. The server is accompanied by several related scripts that assist you in starting and stopping the server:

- `mysqld`

  The SQL daemon (that is, the MySQL server). To use client programs, `mysqld` must be running, because clients gain access to databases by connecting to the server. See Section 4.3.1, “`mysqld` — The MySQL Server”.

- `mysqld_safe`

  A server startup script. `mysqld_safe` attempts to start `mysqld`. See Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”.

- `mysql.server`

  A server startup script. This script is used on systems that use System V-style run directories containing scripts that start system services for particular run levels. It invokes `mysqld_safe` to start the MySQL server. See Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”.

- `mysqld_multi`

  A server startup script that can start or stop multiple servers installed on the system. See Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”.

Several programs perform setup operations during MySQL installation or upgrading:

- `comp_err`

  This program is used during the MySQL build/installation process. It compiles error message files from the error source files. See Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”.

- `mysql_install_db`

  This program initializes the MySQL data directory, creates the `mysql` database and initializes its grant tables with default privileges, and sets up the `InnoDB` system tablespace. It is usually executed only once, when first installing MySQL on a system. See Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”, and Section 2.10, “Postinstallation Setup and Testing”.

- `mysql_plugin`

  This program configures MySQL server plugins. See Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”.

- `mysql_secure_installation`

  This program enables you to improve the security of your MySQL installation. See Section 4.4.5, “`mysql_secure_installation` — Improve MySQL Installation Security”.

- `mysql_tzinfo_to_sql`

  This program loads the time zone tables in the `mysql` database using the contents of the host system *zoneinfo* database (the set of files describing time zones). See Section 4.4.6, “`mysql_tzinfo_to_sql` — Load the Time Zone Tables”.

- `mysql_upgrade`

  This program is used after a MySQL upgrade operation. It updates the grant tables with any changes that have been made in newer versions of MySQL, and checks tables for incompatibilities and repairs them if necessary. See Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”.

MySQL client programs that connect to the MySQL server:

- `mysql`

  The command-line tool for interactively entering SQL statements or executing them from a file in batch mode. See Section 4.5.1, “`mysql` — The MySQL Command-Line Client”.

- `mysqladmin`

  A client that performs administrative operations, such as creating or dropping databases, reloading the grant tables, flushing tables to disk, and reopening log files. `mysqladmin` can also be used to retrieve version, process, and status information from the server. See Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”.

- `mysqlcheck`

  A table-maintenance client that checks, repairs, analyzes, and optimizes tables. See Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”.

- `mysqldump`

  A client that dumps a MySQL database into a file as SQL, text, or XML. See Section 4.5.4, “`mysqldump` — A Database Backup Program”.

- `mysqlimport`

A client that imports text files into their respective tables using `LOAD DATA`. See Section 4.5.5, “`mysqlimport` — A Data Import Program”.

- `mysqlshow`

  A client that displays information about databases, tables, columns, and indexes. See Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”.

- `mysqlslap`

  A client that is designed to emulate client load for a MySQL server and report the timing of each stage. It works as if multiple clients are accessing the server. See Section 4.5.7, “`mysqlslap` — A Load Emulation Client”.

MySQL administrative and utility programs:

- `innochecksum`

  An offline `InnoDB` offline file checksum utility. See Section 4.6.1, “`innochecksum` — Offline InnoDB File Checksum Utility”.

- `myisam_ftdump`

  A utility that displays information about full-text indexes in `MyISAM` tables. See Section 4.6.2, “`myisam_ftdump` — Display Full-Text Index information”.

- `myisamchk`

  A utility to describe, check, optimize, and repair `MyISAM` tables. See Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”.

- `myisamlog`

  A utility that processes the contents of a `MyISAM` log file. See Section 4.6.4, “`myisamlog` — Display MyISAM Log File Contents”.

- `myisampack`

  A utility that compresses `MyISAM` tables to produce smaller read-only tables. See Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”.

- `mysql_config_editor`

  A utility that enables you to store authentication credentials in a secure, encrypted login path file named `.mylogin.cnf`. See Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”.

- `mysqlaccess`

  A script that checks the access privileges for a host name, user name, and database combination. See Section 4.6.7, “`mysqlaccess` — Client for Checking Access Privileges”.

- `mysqlbinlog`

  A utility for reading statements from a binary log. The log of executed statements contained in the binary log files can be used to help recover from a crash. See Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”.

- `mysqldumpslow`

  A utility to read and summarize the contents of a slow query log. See Section 4.6.9, “`mysqldumpslow` — Summarize Slow Query Log Files”.

- `mysqlhotcopy`

  A utility that quickly makes backups of `MyISAM` tables while the server is running. See Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”.

- `mysql_convert_table_format`

  A utility that converts tables in a database to use a given storage engine. See Section 4.6.11, “`mysql_convert_table_format` — Convert Tables to Use a Given Storage Engine”.

- `mysql_find_rows`

  A utility that reads files containing SQL statements (such as update logs) and extracts statements that match a given regular expression. See Section 4.6.12, “`mysql_find_rows` — Extract SQL Statements from Files”.

- `mysql_fix_extensions`

  A utility that converts the extensions for `MyISAM` table files to lowercase. This can be useful after transferring the files from a system with case-insensitive file names to a system with case-sensitive file names. See Section 4.6.13, “`mysql_fix_extensions` — Normalize Table File Name Extensions”.

- `mysql_setpermission`

  A utility for interactively setting permissions in the MySQL grant tables. See Section 4.6.14, “`mysql_setpermission` — Interactively Set Permissions in Grant Tables”.

- `mysql_waitpid`

  A utility that kills the process with a given process ID. See Section 4.6.15, “`mysql_waitpid` — Kill Process and Wait for Its Termination”.

- `mysql_zap`

  A utility that kills processes that match a pattern. See Section 4.6.16, “`mysql_zap` — Kill Processes That Match a Pattern”.

MySQL program-development utilities:

- `msql2mysql`

  A shell script that converts `mSQL` programs to MySQL. It doesn't handle every case, but it gives a good start when converting. See Section 4.7.1, “`msql2mysql` — Convert mSQL Programs for Use with MySQL”.

- `mysql_config`

  A shell script that produces the option values needed when compiling MySQL programs. See Section 4.7.2, “`mysql_config` — Display Options for Compiling Clients”.

- `my_print_defaults`

  A utility that shows which options are present in option groups of option files. See Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”.

- `resolve_stack_dump`

  A utility program that resolves a numeric stack trace dump to symbols. See Section 4.7.4, “`resolve_stack_dump` — Resolve Numeric Stack Trace Dump to Symbols”.

Miscellaneous utilities:

- `perror`

  A utility that displays the meaning of system or MySQL error codes. See Section 4.8.1, “`perror` — Display MySQL Error Message Information”.

- `replace`

  A utility program that performs string replacement in the input text. See Section 4.8.2, “`replace` — A String-Replacement Utility”.

- `resolveip`

  A utility program that resolves a host name to an IP address or vice versa. See Section 4.8.3, “`resolveip` — Resolve Host name to IP Address or Vice Versa”.

Oracle Corporation also provides the MySQL Workbench GUI tool, which is used to administer MySQL servers and databases, to create, execute, and evaluate queries, and to migrate schemas and data from other relational database management systems for use with MySQL.

MySQL client programs that communicate with the server using the MySQL client/server library use the following environment variables.

Environment Variable	Meaning
`MYSQL_UNIX_PORT`	The default Unix socket file; used for connections to `localhost`
`MYSQL_TCP_PORT`	The default port number; used for TCP/IP connections
`MYSQL_PWD`	The default password
`MYSQL_DEBUG`	Debug trace options when debugging
`TMPDIR`	The directory where temporary tables and files are created

For a full list of environment variables used by MySQL programs, see Section 4.9, “Environment Variables”.

Use of `MYSQL_PWD` is insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

4.2 Using MySQL Programs

4.2.1 Invoking MySQL Programs

To invoke a MySQL program from the command line (that is, from your shell or command prompt), enter the program name followed by any options or other arguments needed to instruct the program what you want it to do. The following commands show some sample program invocations. `shell>` represents the prompt for your command interpreter; it is not part of what you type. The particular prompt you see depends on your command interpreter. Typical prompts are `$` for `sh`, `ksh`, or `bash`, `%` for `csh` or `tcsh`, and `C:\>` for the Windows `command.com` or `cmd.exe` command interpreters.

```
shell> mysql --user=root test
shell> mysqladmin extended-status variables
shell> mysqlshow --help
shell> mysqldump -u root personnel
```

Arguments that begin with a single or double dash (`-`, `--`) specify program options. Options typically indicate the type of connection a program should make to the server or affect its operational mode. Option syntax is described in Section 4.2.2, “Specifying Program Options”.

Nonoption arguments (arguments with no leading dash) provide additional information to the program. For example, the `mysql` program interprets the first nonoption argument as a database name, so the command `mysql --user=root test` indicates that you want to use the `test` database.

Later sections that describe individual programs indicate which options a program supports and describe the meaning of any additional nonoption arguments.

Some options are common to a number of programs. The most frequently used of these are the `--host` (or `-h`), `--user` (or `-u`), and `--password` (or `-p`) options that specify connection parameters. They indicate the host where the MySQL server is running, and the user name and password of your MySQL account. All MySQL client programs understand these options; they enable you to specify which server to connect to and the account to use on that server. Other connection options are `--port` (or `-P`) to specify a TCP/IP port number and `--socket` (or `-S`) to specify a Unix socket file on Unix (or named-pipe name on Windows). For more information on options that specify connection options, see Section 4.2.4, “Connecting to the MySQL Server Using Command Options”.

You may find it necessary to invoke MySQL programs using the path name to the `bin` directory in which they are installed. This is likely to be the case if you get a “program not found” error whenever you attempt to run a MySQL program from any directory other than the `bin` directory. To make it more convenient to use MySQL, you can add the path name of the `bin` directory to your `PATH` environment variable setting. That enables you to run a program by typing only its name, not its entire path name. For example, if `mysql` is installed in `/usr/local/mysql/bin`, you can run the program by invoking it as `mysql`, and it is not necessary to invoke it as `/usr/local/mysql/bin/mysql`.

Consult the documentation for your command interpreter for instructions on setting your `PATH` variable. The syntax for setting environment variables is interpreter-specific. (Some information is given in Section 4.2.7, “Setting Environment Variables”.) After modifying your `PATH` setting, open a new console window on Windows or log in again on Unix so that the setting goes into effect.

4.2.2 Specifying Program Options

There are several ways to specify options for MySQL programs:

- List the options on the command line following the program name. This is common for options that apply to a specific invocation of the program.
- List the options in an option file that the program reads when it starts. This is common for options that you want the program to use each time it runs.
- List the options in environment variables (see Section 4.2.7, “Setting Environment Variables”). This method is useful for options that you want to apply each time the program runs. In practice, option files are used more commonly for this purpose, but Section 5.7.3, “Running Multiple MySQL Instances on Unix”, discusses one situation in which environment variables can be very helpful. It describes a handy technique that uses such variables to specify the TCP/IP port number and Unix socket file for the server and for client programs.

Options are processed in order, so if an option is specified multiple times, the last occurrence takes precedence. The following command causes `mysql` to connect to the server running on `localhost`:

```
mysql -h example.com -h localhost
```

There is one exception: For `mysqld`, the *first* instance of the `--user` option is used as a security precaution, to prevent a user specified in an option file from being overridden on the command line.

If conflicting or related options are given, later options take precedence over earlier options. The following command runs `mysql` in “no column names” mode:

```
mysql --column-names --skip-column-names
```

MySQL programs determine which options are given first by examining environment variables, then by processing option files, and then by checking the command line. Because later options take precedence over earlier ones, the processing order means that environment variables have the lowest precedence and command-line options the highest.

You can take advantage of the way that MySQL programs process options by specifying default option values for a program in an option file. That enables you to avoid typing them each time you run the program while enabling you to override the defaults if necessary by using command-line options.

An option can be specified by writing it in full or as any unambiguous prefix. For example, the `--compress` option can be given to `mysqldump` as `--compr`, but not as `--comp` because the latter is ambiguous:

```
shell> mysqldump --comp
mysqldump: ambiguous option '--comp' (compatible, compress)
```

Be aware that the use of option prefixes can cause problems in the event that new options are implemented for a program. A prefix that is unambiguous now might become ambiguous in the future.

Note

As of MySQL 5.6.13, unambiguous prefixes are deprecated. If an unambiguous prefix is given, a warning occurs to provide feedback. Option prefixes are no longer supported as of MySQL 5.7; only full options are accepted.

4.2.2.1 Using Options on the Command Line

Program options specified on the command line follow these rules:

- Options are given after the command name.
- An option argument begins with one dash or two dashes, depending on whether it is a short form or long form of the option name. Many options have both short and long forms. For example, `-?` and `--help` are the short and long forms of the option that instructs a MySQL program to display its help message.
- Option names are case-sensitive. `-v` and `-V` are both legal and have different meanings. (They are the corresponding short forms of the `--verbose` and `--version` options.)
- Some options take a value following the option name. For example, `-h localhost` or `--host=localhost` indicate the MySQL server host to a client program. The option value tells the program the name of the host where the MySQL server is running.
- For a long option that takes a value, separate the option name and the value by an `=` sign. For a short option that takes a value, the option value can immediately follow the option letter, or there can be a space between: `-hlocalhost` and `-h localhost` are equivalent. An exception to this rule is the option for specifying your MySQL password. This option can be given in long form as `--password=pass_val` or as `--password`. In the latter case (with no password value given), the program interactively prompts you for the password. The password option also may be given in short form as `-ppass_val` or as `-p`. However, for the short form, if the password value is given, it must follow the option letter with *no intervening space*: If a space follows the option letter, the program has no way to tell whether a following argument is supposed to be the password value or some other kind of argument. Consequently, the following two commands have two completely different meanings:

  ```
  mysql -ptest
  mysql -p test
  ```

  The first command instructs `mysql` to use a password value of `test`, but specifies no default database. The second instructs `mysql` to prompt for the password value and to use `test` as the default database.
- Within option names, dash (`-`) and underscore (_) may be used interchangeably. For example, `--skip-grant-tables` and `--skip_grant_tables` are equivalent. (However, the leading dashes cannot be given as underscores.)
- The MySQL server has certain command options that may be specified only at startup, and a set of system variables, some of which may be set at startup, at runtime, or both. System variable names use underscores rather than dashes, and when referenced at runtime (for example, using `SET` or `SELECT` statements), must be written using underscores:

  ```
  SET GLOBAL general_log = ON;
  ```

```
SELECT @@GLOBAL.general_log;
```

At server startup, the syntax for system variables is the same as for command options, so within variable names, dashes and underscores may be used interchangeably. For example, `--general_log=ON` and `--general-log=ON` are equivalent. (This is also true for system variables set within option files.)

- For options that take a numeric value, the value can be given with a suffix of `K`, `M`, or `G` (either uppercase or lowercase) to indicate a multiplier of 1024, 1024^2 or 1024^3. For example, the following command tells `mysqladmin` to ping the server 1024 times, sleeping 10 seconds between each ping:

```
mysqladmin --count=1K --sleep=10 ping
```

- When specifying file names as option values, avoid the use of the `~` shell metacharacter. It might not be interpreted as you expect.

Option values that contain spaces must be quoted when given on the command line. For example, the `--execute` (or `-e`) option can be used with `mysql` to pass one or more semicolon-separated SQL statements to the server. When this option is used, `mysql` executes the statements in the option value and exits. The statements must be enclosed by quotation marks. For example:

```
shell> mysql -u root -p -e "SELECT VERSION();SELECT NOW()"
Enter password: ******
+------------+
| VERSION()  |
+------------+
| 5.6.47     |
+------------+
+---------------------+
| NOW()               |
+---------------------+
| 2019-09-03 10:36:01 |
+---------------------+
shell>
```

Note

The long form (`--execute`) is followed by an equal sign (`=`).

To use quoted values within a statement, you must either escape the inner quotation marks, or use a different type of quotation marks within the statement from those used to quote the statement itself. The capabilities of your command processor dictate your choices for whether you can use single or double quotation marks and the syntax for escaping quote characters. For example, if your command processor supports quoting with single or double quotation marks, you can use double quotation marks around the statement, and single quotation marks for any quoted values within the statement.

4.2.2.2 Using Option Files

Most MySQL programs can read startup options from option files (sometimes called configuration files). Option files provide a convenient way to specify commonly used options so that they need not be entered on the command line each time you run a program.

To determine whether a program reads option files, invoke it with the `--help` option. (For `mysqld`, use `--verbose` and `--help`.) If the program reads option files, the help message indicates which files it looks for and which option groups it recognizes.

Note

A MySQL program started with the `--no-defaults` option reads no option files other than `.mylogin.cnf`.

Many option files are plain text files, created using any text editor. The exception is the `.mylogin.cnf` file that contains login path options. This is an encrypted file created by the `mysql_config_editor`

utility. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”. A “login path” is an option group that permits only certain options: host, user, password, port and socket. Client programs specify which login path to read from .mylogin.cnf using the --login-path option.

To specify an alternative login path file name, set the MYSQL_TEST_LOGIN_FILE environment variable. This variable is used by the mysql-test-run.pl testing utility, but also is recognized by mysql_config_editor and by MySQL clients such as mysql, mysqladmin, and so forth.

MySQL looks for option files in the order described in the following discussion and reads any that exist. If an option file you want to use does not exist, create it using the appropriate method, as just discussed.

Note

For information about option files used with NDB Cluster programs, see Section 18.3, “Configuration of NDB Cluster”.

On Windows, MySQL programs read startup options from the files shown in the following table, in the specified order (files listed first are read first, files read later take precedence).

Table 4.1 Option Files Read on Windows Systems

File Name	Purpose
%WINDIR%\my.ini, %WINDIR%\my.cnf	Global options
C:\my.ini, C:\my.cnf	Global options
BASEDIR\my.ini, *BASEDIR*\my.cnf	Global options
defaults-extra-file	The file specified with --defaults-extra-file, if any
%APPDATA%\MySQL\.mylogin.cnf	Login path options (clients only)

- Option File Processing Order
- Option File Syntax
- Option File Inclusions

Option File Processing Order

In the preceding table, %WINDIR% represents the location of your Windows directory. This is commonly C:\WINDOWS. Use the following command to determine its exact location from the value of the WINDIR environment variable:

```
C:\> echo %WINDIR%
```

%APPDATA% represents the value of the Windows application data directory. Use the following command to determine its exact location from the value of the APPDATA environment variable:

```
C:\> echo %APPDATA%
```

BASEDIR represents the MySQL base installation directory. When MySQL 5.6 has been installed using MySQL Installer, this is typically C:*PROGRAMDIR*\MySQL\MySQL 5.6 Server where *PROGRAMDIR* represents the programs directory (usually Program Files on English-language versions of Windows), See Section 2.3.3, “MySQL Installer for Windows”.

On Unix and Unix-like systems, MySQL programs read startup options from the files shown in the following table, in the specified order (files listed first are read first, files read later take precedence).

Note

On Unix platforms, MySQL ignores configuration files that are world-writable. This is intentional as a security measure.

Table 4.2 Option Files Read on Unix and Unix-Like Systems

File Name	Purpose
`/etc/my.cnf`	Global options
`/etc/mysql/my.cnf`	Global options
`SYSCONFDIR``/my.cnf`	Global options
`$MYSQL_HOME/my.cnf`	Server-specific options (server only)
`defaults-extra-file`	The file specified with `--defaults-extra-file`, if any
`~/.my.cnf`	User-specific options
`~/.mylogin.cnf`	User-specific login path options (clients only)

In the preceding table, `~` represents the current user's home directory (the value of `$HOME`).

`SYSCONFDIR` represents the directory specified with the `SYSCONFDIR` option to `CMake` when MySQL was built. By default, this is the `etc` directory located under the compiled-in installation directory.

`MYSQL_HOME` is an environment variable containing the path to the directory in which the server-specific `my.cnf` file resides. If `MYSQL_HOME` is not set and you start the server using the `mysqld_safe` program, `mysqld_safe` attempts to set `MYSQL_HOME` as follows:

- Let *`BASEDIR`* and *`DATADIR`* represent the path names of the MySQL base directory and data directory, respectively.
- If there is a `my.cnf` file in *`DATADIR`* but not in *`BASEDIR`*, `mysqld_safe` sets `MYSQL_HOME` to *`DATADIR`*.
- Otherwise, if `MYSQL_HOME` is not set and there is no `my.cnf` file in *`DATADIR`*, `mysqld_safe` sets `MYSQL_HOME` to *`BASEDIR`*.

In MySQL 5.6, use of *`DATADIR`* as the location for `my.cnf` is deprecated.

`DATADIR` is commonly `/usr/local/mysql/data`, although this can vary per platform or installation method. The value is the data directory location built in when MySQL was compiled, not the location specified with the `--datadir` option when `mysqld` starts. Use of `--datadir` at runtime has no effect on where the server looks for option files that it reads before processing any options.

If multiple instances of a given option are found, the last instance takes precedence, with one exception: For `mysqld`, the *first* instance of the `--user` option is used as a security precaution, to prevent a user specified in an option file from being overridden on the command line.

Option File Syntax

The following description of option file syntax applies to files that you edit manually. This excludes `.mylogin.cnf`, which is created using `mysql_config_editor` and is encrypted.

Any long option that may be given on the command line when running a MySQL program can be given in an option file as well. To get the list of available options for a program, run it with the `--help` option. (For `mysqld`, use `--verbose` and `--help`.)

The syntax for specifying options in an option file is similar to command-line syntax (see Section 4.2.2.1, “Using Options on the Command Line”). However, in an option file, you omit

the leading two dashes from the option name and you specify only one option per line. For example, `--quick` and `--host=localhost` on the command line should be specified as `quick` and `host=localhost` on separate lines in an option file. To specify an option of the form `--loose-opt_name` in an option file, write it as `loose-opt_name`.

Empty lines in option files are ignored. Nonempty lines can take any of the following forms:

- `#comment`, `;comment`

  Comment lines start with `#` or `;`. A `#` comment can start in the middle of a line as well.

- `[group]`

  `group` is the name of the program or group for which you want to set options. After a group line, any option-setting lines apply to the named group until the end of the option file or another group line is given. Option group names are not case-sensitive.

- `opt_name`

  This is equivalent to `--opt_name` on the command line.

- `opt_name=value`

  This is equivalent to `--opt_name=value` on the command line. In an option file, you can have spaces around the `=` character, something that is not true on the command line. The value optionally can be enclosed within single quotation marks or double quotation marks, which is useful if the value contains a `#` comment character.

Leading and trailing spaces are automatically deleted from option names and values.

You can use the escape sequences `\b`, `\t`, `\n`, `\r`, `\\`, and `\s` in option values to represent the backspace, tab, newline, carriage return, backslash, and space characters. In option files, these escaping rules apply:

- A backslash followed by a valid escape sequence character is converted to the character represented by the sequence. For example, `\s` is converted to a space.

- A backslash not followed by a valid escape sequence character remains unchanged. For example, `\S` is retained as is.

The preceding rules mean that a literal backslash can be given as `\\`, or as `\` if it is not followed by a valid escape sequence character.

The rules for escape sequences in option files differ slightly from the rules for escape sequences in string literals in SQL statements. In the latter context, if "*x*" is not a valid escape sequence character, `\x` becomes "*x*" rather than `\x`. See Section 9.1.1, "String Literals".

The escaping rules for option file values are especially pertinent for Windows path names, which use `\` as a path name separator. A separator in a Windows path name must be written as `\\` if it is followed by an escape sequence character. It can be written as `\\` or `\` if it is not. Alternatively, `/` may be used in Windows path names and are treated as `\`. Suppose that you want to specify a base directory of `C:\Program Files\MySQL\MySQL Server 5.6` in an option file. This can be done several ways. Some examples:

```
basedir="C:\Program Files\MySQL\MySQL Server 5.6"
basedir="C:\\Program Files\\MySQL\\MySQL Server 5.6"
basedir="C:/Program Files/MySQL/MySQL Server 5.6"
basedir=C:\\Program\sFiles\\MySQL\\MySQL\sServer\s5.6
```

If an option group name is the same as a program name, options in the group apply specifically to that program. For example, the `[mysqld]` and `[mysql]` groups apply to the `mysqld` server and the `mysql` client program, respectively.

The `[client]` option group is read by all client programs provided in MySQL distributions (but *not* by `mysqld`). To understand how third-party client programs that use the C API can use option files, see the C API documentation at mysql_options().

The `[client]` group enables you to specify options that apply to all clients. For example, `[client]` is the appropriate group to use to specify the password for connecting to the server. (But make sure that the option file is accessible only by yourself, so that other people cannot discover your password.) Be sure not to put an option in the `[client]` group unless it is recognized by *all* client programs that you use. Programs that do not understand the option quit after displaying an error message if you try to run them.

List more general option groups first and more specific groups later. For example, a `[client]` group is more general because it is read by all client programs, whereas a `[mysqldump]` group is read only by `mysqldump`. Options specified later override options specified earlier, so putting the option groups in the order `[client]`, `[mysqldump]` enables `mysqldump`-specific options to override `[client]` options.

Here is a typical global option file:

```
[client]
port=3306
socket=/tmp/mysql.sock

[mysqld]
port=3306
socket=/tmp/mysql.sock
key_buffer_size=16M
max_allowed_packet=8M

[mysqldump]
quick
```

Here is a typical user option file:

```
[client]
# The following password is sent to all standard MySQL clients
password="my password"

[mysql]
no-auto-rehash
connect_timeout=2

[mysqlhotcopy]
interactive-timeout
```

To create option groups to be read only by `mysqld` servers from specific MySQL release series, use groups with names of `[mysqld-5.5]`, `[mysqld-5.6]`, and so forth. The following group indicates that the `sql_mode` setting should be used only by MySQL servers with 5.6.x version numbers:

```
[mysqld-5.6]
sql_mode=TRADITIONAL
```

Option File Inclusions

It is possible to use `!include` directives in option files to include other option files and `!includedir` to search specific directories for option files. For example, to include the `/home/mydir/myopt.cnf` file, use the following directive:

```
!include /home/mydir/myopt.cnf
```

To search the `/home/mydir` directory and read option files found there, use this directive:

```
!includedir /home/mydir
```

MySQL makes no guarantee about the order in which option files in the directory are read.

Note

Any files to be found and included using the `!includedir` directive on Unix operating systems *must* have file names ending in `.cnf`. On Windows, this directive checks for files with the `.ini` or `.cnf` extension.

Write the contents of an included option file like any other option file. That is, it should contain groups of options, each preceded by a `[group]` line that indicates the program to which the options apply.

While an included file is being processed, only those options in groups that the current program is looking for are used. Other groups are ignored. Suppose that a `my.cnf` file contains this line:

```
!include /home/mydir/myopt.cnf
```

And suppose that `/home/mydir/myopt.cnf` looks like this:

```
[mysqladmin]
force

[mysqld]
key_buffer_size=16M
```

If `my.cnf` is processed by `mysqld`, only the `[mysqld]` group in `/home/mydir/myopt.cnf` is used. If the file is processed by `mysqladmin`, only the `[mysqladmin]` group is used. If the file is processed by any other program, no options in `/home/mydir/myopt.cnf` are used.

The `!includedir` directive is processed similarly except that all option files in the named directory are read.

If an option file contains `!include` or `!includedir` directives, files named by those directives are processed whenever the option file is processed, no matter where they appear in the file.

For inclusion directives to work, the file path should not be specified within quotes and should have no escape sequences. For example, the following statements provided in `my.ini` read the option file `myopts.ini`:

```
!include C:/ProgramData/MySQL/MySQL Server/myopts.ini
!include C:\ProgramData\MySQL\MySQL Server\myopts.ini
!include C:\\ProgramData\\MySQL\\MySQL Server\\myopts.ini
```

On Windows, if `!include /path/to/extra.ini` is the last line in the file, make sure that a newline is appended at the end or the line is ignored.

4.2.2.3 Command-Line Options that Affect Option-File Handling

Most MySQL programs that support option files handle the following options. Because these options affect option-file handling, they must be given on the command line and not in an option file. To work properly, each of these options must be given before other options, with these exceptions:

- `--print-defaults` may be used immediately after `--defaults-file`, `--defaults-extra-file`, or `--login-path`.
- On Windows, if the server is started with the `--defaults-file` and `--install` options, `--install` must be first. See Section 2.3.4.7, “Starting MySQL as a Windows Service”.

When specifying file names as option values, avoid the use of the ~ shell metacharacter because it might not be interpreted as you expect.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file and (on all platforms) before the login path file. (For information about the order in which option files are used, see Section 4.2.2.2, “Using Option Files”.) If the file does not exist or is otherwise inaccessible, an

error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

See the introduction to this section regarding constraints on the position in which this option may be specified.

- `--defaults-file=`*`file_name`*

  Read only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  Exception: Even with `--defaults-file`, client programs read `.mylogin.cnf`.

  See the introduction to this section regarding constraints on the position in which this option may be specified.

- `--defaults-group-suffix=`*`str`*

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, the `mysql` client normally reads the `[client]` and `[mysql]` groups. If the `--defaults-group-suffix=_other` option is given, `mysql` also reads the `[client_other]` and `[mysql_other]` groups.

- `--login-path=`*`name`*

  Read options from the named login path in the `.mylogin.cnf` login path file. A “login path” is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the `mysql_config_editor` utility. See Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”.

  A client program reads the option group corresponding to the named login path, in addition to option groups that the program reads by default. Consider this command:

  ```
  mysql --login-path=mypath
  ```

  By default, the `mysql` client reads the `[client]` and `[mysql]` option groups. So for the command shown, `mysql` reads `[client]` and `[mysql]` from other option files, and `[client]`, `[mysql]`, and `[mypath]` from the login path file.

  Client programs read the login path file even when the `--no-defaults` option is used.

  To specify an alternate login path file name, set the `MYSQL_TEST_LOGIN_FILE` environment variable.

  See the introduction to this section regarding constraints on the position in which this option may be specified.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

  The exception is that client programs read the `.mylogin.cnf` login path file, if it exists, even when `--no-defaults` is used. This permits passwords to be specified in a safer way than on the command line even if `--no-defaults` is present. (`.mylogin.cnf` is created by the `mysql_config_editor` utility. See Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”.)

- `--print-defaults`

  Print the program name and all options that it gets from option files.

See the introduction to this section regarding constraints on the position in which this option may be specified.

4.2.2.4 Program Option Modifiers

Some options are “boolean” and control behavior that can be turned on or off. For example, the `mysql` client supports a `--column-names` option that determines whether or not to display a row of column names at the beginning of query results. By default, this option is enabled. However, you may want to disable it in some instances, such as when sending the output of `mysql` into another program that expects to see only data and not an initial header line.

To disable column names, you can specify the option using any of these forms:

```
--disable-column-names
--skip-column-names
--column-names=0
```

The `--disable` and `--skip` prefixes and the `=0` suffix all have the same effect: They turn the option off.

The “enabled” form of the option may be specified in any of these ways:

```
--column-names
--enable-column-names
--column-names=1
```

The values `ON`, `TRUE`, `OFF`, and `FALSE` are also recognized for boolean options (not case-sensitive).

If an option is prefixed by `--loose`, a program does not exit with an error if it does not recognize the option, but instead issues only a warning:

```
shell> mysql --loose-no-such-option
mysql: WARNING: unknown option '--loose-no-such-option'
```

The `--loose` prefix can be useful when you run programs from multiple installations of MySQL on the same machine and list options in an option file. An option that may not be recognized by all versions of a program can be given using the `--loose` prefix (or `loose` in an option file). Versions of the program that recognize the option process it normally, and versions that do not recognize it issue a warning and ignore it.

The `--maximum` prefix is available for `mysqld` only and permits a limit to be placed on how large client programs can set session system variables. To do this, use a `--maximum` prefix with the variable name. For example, `--maximum-max_heap_table_size=32M` prevents any client from making the heap table size limit larger than 32M.

The `--maximum` prefix is intended for use with system variables that have a session value. If applied to a system variable that has only a global value, an error occurs. For example, with `--maximum-back_log=200`, the server produces this error:

```
Maximum value of 'back_log' cannot be set
```

4.2.2.5 Using Options to Set Program Variables

Many MySQL programs have internal variables that can be set at runtime using the `SET` statement. See Section 13.7.4.1, “SET Syntax for Variable Assignment”, and Section 5.1.8, “Using System Variables”.

Most of these program variables also can be set at server startup by using the same syntax that applies to specifying program options. For example, `mysql` has a `max_allowed_packet` variable that controls the maximum size of its communication buffer. To set the `max_allowed_packet` variable for `mysql` to a value of 16MB, use either of the following commands:

```
mysql --max_allowed_packet=16777216
```

```
mysql --max_allowed_packet=16M
```

The first command specifies the value in bytes. The second specifies the value in megabytes. For variables that take a numeric value, the value can be given with a suffix of `K`, `M`, or `G` (either uppercase or lowercase) to indicate a multiplier of 1024, 1024^2 or 1024^3. (For example, when used to set `max_allowed_packet`, the suffixes indicate units of kilobytes, megabytes, or gigabytes.)

In an option file, variable settings are given without the leading dashes:

```
[mysql]
max_allowed_packet=16777216
```

Or:

```
[mysql]
max_allowed_packet=16M
```

If you like, underscores in an option name can be specified as dashes. The following option groups are equivalent. Both set the size of the server's key buffer to 512MB:

```
[mysqld]
key_buffer_size=512M

[mysqld]
key-buffer-size=512M
```

A variable can be specified by writing it in full or as any unambiguous prefix. For example, the `max_allowed_packet` variable can be set for `mysql` as `--max_a`, but not as `--max` because the latter is ambiguous:

```
shell> mysql --max=1000000
mysql: ambiguous option '--max=1000000' (max_allowed_packet, max_join_size)
```

Be aware that the use of variable prefixes can cause problems in the event that new variables are implemented for a program. A prefix that is unambiguous now might become ambiguous in the future.

Suffixes for specifying a value multiplier can be used when setting a variable at program invocation time, but not to set the value with `SET` at runtime. On the other hand, with `SET`, you can assign a variable's value using an expression, which is not true when you set a variable at server startup. For example, the first of the following lines is legal at program invocation time, but the second is not:

```
shell> mysql --max_allowed_packet=16M
shell> mysql --max_allowed_packet=16*1024*1024
```

Conversely, the second of the following lines is legal at runtime, but the first is not:

```
mysql> SET GLOBAL max_allowed_packet=16M;
mysql> SET GLOBAL max_allowed_packet=16*1024*1024;
```

4.2.2.6 Option Defaults, Options Expecting Values, and the = Sign

By convention, long forms of options that assign a value are written with an equals (`=`) sign, like this:

```
mysql --host=tonfisk --user=jon
```

For options that require a value (that is, not having a default value), the equal sign is not required, and so the following is also valid:

```
mysql --host tonfisk --user jon
```

In both cases, the `mysql` client attempts to connect to a MySQL server running on the host named “tonfisk” using an account with the user name “jon”.

Due to this behavior, problems can occasionally arise when no value is provided for an option that expects one. Consider the following example, where a user connects to a MySQL server running on host `tonfisk` as user `jon`:

```
shell> mysql --host 85.224.35.45 --user jon
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 3
Server version: 5.6.51 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT CURRENT_USER();
+----------------+
| CURRENT_USER() |
+----------------+
| jon@%          |
+----------------+
1 row in set (0.00 sec)
```

Omitting the required value for one of these option yields an error, such as the one shown here:

```
shell> mysql --host 85.224.35.45 --user
mysql: option '--user' requires an argument
```

In this case, `mysql` was unable to find a value following the `--user` option because nothing came after it on the command line. However, if you omit the value for an option that is *not* the last option to be used, you obtain a different error that you may not be expecting:

```
shell> mysql --host --user jon
ERROR 2005 (HY000): Unknown MySQL server host '--user' (1)
```

Because `mysql` assumes that any string following `--host` on the command line is a host name, `--host --user` is interpreted as `--host=--user`, and the client attempts to connect to a MySQL server running on a host named “--user”.

Options having default values always require an equal sign when assigning a value; failing to do so causes an error. For example, the MySQL server `--log-error` option has the default value *`host_name`*`.err`, where *`host_name`* is the name of the host on which MySQL is running. Assume that you are running MySQL on a computer whose host name is “tonfisk”, and consider the following invocation of `mysqld_safe`:

```
shell> mysqld_safe &
[1] 11699
shell> 080112 12:53:40 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080112 12:53:40 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
shell>
```

After shutting down the server, restart it as follows:

```
shell> mysqld_safe --log-error &
[1] 11699
shell> 080112 12:53:40 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080112 12:53:40 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
shell>
```

The result is the same, since `--log-error` is not followed by anything else on the command line, and it supplies its own default value. (The `&` character tells the operating system to run MySQL in the background; it is ignored by MySQL itself.) Now suppose that you wish to log errors to a file named `my-errors.err`. You might try starting the server with `--log-error my-errors`, but this does not have the intended effect, as shown here:

```
shell> mysqld_safe --log-error my-errors &
[1] 31357
shell> 080111 22:53:31 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
080111 22:53:32 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var
080111 22:53:34 mysqld_safe mysqld from pid file /usr/local/mysql/var/tonfisk.pid ended

[1]+  Done                    ./mysqld_safe --log-error my-errors
```

The server attempted to start using `/usr/local/mysql/var/tonfisk.err` as the error log, but then shut down. Examining the last few lines of this file shows the reason:

```
shell> tail /usr/local/mysql/var/tonfisk.err
080111 22:53:32  InnoDB: Started; log sequence number 0 46409
/usr/local/mysql/libexec/mysqld: Too many arguments (first extra is 'my-errors').
Use --verbose --help to get a list of available options
080111 22:53:32 [ERROR] Aborting

080111 22:53:32  InnoDB: Starting shutdown...
080111 22:53:34  InnoDB: Shutdown completed; log sequence number 0 46409
080111 22:53:34 [Note] /usr/local/mysql/libexec/mysqld: Shutdown complete

080111 22:53:34 mysqld_safe mysqld from pid file /usr/local/mysql/var/tonfisk.pid ended
```

Because the `--log-error` option supplies a default value, you must use an equal sign to assign a different value to it, as shown here:

```
shell> mysqld_safe --log-error=my-errors &
[1] 31437
shell> 080111 22:54:15 mysqld_safe Logging to '/usr/local/mysql/var/my-errors.err'.
080111 22:54:15 mysqld_safe Starting mysqld daemon with databases from /usr/local/mysql/var

shell>
```

Now the server has been started successfully, and is logging errors to the file `/usr/local/mysql/var/my-errors.err`.

Similar issues can arise when specifying option values in option files. For example, consider a `my.cnf` file that contains the following:

```
[mysql]

host
user
```

When the `mysql` client reads this file, these entries are parsed as `--host --user` or `--host=--user`, with the result shown here:

```
shell> mysql
ERROR 2005 (HY000): Unknown MySQL server host '--user' (1)
```

However, in option files, an equal sign is not assumed. Suppose the `my.cnf` file is as shown here:

```
[mysql]

user jon
```

Trying to start `mysql` in this case causes a different error:

```
shell> mysql
mysql: unknown option '--user jon'
```

A similar error would occur if you were to write `host tonfisk` in the option file rather than `host=tonfisk`. Instead, you must use the equal sign:

```
[mysql]

user=jon
```

Now the login attempt succeeds:

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.6.51 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT USER();
+---------------+
```

```
| USER()        |
+---------------+
| jon@localhost |
+---------------+
1 row in set (0.00 sec)
```

This is not the same behavior as with the command line, where the equal sign is not required:

```
shell> mysql --user jon --host tonfisk
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 6
Server version: 5.6.51 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT USER();
+---------------+
| USER()        |
+---------------+
| jon@tonfisk   |
+---------------+
1 row in set (0.00 sec)
```

Specifying an option requiring a value without a value in an option file causes the server to abort with an error.

4.2.3 Command Options for Connecting to the Server

This section describes options supported by most MySQL client programs that control how client programs establish connections to the server and whether connections are encrypted. These options can be given on the command line or in an option file.

- Command Options for Connection Establishment
- Command Options for Encrypted Connections

Command Options for Connection Establishment

This section describes options that control how client programs establish connections to the server. For additional information and examples showing how to use them, see Section 4.2.4, “Connecting to the MySQL Server Using Command Options”.

Table 4.3 Connection-Establishment Option Summary

Option Name	Description
--default-auth	Authentication plugin to use
--host	Host on which MySQL server is located
--password	Password to use when connecting to server
--pipe	Connect to server using named pipe (Windows only)
--plugin-dir	Directory where plugins are installed
--port	TCP/IP port number for connection
--protocol	Transport protocol to use
--secure-auth	Do not send passwords to server in old (pre-4.1) format
--shared-memory-base-name	Shared-memory name for shared-memory connections (Windows only)
--socket	Unix socket file or Windows named pipe to use
--user	MySQL user name to use when connecting to server

- `--default-auth=`*`plugin`*

A hint about which client-side authentication plugin to use. See Section 6.2.11, “Pluggable Authentication”.

- `--host=host_name`, `-h host_name`

  The host on which the MySQL server is running. The value can be a host name, IPv4 address, or IPv6 address. The default value is `localhost`.

- `--password[=pass_val]`, `-p[pass_val]`

  The password of the MySQL account used for connecting to the server. The password value is optional. If not given, the program prompts for one. If given, there must be *no space* between `--password=` or `-p` and the password following it. If no password option is specified, the default is to send no password.

  Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

  To explicitly specify that there is no password and that the client program should not prompt for one, use the `--skip-password` option.

- `--pipe`, `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--plugin-dir=dir_name`

  The directory in which to look for plugins. Specify this option if the `--default-auth` option is used to specify an authentication plugin but the client program does not find it. See Section 6.2.11, “Pluggable Authentication”.

- `--port=port_num`, `-P port_num`

  For TCP/IP connections, the port number to use. The default port number is 3306.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  This option explicitly specifies which transport protocol to use for connecting to the server. It is useful when other connection parameters normally result in use of a protocol other than the one you want. For example, connections on Unix to `localhost` are made using a Unix socket file by default:

```
mysql --host=localhost
```

  To force TCP/IP transport to be used instead, specify a `--protocol` option:

```
mysql --host=localhost --protocol=TCP
```

  The following table shows the permissible `--protocol` option values and indicates the applicable platforms for each value. The values are not case-sensitive.

`--protocol` Value	**Transport Protocol Used**	**Applicable Platforms**
`TCP`	TCP/IP transport to local or remote server	All
`SOCKET`	Unix socket-file transport to local server	Unix and Unix-like systems
`PIPE`	Named-pipe transport to local server	Windows
`MEMORY`	Shared-memory transport to local server	Windows

See also Section 4.2.5, “Connection Transport Protocols”

- `--secure-auth`

  Do not send passwords to the server in old (pre-4.1) format. This prevents connections except for servers that use the newer password format. This option is enabled by default; use `--skip-secure-auth` to disable it.

  Note

  Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated; expect support for them to be removed in a future MySQL release. For account upgrade instructions, see Section 6.4.1.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”.

  Note

  This option is deprecated; expect it to be removed in a future release. As of MySQL 5.7.5, it is always enabled and attempting to disable it produces an error.

- `--shared-memory-base-name=`*`name`*

  On Windows, the shared-memory name to use for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case-sensitive.

  This option applies only if the server was started with the `shared_memory` system variable enabled to support shared-memory connections.

- `--socket=`*`path`*, `-S` *`path`*

  On Unix, the name of the Unix socket file to use for connections made using a named pipe to a local server. The default Unix socket file name is `/tmp/mysql.sock`.

  On Windows, the name of the named pipe to use for connections to a local server. The default Windows pipe name is `MySQL`. The pipe name is not case-sensitive.

  On Windows, this option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--user=`*`user_name`*, `-u` *`user_name`*

  The user name of the MySQL account to use for connecting to the server. The default user name is `ODBC` on Windows or your Unix login name on Unix.

Command Options for Encrypted Connections

This section describes options for client programs that specify whether to use encrypted connections to the server, the names of certificate and key files, and other parameters related to encrypted-connection support. For examples of suggested use and how to check whether a connection is encrypted, see Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

Note

These options have an effect only for connections that use a transport protocol subject to encryption; that is, TCP/IP and Unix socket-file connections. See Section 4.2.5, “Connection Transport Protocols”

For information about using encrypted connections from the MySQL C API, see C API Support for Encrypted Connections.

Table 4.4 Connection-Encryption Option Summary

Option Name	Description	Introduced
--server-public-key-path	Path name to file containing RSA public key	
--skip-ssl	Disable connection encryption	
--ssl	Enable connection encryption	
--ssl-ca	File that contains list of trusted SSL Certificate Authorities	
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files	
--ssl-cert	File that contains X.509 certificate	
--ssl-cipher	Permissible ciphers for connection encryption	
--ssl-crl	File that contains certificate revocation lists	
--ssl-crlpath	Directory that contains certificate revocation-list files	
--ssl-key	File that contains X.509 key	
--ssl-mode	Desired security state of connection to server	5.6.30
--ssl-verify-server-cert	Verify host name against server certificate Common Name identity	

- `--server-public-key-path=`*`file_name`*

  The path name to a file in PEM format containing a client-side copy of the public key required by the server for RSA key pair-based password exchange. This option applies to clients that connect to the server with the `sha256_password` authentication plugin. This option is ignored for accounts that do not authenticate with that plugin. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

  This option is available only if MySQL was built using OpenSSL.

  For information about the `sha256_password` plugin, see Section 6.4.1.4, “SHA-256 Pluggable Authentication”.

- `--ssl`, `--skip-ssl`

  This option permits but does not require the client to connect to the server using encryption. Therefore, this option is not sufficient in itself to cause an encrypted connection to be used. For example, if you specify this option for a client program but the server has not been configured to support encrypted connections, the client falls back to an unencrypted connection.

  `--ssl` may be implied by other `--ssl-`*`xxx`* options, as indicated in the descriptions for those options.

  To specify additional parameters for encrypted connections, consider setting at least the `ssl_cert` and `ssl_key` system variables on the server side and the `--ssl-ca` option on the client side. See Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

  The `--ssl` option in negated form indicates that encryption should *not* be used and overrides other `--ssl-`*`xxx`* options. Specify the option as `--skip-ssl` or a synonym (`--ssl=0`, `--disable-ssl`). For example, you might have options specified in the `[client]` group of your option file to use encrypted connections by default when you invoke MySQL client programs. To use an unencrypted connection instead, invoke the client program with `--ssl=0` on the command line to override the options in the option file.

To require use of encrypted connections by a MySQL account, use a `GRANT` statement for the account that includes a `REQUIRE SSL` clause. This causes connection attempts by clients that use the account to be rejected unless MySQL supports encrypted connections and an encrypted connection can be established.

The `REQUIRE` clause permits other encryption-related options, which can be used to enforce security requirements stricter than `REQUIRE SSL`. For additional details about which command options may or must be specified by clients that connect using accounts configured using the various `REQUIRE` options, see the description of `REQUIRE` in Section 13.7.1.4, “GRANT Statement”.

- `--ssl-ca=file_name`

  The path name of the Certificate Authority (CA) certificate file in PEM format. The file contains a list of trusted SSL Certificate Authorities. This option implies `--ssl`.

  To tell the client not to authenticate the server certificate when establishing an encrypted connection to the server, specify neither `--ssl-ca` nor `--ssl-capath`. The server still verifies the client according to any applicable requirements established for the client account, and it still uses any `ssl_ca` or `ssl_capath` system variable values specified on the server side.

  To specify the CA file for the server, set the `ssl_ca` system variable.

- `--ssl-capath=dir_name`

  The path name of the directory that contains trusted SSL certificate authority (CA) certificate files in PEM format. Support for this capability depends on the SSL library used to compile MySQL; see Section 6.3.4, “SSL Library-Dependent Capabilities”. This option implies `--ssl`.

  To tell the client not to authenticate the server certificate when establishing an encrypted connection to the server, specify neither `--ssl-ca` nor `--ssl-capath`. The server still verifies the client according to any applicable requirements established for the client account, and it still uses any `ssl_ca` or `ssl_capath` system variable values specified on the server side.

  To specify the CA directory for the server, set the `ssl_capath` system variable.

- `--ssl-cert=file_name`

  The path name of the client SSL public key certificate file in PEM format. This option implies `--ssl`.

  To specify the server SSL public key certificate file, set the `ssl_cert` system variable.

- `--ssl-cipher=cipher_list`

  The list of permissible ciphers for connection encryption. If no cipher in the list is supported, encrypted connections do not work. This option implies `--ssl`.

  For greatest portability, `cipher_list` should be a list of one or more cipher names, separated by colons. This format is understood both by OpenSSL and yaSSL. Examples:

  ```
  --ssl-cipher=AES128-SHA
  --ssl-cipher=DHE-RSA-AES256-SHA:AES128-SHA
  ```

  OpenSSL supports a more flexible syntax for specifying ciphers, as described in the OpenSSL documentation at https://www.openssl.org/docs/manmaster/man1/ciphers.html. yaSSL does not, so attempts to use that extended syntax fail for a MySQL distribution compiled using yaSSL.

  For information about which encryption ciphers MySQL supports, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

  To specify the encryption ciphers for the server, set the `ssl_cipher` system variable.

- `--ssl-crl=file_name`

  The path name of the file containing certificate revocation lists in PEM format. Support for revocation-list capability depends on the SSL library used to compile MySQL. See Section 6.3.4, “SSL Library-Dependent Capabilities”. This option implies `--ssl`.

  If neither `--ssl-crl` nor `--ssl-crlpath` is given, no CRL checks are performed, even if the CA path contains certificate revocation lists.

  To specify the revocation-list file for the server, set the `ssl_crl` system variable.

- `--ssl-crlpath=dir_name`

  The path name of the directory that contains certificate revocation-list files in PEM format. Support for revocation-list capability depends on the SSL library used to compile MySQL. See Section 6.3.4, “SSL Library-Dependent Capabilities”. This option implies `--ssl`.

  If neither `--ssl-crl` nor `--ssl-crlpath` is given, no CRL checks are performed, even if the CA path contains certificate revocation lists.

  To specify the revocation-list directory for the server, set the `ssl_crlpath` system variable.

- `--ssl-key=file_name`

  The path name of the client SSL private key file in PEM format. For better security, use a certificate with an RSA key size of at least 2048 bits. This option implies `--ssl`.

  If the key file is protected by a passphrase, the client program prompts the user for the passphrase. The password must be given interactively; it cannot be stored in a file. If the passphrase is incorrect, the program continues as if it could not read the key.

  To specify the server SSL private key file, set the `ssl_key` system variable.

- `--ssl-mode=mode`

  This option specifies the desired security state of the connection to the server:

  - If this option is not specified, the default is to establish an unencrypted connection. This is like the `--ssl=0` option or its synonyms (`--skip-ssl`, `--disable-ssl`).
  - If this option is specified, the only permissible mode value is `REQUIRED` (establish an encrypted connection if the server supports encrypted connections). The connection attempt fails if an encrypted connection cannot be established.

  The `--ssl-mode` option was added in MySQL 5.6.30.

Note

To require encrypted connections in MySQL 5.6, the standard MySQL client programs check whether the connection is encrypted if `--ssl-mode=REQUIRED` was specified. If not, the client exits with an error. Third-party applications that must be able to require encrypted connections can use the same technique. For details, see mysql_ssl_set().

- `--ssl-verify-server-cert`

  This option causes the client to perform host name identity verification by checking the host name the client uses for connecting to the server against the identity in the certificate that the server sends to the client:

  - As of MySQL 5.6.41, if the client uses OpenSSL 1.0.2 or higher, the client checks whether the host name that it uses for connecting matches either the Subject Alternative Name value or the Common Name value in the server certificate.
  - Otherwise, the client checks whether the host name that it uses for connecting matches the Common Name value in the server certificate.

  The connection fails if there is a mismatch. For encrypted connections, this option helps prevent man-in-the-middle attacks. Host name identity verification is disabled by default.

4.2.4 Connecting to the MySQL Server Using Command Options

This section describes use of command-line options to specify how to establish connections to the MySQL server, for clients such as `mysql` or `mysqldump`. For additional information if you are unable to connect, see Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”.

For a client program to connect to the MySQL server, it must use the proper connection parameters, such as the name of the host where the server is running and the user name and password of your MySQL account. Each connection parameter has a default value, but you can override default values as necessary using program options specified either on the command line or in an option file.

The examples here use the `mysql` client program, but the principles apply to other clients such as `mysqldump`, `mysqladmin`, or `mysqlshow`.

This command invokes `mysql` without specifying any explicit connection parameters:

```
mysql
```

Because there are no parameter options, the default values apply:

- The default host name is `localhost`. On Unix, this has a special meaning, as described later.
- The default user name is `ODBC` on Windows or your Unix login name on Unix.
- No password is sent because neither `--password` nor `-p` is given.
- For `mysql`, the first nonoption argument is taken as the name of the default database. Because there is no such argument, `mysql` selects no default database.

To specify the host name and user name explicitly, as well as a password, supply appropriate options on the command line. To select a default database, add a database-name argument. Examples:

```
mysql --host=localhost --user=myname --password=password mydb
mysql -h localhost -u myname -ppassword mydb
```

For password options, the password value is optional:

- If you use a `--password` or `-p` option and specify a password value, there must be *no space* between `--password=` or `-p` and the password following it.
- If you use `--password` or `-p` but do not specify a password value, the client program prompts you to enter the password. The password is not displayed as you enter it. This is more secure than giving the password on the command line, which might enable other users on your system to see the password line by executing a command such as `ps`. See Section 6.1.2.1, “End-User Guidelines for Password Security”.
- To explicitly specify that there is no password and that the client program should not prompt for one, use the `--skip-password` option.

As just mentioned, including the password value on the command line is a security risk. To avoid this risk, specify the `--password` or `-p` option without any following password value:

```
mysql --host=localhost --user=myname --password mydb
mysql -h localhost -u myname -p mydb
```

When the `--password` or `-p` option is given with no password value, the client program prints a prompt and waits for you to enter the password. (In these examples, `mydb` is *not* interpreted as a password because it is separated from the preceding password option by a space.)

On some systems, the library routine that MySQL uses to prompt for a password automatically limits the password to eight characters. That limitation is a property of the system library, not MySQL. Internally, MySQL does not have any limit for the length of the password. To work around the limitation on systems affected by it, specify your password in an option file (see Section 4.2.2.2, “Using Option Files”). Another workaround is to change your MySQL password to a value that has eight or fewer characters, but that has the disadvantage that shorter passwords tend to be less secure.

Client programs determine what type of connection to make as follows:

- If the host is not specified or is `localhost`, a connection to the local host occurs:
  - On Windows, the client connects using shared memory, if the server was started with the `shared_memory` system variable enabled to support shared-memory connections.
  - On Unix, MySQL programs treat the host name `localhost` specially, in a way that is likely different from what you expect compared to other network-based programs: the client connects using a Unix socket file. The `--socket` option or the `MYSQL_UNIX_PORT` environment variable may be used to specify the socket name.
- On Windows, if `host` is `.` (period), or TCP/IP is not enabled and `--socket` is not specified or the host is empty, the client connects using a named pipe, if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. If named-pipe connections are not supported or if the user making the connection is not a member of the Windows group specified by the `named_pipe_full_access_group` system variable, an error occurs.
- Otherwise, the connection uses TCP/IP.

The `--protocol` option enables you to use a particular transport protocol even when other options normally result in use of a different protocol. That is, `--protocol` specifies the transport protocol explicitly and overrides the preceding rules, even for `localhost`.

Only connection options that are relevant to the selected transport protocol are used or checked. Other connection options are ignored. For example, with `--host=localhost` on Unix, the client attempts to connect to the local server using a Unix socket file, even if a `--port` or `-P` option is given to specify a TCP/IP port number.

To ensure that the client makes a TCP/IP connection to the local server, use `--host` or `-h` to specify a host name value of `127.0.0.1` (instead of `localhost`), or the IP address or name of the local server. You can also specify the transport protocol explicitly, even for `localhost`, by using the `--protocol=TCP` option. Examples:

```
mysql --host=127.0.0.1
mysql --protocol=TCP
```

If the server is configured to accept IPv6 connections, clients can connect to the local server over IPv6 using `--host=::1`. See Section 5.1.12, “IPv6 Support”.

On Windows, to force a MySQL client to use a named-pipe connection, specify the `--pipe` or `--protocol=PIPE` option, or specify `.` (period) as the host name. If the server was not started with the `named_pipe` system variable enabled to support named-pipe connections or if the user making the connection is not a member of the Windows group specified by the `named_pipe_full_access_group` system variable, an error occurs. Use the `--socket` option to specify the name of the pipe if you do not want to use the default pipe name.

Connections to remote servers use TCP/IP. This command connects to the server running on `remote.example.com` using the default port number (3306):

```
mysql --host=remote.example.com
```

To specify a port number explicitly, use the `--port` or `-P` option:

```
mysql --host=remote.example.com --port=13306
```

You can specify a port number for connections to a local server, too. However, as indicated previously, connections to `localhost` on Unix use a socket file by default, so unless you force a TCP/IP connection as previously described, any option that specifies a port number is ignored.

For this command, the program uses a socket file on Unix and the `--port` option is ignored:

```
mysql --port=13306 --host=localhost
```

To cause the port number to be used, force a TCP/IP connection. For example, invoke the program in either of these ways:

```
mysql --port=13306 --host=127.0.0.1
mysql --port=13306 --protocol=TCP
```

For additional information about options that control how client programs establish connections to the server, see Section 4.2.3, “Command Options for Connecting to the Server”.

It is possible to specify connection parameters without entering them on the command line each time you invoke a client program:

- Specify the connection parameters in the `[client]` section of an option file. The relevant section of the file might look like this:

  ```
  [client]
  host=host_name
  user=user_name
  password=password
  ```

  For more information, see Section 4.2.2.2, “Using Option Files”.

- Some connection parameters can be specified using environment variables. Examples:

  - To specify the host for `mysql`, use `MYSQL_HOST`.
  - On Windows, to specify the MySQL user name, use `USER`.
  - To specify the password, use `MYSQL_PWD`. However, this is insecure; see Section 6.1.2.1, “End-User Guidelines for Password Security”.

  For a list of supported environment variables, see Section 4.9, “Environment Variables”.

4.2.5 Connection Transport Protocols

For programs that use the MySQL client library (for example, `mysql` and `mysqldump`), MySQL supports connections to the server based on several transport protocols: TCP/IP, Unix socket file, named pipe, and shared memory. This section describes how to select these protocols, and how they are similar and different.

- Transport Protocol Selection
- Transport Support for Local and Remote Connections
- Interpretation of localhost
- Encryption and Security Characteristics
- Connection Compression

Transport Protocol Selection

For a given connection, if the transport protocol is not specified explicitly, it is determined implicitly. For example, connections to `localhost` result in a socket file connection on Unix and Unix-like systems, and a TCP/IP connection to `127.0.0.1` otherwise. For additional information, see Section 4.2.4, "Connecting to the MySQL Server Using Command Options".

To specify the protocol explicitly, use the `--protocol` command option. The following table shows the permissible values for `--protocol` and indicates the applicable platforms for each value. The values are not case-sensitive.

`--protocol` Value	Transport Protocol Used	Applicable Platforms
`TCP`	TCP/IP	All
`SOCKET`	Unix socket file	Unix and Unix-like systems
`PIPE`	Named pipe	Windows
`MEMORY`	Shared memory	Windows

Transport Support for Local and Remote Connections

TCP/IP transport supports connections to local or remote MySQL servers.

Socket-file, named-pipe, and shared-memory transports support connections only to local MySQL servers. (Named-pipe transport does allow for remote connections, but this capability is not implemented in MySQL.)

Interpretation of localhost

If the transport protocol is not specified explicitly, `localhost` is interpreted as follows:

- On Unix and Unix-like systems, a connection to `localhost` results in a socket-file connection.
- Otherwise, a connection to `localhost` results in a TCP/IP connection to `127.0.0.1`.

If the transport protocol is specified explicitly, `localhost` is interpreted with respect to that protocol. For example, with `--protocol=TCP`, a connection to `localhost` results in a TCP/IP connection to `127.0.0.1` on all platforms.

Encryption and Security Characteristics

TCP/IP and socket-file transports are subject to TLS/SSL encryption, using the options described in Command Options for Encrypted Connections. Named-pipe and shared-memory transports are not subject to TLS/SSL encryption.

A connection is secure by default if made over a transport protocol that is secure by default. Otherwise, for protocols that are subject to TLS/SSL encryption, a connection may be made secure using encryption:

- TCP/IP connections are not secure by default, but can be encrypted to make them secure.
- Socket-file connections are secure by default. They can also be encrypted, but encrypting a socket-file connection makes it no more secure and increases CPU load.
- Named-pipe connections are not secure by default, and are not subject to encryption to make them secure. However, the `named_pipe_full_access_group` system variable is available to control which MySQL users are permitted to use named-pipe connections.
- Shared-memory connections are secure by default.

Per the preceding remarks, connections that use TCP/IP encrypted using TLS/SSL, a socket file, or shared memory are secure connections. TCP/IP connections not encrypted using TLS/SSL and named-pipe connections are not secure.

If the `require_secure_transport` system variable is enabled, the server permits only connections that use some form of secure transport.

See also Configuring Encrypted Connections as Mandatory.

Connection Compression

All transport protocols are subject to use of compression on the traffic between the client and server. If both compression and encryption are used for a given connection, compression occurs before encryption. For more information, see Section 4.2.6, “Connection Compression Control”.

4.2.6 Connection Compression Control

Connections to the server can use compression on the traffic between client and server to reduce the number of bytes sent over the connection. By default, connections are uncompressed, but can be compressed if the server and the client both support compression.

Compressed connections originate on the client side but affect CPU load on both the client and server sides because both sides perform compression and decompression operations. Because enabling compression decreases performance, its benefits occur primarily when there is low network bandwidth, network transfer time dominates the cost of compression and decompression operations, and result sets are large.

Compression control applies to connections to the server by client programs and by servers participating in source/replica replication. Compression control does not apply to connections for `FEDERATED` tables.

These configuration parameters are available for controlling connection compression:

- Client programs support a `--compress` command-line option to specify use of compression for the connection to the server.
- For programs that use the MySQL C API, enabling the `MYSQL_OPT_COMPRESS` option for the `mysql_options()` function specifies use of compression for the connection to the server.
- For source/replica replication, enabling the `slave_compressed_protocol` system variable specifies use of compression for replica connections to the source.

In each case, when use of compression is specified, the connection uses the `zlib` compression algorithm if both sides support it, with fallback to an uncompressed connection otherwise.

4.2.7 Setting Environment Variables

Environment variables can be set at the command prompt to affect the current invocation of your command processor, or set permanently to affect future invocations. To set a variable permanently, you can set it in a startup file or by using the interface provided by your system for this purpose. Consult the documentation for your command interpreter for specific details. Section 4.9, “Environment Variables”, lists all environment variables that affect MySQL program operation.

To specify a value for an environment variable, use the syntax appropriate for your command processor. For example, on Windows, you can set the `USER` variable to specify your MySQL account name. To do so, use this syntax:

```
SET USER=your_name
```

The syntax on Unix depends on your shell. Suppose that you want to specify the TCP/IP port number using the `MYSQL_TCP_PORT` variable. Typical syntax (such as for `sh`, `ksh`, `bash`, `zsh`, and so on) is as follows:

```
MYSQL_TCP_PORT=3306
export MYSQL_TCP_PORT
```

The first command sets the variable, and the `export` command exports the variable to the shell environment so that its value becomes accessible to MySQL and other processes.

For csh and tcsh, use setenv to make the shell variable available to the environment:

```
setenv MYSQL_TCP_PORT 3306
```

The commands to set environment variables can be executed at your command prompt to take effect immediately, but the settings persist only until you log out. To have the settings take effect each time you log in, use the interface provided by your system or place the appropriate command or commands in a startup file that your command interpreter reads each time it starts.

On Windows, you can set environment variables using the System Control Panel (under Advanced).

On Unix, typical shell startup files are .bashrc or .bash_profile for bash, or .tcshrc for tcsh.

Suppose that your MySQL programs are installed in /usr/local/mysql/bin and that you want to make it easy to invoke these programs. To do this, set the value of the PATH environment variable to include that directory. For example, if your shell is bash, add the following line to your .bashrc file:

```
PATH=${PATH}:/usr/local/mysql/bin
```

bash uses different startup files for login and nonlogin shells, so you might want to add the setting to .bashrc for login shells and to .bash_profile for nonlogin shells to make sure that PATH is set regardless.

If your shell is tcsh, add the following line to your .tcshrc file:

```
setenv PATH ${PATH}:/usr/local/mysql/bin
```

If the appropriate startup file does not exist in your home directory, create it with a text editor.

After modifying your PATH setting, open a new console window on Windows or log in again on Unix so that the setting goes into effect.

4.3 Server and Server-Startup Programs

This section describes mysqld, the MySQL server, and several programs that are used to start the server.

4.3.1 mysqld — The MySQL Server

mysqld, also known as MySQL Server, is a single multithreaded program that does most of the work in a MySQL installation. It does not spawn additional processes. MySQL Server manages access to the MySQL data directory that contains databases and tables. The data directory is also the default location for other information such as log files and status files.

Note

Some installation packages contain a debugging version of the server named mysqld-debug. Invoke this version instead of mysqld for debugging support, memory allocation checking, and trace file support (see Section 5.8.1.2, “Creating Trace Files”).

When MySQL server starts, it listens for network connections from client programs and manages access to databases on behalf of those clients.

The mysqld program has many options that can be specified at startup. For a complete list of options, run this command:

```
shell> mysqld --verbose --help
```

MySQL Server also has a set of system variables that affect its operation as it runs. System variables can be set at server startup, and many of them can be changed at runtime to effect dynamic server reconfiguration. MySQL Server also has a set of status variables that provide information about its operation. You can monitor these status variables to access runtime performance characteristics.

For a full description of MySQL Server command options, system variables, and status variables, see Section 5.1, “The MySQL Server”. For information about installing MySQL and setting up the initial configuration, see Chapter 2, *Installing and Upgrading MySQL*.

4.3.2 mysqld_safe — MySQL Server Startup Script

`mysqld_safe` is the recommended way to start a `mysqld` server on Unix. `mysqld_safe` adds some safety features such as restarting the server when an error occurs and logging runtime information to an error log. A description of error logging is given later in this section.

`mysqld_safe` tries to start an executable named `mysqld`. To override the default behavior and specify explicitly the name of the server you want to run, specify a `--mysqld` or `--mysqld-version` option to `mysqld_safe`. You can also use `--ledir` to indicate the directory where `mysqld_safe` should look for the server.

Many of the options to `mysqld_safe` are the same as the options to `mysqld`. See Section 5.1.6, “Server Command Options”.

Options unknown to `mysqld_safe` are passed to `mysqld` if they are specified on the command line, but ignored if they are specified in the `[mysqld_safe]` group of an option file. See Section 4.2.2.2, “Using Option Files”.

`mysqld_safe` reads all options from the `[mysqld]`, `[server]`, and `[mysqld_safe]` sections in option files. For example, if you specify a `[mysqld]` section like this, `mysqld_safe` finds and uses the `--log-error` option:

```
[mysqld]
log-error=error.log
```

For backward compatibility, `mysqld_safe` also reads `[safe_mysqld]` sections, but to be current you should rename such sections to `[mysqld_safe]`.

`mysqld_safe` supports the following options. It also reads option files and supports the options for processing them described at Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

Table 4.5 mysqld_safe Options

Option Name	Description
--basedir	Path to MySQL installation directory
--core-file-size	Size of core file that mysqld should be able to create
--datadir	Path to data directory
--defaults-extra-file	Read named option file in addition to usual option files
--defaults-file	Read only named option file
--help	Display help message and exit
--ledir	Path to directory where server is located
--log-error	Write error log to named file
--malloc-lib	Alternative malloc library to use for mysqld
--mysqld	Name of server program to start (in ledir directory)
--mysqld-version	Suffix for server program name
--nice	Use nice program to set server scheduling priority
--no-defaults	Read no option files
--open-files-limit	Number of files that mysqld should be able to open
--pid-file	Path name of server process ID file
--plugin-dir	Directory where plugins are installed

Option Name	Description
--port	Port number on which to listen for TCP/IP connections
--skip-kill-mysqld	Do not try to kill stray mysqld processes
--skip-syslog	Do not write error messages to syslog; use error log file
--socket	Socket file on which to listen for Unix socket connections
--syslog	Write error messages to syslog
--syslog-tag	Tag suffix for messages written to syslog
--timezone	Set TZ time zone environment variable to named value
--user	Run mysqld as user having name user_name or numeric user ID user_id

- `--help`

  Display a help message and exit.

- `--basedir=dir_name`

  The path to the MySQL installation directory.

- `--core-file-size=size`

  The size of the core file that `mysqld` should be able to create. The option value is passed to `ulimit -c`.

- `--datadir=dir_name`

  The path to the data directory.

- `--defaults-extra-file=file_name`

  Read this option file in addition to the usual option files. If the file does not exist or is otherwise inaccessible, the server exits with an error. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name. This must be the first option on the command line if it is used.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, the server exits with an error. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name. This must be the first option on the command line if it is used.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--ledir=dir_name`

  If `mysqld_safe` cannot find the server, use this option to indicate the path name to the directory where the server is located.

  As of MySQL 5.6.35, this option is accepted only on the command line, not in option files.

- `--log-error=file_name`

  Write the error log to the given file. See Section 5.4.2, “The Error Log”.

- `--malloc-lib=[lib_name]`

The name of the library to use for memory allocation instead of the system `malloc()` library. As of MySQL 5.6.33, the option value must be one of the directories `/usr/lib`, `/usr/lib64`, `/usr/lib/i386-linux-gnu`, or `/usr/lib/x86_64-linux-gnu`. Prior to MySQL 5.6.33, any library can be used by specifying its path name, but there is a shortcut form to enable use of the `tcmalloc` library that is shipped with binary MySQL distributions for Linux in MySQL 5.6. It is possible that the shortcut form may not work under certain configurations, in which case you should specify a path name instead.

Note

As of MySQL 5.6.31, MySQL distributions no longer include a `tcmalloc` library.

The `--malloc-lib` option works by modifying the `LD_PRELOAD` environment value to affect dynamic linking to enable the loader to find the memory-allocation library when `mysqld` runs:

- If the option is not given, or is given without a value (`--malloc-lib=`), `LD_PRELOAD` is not modified and no attempt is made to use `tcmalloc`.
- Prior to MySQL 5.6.49, if the option is given as `--malloc-lib=tcmalloc`, `mysqld_safe` looks for a `tcmalloc` library in `/usr/lib` and then in the MySQL `pkglibdir` location (for example, `/usr/local/mysql/lib` or whatever is appropriate). If `tmalloc` is found, its path name is added to the beginning of the `LD_PRELOAD` value for `mysqld`. If `tcmalloc` is not found, `mysqld_safe` aborts with an error.

  As of MySQL 5.6.49, `tcmalloc` is not a permitted value for the `--malloc-lib` option.
- If the option is given as `--malloc-lib=`*`/path/to/some/library`*, that full path is added to the beginning of the `LD_PRELOAD` value. If the full path points to a nonexistent or unreadable file, `mysqld_safe` aborts with an error.
- For cases where `mysqld_safe` adds a path name to `LD_PRELOAD`, it adds the path to the beginning of any existing value the variable already has.

Linux users can use the `libtcmalloc_minimal.so` included in binary packages by adding these lines to the `my.cnf` file:

```
[mysqld_safe]
malloc-lib=tcmalloc
```

Those lines also suffice for users on any platform who have installed a `tcmalloc` package in `/usr/lib`. To use a specific `tcmalloc` library, specify its full path name. Example:

```
[mysqld_safe]
malloc-lib=/opt/lib/libtcmalloc_minimal.so
```

- `--mysqld=`*`prog_name`*

  The name of the server program (in the `ledir` directory) that you want to start. This option is needed if you use the MySQL binary distribution but have the data directory outside of the binary distribution. If `mysqld_safe` cannot find the server, use the `--ledir` option to indicate the path name to the directory where the server is located.

  As of MySQL 5.6.33, this option is accepted only on the command line, not in option files.

- `--mysqld-version=`*`suffix`*

  This option is similar to the `--mysqld` option, but you specify only the suffix for the server program name. The base name is assumed to be `mysqld`. For example, if you use `--mysqld-version=debug`, `mysqld_safe` starts the `mysqld-debug` program in the `ledir` directory. If the argument to `--mysqld-version` is empty, `mysqld_safe` uses `mysqld` in the `ledir` directory.

As of MySQL 5.6.33, this option is accepted only on the command line, not in option files.

- `--nice=priority`

  Use the `nice` program to set the server's scheduling priority to the given value.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read. This must be the first option on the command line if it is used.

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- `--open-files-limit=count`

  The number of files that `mysqld` should be able to open. The option value is passed to `ulimit -n`.

  Note

  You must start `mysqld_safe` as `root` for this to function properly.

- `--pid-file=file_name`

  The path name that `mysqld` should use for its process ID file.

- `--plugin-dir=dir_name`

  The path name of the plugin directory.

- `--port=port_num`

  The port number that the server should use when listening for TCP/IP connections. The port number must be 1024 or higher unless the server is started by the `root` operating system user.

- `--skip-kill-mysqld`

  Do not try to kill stray `mysqld` processes at startup. This option works only on Linux.

- `--socket=path`

  The Unix socket file that the server should use when listening for local connections.

- `--syslog`, `--skip-syslog`

  `--syslog` causes error messages to be sent to `syslog` on systems that support the `logger` program. `--skip-syslog` suppresses the use of `syslog`; messages are written to an error log file.

  When `syslog` is used, the `daemon.err` syslog facility/severity is used for all log messages.

  `mysqld_safe` ignores `--syslog` if `--log-error` is also given.

- `--syslog-tag=tag`

  For logging to `syslog`, messages from `mysqld_safe` and `mysqld` are written with identifiers of `mysqld_safe` and `mysqld`, respectively. To specify a suffix for the identifiers, use `--syslog-tag=tag`, which modifies the identifiers to be `mysqld_safe-tag` and `mysqld-tag`.

- `--timezone=timezone`

  Set the `TZ` time zone environment variable to the given option value. Consult your operating system documentation for legal time zone specification formats.

- `--user={user_name|user_id}`

  Run the `mysqld` server as the user having the name *`user_name`* or the numeric user ID *`user_id`*. (“User” in this context refers to a system login account, not a MySQL user listed in the grant tables.)

If you execute `mysqld_safe` with the `--defaults-file` or `--defaults-extra-file` option to name an option file, the option must be the first one given on the command line or the option file is not used. For example, this command does not use the named option file:

```
mysql> mysqld_safe --port=port_num --defaults-file=file_name
```

Instead, use the following command:

```
mysql> mysqld_safe --defaults-file=file_name --port=port_num
```

The `mysqld_safe` script is written so that it normally can start a server that was installed from either a source or a binary distribution of MySQL, even though these types of distributions typically install the server in slightly different locations. (See Section 2.1.5, “Installation Layouts”.) `mysqld_safe` expects one of the following conditions to be true:

- The server and databases can be found relative to the working directory (the directory from which `mysqld_safe` is invoked). For binary distributions, `mysqld_safe` looks under its working directory for `bin` and `data` directories. For source distributions, it looks for `libexec` and `var` directories. This condition should be met if you execute `mysqld_safe` from your MySQL installation directory (for example, `/usr/local/mysql` for a binary distribution).
- If the server and databases cannot be found relative to the working directory, `mysqld_safe` attempts to locate them by absolute path names. Typical locations are `/usr/local/libexec` and `/usr/local/var`. The actual locations are determined from the values configured into the distribution at the time it was built. They should be correct if MySQL is installed in the location specified at configuration time.

Because `mysqld_safe` tries to find the server and databases relative to its own working directory, you can install a binary distribution of MySQL anywhere, as long as you run `mysqld_safe` from the MySQL installation directory:

```
shell> cd mysql_installation_directory
shell> bin/mysqld_safe &
```

If `mysqld_safe` fails, even when invoked from the MySQL installation directory, specify the `--ledir` and `--datadir` options to indicate the directories in which the server and databases are located on your system.

In MySQL 5.6.5 and later, `mysqld_safe` tries to use the `sleep` and `date` system utilities to determine how many times it has attempted to start this second, and—if these are present and this is greater than 5 times—is forced to wait 1 full second before starting again. This is intended to prevent excessive CPU usage in the event of repeated failures. (Bug #11761530, Bug #54035)

When you use `mysqld_safe` to start `mysqld`, `mysqld_safe` arranges for error (and notice) messages from itself and from `mysqld` to go to the same destination.

There are several `mysqld_safe` options for controlling the destination of these messages:

- `--log-error=file_name`: Write error messages to the named error file.
- `--syslog`: Write error messages to `syslog` on systems that support the `logger` program.
- `--skip-syslog`: Do not write error messages to `syslog`. Messages are written to the default error log file (*`host_name`*`.err` in the data directory), or to a named file if the `--log-error` option is given.

If none of these options is given, the default is `--skip-syslog`.

If --log-error and --syslog are both given, a warning is issued and --log-error takes precedence.

When mysqld_safe writes a message, notices go to the logging destination (syslog or the error log file) and stdout. Errors go to the logging destination and stderr.

4.3.3 mysql.server — MySQL Server Startup Script

MySQL distributions on Unix and Unix-like system include a script named mysql.server, which starts the MySQL server using mysqld_safe. It can be used on systems such as Linux and Solaris that use System V-style run directories to start and stop system services. It is also used by the macOS Startup Item for MySQL.

mysql.server is the script name as used within the MySQL source tree. The installed name might be different (for example, mysqld or mysql). In the following discussion, adjust the name mysql.server as appropriate for your system.

To start or stop the server manually using the mysql.server script, invoke it from the command line with start or stop arguments:

```
shell> mysql.server start
shell> mysql.server stop
```

mysql.server changes location to the MySQL installation directory, then invokes mysqld_safe. To run the server as some specific user, add an appropriate user option to the [mysqld] group of the global /etc/my.cnf option file, as shown later in this section. (It is possible that you must edit mysql.server if you've installed a binary distribution of MySQL in a nonstandard location. Modify it to change location into the proper directory before it runs mysqld_safe. If you do this, your modified version of mysql.server may be overwritten if you upgrade MySQL in the future; make a copy of your edited version that you can reinstall.)

mysql.server stop stops the server by sending a signal to it. You can also stop the server manually by executing mysqladmin shutdown.

To start and stop MySQL automatically on your server, you must add start and stop commands to the appropriate places in your /etc/rc* files:

- If you use the Linux server RPM package (MySQL-server-*VERSION*.rpm), or a native Linux package installation, the mysql.server script may be installed in the /etc/init.d directory with the name mysqld or mysql. See Section 2.5.5, “Installing MySQL on Linux Using RPM Packages from Oracle”, for more information on the Linux RPM packages.

- If you install MySQL from a source distribution or using a binary distribution format that does not install mysql.server automatically, you can install the script manually. It can be found in the support-files directory under the MySQL installation directory or in a MySQL source tree. Copy the script to the /etc/init.d directory with the name mysql and make it executable:

  ```
  shell> cp mysql.server /etc/init.d/mysql
  shell> chmod +x /etc/init.d/mysql
  ```

  After installing the script, the commands needed to activate it to run at system startup depend on your operating system. On Linux, you can use chkconfig:

  ```
  shell> chkconfig --add mysql
  ```

  On some Linux systems, the following command also seems to be necessary to fully enable the mysql script:

  ```
  shell> chkconfig --level 345 mysql on
  ```

- On FreeBSD, startup scripts generally should go in /usr/local/etc/rc.d/. Install the mysql.server script as /usr/local/etc/rc.d/mysql.server.sh to enable automatic

startup. The rc(8) manual page states that scripts in this directory are executed only if their base name matches the *.sh shell file name pattern. Any other files or directories present within the directory are silently ignored.

- As an alternative to the preceding setup, some operating systems also use /etc/rc.local or /etc/init.d/boot.local to start additional services on startup. To start up MySQL using this method, append a command like the one following to the appropriate startup file:

```
/bin/sh -c 'cd /usr/local/mysql; ./bin/mysqld_safe --user=mysql &'
```

- For other systems, consult your operating system documentation to see how to install startup scripts.

mysql.server reads options from the [mysql.server] and [mysqld] sections of option files. For backward compatibility, it also reads [mysql_server] sections, but to be current you should rename such sections to [mysql.server].

You can add options for mysql.server in a global /etc/my.cnf file. A typical my.cnf file might look like this:

```
[mysqld]
datadir=/usr/local/mysql/var
socket=/var/tmp/mysql.sock
port=3306
user=mysql

[mysql.server]
basedir=/usr/local/mysql
```

The mysql.server script supports the options shown in the following table. If specified, they *must* be placed in an option file, not on the command line. mysql.server supports only start and stop as command-line arguments.

Table 4.6 mysql.server Option-File Options

Option Name	Description	Type
basedir	Path to MySQL installation directory	Directory name
datadir	Path to MySQL data directory	Directory name
pid-file	File in which server should write its process ID	File name
service-startup-timeout	How long to wait for server startup	Integer

- basedir=*dir_name*

  The path to the MySQL installation directory.

- datadir=*dir_name*

  The path to the MySQL data directory.

- pid-file=*file_name*

  The path name of the file in which the server should write its process ID. The server creates the file in the data directory unless an absolute path name is given to specify a different directory.

  If this option is not given, mysql.server uses a default value of *host_name*.pid. The PID file value passed to mysqld_safe overrides any value specified in the [mysqld_safe] option file group. Because mysql.server reads the [mysqld] option file group but not the [mysqld_safe] group, you can ensure that mysqld_safe gets the same value when invoked from mysql.server

as when invoked manually by putting the same pid-file setting in both the [mysqld_safe] and [mysqld] groups.

- service-startup-timeout=*seconds*

  How long in seconds to wait for confirmation of server startup. If the server does not start within this time, mysql.server exits with an error. The default value is 900. A value of 0 means not to wait at all for startup. Negative values mean to wait forever (no timeout).

4.3.4 mysqld_multi — Manage Multiple MySQL Servers

mysqld_multi is designed to manage several mysqld processes that listen for connections on different Unix socket files and TCP/IP ports. It can start or stop servers, or report their current status.

mysqld_multi searches for groups named [mysqld*N*] in my.cnf (or in the file named by the --defaults-file option). *N* can be any positive integer. This number is referred to in the following discussion as the option group number, or *GNR*. Group numbers distinguish option groups from one another and are used as arguments to mysqld_multi to specify which servers you want to start, stop, or obtain a status report for. Options listed in these groups are the same that you would use in the [mysqld] group used for starting mysqld. (See, for example, Section 2.10.5, “Starting and Stopping MySQL Automatically”.) However, when using multiple servers, it is necessary that each one use its own value for options such as the Unix socket file and TCP/IP port number. For more information on which options must be unique per server in a multiple-server environment, see Section 5.7, “Running Multiple MySQL Instances on One Machine”.

To invoke mysqld_multi, use the following syntax:

```
shell> mysqld_multi [options] {start|stop|reload|report} [GNR[,GNR] ...]
```

start, stop, reload (stop and restart), and report indicate which operation to perform. (reload is available as of MySQL 5.6.3.) You can perform the designated operation for a single server or multiple servers, depending on the *GNR* list that follows the option name. If there is no list, mysqld_multi performs the operation for all servers in the option file.

Each *GNR* value represents an option group number or range of group numbers. The value should be the number at the end of the group name in the option file. For example, the *GNR* for a group named [mysqld17] is 17. To specify a range of numbers, separate the first and last numbers by a dash. The *GNR* value 10-13 represents groups [mysqld10] through [mysqld13]. Multiple groups or group ranges can be specified on the command line, separated by commas. There must be no whitespace characters (spaces or tabs) in the *GNR* list; anything after a whitespace character is ignored.

This command starts a single server using option group [mysqld17]:

```
shell> mysqld_multi start 17
```

This command stops several servers, using option groups [mysqld8] and [mysqld10] through [mysqld13]:

```
shell> mysqld_multi stop 8,10-13
```

For an example of how you might set up an option file, use this command:

```
shell> mysqld_multi --example
```

mysqld_multi searches for option files as follows:

- With --no-defaults, no option files are read.
- With --defaults-file=*file_name*, only the named file is read.
- Otherwise, option files in the standard list of locations are read, including any file named by the --defaults-extra-file=*file_name* option, if one is given. (If the option is given multiple times, the last value is used.)

For additional information about these and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

Option files read are searched for [mysqld_multi] and [mysqldN] option groups. The [mysqld_multi] group can be used for options to mysqld_multi itself. [mysqldN] groups can be used for options passed to specific mysqld instances.

The [mysqld] or [mysqld_safe] groups can be used for common options read by all instances of mysqld or mysqld_safe. You can specify a --defaults-file=*file_name* option to use a different configuration file for that instance, in which case the [mysqld] or [mysqld_safe] groups from that file are used for that instance.

mysqld_multi supports the following options.

- --help

  Display a help message and exit.

- --example

  Display a sample option file.

- --log=*file_name*

  Specify the name of the log file. If the file exists, log output is appended to it.

- --mysqladmin=*prog_name*

  The mysqladmin binary to be used to stop servers.

- --mysqld=*prog_name*

  The mysqld binary to be used. You can specify mysqld_safe as the value for this option. If you use mysqld_safe to start the server, you can include the mysqld or ledir options in the corresponding [mysqldN] option group. These options indicate the name of the server that mysqld_safe should start and the path name of the directory where the server is located. (See the descriptions for these options in Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”.) Example:

  ```
  [mysqld38]
  mysqld = mysqld-debug
  ledir  = /opt/local/mysql/libexec
  ```

- --no-log

  Print log information to stdout rather than to the log file. By default, output goes to the log file.

- --password=*password*

  The password of the MySQL account to use when invoking mysqladmin. The password value is not optional for this option, unlike for other MySQL programs.

- --silent

  Silent mode; disable warnings.

- --tcp-ip

  Connect to each MySQL server through the TCP/IP port instead of the Unix socket file. (If a socket file is missing, the server might still be running, but accessible only through the TCP/IP port.) By default, connections are made using the Unix socket file. This option affects stop and report operations.

- --user=*user_name*

The user name of the MySQL account to use when invoking mysqladmin.

- --verbose

  Be more verbose.

- --version

  Display version information and exit.

Some notes about mysqld_multi:

- **Most important**: Before using mysqld_multi be sure that you understand the meanings of the options that are passed to the mysqld servers and *why* you would want to have separate mysqld processes. Beware of the dangers of using multiple mysqld servers with the same data directory. Use separate data directories, unless you *know* what you are doing. Starting multiple servers with the same data directory does *not* give you extra performance in a threaded system. See Section 5.7, "Running Multiple MySQL Instances on One Machine".

  Important

  Make sure that the data directory for each server is fully accessible to the Unix account that the specific mysqld process is started as. *Do not* use the Unix *root* account for this, unless you *know* what you are doing. See Section 6.1.5, "How to Run MySQL as a Normal User".

- Make sure that the MySQL account used for stopping the mysqld servers (with the mysqladmin program) has the same user name and password for each server. Also, make sure that the account has the SHUTDOWN privilege. If the servers that you want to manage have different user names or passwords for the administrative accounts, you might want to create an account on each server that has the same user name and password. For example, you might set up a common multi_admin account by executing the following commands for each server:

  ```
  shell> mysql -u root -S /tmp/mysql.sock -p
  Enter password:
  mysql> CREATE USER 'multi_admin'@'localhost' IDENTIFIED BY 'multipass';
  mysql> GRANT SHUTDOWN ON *.* TO 'multi_admin'@'localhost';
  ```

  See Section 6.2, "Access Control and Account Management". You have to do this for each mysqld server. Change the connection parameters appropriately when connecting to each one. The host name part of the account name must permit you to connect as multi_admin from the host where you want to run mysqld_multi.

- The Unix socket file and the TCP/IP port number must be different for every mysqld. (Alternatively, if the host has multiple network addresses, you can set the bind_address system variable to cause different servers to listen to different interfaces.)

- The --pid-file option is very important if you are using mysqld_safe to start mysqld (for example, --mysqld=mysqld_safe) Every mysqld should have its own process ID file. The advantage of using mysqld_safe instead of mysqld is that mysqld_safe monitors its mysqld process and restarts it if the process terminates due to a signal sent using kill -9 or for other reasons, such as a segmentation fault.

- You might want to use the --user option for mysqld, but to do this you need to run the mysqld_multi script as the Unix superuser (root). Having the option in the option file doesn't matter; you just get a warning if you are not the superuser and the mysqld processes are started under your own Unix account.

The following example shows how you might set up an option file for use with mysqld_multi. The order in which the mysqld programs are started or stopped depends on the order in which they appear in the option file. Group numbers need not form an unbroken sequence. The first and fifth [mysqld*N*]

groups were intentionally omitted from the example to illustrate that you can have “gaps” in the option file. This gives you more flexibility.

```
# This is an example of a my.cnf file for mysqld_multi.
# Usually this file is located in home dir ~/.my.cnf or /etc/my.cnf

[mysqld_multi]
mysqld     = /usr/local/mysql/bin/mysqld_safe
mysqladmin = /usr/local/mysql/bin/mysqladmin
user       = multi_admin
password   = my_password

[mysqld2]
socket     = /tmp/mysql.sock2
port       = 3307
pid-file   = /usr/local/mysql/data2/hostname.pid2
datadir    = /usr/local/mysql/data2
language   = /usr/local/mysql/share/mysql/english
user       = unix_user1

[mysqld3]
mysqld     = /path/to/mysqld_safe
ledir      = /path/to/mysqld-binary/
mysqladmin = /path/to/mysqladmin
socket     = /tmp/mysql.sock3
port       = 3308
pid-file   = /usr/local/mysql/data3/hostname.pid3
datadir    = /usr/local/mysql/data3
language   = /usr/local/mysql/share/mysql/swedish
user       = unix_user2

[mysqld4]
socket     = /tmp/mysql.sock4
port       = 3309
pid-file   = /usr/local/mysql/data4/hostname.pid4
datadir    = /usr/local/mysql/data4
language   = /usr/local/mysql/share/mysql/estonia
user       = unix_user3

[mysqld6]
socket     = /tmp/mysql.sock6
port       = 3311
pid-file   = /usr/local/mysql/data6/hostname.pid6
datadir    = /usr/local/mysql/data6
language   = /usr/local/mysql/share/mysql/japanese
user       = unix_user4
```

See Section 4.2.2.2, “Using Option Files”.

4.4 Installation-Related Programs

The programs in this section are used when installing or upgrading MySQL.

4.4.1 `comp_err` — Compile MySQL Error Message File

`comp_err` creates the `errmsg.sys` file that is used by `mysqld` to determine the error messages to display for different error codes. `comp_err` normally is run automatically when MySQL is built. It compiles the `errmsg.sys` file from the text-format error information file located at `sql/share/errmsg-utf8.txt` in MySQL source distributions.

`comp_err` also generates the `mysqld_error.h`, `mysqld_ername.h`, and `sql_state.h` header files.

For more information about how error messages are defined, see the MySQL Internals Manual.

Invoke `comp_err` like this:

```
shell> comp_err [options]
```

comp_err supports the following options.

- --help, -?

  Display a help message and exit.

- --charset=*dir_name*, -C *dir_name*

  The character set directory. The default is ../sql/share/charsets.

- --debug=*debug_options*, -# *debug_options*

  Write a debugging log. A typical *debug_options* string is d:t:O,*file_name*. The default is d:t:O,/tmp/comp_err.trace.

- --debug-info, -T

  Print some debugging information when the program exits.

- --header-file=*file_name*, -H *file_name*

  The name of the error header file. The default is mysqld_error.h.

- --in-file=*file_name*, -F *file_name*

  The name of the input file that defines error messages. The default is ../sql/share/errmsg-utf8.txt.

- --name-file=*file_name*, -N *file_name*

  The name of the error name file. The default is mysqld_ername.h.

- --out-dir=*dir_name*, -D *dir_name*

  The name of the output base directory. The default is ../sql/share/.

- --out-file=*file_name*, -O *file_name*

  The name of the output file. The default is errmsg.sys.

- --state-file=*file_name*, -S *file_name*

  The name for the SQLSTATE header file. The default is sql_state.h.

- --version, -V

  Display version information and exit.

4.4.2 mysqlbug — Generate Bug Report

This program is obsolete. It is deprecated as of MySQL 5.6.19 and is removed in MySQL 5.7.

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

4.4.3 mysql_install_db — Initialize MySQL Data Directory

mysql_install_db initializes the MySQL data directory and creates the system tables that it contains, if they do not exist. It also initializes the system tablespace and related data structures needed to manage InnoDB tables. As of MySQL 5.6.8, mysql_install_db is a Perl script and can be used on any system with Perl installed. Before 5.6.8, it is a shell script and is available only on Unix platforms.

As of MySQL 5.6.8, on Unix platforms, mysql_install_db creates a default option file named my.cnf in the base installation directory. This file is created from a template included in the distribution package named my-default.cnf. You can find the template in or under the base installation directory. When started using mysqld_safe, the server uses my.cnf file by default. If my.cnf already exists, mysql_install_db assumes it to be in use and writes a new file named my-new.cnf instead.

With one exception, the settings in the default option file are commented and have no effect. The exception is that the file sets the sql_mode system variable to NO_ENGINE_SUBSTITUTION,STRICT_TRANS_TABLES. This setting produces a server configuration that results in errors rather than warnings for bad data in operations that modify transactional tables. See Section 5.1.10, “Server SQL Modes”.

To invoke mysql_install_db, use the following syntax:

```
shell> mysql_install_db [options]
```

Because the MySQL server, mysqld, must access the data directory when it runs later, you should either run mysql_install_db from the same system account that is used for running mysqld, or run it as root and specify the --user option to indicate the user name that mysqld runs as. It might be necessary to specify other options such as --basedir or --datadir if mysql_install_db does not use the correct locations for the installation directory or data directory. For example:

```
shell> scripts/mysql_install_db --user=mysql \
         --basedir=/opt/mysql/mysql \
         --datadir=/opt/mysql/mysql/data
```

Note

After mysql_install_db sets up the InnoDB system tablespace, changes to some tablespace characteristics require setting up a whole new instance. This includes the file name of the first file in the system tablespace and the number of undo logs. If you do not want to use the default values, make sure that the settings for the innodb_data_file_path and innodb_log_file_size configuration parameters are in place in the MySQL configuration file before running mysql_install_db. Also make sure to specify as necessary other parameters that affect the creation and location of InnoDB files, such as innodb_data_home_dir and innodb_log_group_home_dir.

If those options are in your configuration file but that file is not in a location that MySQL reads by default, specify the file location using the --defaults-extra-file option when you run mysql_install_db.

Note

If you have set a custom TMPDIR environment variable when performing the installation, and the specified directory is not accessible, mysql_install_db may fail. If so, unset TMPDIR or set TMPDIR to point to the system temporary directory (usually /tmp).

mysql_install_db supports the following options, which can be specified on the command line or in the [mysql_install_db] group of an option file. (Options that are common to mysqld can also be specified in the [mysqld] group.) Other options are passed to mysqld. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.7 mysql_install_db Options

Option Name	Description	Introduced	Deprecated
--basedir	Path to base directory		

Option Name	Description	Introduced	Deprecated
--builddir	Path to build directory (for out-of-source builds)		
--cross-bootstrap	For internal use		
--datadir	Path to data directory		
--defaults-extra-file	Read named option file in addition to usual option files		
--defaults-file	Read only named option file		
--force	Run even if DNS does not work		
--help	Display help message and exit		
--keep-my-cnf	Keep existing my.cnf file, do not create new one	5.6.20	5.6.20
--ldata	Synonym for --datadir		
--no-defaults	Read no option files		
--random-passwords	Generate administrative account random password		
--rpm	For internal use		
--skip-name-resolve	Use IP addresses rather than host names in grant tables		
--srcdir	For internal use		
--user	Operating system user under which to execute mysqld		
--verbose	Verbose mode		
--windows	For internal use		

- `--help`

  Display a help message and exit.

- `--basedir=`*`dir_name`*

  The path to the MySQL installation directory.

- `--builddir=`*`dir_name`*

  For use with `--srcdir` and out-of-source builds. Set this to the location of the directory where the built files reside.

- `--cross-bootstrap`

  For internal use. This option is used for building system tables on one host intended for another.

- `--datadir=`*`dir_name`*

  The path to the MySQL data directory. Beginning with MySQL 5.6.8, `mysql_install_db` is more strict about the option value. Only the last component of the path name is created if it does not exist; the parent directory must already exist or an error occurs.

- `--defaults-extra-file=`*`file_name`*

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--force`

  Cause `mysql_install_db` to run even if DNS does not work. Grant table entries normally created using host names use IP addresses instead.

- `--keep-my-cnf`

  Tell `mysql_install_db` to preserve any existing `my.cnf` file and not create a new default `my.cnf` file. This option was added in MySQL 5.6.20.

- `--ldata=dir_name`

  A synonym for `--datadir`.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--random-passwords`

  On Unix platforms, this option provides for more secure MySQL installation. Invoking `mysql_install_db` with `--random-passwords` causes it to perform the following actions in addition to its normal operation:

  - The installation process creates a random password, assigns it to the initial MySQL `root` accounts, and sets the “password expired” flag for those accounts.
  - The initial random `root` password is written to the `.mysql_secret` file in the directory named by the `HOME` environment variable. Depending on operating system, using a command such as `sudo` may cause the value of `HOME` to refer to the home directory of the `root` system user.

    If `.mysql_secret` already exists, the new password information is appended to it. Each password entry includes a timestamp so that in the event of multiple install operations it is possible to determine the password associated with each one.

    `.mysql_secret` is created with mode 600 to be accessible only to the operating system user for whom it is created.
  - No anonymous-user MySQL accounts are created.

  As a result of these actions, it is necessary after installation to start the server, connect as `root` using the password written to the `.mysql_secret` file, and specify a new `root` password. Until this is done, `root` cannot do anything else. This must be done for each `root` account you intend to use. To change the password, you can use the `SET PASSWORD` statement (for example, with the `mysql` client). You can also use `mysqladmin` or `mysql_secure_installation`.

  New RPM install operations (not upgrades) invoke `mysql_install_db` with the `--random-passwords` option. (Install operations using RPMs for Unbreakable Linux Network are unaffected because they do not use `mysql_install_db`.)

As of MySQL 5.6.9, new Solaris PKG install operations (not upgrades) invoke mysql_install_db with the --random-passwords option.

For install operations using a binary .tar.gz distribution or a source distribution, you can invoke mysql_install_db with the --random-passwords option manually to make your MySQL installation more secure. This is recommended, particularly for sites with sensitive data.

- --rpm

  For internal use. This option is used during the MySQL installation process for install operations performed using RPM packages.

- --skip-name-resolve

  Use IP addresses rather than host names when creating grant table entries. This option can be useful if your DNS does not work.

- --srcdir=*dir_name*

  For internal use. This option specifies the directory under which mysql_install_db looks for support files such as the error message file and the file for populating the help tables.

- --user=*user_name*

  The system (login) user name to use for running mysqld. Files and directories created by mysqld are owned by this user. You must be the system root user to use this option. By default, mysqld runs using your current login name and files and directories that it creates are owned by you.

- --verbose

  Verbose mode. Print more information about what the program does.

- --windows

  For internal use. This option is used for creating Windows distributions.

4.4.4 mysql_plugin — Configure MySQL Server Plugins

The mysql_plugin utility enables MySQL administrators to manage which plugins a MySQL server loads. It provides an alternative to manually specifying the --plugin-load option at server startup or using the INSTALL PLUGIN and UNINSTALL PLUGIN statements at runtime. mysql_plugin is available as of MySQL 5.6.3.

Depending on whether mysql_plugin is invoked to enable or disable plugins, it inserts or deletes rows in the mysql.plugin table that serves as a plugin registry. (To perform this operation, mysql_plugin invokes the MySQL server in bootstrap mode. This means that the server must not already be running.) For normal server startups, the server loads and enables plugins listed in mysql.plugin automatically. For additional control over plugin activation, use --*plugin_name* options named for specific plugins, as described in Section 5.5.1, “Installing and Uninstalling Plugins”.

Each invocation of mysql_plugin reads a configuration file to determine how to configure the plugins contained in a single plugin library file. To invoke mysql_plugin, use this syntax:

```
mysql_plugin [options] plugin {ENABLE|DISABLE}
```

plugin is the name of the plugin to configure. ENABLE or DISABLE (not case-sensitive) specify whether to enable or disable components of the plugin library named in the configuration file. The order of the *plugin* and ENABLE or DISABLE arguments does not matter.

For example, to configure components of a plugin library file named myplugins.so on Linux or myplugins.dll on Windows, specify a *plugin* value of myplugins. Suppose that this plugin

library contains three plugins, plugin1, plugin2, and plugin3, all of which should be configured under mysql_plugin control. By convention, configuration files have a suffix of .ini and the same base name as the plugin library, so the default configuration file name for this plugin library is myplugins.ini. The configuration file contents look like this:

```
myplugins
plugin1
plugin2
plugin3
```

The first line in the myplugins.ini file is the name of the library file, without any extension such as .so or .dll. The remaining lines are the names of the components to be enabled or disabled. Each value in the file should be on a separate line. Lines on which the first character is '#' are taken as comments and ignored.

To enable the plugins listed in the configuration file, invoke mysql_plugin this way:

```
shell> mysql_plugin myplugins ENABLE
```

To disable the plugins, use DISABLE rather than ENABLE.

An error occurs if mysql_plugin cannot find the configuration file or plugin library file, or if mysql_plugin cannot start the MySQL server.

mysql_plugin supports the following options, which can be specified on the command line or in the [mysqld] group of any option file. For options specified in a [mysqld] group, mysql_plugin recognizes the --basedir, --datadir, and --plugin-dir options and ignores others. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.8 mysql_plugin Options

Option Name	Description
--basedir	The server base directory
--datadir	The server data directory
--help	Display help message and exit
--my-print-defaults	Path to my_print_defaults
--mysqld	Path to server
--no-defaults	Do not read configuration file
--plugin-dir	Directory where plugins are installed
--plugin-ini	The plugin configuration file
--print-defaults	Show configuration file defaults
--verbose	Verbose mode
--version	Display version information and exit

- --help, -?

  Display a help message and exit.

- --basedir=*dir_name*, -b *dir_name*

  The server base directory.

- --datadir=*dir_name*, -d *dir_name*

  The server data directory.

- --my-print-defaults=*file_name*, -b *file_name*

The path to the my_print_defaults program.

- --mysqld=*file_name*, -b *file_name*

  The path to the mysqld server.

- --no-defaults, -p

  Do not read values from the configuration file. This option enables an administrator to skip reading defaults from the configuration file.

  With mysql_plugin, this option need not be given first on the command line, unlike most other MySQL programs that support --no-defaults.

- --plugin-dir=*dir_name*, -p *dir_name*

  The server plugin directory.

- --plugin-ini=*file_name*, -i *file_name*

  The mysql_plugin configuration file. Relative path names are interpreted relative to the current directory. If this option is not given, the default is *plugin*.ini in the plugin directory, where *plugin* is the *plugin* argument on the command line.

- --print-defaults, -P

  Display the default values from the configuration file. This option causes mysql_plugin to print the defaults for --basedir, --datadir, and --plugin-dir if they are found in the configuration file. If no value for a variable is found, nothing is shown.

  With mysql_plugin, this option need not be given first on the command line, unlike most other MySQL programs that support --print-defaults.

- --verbose, -v

  Verbose mode. Print more information about what the program does. This option can be used multiple times to increase the amount of information.

- --version, -V

  Display version information and exit.

4.4.5 mysql_secure_installation — Improve MySQL Installation Security

This program enables you to improve the security of your MySQL installation in the following ways:

- You can set a password for root accounts.
- You can remove root accounts that are accessible from outside the local host.
- You can remove anonymous-user accounts.
- You can remove the test database (which by default can be accessed by all users, even anonymous users), and privileges that permit anyone to access databases with names that start with test_.

mysql_secure_installation helps you implement security recommendations similar to those described at Section 2.10.4, “Securing the Initial MySQL Accounts”.

Invoke mysql_secure_installation without arguments:

```
shell> mysql_secure_installation
```

When executed, the script prompts you to determine which actions to perform.

4.4.6 mysql_tzinfo_to_sql — Load the Time Zone Tables

The mysql_tzinfo_to_sql program loads the time zone tables in the mysql database. It is used on systems that have a *zoneinfo* database (the set of files describing time zones). Examples of such systems are Linux, FreeBSD, Solaris, and macOS. One likely location for these files is the /usr/share/zoneinfo directory (/usr/share/lib/zoneinfo on Solaris). If your system does not have a zoneinfo database, you can use the downloadable package described in Section 5.1.13, "MySQL Server Time Zone Support".

mysql_tzinfo_to_sql can be invoked several ways:

```
shell> mysql_tzinfo_to_sql tz_dir
shell> mysql_tzinfo_to_sql tz_file tz_name
shell> mysql_tzinfo_to_sql --leap tz_file
```

For the first invocation syntax, pass the zoneinfo directory path name to mysql_tzinfo_to_sql and send the output into the mysql program. For example:

```
shell> mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root mysql
```

mysql_tzinfo_to_sql reads your system's time zone files and generates SQL statements from them. mysql processes those statements to load the time zone tables.

The second syntax causes mysql_tzinfo_to_sql to load a single time zone file *tz_file* that corresponds to a time zone name *tz_name*:

```
shell> mysql_tzinfo_to_sql tz_file tz_name | mysql -u root mysql
```

If your time zone needs to account for leap seconds, invoke mysql_tzinfo_to_sql using the third syntax, which initializes the leap second information. *tz_file* is the name of your time zone file:

```
shell> mysql_tzinfo_to_sql --leap tz_file | mysql -u root mysql
```

After running mysql_tzinfo_to_sql, it is best to restart the server so that it does not continue to use any previously cached time zone data.

4.4.7 mysql_upgrade — Check and Upgrade MySQL Tables

Each time you upgrade MySQL, you should execute mysql_upgrade, which looks for incompatibilities with the upgraded MySQL server:

- It upgrades the system tables in the mysql schema so that you can take advantage of new privileges or capabilities that might have been added.
- It upgrades the Performance Schema.
- It examines user schemas.

If mysql_upgrade finds that a table has a possible incompatibility, it performs a table check and, if problems are found, attempts a table repair. If the table cannot be repaired, see Section 2.11.10, "Rebuilding or Repairing Tables or Indexes" for manual table repair strategies.

If you install MySQL from RPM packages on Linux, you must install the server and client RPMs. mysql_upgrade is included in the server RPM but requires the client RPM because the latter includes mysqlcheck. (See Section 2.5.5, "Installing MySQL on Linux Using RPM Packages from Oracle".)

Note

On Windows, you must run mysql_upgrade with administrator privileges. You can do this by running a Command Prompt as Administrator and running

the command. Failure to do so may result in the upgrade failing to execute correctly.

Caution

You should always back up your current MySQL installation *before* performing an upgrade. See Section 7.2, “Database Backup Methods”.

Some upgrade incompatibilities may require special handling *before* upgrading your MySQL installation and running `mysql_upgrade`. See Section 2.11, “Upgrading MySQL”, for instructions on determining whether any such incompatibilities apply to your installation and how to handle them.

Use `mysql_upgrade` like this:

1. Ensure that the server is running.
2. Invoke `mysql_upgrade` to upgrade the system tables in the `mysql` schema and check and repair tables in other schemas:

   ```
   shell> mysql_upgrade [options]
   ```

3. Stop the server and restart it so that any system table changes take effect.

If you have multiple MySQL server instances to upgrade, invoke `mysql_upgrade` with connection parameters appropriate for connecting to each of the desired servers. For example, with servers running on the local host on parts 3306 through 3308, upgrade each of them by connecting to the appropriate port:

```
shell> mysql_upgrade --protocol=tcp -P 3306 [other_options]
shell> mysql_upgrade --protocol=tcp -P 3307 [other_options]
shell> mysql_upgrade --protocol=tcp -P 3308 [other_options]
```

For local host connections on Unix, the `--protocol=tcp` option forces a connection using TCP/IP rather than the Unix socket file.

By default, `mysql_upgrade` runs as the MySQL `root` user. If the `root` password is expired when you run `mysql_upgrade`, you should see a message that your password is expired and that `mysql_upgrade` failed as a result. To correct this, reset the `root` password to unexpire it and run `mysql_upgrade` again:

```
shell> mysql -u root -p
Enter password: ****  <- enter root password here
mysql> SET PASSWORD = PASSWORD('root-password');
mysql> quit

shell> mysql_upgrade [options]
```

`mysql_upgrade` executes the following commands to check and repair tables and to upgrade the system tables:

```
mysqlcheck --no-defaults --databases
 --fix-db-names --fix-table-names mysql
mysqlcheck --no-defaults --check-upgrade --databases
 --auto-repair mysql
mysql < fix_priv_tables
mysqlcheck --no-defaults --all-databases
 --skip-database=mysql --fix-db-names --fix-table-names
mysqlcheck --no-defaults --check-upgrade --all-databases
 --skip-database=mysql --auto-repair
```

Notes about the preceding commands:

- `mysql_upgrade` also adds `--write-binlog` or `--skip-write-binlog` to the `mysqlcheck` commands, depending on whether the `--write-binlog` option was specified on the `mysql_upgrade` command.

- Unless invoked with the `--upgrade-system-tables` option, `mysql_upgrade` processes all tables in all user schemas as necessary. Table checking might take a long time to complete. Each table is locked and therefore unavailable to other sessions while it is being processed. Check and repair operations can be time-consuming, particularly for large tables. Table checking uses `mysqlcheck` with the `--all-databases` and `--check-upgrade` option. For details about what `--check-upgrade` entails, see the description of the `FOR UPGRADE` option of the `CHECK TABLE` statement in Section 13.7.2.2, “CHECK TABLE Statement”.

- *`fix_priv_tables`* represents a script generated internally by `mysql_upgrade` that contains SQL statements to upgrade the tables in the `mysql` database.

`mysql_upgrade` marks all checked and repaired tables with the current MySQL version number. This ensures that the next time you run `mysql_upgrade` with the same version of the server, it can be determined whether there is any need to check or repair a given table again.

`mysql_upgrade` saves the MySQL version number in a file named `mysql_upgrade_info` in the data directory. This is used to quickly check whether all tables have been checked for this release so that table-checking can be skipped. To ignore this file and perform the check regardless, use the `--force` option.

`mysql_upgrade` does not upgrade the contents of the time zone tables or help tables. For upgrade instructions, see Section 5.1.13, “MySQL Server Time Zone Support”, and Section 5.1.14, “Server-Side Help Support”.

`mysql_upgrade` supports the following options, which can be specified on the command line or in the `[mysql_upgrade]` and `[client]` groups of an option file. Unrecognized options are passed to `mysqlcheck`. For information about option files, see Section 4.2.2.2, “Using Option Files”.

Table 4.9 mysql_upgrade Options

Option Name	Description	Introduced
--basedir	Not used	
--character-sets-dir	Directory where character sets are installed	
--compress	Compress all information sent between client and server	
--datadir	Not used	
--debug	Write debugging log	
--debug-check	Print debugging information when program exits	
--debug-info	Print debugging information, memory, and CPU statistics when program exits	
--default-auth	Authentication plugin to use	
--default-character-set	Specify default character set	
--defaults-extra-file	Read named option file in addition to usual option files	
--defaults-file	Read only named option file	
--defaults-group-suffix	Option group suffix value	
--force	Force execution even if mysql_upgrade has already been executed for current MySQL version	
--help	Display help message and exit	
--host	Host on which MySQL server is located	
--login-path	Read login path options from .mylogin.cnf	
--no-defaults	Read no option files	
--password	Password to use when connecting to server	

Option Name	Description	Introduced
--pipe	Connect to server using named pipe (Windows only)	
--plugin-dir	Directory where plugins are installed	
--port	TCP/IP port number for connection	
--print-defaults	Print default options	
--protocol	Transport protocol to use	
--shared-memory-base-name	Shared-memory name for shared-memory connections (Windows only)	
--socket	Unix socket file or Windows named pipe to use	
--ssl	Enable connection encryption	
--ssl-ca	File that contains list of trusted SSL Certificate Authorities	
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files	
--ssl-cert	File that contains X.509 certificate	
--ssl-cipher	Permissible ciphers for connection encryption	
--ssl-crl	File that contains certificate revocation lists	
--ssl-crlpath	Directory that contains certificate revocation-list files	
--ssl-key	File that contains X.509 key	
--ssl-mode	Desired security state of connection to server	5.6.30
--ssl-verify-server-cert	Verify host name against server certificate Common Name identity	
--tmpdir	Directory for temporary files	
--upgrade-system-tables	Update only system tables, not user schemas	
--user	MySQL user name to use when connecting to server	
--verbose	Verbose mode	
--version-check	Check for proper server version	5.6.12
--write-binlog	Write all statements to binary log	

- `--help`

  Display a short help message and exit.

- `--basedir=`*`dir_name`*

  The path to the MySQL installation directory. This option is accepted for backward compatibility but ignored. It is removed in MySQL 5.7.

- `--character-sets-dir=`*`dir_name`*

  The directory where character sets are installed. See Section 10.15, “Character Set Configuration”.

- `--compress`

  Compress all information sent between the client and the server if possible. See Section 4.2.6, “Connection Compression Control”.

- `--datadir=`*`dir_name`*

  The path to the data directory. This option is accepted for backward compatibility but ignored. It is removed in MySQL 5.7.

- `--debug[=debug_options]`, `-# [debug_options]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,file_name`. The default is `d:t:O,/tmp/mysql_upgrade.trace`.

- `--debug-check`

  Print some debugging information when the program exits.

- `--debug-info`, `-T`

  Print debugging information and memory and CPU usage statistics when the program exits.

- `--default-auth=plugin`

  A hint about which client-side authentication plugin to use. See Section 6.2.11, “Pluggable Authentication”.

- `--default-character-set=charset_name`

  Use *`charset_name`* as the default character set. See Section 10.15, “Character Set Configuration”.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysql_upgrade` normally reads the `[client]` and `[mysql_upgrade]` groups. If the `--defaults-group-suffix=_other` option is given, `mysql_upgrade` also reads the `[client_other]` and `[mysql_upgrade_other]` groups.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--force`

  Ignore the `mysql_upgrade_info` file and force execution even if `mysql_upgrade` has already been executed for the current version of MySQL.

- `--host=host_name`, `-h host_name`

  Connect to the MySQL server on the given host.

- `--login-path=name`

  Read options from the named login path in the `.mylogin.cnf` login path file. A “login path” is an option group containing options that specify which MySQL server to connect to and which account to

authenticate as. To create or modify a login path file, use the mysql_config_editor utility. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- --no-defaults

  Do not read any option files. If program startup fails due to reading unknown options from an option file, --no-defaults can be used to prevent them from being read.

  The exception is that the .mylogin.cnf file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when --no-defaults is used. (.mylogin.cnf is created by the mysql_config_editor utility. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”.)

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- --password[=*password*], -p[*password*]

  The password of the MySQL account used for connecting to the server. The password value is optional. If not given, mysql_upgrade prompts for one. If given, there must be *no space* between --password= or -p and the password following it. If no password option is specified, the default is to send no password.

  Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

  To explicitly specify that there is no password and that mysql_upgrade should not prompt for one, use the --skip-password option.

- --pipe, -W

  On Windows, connect to the server using a named pipe. This option applies only if the server was started with the named_pipe system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the named_pipe_full_access_group system variable.

- --plugin-dir=*dir_name*

  The directory in which to look for plugins. Specify this option if the --default-auth option is used to specify an authentication plugin but mysql_upgrade does not find it. See Section 6.2.11, “Pluggable Authentication”.

- --port=*port_num*, -P *port_num*

  For TCP/IP connections, the port number to use.

- --print-defaults

  Print the program name and all options that it gets from option files.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- --protocol={TCP|SOCKET|PIPE|MEMORY}

  The transport protocol to use for connecting to the server. It is useful when the other connection parameters normally result in use of a protocol other than the one you want. For details on the permissible values, see Section 4.2.5, “Connection Transport Protocols”.

- `--shared-memory-base-name=name`

  On Windows, the shared-memory name to use for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case-sensitive.

  This option applies only if the server was started with the `shared_memory` system variable enabled to support shared-memory connections.

- `--socket=path`, `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

  On Windows, this option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Command Options for Encrypted Connections.

- `--tmpdir=dir_name`, `-t dir_name`

  The path name of the directory to use for creating temporary files.

- `--upgrade-system-tables`, `-s`

  Upgrade only the system tables in the `mysql` schema, do not upgrade user schemas.

- `--user=user_name`, `-u user_name`

  The user name of the MySQL account to use for connecting to the server. The default user name is `root`.

- `--verbose`

  Verbose mode. Print more information about what the program does.

- `--version-check`, `-k`

  Check the version of the server to which `mysql_upgrade` is connecting to verify that it is the same as the version for which `mysql_upgrade` was built. If not, `mysql_upgrade` exits. This option is enabled by default; to disable the check, use `--skip-version-check`. This option was added in MySQL 5.6.12.

- `--write-binlog`

  Cause binary logging to be enabled while `mysql_upgrade` runs. In MySQL 5.6.6 and earlier, this was the default behavior. (To disable binary logging during the upgrade, it was necessary to use the inverse of this option, by starting the program with `--skip-write-binlog`.) Beginning with MySQL 5.6.7, binary logging by `mysql_upgrade` is disabled by default (Bug #14221043). Invoke the program explicitly with `--write-binlog` if you want its actions to be written to the binary log. (Also beginning with MySQL 5.6.7, the `--skip-write-binlog` option effectively does nothing.)

  When the server is running with global transaction identifiers (GTIDs) enabled (`gtid_mode=ON`), do not enable binary logging by `mysql_upgrade`.

4.5 Client Programs

This section describes client programs that connect to the MySQL server.

4.5.1 mysql — The MySQL Command-Line Client

mysql is a simple SQL shell with input line editing capabilities. It supports interactive and noninteractive use. When used interactively, query results are presented in an ASCII-table format. When used noninteractively (for example, as a filter), the result is presented in tab-separated format. The output format can be changed using command options.

If you have problems due to insufficient memory for large result sets, use the --quick option. This forces mysql to retrieve results from the server a row at a time rather than retrieving the entire result set and buffering it in memory before displaying it. This is done by returning the result set using the mysql_use_result() C API function in the client/server library rather than mysql_store_result().

Using mysql is very easy. Invoke it from the prompt of your command interpreter as follows:

```
shell> mysql db_name
```

Or:

```
shell> mysql --user=user_name --password db_name
Enter password: your_password
```

Then type an SQL statement, end it with ;, \g, or \G and press Enter.

Typing **Control+C** causes mysql to attempt to kill the current statement. If this cannot be done, or **Control+C** is typed again before the statement is killed, mysql exits.

You can execute SQL statements in a script file (batch file) like this:

```
shell> mysql db_name < script.sql > output.tab
```

On Unix, the mysql client logs statements executed interactively to a history file. See Section 4.5.1.3, “mysql Client Logging”.

4.5.1.1 mysql Client Options

mysql supports the following options, which can be specified on the command line or in the [mysql] and [client] groups of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.10 mysql Client Options

Option Name	Description	Introduced
--auto-rehash	Enable automatic rehashing	
--auto-vertical-output	Enable automatic vertical result set display	
--batch	Do not use history file	
--binary-as-hex	Display binary values in hexadecimal notation	5.6.37
--binary-mode	Disable \r\n - to - \n translation and treatment of \0 as end-of-query	
--bind-address	Use specified network interface to connect to MySQL Server	
--character-sets-dir	Directory where character sets are installed	
--column-names	Write column names in results	
--column-type-info	Display result set metadata	
--comments	Whether to retain or strip comments in statements sent to the server	
--compress	Compress all information sent between client and server	

Option Name	Description	Introduced
--connect-expired-password	Indicate to server that client can handle expired-password sandbox mode	5.6.12
--connect-timeout	Number of seconds before connection timeout	
--database	The database to use	
--debug	Write debugging log; supported only if MySQL was built with debugging support	
--debug-check	Print debugging information when program exits	
--debug-info	Print debugging information, memory, and CPU statistics when program exits	
--default-auth	Authentication plugin to use	
--default-character-set	Specify default character set	
--defaults-extra-file	Read named option file in addition to usual option files	
--defaults-file	Read only named option file	
--defaults-group-suffix	Option group suffix value	
--delimiter	Set the statement delimiter	
--enable-cleartext-plugin	Enable cleartext authentication plugin	
--execute	Execute the statement and quit	
--force	Continue even if an SQL error occurs	
--help	Display help message and exit	
--histignore	Patterns specifying which statements to ignore for logging	
--host	Host on which MySQL server is located	
--html	Produce HTML output	
--ignore-spaces	Ignore spaces after function names	
--init-command	SQL statement to execute after connecting	
--line-numbers	Write line numbers for errors	
--local-infile	Enable or disable for LOCAL capability for LOAD DATA	
--login-path	Read login path options from .mylogin.cnf	
--max-allowed-packet	Maximum packet length to send to or receive from server	
--max-join-size	The automatic limit for rows in a join when using --safe-updates	
--named-commands	Enable named mysql commands	
--net-buffer-length	Buffer size for TCP/IP and socket communication	
--no-auto-rehash	Disable automatic rehashing	
--no-beep	Do not beep when errors occur	
--no-defaults	Read no option files	
--one-database	Ignore statements except those for the default database named on the command line	
--pager	Use the given command for paging query output	
--password	Password to use when connecting to server	
--pipe	Connect to server using named pipe (Windows only)	
--plugin-dir	Directory where plugins are installed	

Option Name	Description	Introduced
--port	TCP/IP port number for connection	
--print-defaults	Print default options	
--prompt	Set the prompt to the specified format	
--protocol	Transport protocol to use	
--quick	Do not cache each query result	
--raw	Write column values without escape conversion	
--reconnect	If the connection to the server is lost, automatically try to reconnect	
--safe-updates, --i-am-a-dummy	Allow only UPDATE and DELETE statements that specify key values	
--secure-auth	Do not send passwords to server in old (pre-4.1) format	
--select-limit	The automatic limit for SELECT statements when using --safe-updates	
--server-public-key-path	Path name to file containing RSA public key	
--shared-memory-base-name	Shared-memory name for shared-memory connections (Windows only)	
--show-warnings	Show warnings after each statement if there are any	
--sigint-ignore	Ignore SIGINT signals (typically the result of typing Control+C)	
--silent	Silent mode	
--skip-auto-rehash	Disable automatic rehashing	
--skip-column-names	Do not write column names in results	
--skip-line-numbers	Skip line numbers for errors	
--skip-named-commands	Disable named mysql commands	
--skip-pager	Disable paging	
--skip-reconnect	Disable reconnecting	
--socket	Unix socket file or Windows named pipe to use	
--ssl	Enable connection encryption	
--ssl-ca	File that contains list of trusted SSL Certificate Authorities	
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files	
--ssl-cert	File that contains X.509 certificate	
--ssl-cipher	Permissible ciphers for connection encryption	
--ssl-crl	File that contains certificate revocation lists	
--ssl-crlpath	Directory that contains certificate revocation-list files	
--ssl-key	File that contains X.509 key	
--ssl-mode	Desired security state of connection to server	5.6.30
--ssl-verify-server-cert	Verify host name against server certificate Common Name identity	
--table	Display output in tabular format	
--tee	Append a copy of output to named file	
--unbuffered	Flush the buffer after each query	

Option Name	Description	Introduced
--user	MySQL user name to use when connecting to server	
--verbose	Verbose mode	
--version	Display version information and exit	
--vertical	Print query output rows vertically (one line per column value)	
--wait	If the connection cannot be established, wait and retry instead of aborting	
--xml	Produce XML output	

- `--help`, `-?`

  Display a help message and exit.

- `--auto-rehash`

  Enable automatic rehashing. This option is on by default, which enables database, table, and column name completion. Use `--disable-auto-rehash` to disable rehashing. That causes `mysql` to start faster, but you must issue the `rehash` command or its `\#` shortcut if you want to use name completion.

  To complete a name, enter the first part and press Tab. If the name is unambiguous, `mysql` completes it. Otherwise, you can press Tab again to see the possible names that begin with what you have typed so far. Completion does not occur if there is no default database.

  Note

  This feature requires a MySQL client that is compiled with the **readline** library. Typically, the **readline** library is not available on Windows.

- `--auto-vertical-output`

  Cause result sets to be displayed vertically if they are too wide for the current window, and using normal tabular format otherwise. (This applies to statements terminated by `;` or `\G`.)

- `--batch`, `-B`

  Print results using tab as the column separator, with each row on a new line. With this option, `mysql` does not use the history file.

  Batch mode results in nontabular output format and escaping of special characters. Escaping may be disabled by using raw mode; see the description for the `--raw` option.

- `--binary-as-hex`

  When this option is given, `mysql` displays binary data using hexadecimal notation (`0x`*`value`*). This occurs whether the overall output display format is tabular, vertical, HTML, or XML.

  `--binary-as-hex` when enabled affects display of all binary strings, including those returned by functions such as `CHAR()` and `UNHEX()`. The following example demonstrates this using the ASCII code for `A` (65 decimal, 41 hexadecimal):

  - `--binary-as-hex` disabled:

```
mysql> SELECT CHAR(0x41), UNHEX('41');
+------------+-------------+
| CHAR(0x41) | UNHEX('41') |
+------------+-------------+
| A          | A           |
+------------+-------------+
```

- `--binary-as-hex` enabled:

```
mysql> SELECT CHAR(0x41), UNHEX('41');
+----------------------+------------------------+
| CHAR(0x41)           | UNHEX('41')            |
+----------------------+------------------------+
| 0x41                 | 0x41                   |
+----------------------+------------------------+
```

To write a binary string expression so that it displays as a character string regardless of whether `--binary-as-hex` is enabled, use these techniques:

- The `CHAR()` function has a `USING charset` clause:

```
mysql> SELECT CHAR(0x41 USING utf8mb4);
+--------------------------+
| CHAR(0x41 USING utf8mb4) |
+--------------------------+
| A                        |
+--------------------------+
```

- More generally, use `CONVERT()` to convert an expression to a given character set:

```
mysql> SELECT CONVERT(UNHEX('41') USING utf8mb4);
+-----------------------------------+
| CONVERT(UNHEX('41') USING utf8mb4) |
+-----------------------------------+
| A                                 |
+-----------------------------------+
```

This option was added in MySQL 5.6.37.

- `--binary-mode`

  This option helps when processing `mysqlbinlog` output that may contain `BLOB` values. By default, `mysql` translates `\r\n` in statement strings to `\n` and interprets `\0` as the statement terminator. `--binary-mode` disables both features. It also disables all `mysql` commands except `charset` and `delimiter` in noninteractive mode (for input piped to `mysql` or loaded using the `source` command).

- `--bind-address=ip_address`

  On a computer having multiple network interfaces, use this option to select which interface to use for connecting to the MySQL server.

- `--character-sets-dir=dir_name`

  The directory where character sets are installed. See Section 10.15, “Character Set Configuration”.

- `--column-names`

  Write column names in results.

- `--column-type-info`

  Display result set metadata. This information corresponds to the contents of C API `MYSQL_FIELD` data structures. See C API Data Structures.

- `--comments`, `-c`

  Whether to strip or preserve comments in statements sent to the server. The default is `--skip-comments` (strip comments), enable with `--comments` (preserve comments).

- `--compress`, `-C`

  Compress all information sent between the client and the server if possible. See Section 4.2.6, “Connection Compression Control”.

- `--connect-expired-password`

  Indicate to the server that the client can handle sandbox mode if the account used to connect has an expired password. This can be useful for noninteractive invocations of `mysql` because normally the server disconnects noninteractive clients that attempt to connect using an account with an expired password. (See Section 6.2.10, “Server Handling of Expired Passwords”.) This option was added in MySQL 5.6.12.

- `--connect-timeout=value`

  The number of seconds before connection timeout. (Default value is `0`.)

- `--database=db_name`, `-D db_name`

  The database to use. This is useful primarily in an option file.

- `--debug[=debug_options]`, `-# [debug_options]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,file_name`. The default is `d:t:o,/tmp/mysql.trace`.

  This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are *not* built using this option.

- `--debug-check`

  Print some debugging information when the program exits.

  This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are *not* built using this option.

- `--debug-info`, `-T`

  Print debugging information and memory and CPU usage statistics when the program exits.

  This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are *not* built using this option.

- `--default-auth=plugin`

  A hint about which client-side authentication plugin to use. See Section 6.2.11, “Pluggable Authentication”.

- `--default-character-set=charset_name`

  Use *`charset_name`* as the default character set for the client and connection.

  This option can be useful if the operating system uses one character set and the `mysql` client by default uses another. In this case, output may be formatted incorrectly. You can usually fix such issues by using this option to force the client to use the system character set instead.

  For more information, see Section 10.4, “Connection Character Sets and Collations”, and Section 10.15, “Character Set Configuration”.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  Exception: Even with `--defaults-file`, client programs read `.mylogin.cnf`.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysql` normally reads the `[client]` and `[mysql]` groups. If the `--defaults-group-suffix=_other` option is given, `mysql` also reads the `[client_other]` and `[mysql_other]` groups.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--delimiter=str`

  Set the statement delimiter. The default is the semicolon character (`;`).

- `--disable-named-commands`

  Disable named commands. Use the `\*` form only, or use named commands only at the beginning of a line ending with a semicolon (`;`). `mysql` starts with this option *enabled* by default. However, even with this option, long-format commands still work from the first line. See Section 4.5.1.2, “mysql Client Commands”.

- `--enable-cleartext-plugin`

  Enable the `mysql_clear_password` cleartext authentication plugin. (See Section 6.4.1.5, “Client-Side Cleartext Pluggable Authentication”.)

- `--execute=statement`, `-e statement`

  Execute the statement and quit. The default output format is like that produced with `--batch`. See Section 4.2.2.1, “Using Options on the Command Line”, for some examples. With this option, `mysql` does not use the history file.

- `--force`, `-f`

  Continue even if an SQL error occurs.

- `--histignore`

  A list of one or more colon-separated patterns specifying statements to ignore for logging purposes. These patterns are added to the default pattern list (`"*IDENTIFIED*:*PASSWORD*"`). The value specified for this option affects logging of statements written to the history file. For more information, see Section 4.5.1.3, “mysql Client Logging”.

- `--host=host_name`, `-h host_name`

  Connect to the MySQL server on the given host.

- `--html`, `-H`

  Produce HTML output.

- `--ignore-spaces`, `-i`

  Ignore spaces after function names. The effect of this is described in the discussion for the `IGNORE_SPACE` SQL mode (see Section 5.1.10, “Server SQL Modes”).

- `--init-command=str`

  SQL statement to execute after connecting to the server. If auto-reconnect is enabled, the statement is executed again after reconnection occurs.

- `--line-numbers`

  Write line numbers for errors. Disable this with `--skip-line-numbers`.

- `--local-infile[={0|1}]`

  By default, `LOCAL` capability for `LOAD DATA` is determined by the default compiled into the MySQL client library. To enable or disable `LOCAL` data loading explicitly, use the `--local-infile` option. When given with no value, the option enables `LOCAL` data loading. When given as `--local-infile=0` or `--local-infile=1`, the option disables or enables `LOCAL` data loading.

  Successful use of `LOCAL` load operations within `mysql` also requires that the server permits local loading; see Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”

- `--login-path=name`

  Read options from the named login path in the `.mylogin.cnf` login path file. A “login path” is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the `mysql_config_editor` utility. See Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--max-allowed-packet=value`

  The maximum size of the buffer for client/server communication. The default is 16MB, the maximum is 1GB.

- `--max-join-size=value`

  The automatic limit for rows in a join when using `--safe-updates`. (Default value is 1,000,000.)

- `--named-commands`, `-G`

  Enable named `mysql` commands. Long-format commands are permitted, not just short-format commands. For example, `quit` and `\q` both are recognized. Use `--skip-named-commands` to disable named commands. See Section 4.5.1.2, “mysql Client Commands”.

- `--net-buffer-length=value`

  The buffer size for TCP/IP and socket communication. (Default value is 16KB.)

- `--no-auto-rehash`, `-A`

  This has the same effect as `--skip-auto-rehash`. See the description for `--auto-rehash`.

- --no-beep, -b

  Do not beep when errors occur.

- --no-defaults

  Do not read any option files. If program startup fails due to reading unknown options from an option file, --no-defaults can be used to prevent them from being read.

  The exception is that the .mylogin.cnf file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when --no-defaults is used. (.mylogin.cnf is created by the mysql_config_editor utility. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”.)

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- --one-database, -o

  Ignore statements except those that occur while the default database is the one named on the command line. This option is rudimentary and should be used with care. Statement filtering is based only on USE statements.

  Initially, mysql executes statements in the input because specifying a database *db_name* on the command line is equivalent to inserting USE *db_name* at the beginning of the input. Then, for each USE statement encountered, mysql accepts or rejects following statements depending on whether the database named is the one on the command line. The content of the statements is immaterial.

  Suppose that mysql is invoked to process this set of statements:

  ```
  DELETE FROM db2.t2;
  USE db2;
  DROP TABLE db1.t1;
  CREATE TABLE db1.t1 (i INT);
  USE db1;
  INSERT INTO t1 (i) VALUES(1);
  CREATE TABLE db2.t1 (j INT);
  ```

  If the command line is mysql --force --one-database db1, mysql handles the input as follows:

  - The DELETE statement is executed because the default database is db1, even though the statement names a table in a different database.
  - The DROP TABLE and CREATE TABLE statements are not executed because the default database is not db1, even though the statements name a table in db1.
  - The INSERT and CREATE TABLE statements are executed because the default database is db1, even though the CREATE TABLE statement names a table in a different database.

- --pager[=*command*]

  Use the given command for paging query output. If the command is omitted, the default pager is the value of your PAGER environment variable. Valid pagers are less, more, cat [> filename], and so forth. This option works only on Unix and only in interactive mode. To disable paging, use --skip-pager. Section 4.5.1.2, “mysql Client Commands”, discusses output paging further.

- --password[=*password*], -p[*password*]

  The password of the MySQL account used for connecting to the server. The password value is optional. If not given, mysql prompts for one. If given, there must be *no space* between --password= or -p and the password following it. If no password option is specified, the default is to send no password.

Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

To explicitly specify that there is no password and that mysql should not prompt for one, use the --skip-password option.

- --pipe, -W

  On Windows, connect to the server using a named pipe. This option applies only if the server was started with the named_pipe system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the named_pipe_full_access_group system variable.

- --plugin-dir=*dir_name*

  The directory in which to look for plugins. Specify this option if the --default-auth option is used to specify an authentication plugin but mysql does not find it. See Section 6.2.11, “Pluggable Authentication”.

- --port=*port_num*, -P *port_num*

  For TCP/IP connections, the port number to use.

- --print-defaults

  Print the program name and all options that it gets from option files.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- --prompt=*format_str*

  Set the prompt to the specified format. The default is mysql>. The special sequences that the prompt can contain are described in Section 4.5.1.2, “mysql Client Commands”.

- --protocol={TCP|SOCKET|PIPE|MEMORY}

  The transport protocol to use for connecting to the server. It is useful when the other connection parameters normally result in use of a protocol other than the one you want. For details on the permissible values, see Section 4.2.5, “Connection Transport Protocols”.

- --quick, -q

  Do not cache each query result, print each row as it is received. This may slow down the server if the output is suspended. With this option, mysql does not use the history file.

- --raw, -r

  For tabular output, the “boxing” around columns enables one column value to be distinguished from another. For nontabular output (such as is produced in batch mode or when the --batch or --silent option is given), special characters are escaped in the output so they can be identified easily. Newline, tab, NUL, and backslash are written as \n, \t, \0, and \\. The --raw option disables this character escaping.

  The following example demonstrates tabular versus nontabular output and the use of raw mode to disable escaping:

```
% mysql
mysql> SELECT CHAR(92);
+----------+
| CHAR(92) |
+----------+
```

```
| \        |
+----------+

% mysql -s
mysql> SELECT CHAR(92);
CHAR(92)
\\

% mysql -s -r
mysql> SELECT CHAR(92);
CHAR(92)
\
```

- `--reconnect`

  If the connection to the server is lost, automatically try to reconnect. A single reconnect attempt is made each time the connection is lost. To suppress reconnection behavior, use `--skip-reconnect`.

- `--safe-updates`, `--i-am-a-dummy`, `-U`

  If this option is enabled, `UPDATE` and `DELETE` statements that do not use a key in the `WHERE` clause or a `LIMIT` clause produce an error. In addition, restrictions are placed on `SELECT` statements that produce (or are estimated to produce) very large result sets. If you have set this option in an option file, you can use `--skip-safe-updates` on the command line to override it. For more information about this option, see Using Safe-Updates Mode (--safe-updates).

- `--secure-auth`

  Do not send passwords to the server in old (pre-4.1) format. This prevents connections except for servers that use the newer password format. This option is enabled by default; use `--skip-secure-auth` to disable it.

  Note

  Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated; expect support for them to be removed in a future MySQL release. For account upgrade instructions, see Section 6.4.1.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”.

  Note

  This option is deprecated; expect it to be removed in a future release. As of MySQL 5.7.5, it is always enabled and attempting to disable it produces an error.

- `--select-limit=value`

  The automatic limit for `SELECT` statements when using `--safe-updates`. (Default value is 1,000.)

- `--server-public-key-path=file_name`

  The path name to a file in PEM format containing a client-side copy of the public key required by the server for RSA key pair-based password exchange. This option applies to clients that authenticate with the `sha256_password` authentication plugin. This option is ignored for accounts that do not

authenticate with that plugin. It is also ignored if RSA-based password exchange is not used, as is the case when the client connects to the server using a secure connection.

This option is available only if MySQL was built using OpenSSL.

For information about the sha256_password plugin, see Section 6.4.1.4, "SHA-256 Pluggable Authentication".

- --shared-memory-base-name=*name*

  On Windows, the shared-memory name to use for connections made using shared memory to a local server. The default value is MYSQL. The shared-memory name is case-sensitive.

  This option applies only if the server was started with the shared_memory system variable enabled to support shared-memory connections.

- --show-warnings

  Cause warnings to be shown after each statement if there are any. This option applies to interactive and batch mode.

- --sigint-ignore

  Ignore SIGINT signals (typically the result of typing **Control+C**).

  Without this option, typing **Control+C** interrupts the current statement if there is one, or cancels any partial input line otherwise.

- --silent, -s

  Silent mode. Produce less output. This option can be given multiple times to produce less and less output.

  This option results in nontabular output format and escaping of special characters. Escaping may be disabled by using raw mode; see the description for the --raw option.

- --skip-column-names, -N

  Do not write column names in results.

- --skip-line-numbers, -L

  Do not write line numbers for errors. Useful when you want to compare result files that include error messages.

- --socket=*path*, -S *path*

  For connections to localhost, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

  On Windows, this option applies only if the server was started with the named_pipe system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the named_pipe_full_access_group system variable.

- --ssl*

  Options that begin with --ssl specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Command Options for Encrypted Connections.

- --table, -t

  Display output in table format. This is the default for interactive use, but can be used to produce table output in batch mode.

- --tee=*file_name*

  Append a copy of output to the given file. This option works only in interactive mode. Section 4.5.1.2, “mysql Client Commands”, discusses tee files further.

- --unbuffered, -n

  Flush the buffer after each query.

- --user=*user_name*, -u *user_name*

  The user name of the MySQL account to use for connecting to the server.

- --verbose, -v

  Verbose mode. Produce more output about what the program does. This option can be given multiple times to produce more and more output. (For example, -v -v -v produces table output format even in batch mode.)

- --version, -V

  Display version information and exit.

- --vertical, -E

  Print query output rows vertically (one line per column value). Without this option, you can specify vertical output for individual statements by terminating them with \G.

- --wait, -w

  If the connection cannot be established, wait and retry instead of aborting.

- --xml, -X

  Produce XML output.

```
<field name="column_name">NULL</field>
```

  The output when --xml is used with mysql matches that of mysqldump --xml. See Section 4.5.4, “mysqldump — A Database Backup Program”, for details.

  The XML output also uses an XML namespace, as shown here:

```
shell> mysql --xml -uroot -e "SHOW VARIABLES LIKE 'version%'"
<?xml version="1.0"?>

<resultset statement="SHOW VARIABLES LIKE 'version%'" xmlns:xsi="http://www.w3.org/2001/XMLSchema-ins
<row>
<field name="Variable_name">version</field>
<field name="Value">5.0.40-debug</field>
</row>

<row>
<field name="Variable_name">version_comment</field>
<field name="Value">Source distribution</field>
</row>

<row>
<field name="Variable_name">version_compile_machine</field>
<field name="Value">i686</field>
</row>
```

```
<row>
<field name="Variable_name">version_compile_os</field>
<field name="Value">suse-linux-gnu</field>
</row>
</resultset>
```

4.5.1.2 mysql Client Commands

mysql sends each SQL statement that you issue to the server to be executed. There is also a set of commands that mysql itself interprets. For a list of these commands, type help or \h at the mysql> prompt:

```
mysql> help

List of all MySQL commands:
Note that all text commands must be first on line and end with ';'
?         (\?) Synonym for `help'.
clear     (\c) Clear the current input statement.
connect   (\r) Reconnect to the server. Optional arguments are db and host.
delimiter (\d) Set statement delimiter.
edit      (\e) Edit command with $EDITOR.
ego       (\G) Send command to mysql server, display result vertically.
exit      (\q) Exit mysql. Same as quit.
go        (\g) Send command to mysql server.
help      (\h) Display this help.
nopager   (\n) Disable pager, print to stdout.
notee     (\t) Don't write into outfile.
pager     (\P) Set PAGER [to_pager]. Print the query results via PAGER.
print     (\p) Print current command.
prompt    (\R) Change your mysql prompt.
quit      (\q) Quit mysql.
rehash    (\#) Rebuild completion hash.
source    (\.) Execute an SQL script file. Takes a file name as an argument.
status    (\s) Get status information from the server.
system    (\!) Execute a system shell command.
tee       (\T) Set outfile [to_outfile]. Append everything into given
               outfile.
use       (\u) Use another database. Takes database name as argument.
charset   (\C) Switch to another charset. Might be needed for processing
               binlog with multi-byte charsets.
warnings  (\W) Show warnings after every statement.
nowarning (\w) Don't show warnings after every statement.

For server side help, type 'help contents'
```

If mysql is invoked with the --binary-mode option, all mysql commands are disabled except charset and delimiter in noninteractive mode (for input piped to mysql or loaded using the source command).

Each command has both a long and short form. The long form is not case-sensitive; the short form is. The long form can be followed by an optional semicolon terminator, but the short form should not.

The use of short-form commands within multiple-line /* ... */ comments is not supported.

- help [*arg*], \h [*arg*], \? [*arg*], ? [*arg*]

  Display a help message listing the available mysql commands.

  If you provide an argument to the help command, mysql uses it as a search string to access server-side help from the contents of the MySQL Reference Manual. For more information, see Section 4.5.1.4, “mysql Client Server-Side Help”.

- charset *charset_name*, \C *charset_name*

  Change the default character set and issue a SET NAMES statement. This enables the character set to remain synchronized on the client and server if mysql is run with auto-reconnect enabled (which is not recommended), because the specified character set is used for reconnects.

- clear, \c

  Clear the current input. Use this if you change your mind about executing the statement that you are entering.

- connect [*db_name* [*host_name*]], \r [*db_name* [*host_name*]]

  Reconnect to the server. The optional database name and host name arguments may be given to specify the default database or the host where the server is running. If omitted, the current values are used.

- delimiter *str*, \d *str*

  Change the string that mysql interprets as the separator between SQL statements. The default is the semicolon character (;).

  The delimiter string can be specified as an unquoted or quoted argument on the delimiter command line. Quoting can be done with either single quote ('), double quote ("), or backtick (`) characters. To include a quote within a quoted string, either quote the string with a different quote character or escape the quote with a backslash (\) character. Backslash should be avoided outside of quoted strings because it is the escape character for MySQL. For an unquoted argument, the delimiter is read up to the first space or end of line. For a quoted argument, the delimiter is read up to the matching quote on the line.

  mysql interprets instances of the delimiter string as a statement delimiter anywhere it occurs, except within quoted strings. Be careful about defining a delimiter that might occur within other words. For example, if you define the delimiter as X, you cannot use the word INDEX in statements. mysql interprets this as INDE followed by the delimiter X.

  When the delimiter recognized by mysql is set to something other than the default of ;, instances of that character are sent to the server without interpretation. However, the server itself still interprets ; as a statement delimiter and processes statements accordingly. This behavior on the server side comes into play for multiple-statement execution (see C API Multiple Statement Execution Support), and for parsing the body of stored procedures and functions, triggers, and events (see Section 20.1, “Defining Stored Programs”).

- edit, \e

  Edit the current input statement. mysql checks the values of the EDITOR and VISUAL environment variables to determine which editor to use. The default editor is vi if neither variable is set.

  The edit command works only in Unix.

- ego, \G

  Send the current statement to the server to be executed and display the result using vertical format.

- exit, \q

  Exit mysql.

- go, \g

  Send the current statement to the server to be executed.

- nopager, \n

  Disable output paging. See the description for pager.

  The nopager command works only in Unix.

- notee, \t

Disable output copying to the tee file. See the description for tee.

- nowarning, \w

  Disable display of warnings after each statement.

- pager [command], \P [command]

  Enable output paging. By using the --pager option when you invoke mysql, it is possible to browse or search query results in interactive mode with Unix programs such as less, more, or any other similar program. If you specify no value for the option, mysql checks the value of the PAGER environment variable and sets the pager to that. Pager functionality works only in interactive mode.

  Output paging can be enabled interactively with the pager command and disabled with nopager. The command takes an optional argument; if given, the paging program is set to that. With no argument, the pager is set to the pager that was set on the command line, or stdout if no pager was specified.

  Output paging works only in Unix because it uses the popen() function, which does not exist on Windows. For Windows, the tee option can be used instead to save query output, although it is not as convenient as pager for browsing output in some situations.

- print, \p

  Print the current input statement without executing it.

- prompt [str], \R [str]

  Reconfigure the mysql prompt to the given string. The special character sequences that can be used in the prompt are described later in this section.

  If you specify the prompt command with no argument, mysql resets the prompt to the default of mysql>.

- quit, \q

  Exit mysql.

- rehash, \#

  Rebuild the completion hash that enables database, table, and column name completion while you are entering statements. (See the description for the --auto-rehash option.)

- source *file_name*, \. *file_name*

  Read the named file and executes the statements contained therein. On Windows, specify path name separators as / or \\.

  Quote characters are taken as part of the file name itself. For best results, the name should not include space characters.

- status, \s

  Provide status information about the connection and the server you are using. If you are running with --safe-updates enabled, status also prints the values for the mysql variables that affect your queries.

- system *command*, \! *command*

  Execute the given command using your default command interpreter.

  The system command works only in Unix.

- `tee [file_name]`, `\T [file_name]`

  By using the `--tee` option when you invoke `mysql`, you can log statements and their output. All the data displayed on the screen is appended into a given file. This can be very useful for debugging purposes also. `mysql` flushes results to the file after each statement, just before it prints its next prompt. Tee functionality works only in interactive mode.

  You can enable this feature interactively with the `tee` command. Without a parameter, the previous file is used. The `tee` file can be disabled with the `notee` command. Executing `tee` again re-enables logging.

- `use db_name`, `\u db_name`

  Use *db_name* as the default database.

- `warnings`, `\W`

  Enable display of warnings after each statement (if there are any).

Here are a few tips about the `pager` command:

- You can use it to write to a file and the results go only to the file:

  ```
  mysql> pager cat > /tmp/log.txt
  ```

  You can also pass any options for the program that you want to use as your pager:

  ```
  mysql> pager less -n -i -S
  ```

- In the preceding example, note the `-S` option. You may find it very useful for browsing wide query results. Sometimes a very wide result set is difficult to read on the screen. The `-S` option to `less` can make the result set much more readable because you can scroll it horizontally using the left-arrow and right-arrow keys. You can also use `-S` interactively within `less` to switch the horizontal-browse mode on and off. For more information, read the `less` manual page:

  ```
  shell> man less
  ```

- The `-F` and `-X` options may be used with `less` to cause it to exit if output fits on one screen, which is convenient when no scrolling is necessary:

  ```
  mysql> pager less -n -i -S -F -X
  ```

- You can specify very complex pager commands for handling query output:

  ```
  mysql> pager cat | tee /dr1/tmp/res.txt \
           | tee /dr2/tmp/res2.txt | less -n -i -S
  ```

  In this example, the command would send query results to two files in two different directories on two different file systems mounted on `/dr1` and `/dr2`, yet still display the results onscreen using `less`.

You can also combine the `tee` and `pager` functions. Have a `tee` file enabled and `pager` set to `less`, and you are able to browse the results using the `less` program and still have everything appended into a file the same time. The difference between the Unix `tee` used with the `pager` command and the `mysql` built-in `tee` command is that the built-in `tee` works even if you do not have the Unix `tee` available. The built-in `tee` also logs everything that is printed on the screen, whereas the Unix `tee` used with `pager` does not log quite that much. Additionally, `tee` file logging can be turned on and off interactively from within `mysql`. This is useful when you want to log some queries to a file, but not others.

The `prompt` command reconfigures the default `mysql>` prompt. The string for defining the prompt can contain the following special sequences.

Option	Description
`\c`	A counter that increments for each statement you issue

Option	Description
\D	The full current date
\d	The default database
\h	The server host
\l	The current delimiter
\m	Minutes of the current time
\n	A newline character
\O	The current month in three-letter format (Jan, Feb, …)
\o	The current month in numeric format
\P	am/pm
\p	The current TCP/IP port or socket file
\R	The current time, in 24-hour military time (0–23)
\r	The current time, standard 12-hour time (1–12)
\S	Semicolon
\s	Seconds of the current time
\t	A tab character
\U	Your full *user_name@host_name* account name
\u	Your user name
\v	The server version
\w	The current day of the week in three-letter format (Mon, Tue, …)
\Y	The current year, four digits
\y	The current year, two digits
_	A space
\	A space (a space follows the backslash)
\'	Single quote
\"	Double quote
\\	A literal \ backslash character
x	*x*, for any "*x*" not listed above

You can set the prompt in several ways:

- *Use an environment variable.* You can set the MYSQL_PS1 environment variable to a prompt string. For example:

```
shell> export MYSQL_PS1="(\u@\h) [\d]> "
```

- *Use a command-line option.* You can set the --prompt option on the command line to mysql. For example:

```
shell> mysql --prompt="(\u@\h) [\d]> "
(user@host) [database]>
```

- *Use an option file.* You can set the prompt option in the [mysql] group of any MySQL option file, such as /etc/my.cnf or the .my.cnf file in your home directory. For example:

```
[mysql]
prompt=(\\u@\\h) [\\d]>\\_
```

  In this example, note that the backslashes are doubled. If you set the prompt using the prompt option in an option file, it is advisable to double the backslashes when using the special prompt

options. There is some overlap in the set of permissible prompt options and the set of special escape sequences that are recognized in option files. (The rules for escape sequences in option files are listed in Section 4.2.2.2, “Using Option Files”.) The overlap may cause you problems if you use single backslashes. For example, `\s` is interpreted as a space rather than as the current seconds value. The following example shows how to define a prompt within an option file to include the current time in *`hh:mm:ss>`* format:

```
[mysql]
prompt="\\r:\\m:\\s> "
```

- *Set the prompt interactively.* You can change your prompt interactively by using the `prompt` (or `\R`) command. For example:

```
mysql> prompt (\u@\h) [\d]>\_
PROMPT set to '(\u@\h) [\d]>\_'
(user@host) [database]>
(user@host) [database]> prompt
Returning to default PROMPT of mysql>
mysql>
```

4.5.1.3 mysql Client Logging

On Unix, the `mysql` client logs statements executed interactively to a history file. By default, this file is named `.mysql_history` in your home directory. To specify a different file, set the value of the `MYSQL_HISTFILE` environment variable.

- How Logging Occurs
- Controlling the History File

How Logging Occurs

Statement logging occurs as follows:

- Statements are logged only when executed interactively. Statements are noninteractive, for example, when read from a file or a pipe. It is also possible to suppress statement logging by using the `--batch` or `--execute` option.
- Statements are ignored and not logged if they match any pattern in the “ignore” list. This list is described later.
- `mysql` logs each nonignored, nonempty statement line individually.
- If a nonignored statement spans multiple lines (not including the terminating delimiter), `mysql` concatenates the lines to form the complete statement, maps newlines to spaces, and logs the result, plus a delimiter.

Consequently, an input statement that spans multiple lines can be logged twice. Consider this input:

```
mysql> SELECT
    -> 'Today is'
    -> ,
    -> CURDATE()
    -> ;
```

In this case, `mysql` logs the “SELECT”, “'Today is'”, “,”, “CURDATE()”, and “;” lines as it reads them. It also logs the complete statement, after mapping `SELECT\n'Today is'\n,\nCURDATE()` to `SELECT 'Today is' , CURDATE()`, plus a delimiter. Thus, these lines appear in logged output:

```
SELECT
'Today is'
,
CURDATE()
;
SELECT 'Today is' , CURDATE();
```

mysql ignores for logging purposes statements that match any pattern in the “ignore” list. By default, the pattern list is "*IDENTIFIED*:*PASSWORD*", to ignore statements that refer to passwords. Pattern matching is not case-sensitive. Within patterns, two characters are special:

- ? matches any single character.
- * matches any sequence of zero or more characters.

To specify additional patterns, use the --histignore option or set the MYSQL_HISTIGNORE environment variable. (If both are specified, the option value takes precedence.) The value should be a list of one or more colon-separated patterns, which are appended to the default pattern list.

Patterns specified on the command line might need to be quoted or escaped to prevent your command interpreter from treating them specially. For example, to suppress logging for UPDATE and DELETE statements in addition to statements that refer to passwords, invoke mysql like this:

```
shell> mysql --histignore="*UPDATE*:*DELETE*"
```

Controlling the History File

The .mysql_history file should be protected with a restrictive access mode because sensitive information might be written to it, such as the text of SQL statements that contain passwords. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

If you do not want to maintain a history file, first remove .mysql_history if it exists. Then use either of the following techniques to prevent it from being created again:

- Set the MYSQL_HISTFILE environment variable to /dev/null. To cause this setting to take effect each time you log in, put it in one of your shell's startup files.
- Create .mysql_history as a symbolic link to /dev/null; this need be done only once:

  ```
  shell> ln -s /dev/null $HOME/.mysql_history
  ```

4.5.1.4 mysql Client Server-Side Help

```
mysql> help search_string
```

If you provide an argument to the help command, mysql uses it as a search string to access server-side help from the contents of the MySQL Reference Manual. The proper operation of this command requires that the help tables in the mysql database be initialized with help topic information (see Section 5.1.14, “Server-Side Help Support”).

If there is no match for the search string, the search fails:

```
mysql> help me

Nothing found
Please try to run 'help contents' for a list of all accessible topics
```

Use help contents to see a list of the help categories:

```
mysql> help contents
You asked for help about help category: "Contents"
For more information, type 'help <item>', where <item> is one of the
following categories:
   Account Management
   Administration
   Data Definition
   Data Manipulation
   Data Types
   Functions
   Functions and Modifiers for Use with GROUP BY
   Geographic Features
   Language Structure
   Plugins
   Storage Engines
   Stored Routines
```

```
   Table Maintenance
   Transactions
   Triggers
```

If the search string matches multiple items, mysql shows a list of matching topics:

```
mysql> help logs
Many help items for your request exist.
To make a more specific request, please type 'help <item>',
where <item> is one of the following topics:
   SHOW
   SHOW BINARY LOGS
   SHOW ENGINE
   SHOW LOGS
```

Use a topic as the search string to see the help entry for that topic:

```
mysql> help show binary logs
Name: 'SHOW BINARY LOGS'
Description:
Syntax:
SHOW BINARY LOGS
SHOW MASTER LOGS

Lists the binary log files on the server. This statement is used as
part of the procedure described in [purge-binary-logs], that shows how
to determine which logs can be purged.

mysql> SHOW BINARY LOGS;
+---------------+-----------+
| Log_name      | File_size |
+---------------+-----------+
| binlog.000015 |    724935 |
| binlog.000016 |    733481 |
+---------------+-----------+
```

The search string can contain the wildcard characters % and _. These have the same meaning as for pattern-matching operations performed with the LIKE operator. For example, HELP rep% returns a list of topics that begin with rep:

```
mysql> HELP rep%
Many help items for your request exist.
To make a more specific request, please type 'help <item>',
where <item> is one of the following
topics:
   REPAIR TABLE
   REPEAT FUNCTION
   REPEAT LOOP
   REPLACE
   REPLACE FUNCTION
```

4.5.1.5 Executing SQL Statements from a Text File

The mysql client typically is used interactively, like this:

```
shell> mysql db_name
```

However, it is also possible to put your SQL statements in a file and then tell mysql to read its input from that file. To do so, create a text file *text_file* that contains the statements you wish to execute. Then invoke mysql as shown here:

```
shell> mysql db_name < text_file
```

If you place a USE *db_name* statement as the first statement in the file, it is unnecessary to specify the database name on the command line:

```
shell> mysql < text_file
```

If you are already running mysql, you can execute an SQL script file using the source command or \. command:

```
mysql> source file_name
mysql> \. file_name
```

Sometimes you may want your script to display progress information to the user. For this you can insert statements like this:

```
SELECT '<info_to_display>' AS ' ';
```

The statement shown outputs <info_to_display>.

You can also invoke mysql with the --verbose option, which causes each statement to be displayed before the result that it produces.

mysql ignores Unicode byte order mark (BOM) characters at the beginning of input files. Previously, it read them and sent them to the server, resulting in a syntax error. Presence of a BOM does not cause mysql to change its default character set. To do that, invoke mysql with an option such as --default-character-set=utf8.

For more information about batch mode, see Section 3.5, “Using mysql in Batch Mode”.

4.5.1.6 mysql Client Tips

This section provides information about techniques for more effective use of mysql and about mysql operational behavior.

- Input-Line Editing
- Unicode Support on Windows
- Displaying Query Results Vertically
- Using Safe-Updates Mode (--safe-updates)
- Disabling mysql Auto-Reconnect
- mysql Client Parser Versus Server Parser

Input-Line Editing

mysql supports input-line editing, which enables you to modify the current input line in place or recall previous input lines. For example, the **left-arrow** and **right-arrow** keys move horizontally within the current input line, and the **up-arrow** and **down-arrow** keys move up and down through the set of previously entered lines. **Backspace** deletes the character before the cursor and typing new characters enters them at the cursor position. To enter the line, press **Enter**.

On Windows, the editing key sequences are the same as supported for command editing in console windows. On Unix, the key sequences depend on the input library used to build mysql (for example, the libedit or readline library).

Documentation for the libedit and readline libraries is available online. To change the set of key sequences permitted by a given input library, define key bindings in the library startup file. This is a file in your home directory: .editrc for libedit and .inputrc for readline.

For example, in libedit, **Control+W** deletes everything before the current cursor position and **Control+U** deletes the entire line. In readline, **Control+W** deletes the word before the cursor and **Control+U** deletes everything before the current cursor position. If mysql was built using libedit, a user who prefers the readline behavior for these two keys can put the following lines in the .editrc file (creating the file if necessary):

```
bind "^W" ed-delete-prev-word
bind "^U" vi-kill-line-prev
```

To see the current set of key bindings, temporarily put a line that says only bind at the end of .editrc. mysql then shows the bindings when it starts.

Unicode Support on Windows

Windows provides APIs based on UTF-16LE for reading from and writing to the console; the `mysql` client for Windows is able to use these APIs. The Windows installer creates an item in the MySQL menu named `MySQL command line client - Unicode`. This item invokes the `mysql` client with properties set to communicate through the console to the MySQL server using Unicode.

To take advantage of this support manually, run `mysql` within a console that uses a compatible Unicode font and set the default character set to a Unicode character set that is supported for communication with the server:

1. Open a console window.
2. Go to the console window properties, select the font tab, and choose Lucida Console or some other compatible Unicode font. This is necessary because console windows start by default using a DOS raster font that is inadequate for Unicode.
3. Execute `mysql.exe` with the `--default-character-set=utf8` (or `utf8mb4`) option. This option is necessary because `utf16le` is one of the character sets that cannot be used as the client character set. See Impermissible Client Character Sets.

With those changes, `mysql` uses the Windows APIs to communicate with the console using UTF-16LE, and communicates with the server using UTF-8. (The menu item mentioned previously sets the font and character set as just described.)

To avoid those steps each time you run `mysql`, you can create a shortcut that invokes `mysql.exe`. The shortcut should set the console font to Lucida Console or some other compatible Unicode font, and pass the `--default-character-set=utf8` (or `utf8mb4`) option to `mysql.exe`.

Alternatively, create a shortcut that only sets the console font, and set the character set in the `[mysql]` group of your `my.ini` file:

```
[mysql]
default-character-set=utf8
```

Displaying Query Results Vertically

Some query results are much more readable when displayed vertically, instead of in the usual horizontal table format. Queries can be displayed vertically by terminating the query with \G instead of a semicolon. For example, longer text values that include newlines often are much easier to read with vertical output:

```
mysql> SELECT * FROM mails WHERE LENGTH(txt) < 300 LIMIT 300,1\G
*************************** 1. row ***************************
  msg_nro: 3068
     date: 2000-03-01 23:29:50
time_zone: +0200
mail_from: Jones
    reply: jones@example.com
  mail_to: "John Smith" <smith@example.com>
      sbj: UTF-8
      txt: >>>>> "John" == John Smith writes:

John> Hi.  I think this is a good idea.  Is anyone familiar
John> with UTF-8 or Unicode? Otherwise, I'll put this on my
John> TODO list and see what happens.

Yes, please do that.

Regards,
Jones
     file: inbox-jani-1
     hash: 190402944
1 row in set (0.09 sec)
```

Using Safe-Updates Mode (--safe-updates)

For beginners, a useful startup option is `--safe-updates` (or `--i-am-a-dummy`, which has the same effect). Safe-updates mode is helpful for cases when you might have issued an `UPDATE` or `DELETE` statement but forgotten the `WHERE` clause indicating which rows to modify. Normally, such statements update or delete all rows in the table. With `--safe-updates`, you can modify rows only by specifying the key values that identify them, or a `LIMIT` clause, or both. This helps prevent accidents. Safe-updates mode also restricts `SELECT` statements that produce (or are estimated to produce) very large result sets.

The `--safe-updates` option causes `mysql` to execute the following statement when it connects to the MySQL server, to set the session values of the `sql_safe_updates`, `sql_select_limit`, and `max_join_size` system variables:

```
SET sql_safe_updates=1, sql_select_limit=1000, max_join_size=1000000;
```

The `SET` statement affects statement processing as follows:

- Enabling `sql_safe_updates` causes `UPDATE` and `DELETE` statements to produce an error if they do not specify a key constraint in the `WHERE` clause, or provide a `LIMIT` clause, or both. For example:

  ```
  UPDATE tbl_name SET not_key_column=val WHERE key_column=val;

  UPDATE tbl_name SET not_key_column=val LIMIT 1;
  ```

- Setting `sql_select_limit` to 1,000 causes the server to limit all `SELECT` result sets to 1,000 rows unless the statement includes a `LIMIT` clause.

- Setting `max_join_size` to 1,000,000 causes multiple-table `SELECT` statements to produce an error if the server estimates it must examine more than 1,000,000 row combinations.

To specify result set limits different from 1,000 and 1,000,000, you can override the defaults by using the `--select-limit` and `--max-join-size` options when you invoke `mysql`:

```
mysql --safe-updates --select-limit=500 --max-join-size=10000
```

It is possible for `UPDATE` and `DELETE` statements to produce an error in safe-updates mode even with a key specified in the `WHERE` clause, if the optimizer decides not to use the index on the key column. For example, if key comparisons require type conversion, the index may not be used (see Section 8.3.1, “How MySQL Uses Indexes”). Suppose that an indexed string column `c1` is compared to a numeric value using `WHERE c1 = 2222`. For such comparisons, the string value is converted to a number and the operands are compared numerically (see Section 12.3, “Type Conversion in Expression Evaluation”), preventing use of the index. If safe-updates mode is enabled, an error occurs.

Disabling mysql Auto-Reconnect

If the `mysql` client loses its connection to the server while sending a statement, it immediately and automatically tries to reconnect once to the server and send the statement again. However, even if `mysql` succeeds in reconnecting, your first connection has ended and all your previous session objects and settings are lost: temporary tables, the autocommit mode, and user-defined and session variables. Also, any current transaction rolls back. This behavior may be dangerous for you, as in the following example where the server was shut down and restarted between the first and second statements without you knowing it:

```
mysql> SET @a=1;
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO t VALUES(@a);
ERROR 2006: MySQL server has gone away
No connection. Trying to reconnect...
Connection id:    1
Current database: test

Query OK, 1 row affected (1.30 sec)

mysql> SELECT * FROM t;
```

```
+------+
| a    |
+------+
| NULL |
+------+
1 row in set (0.05 sec)
```

The @a user variable has been lost with the connection, and after the reconnection it is undefined. If it is important to have mysql terminate with an error if the connection has been lost, you can start the mysql client with the --skip-reconnect option.

For more information about auto-reconnect and its effect on state information when a reconnection occurs, see C API Automatic Reconnection Control.

mysql Client Parser Versus Server Parser

The mysql client uses a parser on the client side that is not a duplicate of the complete parser used by the mysqld server on the server side. This can lead to differences in treatment of certain constructs. Examples:

- The server parser treats strings delimited by " characters as identifiers rather than as plain strings if the ANSI_QUOTES SQL mode is enabled.

  The mysql client parser does not take the ANSI_QUOTES SQL mode into account. It treats strings delimited by ", ', and ` characters the same, regardless of whether ANSI_QUOTES is enabled.

- Within /*! ... */ comments, the mysql client parser interprets short-form mysql commands. The server parser does not interpret them because these commands have no meaning on the server side.

  If it is desirable for mysql not to interpret short-form commands within comments, a partial workaround is to use the --binary-mode option, which causes all mysql commands to be disabled except \C and \d in noninteractive mode (for input piped to mysql or loaded using the source command).

4.5.2 mysqladmin — A MySQL Server Administration Program

mysqladmin is a client for performing administrative operations. You can use it to check the server's configuration and current status, to create and drop databases, and more.

Invoke mysqladmin like this:

```
shell> mysqladmin [options] command [command-arg] [command [command-arg]] ...
```

mysqladmin supports the following commands. Some of the commands take an argument following the command name.

- create *db_name*

  Create a new database named *db_name*.

- debug

  Tell the server to write debug information to the error log. The connected user must have the SUPER privilege. Format and content of this information is subject to change.

  This includes information about the Event Scheduler. See Section 20.4.5, “Event Scheduler Status”.

- drop *db_name*

  Delete the database named *db_name* and all its tables.

- extended-status

  Display the server status variables and their values.

- `flush-hosts`

  Flush all information in the host cache. See Section 5.1.11.2, “DNS Lookups and the Host Cache”.

- `flush-logs`

  Flush all logs.

- `flush-privileges`

  Reload the grant tables (same as `reload`).

- `flush-status`

  Clear status variables.

- `flush-tables`

  Flush all tables.

- `flush-threads`

  Flush the thread cache.

- `kill` *`id,id,...`*

  Kill server threads. If multiple thread ID values are given, there must be no spaces in the list.

  To kill threads belonging to other users, the connected user must have the `SUPER` privilege.

- `old-password` *`new_password`*

  This is like the `password` command but stores the password using the old (pre-4.1) password-hashing format. (See Section 6.1.2.4, “Password Hashing in MySQL”.)

- `password` *`new_password`*

  Set a new password. This changes the password to *`new_password`* for the account that you use with `mysqladmin` for connecting to the server. Thus, the next time you invoke `mysqladmin` (or any other client program) using the same account, you must specify the new password.

  Warning

  Setting a password using `mysqladmin` should be considered *insecure*. On some systems, your password becomes visible to system status programs such as `ps` that may be invoked by other users to display command lines. MySQL clients typically overwrite the command-line password argument with zeros during their initialization sequence. However, there is still a brief interval during which the value is visible. Also, on some systems this overwriting strategy is ineffective and the password remains visible to `ps`. (SystemV Unix systems and perhaps others are subject to this problem.)

  If the *`new_password`* value contains spaces or other characters that are special to your command interpreter, you need to enclose it within quotation marks. On Windows, be sure to use double quotation marks rather than single quotation marks; single quotation marks are not stripped from the password, but rather are interpreted as part of the password. For example:

```
shell> mysqladmin password "my new password"
```

  The new password can be omitted following the `password` command. In this case, `mysqladmin` prompts for the password value, which enables you to avoid specifying the password on the command line. Omitting the password value should be done only if `password` is the final command on the `mysqladmin` command line. Otherwise, the next argument is taken as the password.

Caution

Do not use this command used if the server was started with the `--skip-grant-tables` option. No password change is applied. This is true even if you precede the `password` command with `flush-privileges` on the same command line to re-enable the grant tables because the flush operation occurs after you connect. However, you can use `mysqladmin flush-privileges` to re-enable the grant table and then use a separate `mysqladmin password` command to change the password.

- `ping`

  Check whether the server is available. The return status from `mysqladmin` is 0 if the server is running, 1 if it is not. This is 0 even in case of an error such as `Access denied`, because this means that the server is running but refused the connection, which is different from the server not running.

- `processlist`

  Show a list of active server threads. This is like the output of the `SHOW PROCESSLIST` statement. If the `--verbose` option is given, the output is like that of `SHOW FULL PROCESSLIST`. (See Section 13.7.5.30, “SHOW PROCESSLIST Statement”.)

- `reload`

  Reload the grant tables.

- `refresh`

  Flush all tables and close and open log files.

- `shutdown`

  Stop the server.

- `start-slave`

  Start replication on a replica server.

- `status`

  Display a short server status message.

- `stop-slave`

  Stop replication on a replica server.

- `variables`

  Display the server system variables and their values.

- `version`

  Display version information from the server.

All commands can be shortened to any unique prefix. For example:

```
shell> mysqladmin proc stat
+----+-------+-----------+----+---------+------+-------+------------------+
| Id | User  | Host      | db | Command | Time | State | Info             |
+----+-------+-----------+----+---------+------+-------+------------------+
| 51 | jones | localhost |    | Query   | 0    |       | show processlist |
+----+-------+-----------+----+---------+------+-------+------------------+
Uptime: 1473624  Threads: 1  Questions: 39487
```

```
Slow queries: 0  Opens: 541  Flush tables: 1
Open tables: 19  Queries per second avg: 0.0268
```

The `mysqladmin status` command result displays the following values:

- `Uptime`

  The number of seconds the MySQL server has been running.

- `Threads`

  The number of active threads (clients).

- `Questions`

  The number of questions (queries) from clients since the server was started.

- `Slow queries`

  The number of queries that have taken more than `long_query_time` seconds. See Section 5.4.5, “The Slow Query Log”.

- `Opens`

  The number of tables the server has opened.

- `Flush tables`

  The number of `flush-*`, `refresh`, and `reload` commands the server has executed.

- `Open tables`

  The number of tables that currently are open.

If you execute `mysqladmin shutdown` when connecting to a local server using a Unix socket file, `mysqladmin` waits until the server's process ID file has been removed, to ensure that the server has stopped properly.

`mysqladmin` supports the following options, which can be specified on the command line or in the `[mysqladmin]` and `[client]` groups of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.11 mysqladmin Options

Option Name	Description	Introduced
--bind-address	Use specified network interface to connect to MySQL Server	
--compress	Compress all information sent between client and server	
--connect-timeout	Number of seconds before connection timeout	
--count	Number of iterations to make for repeated command execution	
--debug	Write debugging log	
--debug-check	Print debugging information when program exits	
--debug-info	Print debugging information, memory, and CPU statistics when program exits	
--default-auth	Authentication plugin to use	
--default-character-set	Specify default character set	
--defaults-extra-file	Read named option file in addition to usual option files	
--defaults-file	Read only named option file	

Option Name	Description	Introduced
--defaults-group-suffix	Option group suffix value	
--enable-cleartext-plugin	Enable cleartext authentication plugin	
--force	Continue even if an SQL error occurs	
--help	Display help message and exit	
--host	Host on which MySQL server is located	
--login-path	Read login path options from .mylogin.cnf	
--no-beep	Do not beep when errors occur	
--no-defaults	Read no option files	
--password	Password to use when connecting to server	
--pipe	Connect to server using named pipe (Windows only)	
--plugin-dir	Directory where plugins are installed	
--port	TCP/IP port number for connection	
--print-defaults	Print default options	
--protocol	Transport protocol to use	
--relative	Show the difference between the current and previous values when used with the --sleep option	
--secure-auth	Do not send passwords to server in old (pre-4.1) format	5.6.17
--shared-memory-base-name	Shared-memory name for shared-memory connections (Windows only)	
--shutdown-timeout	The maximum number of seconds to wait for server shutdown	
--silent	Silent mode	
--sleep	Execute commands repeatedly, sleeping for delay seconds in between	
--socket	Unix socket file or Windows named pipe to use	
--ssl	Enable connection encryption	
--ssl-ca	File that contains list of trusted SSL Certificate Authorities	
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files	
--ssl-cert	File that contains X.509 certificate	
--ssl-cipher	Permissible ciphers for connection encryption	
--ssl-crl	File that contains certificate revocation lists	
--ssl-crlpath	Directory that contains certificate revocation-list files	
--ssl-key	File that contains X.509 key	
--ssl-mode	Desired security state of connection to server	5.6.30
--ssl-verify-server-cert	Verify host name against server certificate Common Name identity	
--user	MySQL user name to use when connecting to server	
--verbose	Verbose mode	
--version	Display version information and exit	
--vertical	Print query output rows vertically (one line per column value)	

Option Name	Description	Introduced
--wait	If the connection cannot be established, wait and retry instead of aborting	

- `--help`, `-?`

  Display a help message and exit.

- `--bind-address=ip_address`

  On a computer having multiple network interfaces, use this option to select which interface to use for connecting to the MySQL server.

- `--character-sets-dir=dir_name`

  The directory where character sets are installed. See Section 10.15, “Character Set Configuration”.

- `--compress`, `-C`

  Compress all information sent between the client and the server if possible. See Section 4.2.6, “Connection Compression Control”.

- `--connect-timeout=value`

  The maximum number of seconds before connection timeout. The default value is 43200 (12 hours).

- `--count=N`, `-c N`

  The number of iterations to make for repeated command execution if the `--sleep` option is given.

- `--debug[=debug_options]`, `-# [debug_options]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,file_name`. The default is `d:t:o,/tmp/mysqladmin.trace`.

  This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are *not* built using this option.

- `--debug-check`

  Print some debugging information when the program exits.

  This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are *not* built using this option.

- `--debug-info`

  Print debugging information and memory and CPU usage statistics when the program exits.

  This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are *not* built using this option.

- `--default-auth=plugin`

  A hint about which client-side authentication plugin to use. See Section 6.2.11, “Pluggable Authentication”.

- `--default-character-set=charset_name`

  Use *`charset_name`* as the default character set. See Section 10.15, “Character Set Configuration”.

- `--defaults-extra-file=file_name`

Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  Exception: Even with `--defaults-file`, client programs read `.mylogin.cnf`.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *str*. For example, `mysqladmin` normally reads the `[client]` and `[mysqladmin]` groups. If the `--defaults-group-suffix=_other` option is given, `mysqladmin` also reads the `[client_other]` and `[mysqladmin_other]` groups.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--enable-cleartext-plugin`

  Enable the `mysql_clear_password` cleartext authentication plugin. (See Section 6.4.1.5, “Client-Side Cleartext Pluggable Authentication”.)

- `--force`, `-f`

  Do not ask for confirmation for the `drop db_name` command. With multiple commands, continue even if an error occurs.

- `--host=host_name`, `-h host_name`

  Connect to the MySQL server on the given host.

- `--login-path=name`

  Read options from the named login path in the `.mylogin.cnf` login path file. A “login path” is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the `mysql_config_editor` utility. See Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--no-beep`, `-b`

  Suppress the warning beep that is emitted by default for errors such as a failure to connect to the server.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

The exception is that the .mylogin.cnf file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when --no-defaults is used. (.mylogin.cnf is created by the mysql_config_editor utility. See Section 4.6.6, "mysql_config_editor — MySQL Configuration Utility".)

For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- --password[=*password*], -p[*password*]

  The password of the MySQL account used for connecting to the server. The password value is optional. If not given, mysqladmin prompts for one. If given, there must be *no space* between --password= or -p and the password following it. If no password option is specified, the default is to send no password.

  Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, "End-User Guidelines for Password Security".

  To explicitly specify that there is no password and that mysqladmin should not prompt for one, use the --skip-password option.

- --pipe, -W

  On Windows, connect to the server using a named pipe. This option applies only if the server was started with the named_pipe system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the named_pipe_full_access_group system variable.

- --plugin-dir=*dir_name*

  The directory in which to look for plugins. Specify this option if the --default-auth option is used to specify an authentication plugin but mysqladmin does not find it. See Section 6.2.11, "Pluggable Authentication".

- --port=*port_num*, -P *port_num*

  For TCP/IP connections, the port number to use.

- --print-defaults

  Print the program name and all options that it gets from option files.

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- --protocol={TCP|SOCKET|PIPE|MEMORY}

  The transport protocol to use for connecting to the server. It is useful when the other connection parameters normally result in use of a protocol other than the one you want. For details on the permissible values, see Section 4.2.5, "Connection Transport Protocols".

- --relative, -r

  Show the difference between the current and previous values when used with the --sleep option. This option works only with the extended-status command.

- `--secure-auth`

  Do not send passwords to the server in old (pre-4.1) format. This prevents connections except for servers that use the newer password format. This option is enabled by default; use `--skip-secure-auth` to disable it. This option was added in MySQL 5.6.17.

  Note

  Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated; expect support for them to be removed in a future MySQL release. For account upgrade instructions, see Section 6.4.1.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”.

  Note

  This option is deprecated; expect it to be removed in a future release. As of MySQL 5.7.5, it is always enabled and attempting to disable it produces an error.

- `--shared-memory-base-name=`*`name`*

  On Windows, the shared-memory name to use for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case-sensitive.

  This option applies only if the server was started with the `shared_memory` system variable enabled to support shared-memory connections.

- `--shutdown-timeout=`*`value`*

  The maximum number of seconds to wait for server shutdown. The default value is 3600 (1 hour).

- `--silent`, `-s`

  Exit silently if a connection to the server cannot be established.

- `--sleep=`*`delay`*, `-i` *`delay`*

  Execute commands repeatedly, sleeping for *`delay`* seconds in between. The `--count` option determines the number of iterations. If `--count` is not given, `mysqladmin` executes commands indefinitely until interrupted.

- `--socket=`*`path`*, `-S` *`path`*

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

  On Windows, this option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Command Options for Encrypted Connections.

- `--user=`*`user_name`*, `-u` *`user_name`*

  The user name of the MySQL account to use for connecting to the server.

- `--verbose`, `-v`

Verbose mode. Print more information about what the program does.

- `--version`, `-V`

  Display version information and exit.

- `--vertical`, `-E`

  Print output vertically. This is similar to `--relative`, but prints output vertically.

- `--wait[=count]`, `-w[count]`

  If the connection cannot be established, wait and retry instead of aborting. If a *`count`* value is given, it indicates the number of times to retry. The default is one time.

4.5.3 mysqlcheck — A Table Maintenance Program

The `mysqlcheck` client performs table maintenance: It checks, repairs, optimizes, or analyzes tables.

Each table is locked and therefore unavailable to other sessions while it is being processed, although for check operations, the table is locked with a `READ` lock only (see Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”, for more information about `READ` and `WRITE` locks). Table maintenance operations can be time-consuming, particularly for large tables. If you use the `--databases` or `--all-databases` option to process all tables in one or more databases, an invocation of `mysqlcheck` might take a long time. (This is also true for the MySQL upgrade procedure if it determines that table checking is needed because it processes tables the same way.)

`mysqlcheck` is similar in function to `myisamchk`, but works differently. The main operational difference is that `mysqlcheck` must be used when the `mysqld` server is running, whereas `myisamchk` should be used when it is not. The benefit of using `mysqlcheck` is that you do not have to stop the server to perform table maintenance.

`mysqlcheck` uses the SQL statements `CHECK TABLE`, `REPAIR TABLE`, `ANALYZE TABLE`, and `OPTIMIZE TABLE` in a convenient way for the user. It determines which statements to use for the operation you want to perform, and then sends the statements to the server to be executed. For details about which storage engines each statement works with, see the descriptions for those statements in Section 13.7.2, “Table Maintenance Statements”.

The `MyISAM` storage engine supports all four maintenance operations, so `mysqlcheck` can be used to perform any of them on `MyISAM` tables. Other storage engines do not necessarily support all operations. In such cases, an error message is displayed. For example, if `test.t` is a `MEMORY` table, an attempt to check it produces this result:

```
shell> mysqlcheck test t
test.t
note     : The storage engine for the table doesn't support check
```

If `mysqlcheck` is unable to repair a table, see Section 2.11.10, “Rebuilding or Repairing Tables or Indexes” for manual table repair strategies. This is the case, for example, for `InnoDB` tables, which can be checked with `CHECK TABLE`, but not repaired with `REPAIR TABLE`.

Caution

It is best to make a backup of a table before performing a table repair operation; under some circumstances the operation might cause data loss. Possible causes include but are not limited to file system errors.

There are three general ways to invoke `mysqlcheck`:

```
shell> mysqlcheck [options] db_name [tbl_name ...]
shell> mysqlcheck [options] --databases db_name ...
shell> mysqlcheck [options] --all-databases
```

If you do not name any tables following *db_name* or if you use the `--databases` or `--all-databases` option, entire databases are checked.

`mysqlcheck` has a special feature compared to other client programs. The default behavior of checking tables (`--check`) can be changed by renaming the binary. If you want to have a tool that repairs tables by default, you should just make a copy of `mysqlcheck` named `mysqlrepair`, or make a symbolic link to `mysqlcheck` named `mysqlrepair`. If you invoke `mysqlrepair`, it repairs tables.

The names shown in the following table can be used to change `mysqlcheck` default behavior.

Command	Meaning
`mysqlrepair`	The default option is `--repair`
`mysqlanalyze`	The default option is `--analyze`
`mysqloptimize`	The default option is `--optimize`

`mysqlcheck` supports the following options, which can be specified on the command line or in the `[mysqlcheck]` and `[client]` groups of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.12 mysqlcheck Options

Option Name	Description	Introduced
--all-databases	Check all tables in all databases	
--all-in-1	Execute a single statement for each database that names all the tables from that database	
--analyze	Analyze the tables	
--auto-repair	If a checked table is corrupted, automatically fix it	
--bind-address	Use specified network interface to connect to MySQL Server	
--character-sets-dir	Directory where character sets are installed	
--check	Check the tables for errors	
--check-only-changed	Check only tables that have changed since the last check	
--check-upgrade	Invoke CHECK TABLE with the FOR UPGRADE option	
--compress	Compress all information sent between client and server	
--databases	Interpret all arguments as database names	
--debug	Write debugging log	
--debug-check	Print debugging information when program exits	
--debug-info	Print debugging information, memory, and CPU statistics when program exits	
--default-auth	Authentication plugin to use	
--default-character-set	Specify default character set	
--defaults-extra-file	Read named option file in addition to usual option files	
--defaults-file	Read only named option file	
--defaults-group-suffix	Option group suffix value	
--enable-cleartext-plugin	Enable cleartext authentication plugin	5.6.28
--extended	Check and repair tables	
--fast	Check only tables that have not been closed properly	
--fix-db-names	Convert database names to 5.1 format	
--fix-table-names	Convert table names to 5.1 format	

Option Name	Description	Introduced
--force	Continue even if an SQL error occurs	
--help	Display help message and exit	
--host	Host on which MySQL server is located	
--login-path	Read login path options from .mylogin.cnf	
--medium-check	Do a check that is faster than an --extended operation	
--no-defaults	Read no option files	
--optimize	Optimize the tables	
--password	Password to use when connecting to server	
--pipe	Connect to server using named pipe (Windows only)	
--plugin-dir	Directory where plugins are installed	
--port	TCP/IP port number for connection	
--print-defaults	Print default options	
--protocol	Transport protocol to use	
--quick	The fastest method of checking	
--repair	Perform a repair that can fix almost anything except unique keys that are not unique	
--secure-auth	Do not send passwords to server in old (pre-4.1) format	5.6.17
--shared-memory-base-name	Shared-memory name for shared-memory connections (Windows only)	
--silent	Silent mode	
--skip-database	Omit this database from performed operations	5.6.11
--socket	Unix socket file or Windows named pipe to use	
--ssl	Enable connection encryption	
--ssl-ca	File that contains list of trusted SSL Certificate Authorities	
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files	
--ssl-cert	File that contains X.509 certificate	
--ssl-cipher	Permissible ciphers for connection encryption	
--ssl-crl	File that contains certificate revocation lists	
--ssl-crlpath	Directory that contains certificate revocation-list files	
--ssl-key	File that contains X.509 key	
--ssl-mode	Desired security state of connection to server	5.6.30
--ssl-verify-server-cert	Verify host name against server certificate Common Name identity	
--tables	Overrides the --databases or -B option	
--use-frm	For repair operations on MyISAM tables	
--user	MySQL user name to use when connecting to server	
--verbose	Verbose mode	
--version	Display version information and exit	
--write-binlog	Log ANALYZE, OPTIMIZE, REPAIR statements to binary log. --skip-write-binlog adds NO_WRITE_TO_BINLOG to these statements	

- `--help`, `-?`

  Display a help message and exit.

- `--all-databases`, `-A`

  Check all tables in all databases. This is the same as using the `--databases` option and naming all the databases on the command line, except that the `INFORMATION_SCHEMA` and `performance_schema` databases are not checked. They can be checked by explicitly naming them with the `--databases` option.

- `--all-in-1`, `-1`

  Instead of issuing a statement for each table, execute a single statement for each database that names all the tables from that database to be processed.

- `--analyze`, `-a`

  Analyze the tables.

- `--auto-repair`

  If a checked table is corrupted, automatically fix it. Any necessary repairs are done after all tables have been checked.

- `--bind-address=`*`ip_address`*

  On a computer having multiple network interfaces, use this option to select which interface to use for connecting to the MySQL server.

- `--character-sets-dir=`*`dir_name`*

  The directory where character sets are installed. See Section 10.15, “Character Set Configuration”.

- `--check`, `-c`

  Check the tables for errors. This is the default operation.

- `--check-only-changed`, `-C`

  Check only tables that have changed since the last check or that have not been closed properly.

- `--check-upgrade`, `-g`

  Invoke `CHECK TABLE` with the `FOR UPGRADE` option to check tables for incompatibilities with the current version of the server. This option automatically enables the `--fix-db-names` and `--fix-table-names` options.

- `--compress`

  Compress all information sent between the client and the server if possible. See Section 4.2.6, “Connection Compression Control”.

- `--databases`, `-B`

  Process all tables in the named databases. Normally, `mysqlcheck` treats the first name argument on the command line as a database name and any following names as table names. With this option, it treats all name arguments as database names.

- `--debug[=`*`debug_options`*`]`, `-# [`*`debug_options`*`]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,`*`file_name`*. The default is `d:t:o`.

This option is available only if MySQL was built using WITH_DEBUG. MySQL release binaries provided by Oracle are *not* built using this option.

- --debug-check

  Print some debugging information when the program exits.

  This option is available only if MySQL was built using WITH_DEBUG. MySQL release binaries provided by Oracle are *not* built using this option.

- --debug-info

  Print debugging information and memory and CPU usage statistics when the program exits.

  This option is available only if MySQL was built using WITH_DEBUG. MySQL release binaries provided by Oracle are *not* built using this option.

- --default-character-set=*charset_name*

  Use *charset_name* as the default character set. See Section 10.15, "Character Set Configuration".

- --defaults-extra-file=*file_name*

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- --defaults-file=*file_name*

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  Exception: Even with --defaults-file, client programs read .mylogin.cnf.

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- --defaults-group-suffix=*str*

  Read not only the usual option groups, but also groups with the usual names and a suffix of *str*. For example, mysqlcheck normally reads the [client] and [mysqlcheck] groups. If the --defaults-group-suffix=_other option is given, mysqlcheck also reads the [client_other] and [mysqlcheck_other] groups.

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- --extended, -e

  If you are using this option to check tables, it ensures that they are 100% consistent but takes a long time.

  If you are using this option to repair tables, it runs an extended repair that may not only take a long time to execute, but may produce a lot of garbage rows also!

- `--default-auth=plugin`

  A hint about which client-side authentication plugin to use. See Section 6.2.11, “Pluggable Authentication”.

- `--enable-cleartext-plugin`

  Enable the `mysql_clear_password` cleartext authentication plugin. (See Section 6.4.1.5, “Client-Side Cleartext Pluggable Authentication”.)

  This option was added in MySQL 5.6.28.

- `--fast`, `-F`

  Check only tables that have not been closed properly.

- `--fix-db-names`

  Convert database names to 5.1 format. Only database names that contain special characters are affected.

- `--fix-table-names`

  Convert table names to 5.1 format. Only table names that contain special characters are affected. This option also applies to views.

- `--force`, `-f`

  Continue even if an SQL error occurs.

- `--host=host_name`, `-h host_name`

  Connect to the MySQL server on the given host.

- `--login-path=name`

  Read options from the named login path in the `.mylogin.cnf` login path file. A “login path” is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the `mysql_config_editor` utility. See Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--medium-check`, `-m`

  Do a check that is faster than an `--extended` operation. This finds only 99.99% of all errors, which should be good enough in most cases.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

  The exception is that the `.mylogin.cnf` file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when `--no-defaults` is used. (`.mylogin.cnf` is created by the `mysql_config_editor` utility. See Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”.)

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--optimize`, `-o`

Optimize the tables.

- `--password[=password]`, `-p[password]`

  The password of the MySQL account used for connecting to the server. The password value is optional. If not given, `mysqlcheck` prompts for one. If given, there must be *no space* between `--password=` or `-p` and the password following it. If no password option is specified, the default is to send no password.

  Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

  To explicitly specify that there is no password and that `mysqlcheck` should not prompt for one, use the `--skip-password` option.

- `--pipe`, `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--plugin-dir=dir_name`

  The directory in which to look for plugins. Specify this option if the `--default-auth` option is used to specify an authentication plugin but `mysqlcheck` does not find it. See Section 6.2.11, “Pluggable Authentication”.

- `--port=port_num`, `-P port_num`

  For TCP/IP connections, the port number to use.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  The transport protocol to use for connecting to the server. It is useful when the other connection parameters normally result in use of a protocol other than the one you want. For details on the permissible values, see Section 4.2.5, “Connection Transport Protocols”.

- `--quick`, `-q`

  If you are using this option to check tables, it prevents the check from scanning the rows to check for incorrect links. This is the fastest check method.

  If you are using this option to repair tables, it tries to repair only the index tree. This is the fastest repair method.

- `--repair`, `-r`

  Perform a repair that can fix almost anything except unique keys that are not unique.

- `--secure-auth`

  Do not send passwords to the server in old (pre-4.1) format. This prevents connections except for servers that use the newer password format. This option is enabled by default; use `--skip-secure-auth` to disable it. This option was added in MySQL 5.6.17.

  Note

  Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated; expect support for them to be removed in a future MySQL release. For account upgrade instructions, see Section 6.4.1.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”.

  Note

  This option is deprecated; expect support for it to be removed in a future release. As of MySQL 5.7.5, it is always enabled and attempting to disable it produces an error.

- `--shared-memory-base-name=`*`name`*

  On Windows, the shared-memory name to use for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case-sensitive.

  This option applies only if the server was started with the `shared_memory` system variable enabled to support shared-memory connections.

- `--silent`, `-s`

  Silent mode. Print only error messages.

- `--skip-database=`*`db_name`*

  Do not include the named database (case-sensitive) in the operations performed by `mysqlcheck`. This option was added in MySQL 5.6.11.

- `--socket=`*`path`*, `-S` *`path`*

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

  On Windows, this option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Command Options for Encrypted Connections.

- `--tables`

  Override the `--databases` or `-B` option. All name arguments following the option are regarded as table names.

- `--use-frm`

  For repair operations on `MyISAM` tables, get the table structure from the `.frm` file so that the table can be repaired even if the `.MYI` header is corrupted.

- `--user=user_name`, `-u user_name`

  The user name of the MySQL account to use for connecting to the server.

- `--verbose`, `-v`

  Verbose mode. Print information about the various stages of program operation.

- `--version`, `-V`

  Display version information and exit.

- `--write-binlog`

  This option is enabled by default, so that `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements generated by `mysqlcheck` are written to the binary log. Use `--skip-write-binlog` to cause `NO_WRITE_TO_BINLOG` to be added to the statements so that they are not logged. Use the `--skip-write-binlog` when these statements should not be sent to replicas or run when using the binary logs for recovery from backup.

4.5.4 mysqldump — A Database Backup Program

The `mysqldump` client utility performs logical backups, producing a set of SQL statements that can be executed to reproduce the original database object definitions and table data. It dumps one or more MySQL databases for backup or transfer to another SQL server. The `mysqldump` command can also generate output in CSV, other delimited text, or XML format.

- Performance and Scalability Considerations
- Invocation Syntax
- Option Syntax - Alphabetical Summary
- Connection Options
- Option-File Options
- DDL Options
- Debug Options
- Help Options
- Internationalization Options
- Replication Options
- Format Options
- Filtering Options
- Performance Options
- Transactional Options
- Option Groups
- Examples
- Restrictions

`mysqldump` requires at least the `SELECT` privilege for dumped tables, `SHOW VIEW` for dumped views, `TRIGGER` for dumped triggers, `LOCK TABLES` if the `--single-transaction` option is not used, and (as of MySQL 5.6.49) `PROCESS` if the `--no-tablespaces` option is not used. Certain options might require other privileges as noted in the option descriptions.

To reload a dump file, you must have the privileges required to execute the statements that it contains, such as the appropriate CREATE privileges for objects created by those statements.

mysqldump output can include ALTER DATABASE statements that change the database collation. These may be used when dumping stored programs to preserve their character encodings. To reload a dump file containing such statements, the ALTER privilege for the affected database is required.

Note

A dump made using PowerShell on Windows with output redirection creates a file that has UTF-16 encoding:

```
shell> mysqldump [options] > dump.sql
```

However, UTF-16 is not permitted as a connection character set (see Impermissible Client Character Sets), so the dump file cannot be loaded correctly. To work around this issue, use the --result-file option, which creates the output in ASCII format:

```
shell> mysqldump [options] --result-file=dump.sql
```

Performance and Scalability Considerations

mysqldump advantages include the convenience and flexibility of viewing or even editing the output before restoring. You can clone databases for development and DBA work, or produce slight variations of an existing database for testing. It is not intended as a fast or scalable solution for backing up substantial amounts of data. With large data sizes, even if the backup step takes a reasonable time, restoring the data can be very slow because replaying the SQL statements involves disk I/O for insertion, index creation, and so on.

For large-scale backup and restore, a physical backup is more appropriate, to copy the data files in their original format that can be restored quickly:

- If your tables are primarily InnoDB tables, or if you have a mix of InnoDB and MyISAM tables, consider using the mysqlbackup command of the MySQL Enterprise Backup product. (Available as part of the Enterprise subscription.) It provides the best performance for InnoDB backups with minimal disruption; it can also back up tables from MyISAM and other storage engines; and it provides a number of convenient options to accommodate different backup scenarios. See Section 24.2, “MySQL Enterprise Backup Overview”.
- If your tables are primarily MyISAM tables, consider using the mysqlhotcopy instead, for better performance than mysqldump of backup and restore operations. See Section 4.6.10, “mysqlhotcopy — A Database Backup Program”.

mysqldump can retrieve and dump table contents row by row, or it can retrieve the entire content from a table and buffer it in memory before dumping it. Buffering in memory can be a problem if you are dumping large tables. To dump tables row by row, use the --quick option (or --opt, which enables --quick). The --opt option (and hence --quick) is enabled by default, so to enable memory buffering, use --skip-quick.

If you are using a recent version of mysqldump to generate a dump to be reloaded into a very old MySQL server, use the --skip-opt option instead of the --opt or --extended-insert option.

For additional information about mysqldump, see Section 7.4, “Using mysqldump for Backups”.

Invocation Syntax

There are in general three ways to use mysqldump—in order to dump a set of one or more tables, a set of one or more complete databases, or an entire MySQL server—as shown here:

```
shell> mysqldump [options] db_name [tbl_name ...]
shell> mysqldump [options] --databases db_name ...
shell> mysqldump [options] --all-databases
```

To dump entire databases, do not name any tables following *db_name*, or use the `--databases` or `--all-databases` option.

To see a list of the options your version of `mysqldump` supports, issue the command `mysqldump --help`.

Option Syntax - Alphabetical Summary

`mysqldump` supports the following options, which can be specified on the command line or in the `[mysqldump]` and `[client]` groups of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.13 mysqldump Options

Option Name	Description	Introduced
--add-drop-database	Add DROP DATABASE statement before each CREATE DATABASE statement	
--add-drop-table	Add DROP TABLE statement before each CREATE TABLE statement	
--add-drop-trigger	Add DROP TRIGGER statement before each CREATE TRIGGER statement	
--add-locks	Surround each table dump with LOCK TABLES and UNLOCK TABLES statements	
--all-databases	Dump all tables in all databases	
--allow-keywords	Allow creation of column names that are keywords	
--apply-slave-statements	Include STOP SLAVE prior to CHANGE MASTER statement and START SLAVE at end of output	
--bind-address	Use specified network interface to connect to MySQL Server	
--character-sets-dir	Directory where character sets are installed	
--comments	Add comments to dump file	
--compact	Produce more compact output	
--compatible	Produce output that is more compatible with other database systems or with older MySQL servers	
--complete-insert	Use complete INSERT statements that include column names	
--compress	Compress all information sent between client and server	
--create-options	Include all MySQL-specific table options in CREATE TABLE statements	
--databases	Interpret all name arguments as database names	
--debug	Write debugging log	
--debug-check	Print debugging information when program exits	
--debug-info	Print debugging information, memory, and CPU statistics when program exits	
--default-auth	Authentication plugin to use	
--default-character-set	Specify default character set	
--defaults-extra-file	Read named option file in addition to usual option files	
--defaults-file	Read only named option file	
--defaults-group-suffix	Option group suffix value	

Option Name	Description	Introduced
--delayed-insert	Write INSERT DELAYED statements rather than INSERT statements	
--delete-master-logs	On a master replication server, delete the binary logs after performing the dump operation	
--disable-keys	For each table, surround INSERT statements with statements to disable and enable keys	
--dump-date	Include dump date as "Dump completed on" comment if --comments is given	
--dump-slave	Include CHANGE MASTER statement that lists binary log coordinates of slave's master	
--enable-cleartext-plugin	Enable cleartext authentication plugin	5.6.28
--events	Dump events from dumped databases	
--extended-insert	Use multiple-row INSERT syntax	
--fields-enclosed-by	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA	
--fields-escaped-by	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA	
--fields-optionally-enclosed-by	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA	
--fields-terminated-by	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA	
--flush-logs	Flush MySQL server log files before starting dump	
--flush-privileges	Emit a FLUSH PRIVILEGES statement after dumping mysql database	
--force	Continue even if an SQL error occurs during a table dump	
--help	Display help message and exit	
--hex-blob	Dump binary columns using hexadecimal notation	
--host	Host on which MySQL server is located	
--ignore-table	Do not dump given table	
--include-master-host-port	Include MASTER_HOST/MASTER_PORT options in CHANGE MASTER statement produced with --dump-slave	
--insert-ignore	Write INSERT IGNORE rather than INSERT statements	
--lines-terminated-by	This option is used with the --tab option and has the same meaning as the corresponding clause for LOAD DATA	
--lock-all-tables	Lock all tables across all databases	
--lock-tables	Lock all tables before dumping them	
--log-error	Append warnings and errors to named file	
--login-path	Read login path options from .mylogin.cnf	
--master-data	Write the binary log file name and position to the output	

Option Name	Description	Introduced
--max-allowed-packet	Maximum packet length to send to or receive from server	
--net-buffer-length	Buffer size for TCP/IP and socket communication	
--no-autocommit	Enclose the INSERT statements for each dumped table within SET autocommit = 0 and COMMIT statements	
--no-create-db	Do not write CREATE DATABASE statements	
--no-create-info	Do not write CREATE TABLE statements that re-create each dumped table	
--no-data	Do not dump table contents	
--no-defaults	Read no option files	
--no-set-names	Same as --skip-set-charset	
--no-tablespaces	Do not write any CREATE LOGFILE GROUP or CREATE TABLESPACE statements in output	
--opt	Shorthand for --add-drop-table --add-locks --create-options --disable-keys --extended-insert --lock-tables --quick --set-charset	
--order-by-primary	Dump each table's rows sorted by its primary key, or by its first unique index	
--password	Password to use when connecting to server	
--pipe	Connect to server using named pipe (Windows only)	
--plugin-dir	Directory where plugins are installed	
--port	TCP/IP port number for connection	
--print-defaults	Print default options	
--protocol	Transport protocol to use	
--quick	Retrieve rows for a table from the server a row at a time	
--quote-names	Quote identifiers within backtick characters	
--replace	Write REPLACE statements rather than INSERT statements	
--result-file	Direct output to a given file	
--routines	Dump stored routines (procedures and functions) from dumped databases	
--secure-auth	Do not send passwords to server in old (pre-4.1) format	5.6.17
--set-charset	Add SET NAMES default_character_set to output	
--set-gtid-purged	Whether to add SET @@GLOBAL.GTID_PURGED to output	
--shared-memory-base-name	Shared-memory name for shared-memory connections (Windows only)	
--single-transaction	Issue a BEGIN SQL statement before dumping data from server	
--skip-add-drop-table	Do not add a DROP TABLE statement before each CREATE TABLE statement	
--skip-add-locks	Do not add locks	
--skip-comments	Do not add comments to dump file	
--skip-compact	Do not produce more compact output	

Option Name	Description	Introduced
--skip-disable-keys	Do not disable keys	
--skip-extended-insert	Turn off extended-insert	
--skip-opt	Turn off options set by --opt	
--skip-quick	Do not retrieve rows for a table from the server a row at a time	
--skip-quote-names	Do not quote identifiers	
--skip-set-charset	Do not write SET NAMES statement	
--skip-triggers	Do not dump triggers	
--skip-tz-utc	Turn off tz-utc	
--socket	Unix socket file or Windows named pipe to use	
--ssl	Enable connection encryption	
--ssl-ca	File that contains list of trusted SSL Certificate Authorities	
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files	
--ssl-cert	File that contains X.509 certificate	
--ssl-cipher	Permissible ciphers for connection encryption	
--ssl-crl	File that contains certificate revocation lists	
--ssl-crlpath	Directory that contains certificate revocation-list files	
--ssl-key	File that contains X.509 key	
--ssl-mode	Desired security state of connection to server	5.6.30
--ssl-verify-server-cert	Verify host name against server certificate Common Name identity	
--tab	Produce tab-separated data files	
--tables	Override --databases or -B option	
--triggers	Dump triggers for each dumped table	
--tz-utc	Add SET TIME_ZONE='+00:00' to dump file	
--user	MySQL user name to use when connecting to server	
--verbose	Verbose mode	
--version	Display version information and exit	
--where	Dump only rows selected by given WHERE condition	
--xml	Produce XML output	

Connection Options

The `mysqldump` command logs into a MySQL server to extract information. The following options specify how to connect to the MySQL server, either on the same machine or a remote system.

- `--bind-address=`*`ip_address`*

  On a computer having multiple network interfaces, use this option to select which interface to use for connecting to the MySQL server.

- `--compress`, `-C`

  Compress all information sent between the client and the server if possible. See Section 4.2.6, “Connection Compression Control”.

- `--default-auth=plugin`

  A hint about which client-side authentication plugin to use. See Section 6.2.11, “Pluggable Authentication”.

- `--enable-cleartext-plugin`

  Enable the `mysql_clear_password` cleartext authentication plugin. (See Section 6.4.1.5, “Client-Side Cleartext Pluggable Authentication”.)

  This option was added in MySQL 5.6.28.

- `--host=host_name`, `-h host_name`

  Dump data from the MySQL server on the given host. The default host is `localhost`.

- `--login-path=name`

  Read options from the named login path in the `.mylogin.cnf` login path file. A “login path” is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the `mysql_config_editor` utility. See Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--password[=password]`, `-p[password]`

  The password of the MySQL account used for connecting to the server. The password value is optional. If not given, `mysqldump` prompts for one. If given, there must be *no space* between `--password=` or `-p` and the password following it. If no password option is specified, the default is to send no password.

  Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

  To explicitly specify that there is no password and that `mysqldump` should not prompt for one, use the `--skip-password` option.

- `--pipe`, `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--plugin-dir=dir_name`

  The directory in which to look for plugins. Specify this option if the `--default-auth` option is used to specify an authentication plugin but `mysqldump` does not find it. See Section 6.2.11, “Pluggable Authentication”.

- `--port=port_num`, `-P port_num`

  For TCP/IP connections, the port number to use.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  The transport protocol to use for connecting to the server. It is useful when the other connection parameters normally result in use of a protocol other than the one you want. For details on the permissible values, see Section 4.2.5, “Connection Transport Protocols”.

- `--secure-auth`

  Do not send passwords to the server in old (pre-4.1) format. This prevents connections except for servers that use the newer password format. This option is enabled by default; use `--skip-secure-auth` to disable it. This option was added in MySQL 5.6.17.

  Note

  Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated; expect support for them to be removed in a future MySQL release. For account upgrade instructions, see Section 6.4.1.3, "Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin".

  Note

  This option is deprecated; expect it to be removed in a future release. As of MySQL 5.7.5, it is always enabled and attempting to disable it produces an error.

- `--socket=`*`path`*, `-S` *`path`*

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

  On Windows, this option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Command Options for Encrypted Connections.

- `--user=`*`user_name`*, `-u` *`user_name`*

  The user name of the MySQL account to use for connecting to the server.

Option-File Options

These options are used to control which option files to read.

- `--defaults-extra-file=`*`file_name`*

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- `--defaults-file=`*`file_name`*

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  Exception: Even with `--defaults-file`, client programs read `.mylogin.cnf`.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysqldump` normally reads the `[client]` and `[mysqldump]` groups. If the `--defaults-group-suffix=_other` option is given, `mysqldump` also reads the `[client_other]` and `[mysqldump_other]` groups.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

  The exception is that the `.mylogin.cnf` file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when `--no-defaults` is used. (`.mylogin.cnf` is created by the `mysql_config_editor` utility. See Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”.)

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

DDL Options

Usage scenarios for `mysqldump` include setting up an entire new MySQL instance (including database tables), and replacing data inside an existing instance with existing databases and tables. The following options let you specify which things to tear down and set up when restoring a dump, by encoding various DDL statements within the dump file.

- `--add-drop-database`

  Write a `DROP DATABASE` statement before each `CREATE DATABASE` statement. This option is typically used in conjunction with the `--all-databases` or `--databases` option because no `CREATE DATABASE` statements are written unless one of those options is specified.

- `--add-drop-table`

  Write a `DROP TABLE` statement before each `CREATE TABLE` statement.

- `--add-drop-trigger`

  Write a `DROP TRIGGER` statement before each `CREATE TRIGGER` statement.

- `--all-tablespaces`, `-Y`

  Adds to a table dump all SQL statements needed to create any tablespaces used by an `NDB` table. This information is not otherwise included in the output from `mysqldump`. This option is currently relevant only to NDB Cluster tables.

- `--no-create-db`, `-n`

Suppress the CREATE DATABASE statements that are otherwise included in the output if the --databases or --all-databases option is given.

- --no-create-info, -t

  Do not write CREATE TABLE statements that create each dumped table.

  Note

  This option does *not* exclude statements creating log file groups or tablespaces from mysqldump output; however, you can use the --no-tablespaces option for this purpose.

- --no-tablespaces, -y

  This option suppresses all CREATE LOGFILE GROUP and CREATE TABLESPACE statements in the output of mysqldump.

- --replace

  Write REPLACE statements rather than INSERT statements.

Debug Options

The following options print debugging information, encode debugging information in the dump file, or let the dump operation proceed regardless of potential problems.

- --allow-keywords

  Permit creation of column names that are keywords. This works by prefixing each column name with the table name.

- --comments, -i

  Write additional information in the dump file such as program version, server version, and host. This option is enabled by default. To suppress this additional information, use --skip-comments.

- --debug[=*debug_options*], -# [*debug_options*]

  Write a debugging log. A typical *debug_options* string is d:t:o,*file_name*. The default value is d:t:o,/tmp/mysqldump.trace.

  This option is available only if MySQL was built using WITH_DEBUG. MySQL release binaries provided by Oracle are *not* built using this option.

- --debug-check

  Print some debugging information when the program exits.

  This option is available only if MySQL was built using WITH_DEBUG. MySQL release binaries provided by Oracle are *not* built using this option.

- --debug-info

  Print debugging information and memory and CPU usage statistics when the program exits.

  This option is available only if MySQL was built using WITH_DEBUG. MySQL release binaries provided by Oracle are *not* built using this option.

- --dump-date

  If the --comments option is given, mysqldump produces a comment at the end of the dump of the following form:

```
-- Dump completed on DATE
```

However, the date causes dump files taken at different times to appear to be different, even if the data are otherwise identical. `--dump-date` and `--skip-dump-date` control whether the date is added to the comment. The default is `--dump-date` (include the date in the comment). `--skip-dump-date` suppresses date printing.

- `--force`, `-f`

  Continue even if an SQL error occurs during a table dump.

  One use for this option is to cause `mysqldump` to continue executing even when it encounters a view that has become invalid because the definition refers to a table that has been dropped. Without `--force`, `mysqldump` exits with an error message. With `--force`, `mysqldump` prints the error message, but it also writes an SQL comment containing the view definition to the dump output and continues executing.

- `--log-error=file_name`

  Log warnings and errors by appending them to the named file. The default is to do no logging.

- `--skip-comments`

  See the description for the `--comments` option.

- `--verbose`, `-v`

  Verbose mode. Print more information about what the program does.

Help Options

The following options display information about the `mysqldump` command itself.

- `--help`, `-?`

  Display a help message and exit.

- `--version`, `-V`

  Display version information and exit.

Internationalization Options

The following options change how the `mysqldump` command represents character data with national language settings.

- `--character-sets-dir=dir_name`

  The directory where character sets are installed. See Section 10.15, “Character Set Configuration”.

- `--default-character-set=charset_name`

  Use `charset_name` as the default character set. See Section 10.15, “Character Set Configuration”. If no character set is specified, `mysqldump` uses `utf8`.

- `--no-set-names`, `-N`

  Turns off the `--set-charset` setting, the same as specifying `--skip-set-charset`.

- `--set-charset`

  Write `SET NAMES default_character_set` to the output. This option is enabled by default. To suppress the `SET NAMES` statement, use `--skip-set-charset`.

Replication Options

The mysqldump command is frequently used to create an empty instance, or an instance including data, on a replica server in a replication configuration. The following options apply to dumping and restoring data on replication source and replica servers.

- --apply-slave-statements

  For a replica dump produced with the --dump-slave option, add a STOP SLAVE statement before the CHANGE MASTER TO statement and a START SLAVE statement at the end of the output.

- --delete-master-logs

  On a source replication server, delete the binary logs by sending a PURGE BINARY LOGS statement to the server after performing the dump operation. This option requires the RELOAD privilege as well as privileges sufficient to execute that statement. This option automatically enables --master-data.

- --dump-slave[=*value*]

  This option is similar to --master-data except that it is used to dump a replica server to produce a dump file that can be used to set up another server as a replica that has the same source as the dumped server. It causes the dump output to include a CHANGE MASTER TO statement that indicates the binary log coordinates (file name and position) of the dumped replica's source. These are the source server coordinates from which the replica should start replicating.

  --dump-slave causes the coordinates from the source to be used rather than those of the dumped server, as is done by the --master-data option. In addition, specifiying this option causes the --master-data option to be overridden, if used, and effectively ignored.

  The option value is handled the same way as for --master-data (setting no value or 1 causes a CHANGE MASTER TO statement to be written to the dump, setting 2 causes the statement to be written but encased in SQL comments) and has the same effect as --master-data in terms of enabling or disabling other options and in how locking is handled.

  This option causes mysqldump to stop the replica SQL thread before the dump and restart it again after.

  --dump-slave sends a SHOW SLAVE STATUS statement to the server to obtain information, so it requires privileges sufficient to execute that statement.

  In conjunction with --dump-slave, the --apply-slave-statements and --include-master-host-port options can also be used.

- --include-master-host-port

  For the CHANGE MASTER TO statement in a replica dump produced with the --dump-slave option, add MASTER_HOST and MASTER_PORT options for the host name and TCP/IP port number of the replica's source.

- --master-data[=*value*]

  Use this option to dump a source replication server to produce a dump file that can be used to set up another server as a replica of the source. It causes the dump output to include a CHANGE MASTER TO statement that indicates the binary log coordinates (file name and position) of the dumped server. These are the source server coordinates from which the replica should start replicating after you load the dump file into the replica.

  If the option value is 2, the CHANGE MASTER TO statement is written as an SQL comment, and thus is informative only; it has no effect when the dump file is reloaded. If the option value is 1, the

statement is not written as a comment and takes effect when the dump file is reloaded. If no option value is specified, the default value is 1.

`--master-data` sends a `SHOW MASTER STATUS` statement to the server to obtain information, so it requires privileges sufficient to execute that statement. This option also requires the `RELOAD` privilege and the binary log must be enabled.

The `--master-data` option automatically turns off `--lock-tables`. It also turns on `--lock-all-tables`, unless `--single-transaction` also is specified, in which case, a global read lock is acquired only for a short time at the beginning of the dump (see the description for `--single-transaction`). In all cases, any action on logs happens at the exact moment of the dump.

It is also possible to set up a replica by dumping an existing replica of the source, using the `--dump-slave` option, which overrides `--master-data` and causes it to be ignored if both options are used.

- `--set-gtid-purged=value`

  This option enables control over global transaction ID (GTID) information written to the dump file, by indicating whether to add a `SET @@GLOBAL.gtid_purged` statement to the output. This option may also cause a statement to be written to the output that disables binary logging while the dump file is being reloaded.

  The following table shows the permitted option values. The default value is `AUTO`.

Value	Meaning
`OFF`	Add no `SET` statement to the output.
`ON`	Add a `SET` statement to the output. An error occurs if GTIDs are not enabled on the server.
`AUTO`	Add a `SET` statement to the output if GTIDs are enabled on the server.

  The `--set-gtid-purged` option has the following effect on binary logging when the dump file is reloaded:

  - `--set-gtid-purged=OFF`: `SET @@SESSION.SQL_LOG_BIN=0;` is not added to the output.
  - `--set-gtid-purged=ON`: `SET @@SESSION.SQL_LOG_BIN=0;` is added to the output.
  - `--set-gtid-purged=AUTO`: `SET @@SESSION.SQL_LOG_BIN=0;` is added to the output if GTIDs are enabled on the server you are backing up (that is, if `AUTO` evaluates to `ON`).

Note

It is not recommended to load a dump file when GTIDs are enabled on the server (`gtid_mode=ON`), if your dump file includes system tables. `mysqldump` issues DML instructions for the system tables which use the non-transactional MyISAM storage engine, and this combination is not permitted when GTIDs are enabled. Also be aware that loading a dump file from a server with GTIDs enabled, into another server with GTIDs enabled, causes different transaction identifiers to be generated.

Format Options

The following options specify how to represent the entire dump file or certain kinds of data in the dump file. They also control whether certain optional information is written to the dump file.

- `--compact`

Produce more compact output. This option enables the `--skip-add-drop-table`, `--skip-add-locks`, `--skip-comments`, `--skip-disable-keys`, and `--skip-set-charset` options.

- `--compatible=name`

  Produce output that is more compatible with other database systems or with older MySQL servers. The value of *name* can be `ansi`, `mysql323`, `mysql40`, `postgresql`, `oracle`, `mssql`, `db2`, `maxdb`, `no_key_options`, `no_table_options`, or `no_field_options`. To use several values, separate them by commas. These values have the same meaning as the corresponding options for setting the server SQL mode. See Section 5.1.10, “Server SQL Modes”.

  This option does not guarantee compatibility with other servers. It only enables those SQL mode values that are currently available for making dump output more compatible. For example, `--compatible=oracle` does not map data types to Oracle types or use Oracle comment syntax.

- `--complete-insert`, `-c`

  Use complete `INSERT` statements that include column names.

- `--create-options`

  Include all MySQL-specific table options in the `CREATE TABLE` statements.

- `--fields-terminated-by=...`, `--fields-enclosed-by=...`, `--fields-optionally-enclosed-by=...`, `--fields-escaped-by=...`

  These options are used with the `--tab` option and have the same meaning as the corresponding `FIELDS` clauses for `LOAD DATA`. See Section 13.2.6, “LOAD DATA Statement”.

- `--hex-blob`

  Dump binary columns using hexadecimal notation (for example, `'abc'` becomes `0x616263`). The affected data types are `BINARY`, `VARBINARY`, `BLOB` types, `BIT`, all spatial data types, and other non-binary data types when used with the `binary` character set.

- `--lines-terminated-by=...`

  This option is used with the `--tab` option and has the same meaning as the corresponding `LINES` clause for `LOAD DATA`. See Section 13.2.6, “LOAD DATA Statement”.

- `--quote-names`, `-Q`

  Quote identifiers (such as database, table, and column names) within `` ` `` characters. If the `ANSI_QUOTES` SQL mode is enabled, identifiers are quoted within `"` characters. This option is enabled by default. It can be disabled with `--skip-quote-names`, but this option should be given after any option such as `--compatible` that may enable `--quote-names`.

- `--result-file=file_name`, `-r file_name`

  Direct output to the named file. The result file is created and its previous contents overwritten, even if an error occurs while generating the dump.

  This option should be used on Windows to prevent newline `\n` characters from being converted to `\r\n` carriage return/newline sequences.

- `--tab=dir_name`, `-T dir_name`

  Produce tab-separated text-format data files. For each dumped table, `mysqldump` creates a *`tbl_name`*`.sql` file that contains the `CREATE TABLE` statement that creates the table, and the server writes a *`tbl_name`*`.txt` file that contains its data. The option value is the directory in which to write the files.

Note

This option should be used only when `mysqldump` is run on the same machine as the `mysqld` server. Because the server creates `*.txt` files in the directory that you specify, the directory must be writable by the server and the MySQL account that you use must have the `FILE` privilege. Because `mysqldump` creates `*.sql` in the same directory, it must be writable by your system login account.

By default, the `.txt` data files are formatted using tab characters between column values and a newline at the end of each line. The format can be specified explicitly using the `--fields-xxx` and `--lines-terminated-by` options.

Column values are converted to the character set specified by the `--default-character-set` option.

- `--tz-utc`

  This option enables `TIMESTAMP` columns to be dumped and reloaded between servers in different time zones. `mysqldump` sets its connection time zone to UTC and adds `SET TIME_ZONE='+00:00'` to the dump file. Without this option, `TIMESTAMP` columns are dumped and reloaded in the time zones local to the source and destination servers, which can cause the values to change if the servers are in different time zones. `--tz-utc` also protects against changes due to daylight saving time. `--tz-utc` is enabled by default. To disable it, use `--skip-tz-utc`.

- `--xml`, `-X`

  Write dump output as well-formed XML.

  `NULL`, `'NULL'`, and Empty Values: For a column named *`column_name`*, the `NULL` value, an empty string, and the string value `'NULL'` are distinguished from one another in the output generated by this option as follows.

Value:	XML Representation:
`NULL` (*unknown value*)	`<field name="column_name" xsi:nil="true" />`
`''` (*empty string*)	`<field name="column_name"></field>`
`'NULL'` (*string value*)	`<field name="column_name">NULL</field>`

  The output from the `mysql` client when run using the `--xml` option also follows the preceding rules. (See Section 4.5.1.1, “mysql Client Options”.)

  XML output from `mysqldump` includes the XML namespace, as shown here:

```
shell> mysqldump --xml -u root world City
<?xml version="1.0"?>
<mysqldump xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
<database name="world">
<table_structure name="City">
<field Field="ID" Type="int(11)" Null="NO" Key="PRI" Extra="auto_increment" />
<field Field="Name" Type="char(35)" Null="NO" Key="" Default="" Extra="" />
<field Field="CountryCode" Type="char(3)" Null="NO" Key="" Default="" Extra="" />
<field Field="District" Type="char(20)" Null="NO" Key="" Default="" Extra="" />
<field Field="Population" Type="int(11)" Null="NO" Key="" Default="0" Extra="" />
<key Table="City" Non_unique="0" Key_name="PRIMARY" Seq_in_index="1" Column_name="ID"
Collation="A" Cardinality="4079" Null="" Index_type="BTREE" Comment="" />
<options Name="City" Engine="MyISAM" Version="10" Row_format="Fixed" Rows="4079"
Avg_row_length="67" Data_length="273293" Max_data_length="18858823439613951"
Index_length="43008" Data_free="0" Auto_increment="4080"
Create_time="2007-03-31 01:47:01" Update_time="2007-03-31 01:47:02"
```

```
Collation="latin1_swedish_ci" Create_options="" Comment="" />
</table_structure>
<table_data name="City">
<row>
<field name="ID">1</field>
<field name="Name">Kabul</field>
<field name="CountryCode">AFG</field>
<field name="District">Kabol</field>
<field name="Population">1780000</field>
</row>

...

<row>
<field name="ID">4079</field>
<field name="Name">Rafah</field>
<field name="CountryCode">PSE</field>
<field name="District">Rafah</field>
<field name="Population">92020</field>
</row>
</table_data>
</database>
</mysqldump>
```

Filtering Options

The following options control which kinds of schema objects are written to the dump file: by category, such as triggers or events; by name, for example, choosing which databases and tables to dump; or even filtering rows from the table data using a `WHERE` clause.

- `--all-databases`, `-A`

  Dump all tables in all databases. This is the same as using the `--databases` option and naming all the databases on the command line.

- `--databases`, `-B`

  Dump several databases. Normally, `mysqldump` treats the first name argument on the command line as a database name and following names as table names. With this option, it treats all name arguments as database names. `CREATE DATABASE` and `USE` statements are included in the output before each new database.

  This option may be used to dump the `INFORMATION_SCHEMA` and `performance_schema` databases, which normally are not dumped even with the `--all-databases` option. (Also use the `--skip-lock-tables` option.)

- `--events`, `-E`

  Include Event Scheduler events for the dumped databases in the output. This option requires the `EVENT` privileges for those databases.

  The output generated by using `--events` contains `CREATE EVENT` statements to create the events. However, these statements do not include attributes such as the event creation and modification timestamps, so when the events are reloaded, they are created with timestamps equal to the reload time.

  If you require events to be created with their original timestamp attributes, do not use `--events`. Instead, dump and reload the contents of the `mysql.event` table directly, using a MySQL account that has appropriate privileges for the `mysql` database.

- `--ignore-table=`*`db_name.tbl_name`*

  Do not dump the given table, which must be specified using both the database and table names. To ignore multiple tables, use this option multiple times. This option also can be used to ignore views.

- `--no-data`, `-d`

  Do not write any table row information (that is, do not dump table contents). This is useful if you want to dump only the `CREATE TABLE` statement for the table (for example, to create an empty copy of the table by loading the dump file).

- `--routines`, `-R`

  Include stored routines (procedures and functions) for the dumped databases in the output. This option requires the `SELECT` privilege for the `mysql.proc` table.

  The output generated by using `--routines` contains `CREATE PROCEDURE` and `CREATE FUNCTION` statements to create the routines. However, these statements do not include attributes such as the routine creation and modification timestamps, so when the routines are reloaded, they are created with timestamps equal to the reload time.

  If you require routines to be created with their original timestamp attributes, do not use `--routines`. Instead, dump and reload the contents of the `mysql.proc` table directly, using a MySQL account that has appropriate privileges for the `mysql` database.

- `--tables`

  Override the `--databases` or `-B` option. `mysqldump` regards all name arguments following the option as table names.

- `--triggers`

  Include triggers for each dumped table in the output. This option is enabled by default; disable it with `--skip-triggers`.

  To be able to dump a table's triggers, you must have the `TRIGGER` privilege for the table.

- `--where='where_condition'`, `-w 'where_condition'`

  Dump only rows selected by the given `WHERE` condition. Quotes around the condition are mandatory if it contains spaces or other characters that are special to your command interpreter.

  Examples:

```
--where="user='jimf'"
-w"userid>1"
-w"userid<1"
```

Performance Options

The following options are the most relevant for the performance particularly of the restore operations. For large data sets, restore operation (processing the `INSERT` statements in the dump file) is the most time-consuming part. When it is urgent to restore data quickly, plan and test the performance of this stage in advance. For restore times measured in hours, you might prefer an alternative backup and restore solution, such as MySQL Enterprise Backup for `InnoDB`-only and mixed-use databases, or `mysqlhotcopy` for `MyISAM`-only databases.

Performance is also affected by the transactional options, primarily for the dump operation.

- `--delayed-insert`

  For those nontransactional tables that support the `INSERT DELAYED` syntax, use that statement rather than regular `INSERT` statements.

  As of MySQL 5.6.6, `DELAYED` inserts are deprecated; expect this option to be removed in a future release.

- `--disable-keys`, `-K`

For each table, surround the INSERT statements with /*!40000 ALTER TABLE *tbl_name* DISABLE KEYS */; and /*!40000 ALTER TABLE *tbl_name* ENABLE KEYS */; statements. This makes loading the dump file faster because the indexes are created after all rows are inserted. This option is effective only for nonunique indexes of MyISAM tables.

- --extended-insert, -e

  Write INSERT statements using multiple-row syntax that includes several VALUES lists. This results in a smaller dump file and speeds up inserts when the file is reloaded.

- --insert-ignore

  Write INSERT IGNORE statements rather than INSERT statements.

- --max-allowed-packet=*value*

  The maximum size of the buffer for client/server communication. The default is 24MB, the maximum is 1GB.

- --net-buffer-length=*value*

  The initial size of the buffer for client/server communication. When creating multiple-row INSERT statements (as with the --extended-insert or --opt option), mysqldump creates rows up to --net-buffer-length bytes long. If you increase this variable, ensure that the MySQL server net_buffer_length system variable has a value at least this large.

- --opt

  This option, enabled by default, is shorthand for the combination of --add-drop-table --add-locks --create-options --disable-keys --extended-insert --lock-tables --quick --set-charset. It gives a fast dump operation and produces a dump file that can be reloaded into a MySQL server quickly.

  Because the --opt option is enabled by default, you only specify its converse, the --skip-opt to turn off several default settings. See the discussion of mysqldump option groups for information about selectively enabling or disabling a subset of the options affected by --opt.

- --quick, -q

  This option is useful for dumping large tables. It forces mysqldump to retrieve rows for a table from the server a row at a time rather than retrieving the entire row set and buffering it in memory before writing it out.

- --skip-opt

  See the description for the --opt option.

Transactional Options

The following options trade off the performance of the dump operation, against the reliability and consistency of the exported data.

- --add-locks

  Surround each table dump with LOCK TABLES and UNLOCK TABLES statements. This results in faster inserts when the dump file is reloaded. See Section 8.2.4.1, “Optimizing INSERT Statements”.

- --flush-logs, -F

  Flush the MySQL server log files before starting the dump. This option requires the RELOAD privilege. If you use this option in combination with the --all-databases option, the logs are flushed *for each database dumped*. The exception is when using --lock-all-tables,

--master-data, or --single-transaction: In this case, the logs are flushed only once, corresponding to the moment that all tables are locked by FLUSH TABLES WITH READ LOCK. If you want your dump and the log flush to happen at exactly the same moment, you should use --flush-logs together with --lock-all-tables, --master-data, or --single-transaction.

- --flush-privileges

  Add a FLUSH PRIVILEGES statement to the dump output after dumping the mysql database. This option should be used any time the dump contains the mysql database and any other database that depends on the data in the mysql database for proper restoration.

  Because the dump file contains a FLUSH PRIVILEGES statement, reloading the file requires privileges sufficient to execute that statement.

- --lock-all-tables, -x

  Lock all tables across all databases. This is achieved by acquiring a global read lock for the duration of the whole dump. This option automatically turns off --single-transaction and --lock-tables.

- --lock-tables, -l

  For each dumped database, lock all tables to be dumped before dumping them. The tables are locked with READ LOCAL to permit concurrent inserts in the case of MyISAM tables. For transactional tables such as InnoDB, --single-transaction is a much better option than --lock-tables because it does not need to lock the tables at all.

  Because --lock-tables locks tables for each database separately, this option does not guarantee that the tables in the dump file are logically consistent between databases. Tables in different databases may be dumped in completely different states.

  Some options, such as --opt, automatically enable --lock-tables. If you want to override this, use --skip-lock-tables at the end of the option list.

- --no-autocommit

  Enclose the INSERT statements for each dumped table within SET autocommit = 0 and COMMIT statements.

- --order-by-primary

  Dump each table's rows sorted by its primary key, or by its first unique index, if such an index exists. This is useful when dumping a MyISAM table to be loaded into an InnoDB table, but makes the dump operation take considerably longer.

- --shared-memory-base-name=*name*

  On Windows, the shared-memory name to use for connections made using shared memory to a local server. The default value is MYSQL. The shared-memory name is case-sensitive.

  This option applies only if the server was started with the shared_memory system variable enabled to support shared-memory connections.

- --single-transaction

  This option sets the transaction isolation mode to REPEATABLE READ and sends a START TRANSACTION SQL statement to the server before dumping data. It is useful only with transactional

tables such as InnoDB, because then it dumps the consistent state of the database at the time when START TRANSACTION was issued without blocking any applications.

When using this option, you should keep in mind that only InnoDB tables are dumped in a consistent state. For example, any MyISAM or MEMORY tables dumped while using this option may still change state.

While a --single-transaction dump is in process, to ensure a valid dump file (correct table contents and binary log coordinates), no other connection should use the following statements: ALTER TABLE, CREATE TABLE, DROP TABLE, RENAME TABLE, TRUNCATE TABLE. A consistent read is not isolated from those statements, so use of them on a table to be dumped can cause the SELECT that is performed by mysqldump to retrieve the table contents to obtain incorrect contents or fail.

The --single-transaction option and the --lock-tables option are mutually exclusive because LOCK TABLES causes any pending transactions to be committed implicitly.

To dump large tables, combine the --single-transaction option with the --quick option.

Option Groups

- The --opt option turns on several settings that work together to perform a fast dump operation. All of these settings are on by default, because --opt is on by default. Thus you rarely if ever specify --opt. Instead, you can turn these settings off as a group by specifying --skip-opt, the optionally re-enable certain settings by specifying the associated options later on the command line.
- The --compact option turns off several settings that control whether optional statements and comments appear in the output. Again, you can follow this option with other options that re-enable certain settings, or turn all the settings on by using the --skip-compact form.

When you selectively enable or disable the effect of a group option, order is important because options are processed first to last. For example, --disable-keys --lock-tables --skip-opt would not have the intended effect; it is the same as --skip-opt by itself.

Examples

To make a backup of an entire database:

```
shell> mysqldump db_name > backup-file.sql
```

To load the dump file back into the server:

```
shell> mysql db_name < backup-file.sql
```

Another way to reload the dump file:

```
shell> mysql -e "source /path-to-backup/backup-file.sql" db_name
```

mysqldump is also very useful for populating databases by copying data from one MySQL server to another:

```
shell> mysqldump --opt db_name | mysql --host=remote_host -C db_name
```

You can dump several databases with one command:

```
shell> mysqldump --databases db_name1 [db_name2 ...] > my_databases.sql
```

To dump all databases, use the --all-databases option:

```
shell> mysqldump --all-databases > all_databases.sql
```

For InnoDB tables, mysqldump provides a way of making an online backup:

```
shell> mysqldump --all-databases --master-data --single-transaction > all_databases.sql
```

This backup acquires a global read lock on all tables (using `FLUSH TABLES WITH READ LOCK`) at the beginning of the dump. As soon as this lock has been acquired, the binary log coordinates are read and the lock is released. If long updating statements are running when the `FLUSH` statement is issued, the MySQL server may get stalled until those statements finish. After that, the dump becomes lock free and does not disturb reads and writes on the tables. If the update statements that the MySQL server receives are short (in terms of execution time), the initial lock period should not be noticeable, even with many updates.

For point-in-time recovery (also known as “roll-forward,” when you need to restore an old backup and replay the changes that happened since that backup), it is often useful to rotate the binary log (see Section 5.4.4, “The Binary Log”) or at least know the binary log coordinates to which the dump corresponds:

```
shell> mysqldump --all-databases --master-data=2 > all_databases.sql
```

Or:

```
shell> mysqldump --all-databases --flush-logs --master-data=2
             > all_databases.sql
```

The `--master-data` and `--single-transaction` options can be used simultaneously, which provides a convenient way to make an online backup suitable for use prior to point-in-time recovery if tables are stored using the `InnoDB` storage engine.

For more information on making backups, see Section 7.2, “Database Backup Methods”, and Section 7.3, “Example Backup and Recovery Strategy”.

- To select the effect of `--opt` except for some features, use the `--skip` option for each feature. To disable extended inserts and memory buffering, use `--opt --skip-extended-insert --skip-quick`. (Actually, `--skip-extended-insert --skip-quick` is sufficient because `--opt` is on by default.)
- To reverse `--opt` for all features except index disabling and table locking, use `--skip-opt --disable-keys --lock-tables`.

Restrictions

`mysqldump` does not dump the `INFORMATION_SCHEMA` or `performance_schema` database by default. To dump either of these, name it explicitly on the command line. You can also name it with the `--databases` option. Also, use the `--skip-lock-tables` option.

`mysqldump` does not dump the NDB Cluster `ndbinfo` information database.

If you encounter problems backing up views due to insufficient privileges, see Section 20.9, “Restrictions on Views” for a workaround.

4.5.5 `mysqlimport` — A Data Import Program

The `mysqlimport` client provides a command-line interface to the `LOAD DATA` SQL statement. Most options to `mysqlimport` correspond directly to clauses of `LOAD DATA` syntax. See Section 13.2.6, “LOAD DATA Statement”.

Invoke `mysqlimport` like this:

```
shell> mysqlimport [options] db_name textfile1 [textfile2 ...]
```

For each text file named on the command line, `mysqlimport` strips any extension from the file name and uses the result to determine the name of the table into which to import the file's contents. For example, files named `patient.txt`, `patient.text`, and `patient` all would be imported into a table named `patient`.

mysqlimport supports the following options, which can be specified on the command line or in the [mysqlimport] and [client] groups of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.14 mysqlimport Options

Option Name	Description	Introduced
--bind-address	Use specified network interface to connect to MySQL Server	
--columns	This option takes a comma-separated list of column names as its value	
--compress	Compress all information sent between client and server	
--debug	Write debugging log	
--debug-check	Print debugging information when program exits	
--debug-info	Print debugging information, memory, and CPU statistics when program exits	
--default-auth	Authentication plugin to use	
--default-character-set	Specify default character set	
--defaults-extra-file	Read named option file in addition to usual option files	
--defaults-file	Read only named option file	
--defaults-group-suffix	Option group suffix value	
--delete	Empty the table before importing the text file	
--enable-cleartext-plugin	Enable cleartext authentication plugin	5.6.28
--fields-enclosed-by	This option has the same meaning as the corresponding clause for LOAD DATA	
--fields-escaped-by	This option has the same meaning as the corresponding clause for LOAD DATA	
--fields-optionally-enclosed-by	This option has the same meaning as the corresponding clause for LOAD DATA	
--fields-terminated-by	This option has the same meaning as the corresponding clause for LOAD DATA	
--force	Continue even if an SQL error occurs	
--help	Display help message and exit	
--host	Host on which MySQL server is located	
--ignore	See the description for the --replace option	
--ignore-lines	Ignore the first N lines of the data file	
--lines-terminated-by	This option has the same meaning as the corresponding clause for LOAD DATA	
--local	Read input files locally from the client host	
--lock-tables	Lock all tables for writing before processing any text files	
--login-path	Read login path options from .mylogin.cnf	
--low-priority	Use LOW_PRIORITY when loading the table	
--no-defaults	Read no option files	
--password	Password to use when connecting to server	
--pipe	Connect to server using named pipe (Windows only)	
--plugin-dir	Directory where plugins are installed	

Option Name	Description	Introduced
--port	TCP/IP port number for connection	
--print-defaults	Print default options	
--protocol	Transport protocol to use	
--replace	The --replace and --ignore options control handling of input rows that duplicate existing rows on unique key values	
--secure-auth	Do not send passwords to server in old (pre-4.1) format	5.6.17
--shared-memory-base-name	Shared-memory name for shared-memory connections (Windows only)	
--silent	Produce output only when errors occur	
--socket	Unix socket file or Windows named pipe to use	
--ssl	Enable connection encryption	
--ssl-ca	File that contains list of trusted SSL Certificate Authorities	
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files	
--ssl-cert	File that contains X.509 certificate	
--ssl-cipher	Permissible ciphers for connection encryption	
--ssl-crl	File that contains certificate revocation lists	
--ssl-crlpath	Directory that contains certificate revocation-list files	
--ssl-key	File that contains X.509 key	
--ssl-mode	Desired security state of connection to server	5.6.30
--ssl-verify-server-cert	Verify host name against server certificate Common Name identity	
--use-threads	Number of threads for parallel file-loading	
--user	MySQL user name to use when connecting to server	
--verbose	Verbose mode	
--version	Display version information and exit	

- `--help`, `-?`

  Display a help message and exit.

- `--bind-address=`*`ip_address`*

  On a computer having multiple network interfaces, use this option to select which interface to use for connecting to the MySQL server.

- `--character-sets-dir=`*`dir_name`*

  The directory where character sets are installed. See Section 10.15, “Character Set Configuration”.

- `--columns=`*`column_list`*, `-c` *`column_list`*

  This option takes a list of comma-separated column names as its value. The order of the column names indicates how to match data file columns with table columns.

- `--compress`, `-C`

  Compress all information sent between the client and the server if possible. See Section 4.2.6, “Connection Compression Control”.

- `--debug[=debug_options]`, `-# [debug_options]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,file_name`. The default is `d:t:o`.

  This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are *not* built using this option.

- `--debug-check`

  Print some debugging information when the program exits.

  This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are *not* built using this option.

- `--debug-info`

  Print debugging information and memory and CPU usage statistics when the program exits.

  This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are *not* built using this option.

- `--default-character-set=charset_name`

  Use *`charset_name`* as the default character set. See Section 10.15, “Character Set Configuration”.

- `--default-auth=plugin`

  A hint about which client-side authentication plugin to use. See Section 6.2.11, “Pluggable Authentication”.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  Exception: Even with `--defaults-file`, client programs read `.mylogin.cnf`.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysqlimport` normally reads the `[client]` and `[mysqlimport]` groups. If the `--defaults-group-suffix=_other` option is given, `mysqlimport` also reads the `[client_other]` and `[mysqlimport_other]` groups.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--delete`, `-D`

Empty the table before importing the text file.

- `--enable-cleartext-plugin`

  Enable the `mysql_clear_password` cleartext authentication plugin. (See Section 6.4.1.5, “Client-Side Cleartext Pluggable Authentication”.)

  This option was added in MySQL 5.6.28.

- `--fields-terminated-by=...`, `--fields-enclosed-by=...`, `--fields-optionally-enclosed-by=...`, `--fields-escaped-by=...`

  These options have the same meaning as the corresponding clauses for `LOAD DATA`. See Section 13.2.6, “LOAD DATA Statement”.

- `--force`, `-f`

  Ignore errors. For example, if a table for a text file does not exist, continue processing any remaining files. Without `--force`, `mysqlimport` exits if a table does not exist.

- `--host=`*`host_name`*, `-h` *`host_name`*

  Import data to the MySQL server on the given host. The default host is `localhost`.

- `--ignore`, `-i`

  See the description for the `--replace` option.

- `--ignore-lines=`*`N`*

  Ignore the first *`N`* lines of the data file.

- `--lines-terminated-by=...`

  This option has the same meaning as the corresponding clause for `LOAD DATA`. For example, to import Windows files that have lines terminated with carriage return/linefeed pairs, use `--lines-terminated-by="\r\n"`. (You might have to double the backslashes, depending on the escaping conventions of your command interpreter.) See Section 13.2.6, “LOAD DATA Statement”.

- `--local`, `-L`

  By default, files are read by the server on the server host. With this option, `mysqlimport` reads input files locally on the client host.

  Successful use of `LOCAL` load operations within `mysqlimport` also requires that the server permits local loading; see Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”

- `--lock-tables`, `-l`

  Lock *all* tables for writing before processing any text files. This ensures that all tables are synchronized on the server.

- `--login-path=`*`name`*

  Read options from the named login path in the `.mylogin.cnf` login path file. A “login path” is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the `mysql_config_editor` utility. See Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--low-priority`

Use LOW_PRIORITY when loading the table. This affects only storage engines that use only table-level locking (such as MyISAM, MEMORY, and MERGE).

- --no-defaults

  Do not read any option files. If program startup fails due to reading unknown options from an option file, --no-defaults can be used to prevent them from being read.

  The exception is that the .mylogin.cnf file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when --no-defaults is used. (.mylogin.cnf is created by the mysql_config_editor utility. See Section 4.6.6, "mysql_config_editor — MySQL Configuration Utility".)

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- --password[=*password*], -p[*password*]

  The password of the MySQL account used for connecting to the server. The password value is optional. If not given, mysqlimport prompts for one. If given, there must be *no space* between --password= or -p and the password following it. If no password option is specified, the default is to send no password.

  Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, "End-User Guidelines for Password Security".

  To explicitly specify that there is no password and that mysqlimport should not prompt for one, use the --skip-password option.

- --pipe, -W

  On Windows, connect to the server using a named pipe. This option applies only if the server was started with the named_pipe system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the named_pipe_full_access_group system variable.

- --plugin-dir=*dir_name*

  The directory in which to look for plugins. Specify this option if the --default-auth option is used to specify an authentication plugin but mysqlimport does not find it. See Section 6.2.11, "Pluggable Authentication".

- --port=*port_num*, -P *port_num*

  For TCP/IP connections, the port number to use.

- --print-defaults

  Print the program name and all options that it gets from option files.

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- --protocol={TCP|SOCKET|PIPE|MEMORY}

  The transport protocol to use for connecting to the server. It is useful when the other connection parameters normally result in use of a protocol other than the one you want. For details on the permissible values, see Section 4.2.5, "Connection Transport Protocols".

- --replace, -r

The `--replace` and `--ignore` options control handling of input rows that duplicate existing rows on unique key values. If you specify `--replace`, new rows replace existing rows that have the same unique key value. If you specify `--ignore`, input rows that duplicate an existing row on a unique key value are skipped. If you do not specify either option, an error occurs when a duplicate key value is found, and the rest of the text file is ignored.

- `--secure-auth`

  Do not send passwords to the server in old (pre-4.1) format. This prevents connections except for servers that use the newer password format. This option is enabled by default; use `--skip-secure-auth` to disable it. This option was added in MySQL 5.6.17.

  Note

  Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated; expect support for them to be removed in a future MySQL release. For account upgrade instructions, see Section 6.4.1.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”.

  Note

  This option is deprecated; expect it to be removed in a future release. As of MySQL 5.7.5, it is always enabled and attempting to disable it produces an error.

- `--shared-memory-base-name=`*`name`*

  On Windows, the shared-memory name to use for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case-sensitive.

  This option applies only if the server was started with the `shared_memory` system variable enabled to support shared-memory connections.

- `--silent`, `-s`

  Silent mode. Produce output only when errors occur.

- `--socket=`*`path`*, `-S` *`path`*

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

  On Windows, this option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Command Options for Encrypted Connections.

- `--user=`*`user_name`*, `-u` *`user_name`*

  The user name of the MySQL account to use for connecting to the server.

- `--use-threads=`*`N`*

  Load files in parallel using *`N`* threads.

- `--verbose`, `-v`

  Verbose mode. Print more information about what the program does.

- `--version`, `-V`

  Display version information and exit.

Here is a sample session that demonstrates use of `mysqlimport`:

```
shell> mysql -e 'CREATE TABLE imptest(id INT, n VARCHAR(30))' test
shell> ed
a
100     Max Sydow
101     Count Dracula
.
w imptest.txt
32
q
shell> od -c imptest.txt
0000000   1   0   0  \t   M   a   x       S   y   d   o   w  \n   1   0
0000020   1  \t   C   o   u   n   t       D   r   a   c   u   l   a  \n
0000040
shell> mysqlimport --local test imptest.txt
test.imptest: Records: 2  Deleted: 0  Skipped: 0  Warnings: 0
shell> mysql -e 'SELECT * FROM imptest' test
+------+---------------+
| id   | n             |
+------+---------------+
|  100 | Max Sydow     |
|  101 | Count Dracula |
+------+---------------+
```

4.5.6 `mysqlshow` — Display Database, Table, and Column Information

The `mysqlshow` client can be used to quickly see which databases exist, their tables, or a table's columns or indexes.

`mysqlshow` provides a command-line interface to several SQL `SHOW` statements. See Section 13.7.5, “SHOW Statements”. The same information can be obtained by using those statements directly. For example, you can issue them from the `mysql` client program.

Invoke `mysqlshow` like this:

```
shell> mysqlshow [options] [db_name [tbl_name [col_name]]]
```

- If no database is given, a list of database names is shown.
- If no table is given, all matching tables in the database are shown.
- If no column is given, all matching columns and column types in the table are shown.

The output displays only the names of those databases, tables, or columns for which you have some privileges.

If the last argument contains shell or SQL wildcard characters (`*`, `?`, `%`, or `_`), only those names that are matched by the wildcard are shown. If a database name contains any underscores, those should be escaped with a backslash (some Unix shells require two) to get a list of the proper tables or columns. `*` and `?` characters are converted into SQL `%` and `_` wildcard characters. This might cause some confusion when you try to display the columns for a table with a `_` in the name, because in this case, `mysqlshow` shows you only the table names that match the pattern. This is easily fixed by adding an extra `%` last on the command line as a separate argument.

`mysqlshow` supports the following options, which can be specified on the command line or in the `[mysqlshow]` and `[client]` groups of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.15 mysqlshow Options

Option Name	Description	Introduced
--bind-address	Use specified network interface to connect to MySQL Server	
--compress	Compress all information sent between client and server	
--count	Show the number of rows per table	
--debug	Write debugging log	
--debug-check	Print debugging information when program exits	
--debug-info	Print debugging information, memory, and CPU statistics when program exits	
--default-auth	Authentication plugin to use	
--default-character-set	Specify default character set	
--defaults-extra-file	Read named option file in addition to usual option files	
--defaults-file	Read only named option file	
--defaults-group-suffix	Option group suffix value	
--enable-cleartext-plugin	Enable cleartext authentication plugin	5.6.28
--help	Display help message and exit	
--host	Host on which MySQL server is located	
--keys	Show table indexes	
--login-path	Read login path options from .mylogin.cnf	
--no-defaults	Read no option files	
--password	Password to use when connecting to server	
--pipe	Connect to server using named pipe (Windows only)	
--plugin-dir	Directory where plugins are installed	
--port	TCP/IP port number for connection	
--print-defaults	Print default options	
--protocol	Transport protocol to use	
--secure-auth	Do not send passwords to server in old (pre-4.1) format	5.6.17
--shared-memory-base-name	Shared-memory name for shared-memory connections (Windows only)	
--show-table-type	Show a column indicating the table type	
--socket	Unix socket file or Windows named pipe to use	
--ssl	Enable connection encryption	
--ssl-ca	File that contains list of trusted SSL Certificate Authorities	
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files	
--ssl-cert	File that contains X.509 certificate	
--ssl-cipher	Permissible ciphers for connection encryption	
--ssl-crl	File that contains certificate revocation lists	
--ssl-crlpath	Directory that contains certificate revocation-list files	
--ssl-key	File that contains X.509 key	
--ssl-mode	Desired security state of connection to server	5.6.30

Option Name	Description	Introduced
--ssl-verify-server-cert	Verify host name against server certificate Common Name identity	
--status	Display extra information about each table	
--user	MySQL user name to use when connecting to server	
--verbose	Verbose mode	
--version	Display version information and exit	

- `--help`, `-?`

  Display a help message and exit.

- `--bind-address=ip_address`

  On a computer having multiple network interfaces, use this option to select which interface to use for connecting to the MySQL server.

- `--character-sets-dir=dir_name`

  The directory where character sets are installed. See Section 10.15, “Character Set Configuration”.

- `--compress`, `-C`

  Compress all information sent between the client and the server if possible. See Section 4.2.6, “Connection Compression Control”.

- `--count`

  Show the number of rows per table. This can be slow for non-`MyISAM` tables.

- `--debug[=debug_options]`, `-# [debug_options]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,file_name`. The default is `d:t:o`.

  This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are *not* built using this option.

- `--debug-check`

  Print some debugging information when the program exits.

  This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are *not* built using this option.

- `--debug-info`

  Print debugging information and memory and CPU usage statistics when the program exits.

  This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are *not* built using this option.

- `--default-character-set=charset_name`

  Use *`charset_name`* as the default character set. See Section 10.15, “Character Set Configuration”.

- `--default-auth=plugin`

  A hint about which client-side authentication plugin to use. See Section 6.2.11, “Pluggable Authentication”.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  Exception: Even with `--defaults-file`, client programs read `.mylogin.cnf`.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysqlshow` normally reads the `[client]` and `[mysqlshow]` groups. If the `--defaults-group-suffix=_other` option is given, `mysqlshow` also reads the `[client_other]` and `[mysqlshow_other]` groups.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--enable-cleartext-plugin`

  Enable the `mysql_clear_password` cleartext authentication plugin. (See Section 6.4.1.5, “Client-Side Cleartext Pluggable Authentication”.)

  This option was added in MySQL 5.6.28.

- `--host=host_name`, `-h host_name`

  Connect to the MySQL server on the given host.

- `--keys`, `-k`

  Show table indexes.

- `--login-path=name`

  Read options from the named login path in the `.mylogin.cnf` login path file. A “login path” is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the `mysql_config_editor` utility. See Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

  The exception is that the `.mylogin.cnf` file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when `--no-defaults`

is used. (`.mylogin.cnf` is created by the `mysql_config_editor` utility. See Section 4.6.6, "`mysql_config_editor` — MySQL Configuration Utility".)

For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- `--password[=password]`, `-p[password]`

  The password of the MySQL account used for connecting to the server. The password value is optional. If not given, `mysqlshow` prompts for one. If given, there must be *no space* between `--password=` or `-p` and the password following it. If no password option is specified, the default is to send no password.

  Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, "End-User Guidelines for Password Security".

  To explicitly specify that there is no password and that `mysqlshow` should not prompt for one, use the `--skip-password` option.

- `--pipe`, `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--plugin-dir=dir_name`

  The directory in which to look for plugins. Specify this option if the `--default-auth` option is used to specify an authentication plugin but `mysqlshow` does not find it. See Section 6.2.11, "Pluggable Authentication".

- `--port=port_num`, `-P port_num`

  For TCP/IP connections, the port number to use.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  The transport protocol to use for connecting to the server. It is useful when the other connection parameters normally result in use of a protocol other than the one you want. For details on the permissible values, see Section 4.2.5, "Connection Transport Protocols".

- `--secure-auth`

  Do not send passwords to the server in old (pre-4.1) format. This prevents connections except for servers that use the newer password format. This option is enabled by default; use `--skip-secure-auth` to disable it. This option was added in MySQL 5.6.17.

Note

Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated; expect support for them to be removed in a future MySQL release. For account upgrade instructions, see

Section 6.4.1.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”.

Note

This option is deprecated; expect it to be removed in a future release. As of MySQL 5.7.5, it is always enabled and attempting to disable it produces an error.

- `--shared-memory-base-name=name`

  On Windows, the shared-memory name to use for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case-sensitive.

  This option applies only if the server was started with the `shared_memory` system variable enabled to support shared-memory connections.

- `--show-table-type`, `-t`

  Show a column indicating the table type, as in `SHOW FULL TABLES`. The type is `BASE TABLE` or `VIEW`.

- `--socket=path`, `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

  On Windows, this option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Command Options for Encrypted Connections.

- `--status`, `-i`

  Display extra information about each table.

- `--user=user_name`, `-u user_name`

  The user name of the MySQL account to use for connecting to the server.

- `--verbose`, `-v`

  Verbose mode. Print more information about what the program does. This option can be used multiple times to increase the amount of information.

- `--version`, `-V`

  Display version information and exit.

4.5.7 `mysqlslap` — A Load Emulation Client

`mysqlslap` is a diagnostic program designed to emulate client load for a MySQL server and to report the timing of each stage. It works as if multiple clients are accessing the server.

Invoke `mysqlslap` like this:

```
shell> mysqlslap [options]
```

Some options such as `--create` or `--query` enable you to specify a string containing an SQL statement or a file containing statements. If you specify a file, by default it must contain one statement per line. (That is, the implicit statement delimiter is the newline character.) Use the `--delimiter` option to specify a different delimiter, which enables you to specify statements that span multiple lines or place multiple statements on a single line. You cannot include comments in a file; `mysqlslap` does not understand them.

`mysqlslap` runs in three stages:

1. Create schema, table, and optionally any stored programs or data to use for the test. This stage uses a single client connection.
2. Run the load test. This stage can use many client connections.
3. Clean up (disconnect, drop table if specified). This stage uses a single client connection.

Examples:

Supply your own create and query SQL statements, with 50 clients querying and 200 selects for each (enter the command on a single line):

```
mysqlslap --delimiter=";"
  --create="CREATE TABLE a (b int);INSERT INTO a VALUES (23)"
  --query="SELECT * FROM a" --concurrency=50 --iterations=200
```

Let `mysqlslap` build the query SQL statement with a table of two `INT` columns and three `VARCHAR` columns. Use five clients querying 20 times each. Do not create the table or insert the data (that is, use the previous test's schema and data):

```
mysqlslap --concurrency=5 --iterations=20
  --number-int-cols=2 --number-char-cols=3
  --auto-generate-sql
```

Tell the program to load the create, insert, and query SQL statements from the specified files, where the `create.sql` file has multiple table creation statements delimited by `';'` and multiple insert statements delimited by `';'`. The `--query` file has multiple queries delimited by `';'`. Run all the load statements, then run all the queries in the query file with five clients (five times each):

```
mysqlslap --concurrency=5
  --iterations=5 --query=query.sql --create=create.sql
  --delimiter=";"
```

`mysqlslap` supports the following options, which can be specified on the command line or in the `[mysqlslap]` and `[client]` groups of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.16 mysqlslap Options

Option Name	Description	Introduced
--auto-generate-sql	Generate SQL statements automatically when they are not supplied in files or using command options	
--auto-generate-sql-add-autoincrement	Add AUTO_INCREMENT column to automatically generated tables	
--auto-generate-sql-execute-number	Specify how many queries to generate automatically	
--auto-generate-sql-guid-primary	Add a GUID-based primary key to automatically generated tables	
--auto-generate-sql-load-type	Specify the test load type	
--auto-generate-sql-secondary-indexes	Specify how many secondary indexes to add to automatically generated tables	
--auto-generate-sql-unique-query-number	How many different queries to generate for automatic tests	

Option Name	Description	Introduced
--auto-generate-sql-unique-write-number	How many different queries to generate for --auto-generate-sql-write-number	
--auto-generate-sql-write-number	How many row inserts to perform on each thread	
--commit	How many statements to execute before committing	
--compress	Compress all information sent between client and server	
--concurrency	Number of clients to simulate when issuing the SELECT statement	
--create	File or string containing the statement to use for creating the table	
--create-schema	Schema in which to run the tests	
--csv	Generate output in comma-separated values format	
--debug	Write debugging log	
--debug-check	Print debugging information when program exits	
--debug-info	Print debugging information, memory, and CPU statistics when program exits	
--default-auth	Authentication plugin to use	
--defaults-extra-file	Read named option file in addition to usual option files	
--defaults-file	Read only named option file	
--defaults-group-suffix	Option group suffix value	
--delimiter	Delimiter to use in SQL statements	
--detach	Detach (close and reopen) each connection after each N statements	
--enable-cleartext-plugin	Enable cleartext authentication plugin	
--engine	Storage engine to use for creating the table	
--help	Display help message and exit	
--host	Host on which MySQL server is located	
--iterations	Number of times to run the tests	
--login-path	Read login path options from .mylogin.cnf	
--no-defaults	Read no option files	
--no-drop	Do not drop any schema created during the test run	
--number-char-cols	Number of VARCHAR columns to use if --auto-generate-sql is specified	
--number-int-cols	Number of INT columns to use if --auto-generate-sql is specified	
--number-of-queries	Limit each client to approximately this number of queries	
--only-print	Do not connect to databases. mysqlslap only prints what it would have done	
--password	Password to use when connecting to server	
--pipe	Connect to server using named pipe (Windows only)	
--plugin-dir	Directory where plugins are installed	
--port	TCP/IP port number for connection	
--post-query	File or string containing the statement to execute after the tests have completed	

Option Name	Description	Introduced
--post-system	String to execute using system() after the tests have completed	
--pre-query	File or string containing the statement to execute before running the tests	
--pre-system	String to execute using system() before running the tests	
--print-defaults	Print default options	
--protocol	Transport protocol to use	
--query	File or string containing the SELECT statement to use for retrieving data	
--secure-auth	Do not send passwords to server in old (pre-4.1) format	5.6.17
--shared-memory-base-name	Shared-memory name for shared-memory connections (Windows only)	
--silent	Silent mode	
--socket	Unix socket file or Windows named pipe to use	
--ssl	Enable connection encryption	
--ssl-ca	File that contains list of trusted SSL Certificate Authorities	
--ssl-capath	Directory that contains trusted SSL Certificate Authority certificate files	
--ssl-cert	File that contains X.509 certificate	
--ssl-cipher	Permissible ciphers for connection encryption	
--ssl-crl	File that contains certificate revocation lists	
--ssl-crlpath	Directory that contains certificate revocation-list files	
--ssl-key	File that contains X.509 key	
--ssl-mode	Desired security state of connection to server	5.6.30
--ssl-verify-server-cert	Verify host name against server certificate Common Name identity	
--user	MySQL user name to use when connecting to server	
--verbose	Verbose mode	
--version	Display version information and exit	

- `--help`, `-?`

  Display a help message and exit.

- `--auto-generate-sql`, `-a`

  Generate SQL statements automatically when they are not supplied in files or using command options.

- `--auto-generate-sql-add-autoincrement`

  Add an `AUTO_INCREMENT` column to automatically generated tables.

- `--auto-generate-sql-execute-number=`*`N`*

  Specify how many queries to generate automatically.

- `--auto-generate-sql-guid-primary`

  Add a GUID-based primary key to automatically generated tables.

- `--auto-generate-sql-load-type=type`

  Specify the test load type. The permissible values are `read` (scan tables), `write` (insert into tables), `key` (read primary keys), `update` (update primary keys), or `mixed` (half inserts, half scanning selects). The default is `mixed`.

- `--auto-generate-sql-secondary-indexes=N`

  Specify how many secondary indexes to add to automatically generated tables. By default, none are added.

- `--auto-generate-sql-unique-query-number=N`

  How many different queries to generate for automatic tests. For example, if you run a `key` test that performs 1000 selects, you can use this option with a value of 1000 to run 1000 unique queries, or with a value of 50 to perform 50 different selects. The default is 10.

- `--auto-generate-sql-unique-write-number=N`

  How many different queries to generate for `--auto-generate-sql-write-number`. The default is 10.

- `--auto-generate-sql-write-number=N`

  How many row inserts to perform. The default is 100.

- `--commit=N`

  How many statements to execute before committing. The default is 0 (no commits are done).

- `--compress`, `-C`

  Compress all information sent between the client and the server if possible. See Section 4.2.6, “Connection Compression Control”.

- `--concurrency=N`, `-c N`

  The number of parallel clients to simulate.

- `--create=value`

  The file or string containing the statement to use for creating the table.

- `--create-schema=value`

  The schema in which to run the tests.

  Note

  If the `--auto-generate-sql` option is also given, `mysqlslap` drops the schema at the end of the test run. To avoid this, use the `--no-drop` option as well.

- `--csv[=file_name]`

  Generate output in comma-separated values format. The output goes to the named file, or to the standard output if no file is given.

- `--debug[=debug_options]`, `-# [debug_options]`

Write a debugging log. A typical *debug_options* string is d:t:o,*file_name*. The default is d:t:o,/tmp/mysqlslap.trace.

This option is available only if MySQL was built using WITH_DEBUG. MySQL release binaries provided by Oracle are *not* built using this option.

- --debug-check

  Print some debugging information when the program exits.

  This option is available only if MySQL was built using WITH_DEBUG. MySQL release binaries provided by Oracle are *not* built using this option.

- --debug-info, -T

  Print debugging information and memory and CPU usage statistics when the program exits.

  This option is available only if MySQL was built using WITH_DEBUG. MySQL release binaries provided by Oracle are *not* built using this option.

- --default-auth=*plugin*

  A hint about which client-side authentication plugin to use. See Section 6.2.11, “Pluggable Authentication”.

- --defaults-extra-file=*file_name*

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- --defaults-file=*file_name*

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *file_name* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  Exception: Even with --defaults-file, client programs read .mylogin.cnf.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- --defaults-group-suffix=*str*

  Read not only the usual option groups, but also groups with the usual names and a suffix of *str*. For example, mysqlslap normally reads the [client] and [mysqlslap] groups. If the --defaults-group-suffix=_other option is given, mysqlslap also reads the [client_other] and [mysqlslap_other] groups.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- --delimiter=*str*, -F *str*

  The delimiter to use in SQL statements supplied in files or using command options.

- `--detach=N`

  Detach (close and reopen) each connection after each *`N`* statements. The default is 0 (connections are not detached).

- `--enable-cleartext-plugin`

  Enable the `mysql_clear_password` cleartext authentication plugin. (See Section 6.4.1.5, “Client-Side Cleartext Pluggable Authentication”.)

- `--engine=engine_name`, `-e engine_name`

  The storage engine to use for creating tables.

- `--host=host_name`, `-h host_name`

  Connect to the MySQL server on the given host.

- `--iterations=N`, `-i N`

  The number of times to run the tests.

- `--login-path=name`

  Read options from the named login path in the `.mylogin.cnf` login path file. A “login path” is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the `mysql_config_editor` utility. See Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--no-drop`

  Prevent `mysqlslap` from dropping any schema it creates during the test run.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

  The exception is that the `.mylogin.cnf` file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when `--no-defaults` is used. (`.mylogin.cnf` is created by the `mysql_config_editor` utility. See Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”.)

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--number-char-cols=N`, `-x N`

  The number of `VARCHAR` columns to use if `--auto-generate-sql` is specified.

- `--number-int-cols=N`, `-y N`

  The number of `INT` columns to use if `--auto-generate-sql` is specified.

- `--number-of-queries=N`

  Limit each client to approximately this many queries. Query counting takes into account the statement delimiter. For example, if you invoke `mysqlslap` as follows, the `;` delimiter is recognized so that each instance of the query string counts as two queries. As a result, 5 rows (not 10) are inserted.

```
shell> mysqlslap --delimiter=";" --number-of-queries=10
         --query="use test;insert into t values(null)"
```

- `--only-print`

  Do not connect to databases. `mysqlslap` only prints what it would have done.

- `--password[=password]`, `-p[password]`

  The password of the MySQL account used for connecting to the server. The password value is optional. If not given, `mysqlslap` prompts for one. If given, there must be *no space* between `--password=` or `-p` and the password following it. If no password option is specified, the default is to send no password.

  Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

  To explicitly specify that there is no password and that `mysqlslap` should not prompt for one, use the `--skip-password` option.

- `--pipe`, `-W`

  On Windows, connect to the server using a named pipe. This option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--plugin-dir=dir_name`

  The directory in which to look for plugins. Specify this option if the `--default-auth` option is used to specify an authentication plugin but `mysqlslap` does not find it. See Section 6.2.11, “Pluggable Authentication”.

- `--port=port_num`, `-P port_num`

  For TCP/IP connections, the port number to use.

- `--post-query=value`

  The file or string containing the statement to execute after the tests have completed. This execution is not counted for timing purposes.

- `--post-system=str`

  The string to execute using `system()` after the tests have completed. This execution is not counted for timing purposes.

- `--pre-query=value`

  The file or string containing the statement to execute before running the tests. This execution is not counted for timing purposes.

- `--pre-system=str`

  The string to execute using `system()` before running the tests. This execution is not counted for timing purposes.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--protocol={TCP|SOCKET|PIPE|MEMORY}`

  The transport protocol to use for connecting to the server. It is useful when the other connection parameters normally result in use of a protocol other than the one you want. For details on the permissible values, see Section 4.2.5, “Connection Transport Protocols”.

- `--query=value`, `-q value`

  The file or string containing the `SELECT` statement to use for retrieving data.

- `--secure-auth`

  Do not send passwords to the server in old (pre-4.1) format. This prevents connections except for servers that use the newer password format. This option is enabled by default; use `--skip-secure-auth` to disable it. This option was added in MySQL 5.6.17.

  Note

  Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated; expect support for them to be removed in a future MySQL release. For account upgrade instructions, see Section 6.4.1.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”.

  Note

  This option is deprecated; expect it to be removed in a future release. As of MySQL 5.7.5, it is always enabled and attempting to disable it produces an error.

- `--shared-memory-base-name=name`

  On Windows, the shared-memory name to use for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case-sensitive.

  This option applies only if the server was started with the `shared_memory` system variable enabled to support shared-memory connections.

- `--silent`, `-s`

  Silent mode. No output.

- `--socket=path`, `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

  On Windows, this option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Command Options for Encrypted Connections.

- `--user=user_name`, `-u user_name`

  The user name of the MySQL account to use for connecting to the server.

- `--verbose`, `-v`

  Verbose mode. Print more information about what the program does. This option can be used multiple times to increase the amount of information.

- `--version`, `-V`

  Display version information and exit.

4.6 Administrative and Utility Programs

This section describes administrative programs and programs that perform miscellaneous utility operations.

4.6.1 `innochecksum` — Offline InnoDB File Checksum Utility

`innochecksum` prints checksums for `InnoDB` files. This tool reads an `InnoDB` tablespace file, calculates the checksum for each page, compares the calculated checksum to the stored checksum, and reports mismatches, which indicate damaged pages. It was originally developed to speed up verifying the integrity of tablespace files after power outages but can also be used after file copies. Because checksum mismatches cause `InnoDB` to deliberately shut down a running server, it may be preferable to use this tool rather than waiting for an in-production server to encounter the damaged pages. As of MySQL 5.6.16, `innochecksum` supports files greater than 2GB in size. Previously, `innochecksum` only supported files up to 2GB in size.

`innochecksum` does not support tablespaces that contain compressed pages.

`innochecksum` cannot be used on tablespace files that the server already has open. For such files, you should use `CHECK TABLE` to check tables within the tablespace.

If checksum mismatches are found, you would normally restore the tablespace from backup or start the server and attempt to use `mysqldump` to make a backup of the tables within the tablespace.

Invoke `innochecksum` like this:

```
shell> innochecksum [options] file_name
```

`innochecksum` supports the following options. For options that refer to page numbers, the numbers are zero-based.

- `-c`

  Print a count of the number of pages in the file.

- `-d`

  Debug mode; prints checksums for each page.

- `-e num`

  End at this page number.

- `-p num`

  Check only this page number.

- `-s num`

  Start at this page number.

- `-v`

  Verbose mode; print a progress indicator every five seconds.

4.6.2 myisam_ftdump — Display Full-Text Index information

`myisam_ftdump` displays information about `FULLTEXT` indexes in `MyISAM` tables. It reads the `MyISAM` index file directly, so it must be run on the server host where the table is located. Before using `myisam_ftdump`, be sure to issue a `FLUSH TABLES` statement first if the server is running.

`myisam_ftdump` scans and dumps the entire index, which is not particularly fast. On the other hand, the distribution of words changes infrequently, so it need not be run often.

Invoke `myisam_ftdump` like this:

```
shell> myisam_ftdump [options] tbl_name index_num
```

The *`tbl_name`* argument should be the name of a `MyISAM` table. You can also specify a table by naming its index file (the file with the `.MYI` suffix). If you do not invoke `myisam_ftdump` in the directory where the table files are located, the table or index file name must be preceded by the path name to the table's database directory. Index numbers begin with 0.

Example: Suppose that the `test` database contains a table named `mytexttable` that has the following definition:

```
CREATE TABLE mytexttable
(
  id   INT NOT NULL,
  txt  TEXT NOT NULL,
  PRIMARY KEY (id),
  FULLTEXT (txt)
) ENGINE=MyISAM;
```

The index on `id` is index 0 and the `FULLTEXT` index on `txt` is index 1. If your working directory is the `test` database directory, invoke `myisam_ftdump` as follows:

```
shell> myisam_ftdump mytexttable 1
```

If the path name to the `test` database directory is `/usr/local/mysql/data/test`, you can also specify the table name argument using that path name. This is useful if you do not invoke `myisam_ftdump` in the database directory:

```
shell> myisam_ftdump /usr/local/mysql/data/test/mytexttable 1
```

You can use `myisam_ftdump` to generate a list of index entries in order of frequency of occurrence like this on Unix-like systems:

```
shell> myisam_ftdump -c mytexttable 1 | sort -r
```

On Windows, use:

```
shell> myisam_ftdump -c mytexttable 1 | sort /R
```

`myisam_ftdump` supports the following options:

- `--help`, `-h -?`

  Display a help message and exit.

- `--count`, `-c`

  Calculate per-word statistics (counts and global weights).

- `--dump`, `-d`

  Dump the index, including data offsets and word weights.

- `--length`, `-l`

  Report the length distribution.

- `--stats`, `-s`

  Report global index statistics. This is the default operation if no other operation is specified.

- `--verbose`, `-v`

  Verbose mode. Print more output about what the program does.

4.6.3 myisamchk — MyISAM Table-Maintenance Utility

The `myisamchk` utility gets information about your database tables or checks, repairs, or optimizes them. `myisamchk` works with `MyISAM` tables (tables that have `.MYD` and `.MYI` files for storing data and indexes).

You can also use the `CHECK TABLE` and `REPAIR TABLE` statements to check and repair `MyISAM` tables. See Section 13.7.2.2, “CHECK TABLE Statement”, and Section 13.7.2.5, “REPAIR TABLE Statement”.

The use of `myisamchk` with partitioned tables is not supported.

Caution

It is best to make a backup of a table before performing a table repair operation; under some circumstances the operation might cause data loss. Possible causes include but are not limited to file system errors.

Invoke `myisamchk` like this:

```
shell> myisamchk [options] tbl_name ...
```

The *`options`* specify what you want `myisamchk` to do. They are described in the following sections. You can also get a list of options by invoking `myisamchk --help`.

With no options, `myisamchk` simply checks your table as the default operation. To get more information or to tell `myisamchk` to take corrective action, specify options as described in the following discussion.

`tbl_name` is the database table you want to check or repair. If you run `myisamchk` somewhere other than in the database directory, you must specify the path to the database directory, because `myisamchk` has no idea where the database is located. In fact, `myisamchk` does not actually care whether the files you are working on are located in a database directory. You can copy the files that correspond to a database table into some other location and perform recovery operations on them there.

You can name several tables on the `myisamchk` command line if you wish. You can also specify a table by naming its index file (the file with the `.MYI` suffix). This enables you to specify all tables in a directory by using the pattern `*.MYI`. For example, if you are in a database directory, you can check all the `MyISAM` tables in that directory like this:

```
shell> myisamchk *.MYI
```

If you are not in the database directory, you can check all the tables there by specifying the path to the directory:

```
shell> myisamchk /path/to/database_dir/*.MYI
```

You can even check all tables in all databases by specifying a wildcard with the path to the MySQL data directory:

```
shell> myisamchk /path/to/datadir/*/*.MYI
```

The recommended way to quickly check all MyISAM tables is:

```
shell> myisamchk --silent --fast /path/to/datadir/*/*.MYI
```

If you want to check all MyISAM tables and repair any that are corrupted, you can use the following command:

```
shell> myisamchk --silent --force --fast --update-state \
          --key_buffer_size=64M --myisam_sort_buffer_size=64M \
          --read_buffer_size=1M --write_buffer_size=1M \
          /path/to/datadir/*/*.MYI
```

This command assumes that you have more than 64MB free. For more information about memory allocation with myisamchk, see Section 4.6.3.6, “myisamchk Memory Usage”.

For additional information about using myisamchk, see Section 7.6, “MyISAM Table Maintenance and Crash Recovery”.

Important

You must ensure that no other program is using the tables while you are running myisamchk. The most effective means of doing so is to shut down the MySQL server while running myisamchk, or to lock all tables that myisamchk is being used on.

Otherwise, when you run myisamchk, it may display the following error message:

```
warning: clients are using or haven't closed the table properly
```

This means that you are trying to check a table that has been updated by another program (such as the mysqld server) that hasn't yet closed the file or that has died without closing the file properly, which can sometimes lead to the corruption of one or more MyISAM tables.

If mysqld is running, you must force it to flush any table modifications that are still buffered in memory by using FLUSH TABLES. You should then ensure that no one is using the tables while you are running myisamchk

However, the easiest way to avoid this problem is to use CHECK TABLE instead of myisamchk to check tables. See Section 13.7.2.2, “CHECK TABLE Statement”.

myisamchk supports the following options, which can be specified on the command line or in the [myisamchk] group of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.17 myisamchk Options

Option Name	Description	Deprecated
--analyze	Analyze the distribution of key values	
--backup	Make a backup of the .MYD file as file_name-time.BAK	
--block-search	Find the record that a block at the given offset belongs to	
--check	Check the table for errors	
--check-only-changed	Check only tables that have changed since the last check	

Option Name	Description	Deprecated
--correct-checksum	Correct the checksum information for the table	
--data-file-length	Maximum length of the data file (when re-creating data file when it is full)	
--debug	Write debugging log	
--decode_bits	Decode_bits	
--defaults-extra-file	Read named option file in addition to usual option files	
--defaults-file	Read only named option file	
--defaults-group-suffix	Option group suffix value	
--description	Print some descriptive information about the table	
--extend-check	Do very thorough table check or repair that tries to recover every possible row from the data file	
--fast	Check only tables that haven't been closed properly	
--force	Do a repair operation automatically if myisamchk finds any errors in the table	
--force	Overwrite old temporary files. For use with the -r or -o option	
--ft_max_word_len	Maximum word length for FULLTEXT indexes	
--ft_min_word_len	Minimum word length for FULLTEXT indexes	
--ft_stopword_file	Use stopwords from this file instead of built-in list	
--HELP	Display help message and exit	
--help	Display help message and exit	
--information	Print informational statistics about the table that is checked	
--key_buffer_size	Size of buffer used for index blocks for MyISAM tables	
--keys-used	A bit-value that indicates which indexes to update	
--max-record-length	Skip rows larger than the given length if myisamchk cannot allocate memory to hold them	
--medium-check	Do a check that is faster than an --extend-check operation	
--myisam_block_size	Block size to be used for MyISAM index pages	
--myisam_sort_buffer_size	The buffer that is allocated when sorting the index when doing a REPAIR or when creating indexes with CREATE INDEX or ALTER TABLE	
--no-defaults	Read no option files	
--parallel-recover	Uses the same technique as -r and -n, but creates all the keys in parallel, using different threads (beta)	
--print-defaults	Print default options	
--quick	Achieve a faster repair by not modifying the data file	
--read_buffer_size	Each thread that does a sequential scan allocates a buffer of this size for each table it scans	
--read-only	Do not mark the table as checked	
--recover	Do a repair that can fix almost any problem except unique keys that aren't unique	

Option Name	Description	Deprecated
--safe-recover	Do a repair using an old recovery method that reads through all rows in order and updates all index trees based on the rows found	
--set-auto-increment	Force AUTO_INCREMENT numbering for new records to start at the given value	
--set-collation	Specify the collation to use for sorting table indexes	
--silent	Silent mode	
--sort_buffer_size	The buffer that is allocated when sorting the index when doing a REPAIR or when creating indexes with CREATE INDEX or ALTER TABLE	Yes
--sort-index	Sort the index tree blocks in high-low order	
--sort_key_blocks	sort_key_blocks	
--sort-records	Sort records according to a particular index	
--sort-recover	Force myisamchk to use sorting to resolve the keys even if the temporary files would be very large	
--stats_method	Specifies how MyISAM index statistics collection code should treat NULLs	
--tmpdir	Directory to be used for storing temporary files	
--unpack	Unpack a table that was packed with myisampack	
--update-state	Store information in the .MYI file to indicate when the table was checked and whether the table crashed	
--verbose	Verbose mode	
--version	Display version information and exit	
--write_buffer_size	Write buffer size	

4.6.3.1 myisamchk General Options

The options described in this section can be used for any type of table maintenance operation performed by `myisamchk`. The sections following this one describe options that pertain only to specific operations, such as table checking or repairing.

- `--help`, `-?`

  Display a help message and exit. Options are grouped by type of operation.

- `--HELP`, `-H`

  Display a help message and exit. Options are presented in a single list.

- `--debug=`*`debug_options`*, `-# `*`debug_options`*

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,`*`file_name`*. The default is `d:t:o,/tmp/myisamchk.trace`.

  This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are *not* built using this option.

- `--defaults-extra-file=`*`file_name`*

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `myisamchk` normally reads the `[myisamchk]` group. If the `--defaults-group-suffix=_other` option is given, `myisamchk` also reads the `[myisamchk_other]` group.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--no-defaults`

  Do not read any option files. If program startup fails due to reading unknown options from an option file, `--no-defaults` can be used to prevent them from being read.

  The exception is that the `.mylogin.cnf` file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when `--no-defaults` is used. (`.mylogin.cnf` is created by the `mysql_config_editor` utility. See Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”.)

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--print-defaults`

  Print the program name and all options that it gets from option files.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--silent`, `-s`

  Silent mode. Write output only when errors occur. You can use `-s` twice (`-ss`) to make `myisamchk` very silent.

- `--verbose`, `-v`

  Verbose mode. Print more information about what the program does. This can be used with `-d` and `-e`. Use `-v` multiple times (`-vv`, `-vvv`) for even more output.

- `--version`, `-V`

  Display version information and exit.

- `--wait`, `-w`

  Instead of terminating with an error if the table is locked, wait until the table is unlocked before continuing. If you are running `mysqld` with external locking disabled, the table can be locked only by another `myisamchk` command.

You can also set the following variables by using `--var_name=value` syntax:

Variable	Default Value
decode_bits	9
ft_max_word_len	version-dependent
ft_min_word_len	4
ft_stopword_file	built-in list
key_buffer_size	523264
myisam_block_size	1024
myisam_sort_key_blocks	16
read_buffer_size	262136
sort_buffer_size	2097144
sort_key_blocks	16
stats_method	nulls_unequal
write_buffer_size	262136

The possible myisamchk variables and their default values can be examined with myisamchk --help:

sort_buffer_size is used when the keys are repaired by sorting keys, which is the normal case when you use --recover. myisam_sort_buffer_size is available as an alternative name to sort_buffer_size. myisam_sort_buffer_size is preferable to sort_buffer_size because its name corresponds to the myisam_sort_buffer_size server system variable that has a similar meaning. sort_buffer_size should be considered deprecated.

key_buffer_size is used when you are checking the table with --extend-check or when the keys are repaired by inserting keys row by row into the table (like when doing normal inserts). Repairing through the key buffer is used in the following cases:

- You use --safe-recover.
- The temporary files needed to sort the keys would be more than twice as big as when creating the key file directly. This is often the case when you have large key values for CHAR, VARCHAR, or TEXT columns, because the sort operation needs to store the complete key values as it proceeds. If you have lots of temporary space and you can force myisamchk to repair by sorting, you can use the --sort-recover option.

Repairing through the key buffer takes much less disk space than using sorting, but is also much slower.

If you want a faster repair, set the key_buffer_size and myisam_sort_buffer_size variables to about 25% of your available memory. You can set both variables to large values, because only one of them is used at a time.

myisam_block_size is the size used for index blocks.

stats_method influences how NULL values are treated for index statistics collection when the --analyze option is given. It acts like the myisam_stats_method system variable. For more information, see the description of myisam_stats_method in Section 5.1.7, “Server System Variables”, and Section 8.3.7, “InnoDB and MyISAM Index Statistics Collection”.

ft_min_word_len and ft_max_word_len indicate the minimum and maximum word length for FULLTEXT indexes on MyISAM tables. ft_stopword_file names the stopword file. These need to be set under the following circumstances.

If you use myisamchk to perform an operation that modifies table indexes (such as repair or analyze), the FULLTEXT indexes are rebuilt using the default full-text parameter values for minimum and

maximum word length and the stopword file unless you specify otherwise. This can result in queries failing.

The problem occurs because these parameters are known only by the server. They are not stored in MyISAM index files. To avoid the problem if you have modified the minimum or maximum word length or the stopword file in the server, specify the same ft_min_word_len, ft_max_word_len, and ft_stopword_file values to myisamchk that you use for mysqld. For example, if you have set the minimum word length to 3, you can repair a table with myisamchk like this:

```
shell> myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

To ensure that myisamchk and the server use the same values for full-text parameters, you can place each one in both the [mysqld] and [myisamchk] sections of an option file:

```
[mysqld]
ft_min_word_len=3

[myisamchk]
ft_min_word_len=3
```

An alternative to using myisamchk is to use the REPAIR TABLE, ANALYZE TABLE, OPTIMIZE TABLE, or ALTER TABLE. These statements are performed by the server, which knows the proper full-text parameter values to use.

4.6.3.2 myisamchk Check Options

myisamchk supports the following options for table checking operations:

- --check, -c

  Check the table for errors. This is the default operation if you specify no option that selects an operation type explicitly.

- --check-only-changed, -C

  Check only tables that have changed since the last check.

- --extend-check, -e

  Check the table very thoroughly. This is quite slow if the table has many indexes. This option should only be used in extreme cases. Normally, myisamchk or myisamchk --medium-check should be able to determine whether there are any errors in the table.

  If you are using --extend-check and have plenty of memory, setting the key_buffer_size variable to a large value helps the repair operation run faster.

  See also the description of this option under table repair options.

  For a description of the output format, see Section 4.6.3.5, “Obtaining Table Information with myisamchk”.

- --fast, -F

  Check only tables that haven't been closed properly.

- --force, -f

  Do a repair operation automatically if myisamchk finds any errors in the table. The repair type is the same as that specified with the --recover or -r option.

- --information, -i

  Print informational statistics about the table that is checked.

- --medium-check, -m

Do a check that is faster than an `--extend-check` operation. This finds only 99.99% of all errors, which should be good enough in most cases.

- `--read-only`, `-T`

  Do not mark the table as checked. This is useful if you use `myisamchk` to check a table that is in use by some other application that does not use locking, such as `mysqld` when run with external locking disabled.

- `--update-state`, `-U`

  Store information in the `.MYI` file to indicate when the table was checked and whether the table crashed. This should be used to get full benefit of the `--check-only-changed` option, but you shouldn't use this option if the `mysqld` server is using the table and you are running it with external locking disabled.

4.6.3.3 myisamchk Repair Options

`myisamchk` supports the following options for table repair operations (operations performed when an option such as `--recover` or `--safe-recover` is given):

- `--backup`, `-B`

  Make a backup of the `.MYD` file as *`file_name-time`*`.BAK`

- `--character-sets-dir=`*`dir_name`*

  The directory where character sets are installed. See Section 10.15, “Character Set Configuration”.

- `--correct-checksum`

  Correct the checksum information for the table.

- `--data-file-length=`*`len`*, `-D` *`len`*

  The maximum length of the data file (when re-creating data file when it is “full”).

- `--extend-check`, `-e`

  Do a repair that tries to recover every possible row from the data file. Normally, this also finds a lot of garbage rows. Do not use this option unless you are desperate.

  See also the description of this option under table checking options.

  For a description of the output format, see Section 4.6.3.5, “Obtaining Table Information with myisamchk”.

- `--force`, `-f`

  Overwrite old intermediate files (files with names like *`tbl_name`*`.TMD`) instead of aborting.

- `--keys-used=`*`val`*, `-k` *`val`*

  For `myisamchk`, the option value is a bit value that indicates which indexes to update. Each binary bit of the option value corresponds to a table index, where the first index is bit 0. An option value of 0 disables updates to all indexes, which can be used to get faster inserts. Deactivated indexes can be reactivated by using `myisamchk -r`.

- `--no-symlinks`, `-l`

  Do not follow symbolic links. Normally `myisamchk` repairs the table that a symlink points to. This option does not exist as of MySQL 4.0 because versions from 4.0 on do not remove symlinks during repair operations.

- `--max-record-length=len`

  Skip rows larger than the given length if `myisamchk` cannot allocate memory to hold them.

- `--parallel-recover`, `-p`

  Use the same technique as `-r` and `-n`, but create all the keys in parallel, using different threads. *This is beta-quality code. Use at your own risk!*

- `--quick`, `-q`

  Achieve a faster repair by modifying only the index file, not the data file. You can specify this option twice to force `myisamchk` to modify the original data file in case of duplicate keys.

- `--recover`, `-r`

  Do a repair that can fix almost any problem except unique keys that are not unique (which is an extremely unlikely error with `MyISAM` tables). If you want to recover a table, this is the option to try first. You should try `--safe-recover` only if `myisamchk` reports that the table cannot be recovered using `--recover`. (In the unlikely case that `--recover` fails, the data file remains intact.)

  If you have lots of memory, you should increase the value of `myisam_sort_buffer_size`.

- `--safe-recover`, `-o`

  Do a repair using an old recovery method that reads through all rows in order and updates all index trees based on the rows found. This is an order of magnitude slower than `--recover`, but can handle a couple of very unlikely cases that `--recover` cannot. This recovery method also uses much less disk space than `--recover`. Normally, you should repair first using `--recover`, and then with `--safe-recover` only if `--recover` fails.

  If you have lots of memory, you should increase the value of `key_buffer_size`.

- `--set-collation=name`

  Specify the collation to use for sorting table indexes. The character set name is implied by the first part of the collation name.

- `--sort-recover`, `-n`

  Force `myisamchk` to use sorting to resolve the keys even if the temporary files would be very large.

- `--tmpdir=dir_name`, `-t dir_name`

  The path of the directory to be used for storing temporary files. If this is not set, `myisamchk` uses the value of the `TMPDIR` environment variable. `--tmpdir` can be set to a list of directory paths that are used successively in round-robin fashion for creating temporary files. The separator character between directory names is the colon (`:`) on Unix and the semicolon (`;`) on Windows.

- `--unpack`, `-u`

  Unpack a table that was packed with `myisampack`.

4.6.3.4 Other myisamchk Options

`myisamchk` supports the following options for actions other than table checks and repairs:

- `--analyze`, `-a`

  Analyze the distribution of key values. This improves join performance by enabling the join optimizer to better choose the order in which to join the tables and which indexes it should use. To obtain information about the key distribution, use a `myisamchk --description --verbose tbl_name` command or the `SHOW INDEX FROM tbl_name` statement.

- `--block-search=offset`, `-b offset`

  Find the record that a block at the given offset belongs to.

- `--description`, `-d`

  Print some descriptive information about the table. Specifying the `--verbose` option once or twice produces additional information. See Section 4.6.3.5, “Obtaining Table Information with myisamchk”.

- `--set-auto-increment[=value]`, `-A[value]`

  Force `AUTO_INCREMENT` numbering for new records to start at the given value (or higher, if there are existing records with `AUTO_INCREMENT` values this large). If *`value`* is not specified, `AUTO_INCREMENT` numbers for new records begin with the largest value currently in the table, plus one.

- `--sort-index`, `-S`

  Sort the index tree blocks in high-low order. This optimizes seeks and makes table scans that use indexes faster.

- `--sort-records=N`, `-R N`

  Sort records according to a particular index. This makes your data much more localized and may speed up range-based `SELECT` and `ORDER BY` operations that use this index. (The first time you use this option to sort a table, it may be very slow.) To determine a table's index numbers, use `SHOW INDEX`, which displays a table's indexes in the same order that `myisamchk` sees them. Indexes are numbered beginning with 1.

  If keys are not packed (`PACK_KEYS=0`), they have the same length, so when `myisamchk` sorts and moves records, it just overwrites record offsets in the index. If keys are packed (`PACK_KEYS=1`), `myisamchk` must unpack key blocks first, then re-create indexes and pack the key blocks again. (In this case, re-creating indexes is faster than updating offsets for each index.)

4.6.3.5 Obtaining Table Information with myisamchk

To obtain a description of a `MyISAM` table or statistics about it, use the commands shown here. The output from these commands is explained later in this section.

- `myisamchk -d tbl_name`

  Runs `myisamchk` in “describe mode” to produce a description of your table. If you start the MySQL server with external locking disabled, `myisamchk` may report an error for a table that is updated while it runs. However, because `myisamchk` does not change the table in describe mode, there is no risk of destroying data.

- `myisamchk -dv tbl_name`

  Adding `-v` runs `myisamchk` in verbose mode so that it produces more information about the table. Adding `-v` a second time produces even more information.

- `myisamchk -eis tbl_name`

  Shows only the most important information from a table. This operation is slow because it must read the entire table.

- `myisamchk -eiv tbl_name`

  This is like `-eis`, but tells you what is being done.

The *`tbl_name`* argument can be either the name of a `MyISAM` table or the name of its index file, as described in Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”. Multiple *`tbl_name`* arguments can be given.

Suppose that a table named person has the following structure. (The MAX_ROWS table option is included so that in the example output from myisamchk shown later, some values are smaller and fit the output format more easily.)

```
CREATE TABLE person
(
  id         INT NOT NULL AUTO_INCREMENT,
  last_name  VARCHAR(20) NOT NULL,
  first_name VARCHAR(20) NOT NULL,
  birth      DATE,
  death      DATE,
  PRIMARY KEY (id),
  INDEX (last_name, first_name),
  INDEX (birth)
) MAX_ROWS = 1000000 ENGINE=MYISAM;
```

Suppose also that the table has these data and index file sizes:

```
-rw-rw----  1 mysql  mysql  9347072 Aug 19 11:47 person.MYD
-rw-rw----  1 mysql  mysql  6066176 Aug 19 11:47 person.MYI
```

Example of myisamchk -dvv output:

```
MyISAM file:         person
Record format:       Packed
Character set:       latin1_swedish_ci (8)
File-version:        1
Creation time:       2009-08-19 16:47:41
Recover time:        2009-08-19 16:47:56
Status:              checked,analyzed,optimized keys
Auto increment key:              1  Last value:                306688
Data records:               306688  Deleted blocks:                 0
Datafile parts:             306688  Deleted data:                   0
Datafile pointer (bytes):        4  Keyfile pointer (bytes):        3
Datafile length:           9347072  Keyfile length:           6066176
Max datafile length:    4294967294  Max keyfile length:   17179868159
Recordlength:                   54

table description:
Key Start Len Index   Type                 Rec/key         Root  Blocksize
1   2     4   unique  long                       1        99328       1024
2   6     20  multip. varchar prefix           512      3563520       1024
    27    20          varchar                  512
3   48    3   multip. uint24 NULL           306688      6065152       1024

Field Start Length Nullpos Nullbit Type
1     1     1
2     2     4                      no zeros
3     6     21                     varchar
4     27    21                     varchar
5     48    3      1       1       no zeros
6     51    3      1       2       no zeros
```

Explanations for the types of information myisamchk produces are given here. “Keyfile” refers to the index file. “Record” and “row” are synonymous, as are “field” and “column.”

The initial part of the table description contains these values:

- MyISAM file

  Name of the MyISAM (index) file.

- Record format

  The format used to store table rows. The preceding examples use Fixed length. Other possible values are Compressed and Packed. (Packed corresponds to what SHOW TABLE STATUS reports as Dynamic.)

- Chararacter set

The table default character set.

- `File-version`

  Version of `MyISAM` format. Always 1.

- `Creation time`

  When the data file was created.

- `Recover time`

  When the index/data file was last reconstructed.

- `Status`

  Table status flags. Possible values are `crashed`, `open`, `changed`, `analyzed`, `optimized keys`, and `sorted index pages`.

- `Auto increment key`, `Last value`

  The key number associated the table's `AUTO_INCREMENT` column, and the most recently generated value for this column. These fields do not appear if there is no such column.

- `Data records`

  The number of rows in the table.

- `Deleted blocks`

  How many deleted blocks still have reserved space. You can optimize your table to minimize this space. See Section 7.6.4, “MyISAM Table Optimization”.

- `Datafile parts`

  For dynamic-row format, this indicates how many data blocks there are. For an optimized table without fragmented rows, this is the same as `Data records`.

- `Deleted data`

  How many bytes of unreclaimed deleted data there are. You can optimize your table to minimize this space. See Section 7.6.4, “MyISAM Table Optimization”.

- `Datafile pointer`

  The size of the data file pointer, in bytes. It is usually 2, 3, 4, or 5 bytes. Most tables manage with 2 bytes, but this cannot be controlled from MySQL yet. For fixed tables, this is a row address. For dynamic tables, this is a byte address.

- `Keyfile pointer`

  The size of the index file pointer, in bytes. It is usually 1, 2, or 3 bytes. Most tables manage with 2 bytes, but this is calculated automatically by MySQL. It is always a block address.

- `Max datafile length`

  How long the table data file can become, in bytes.

- `Max keyfile length`

  How long the table index file can become, in bytes.

- `Recordlength`

How much space each row takes, in bytes.

The `table description` part of the output includes a list of all keys in the table. For each key, `myisamchk` displays some low-level information:

- `Key`

  This key's number. This value is shown only for the first column of the key. If this value is missing, the line corresponds to the second or later column of a multiple-column key. For the table shown in the example, there are two `table description` lines for the second index. This indicates that it is a multiple-part index with two parts.

- `Start`

  Where in the row this portion of the index starts.

- `Len`

  How long this portion of the index is. For packed numbers, this should always be the full length of the column. For strings, it may be shorter than the full length of the indexed column, because you can index a prefix of a string column. The total length of a multiple-part key is the sum of the `Len` values for all key parts.

- `Index`

  Whether a key value can exist multiple times in the index. Possible values are `unique` or `multip.` (multiple).

- `Type`

  What data type this portion of the index has. This is a `MyISAM` data type with the possible values `packed`, `stripped`, or `empty`.

- `Root`

  Address of the root index block.

- `Blocksize`

  The size of each index block. By default this is 1024, but the value may be changed at compile time when MySQL is built from source.

- `Rec/key`

  This is a statistical value used by the optimizer. It tells how many rows there are per value for this index. A unique index always has a value of 1. This may be updated after a table is loaded (or greatly changed) with `myisamchk -a`. If this is not updated at all, a default value of 30 is given.

The last part of the output provides information about each column:

- `Field`

  The column number.

- `Start`

  The byte position of the column within table rows.

- `Length`

  The length of the column in bytes.

- `Nullpos`, `Nullbit`

For columns that can be NULL, MyISAM stores NULL values as a flag in a byte. Depending on how many nullable columns there are, there can be one or more bytes used for this purpose. The Nullpos and Nullbit values, if nonempty, indicate which byte and bit contains that flag indicating whether the column is NULL.

The position and number of bytes used to store NULL flags is shown in the line for field 1. This is why there are six Field lines for the person table even though it has only five columns.

- Type

  The data type. The value may contain any of the following descriptors:

  - constant

    All rows have the same value.

  - no endspace

    Do not store endspace.

  - no endspace, not_always

    Do not store endspace and do not do endspace compression for all values.

  - no endspace, no empty

    Do not store endspace. Do not store empty values.

  - table-lookup

    The column was converted to an ENUM.

  - zerofill(*N*)

    The most significant *N* bytes in the value are always 0 and are not stored.

  - no zeros

    Do not store zeros.

  - always zero

    Zero values are stored using one bit.

- Huff tree

  The number of the Huffman tree associated with the column.

- Bits

  The number of bits used in the Huffman tree.

The Huff tree and Bits fields are displayed if the table has been compressed with myisampack. See Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”, for an example of this information.

Example of myisamchk -eiv output:

```
Checking MyISAM file: person
Data records:  306688   Deleted blocks:       0
- check file-size
- check record delete-chain
No recordlinks
```

```
- check key delete-chain
block_size 1024:
- check index reference
- check data record references index: 1
Key:  1:  Keyblocks used:  98%  Packed:    0%  Max levels:  3
- check data record references index: 2
Key:  2:  Keyblocks used:  99%  Packed:   97%  Max levels:  3
- check data record references index: 3
Key:  3:  Keyblocks used:  98%  Packed:  -14%  Max levels:  3
Total:    Keyblocks used:  98%  Packed:   89%

- check records and index references
*** LOTS OF ROW NUMBERS DELETED ***

Records:            306688  M.recordlength:       25  Packed:            83%
Recordspace used:       97% Empty space:           2% Blocks/Record:   1.00
Record blocks:      306688  Delete blocks:         0
Record data:       7934464  Deleted data:          0
Lost space:         256512  Linkdata:        1156096

User time 43.08, System time 1.68
Maximum resident set size 0, Integral resident set size 0
Non-physical pagefaults 0, Physical pagefaults 0, Swaps 0
Blocks in 0 out 7, Messages in 0 out 0, Signals 0
Voluntary context switches 0, Involuntary context switches 0
Maximum memory usage: 1046926 bytes (1023k)
```

`myisamchk -eiv` output includes the following information:

- `Data records`

  The number of rows in the table.

- `Deleted blocks`

  How many deleted blocks still have reserved space. You can optimize your table to minimize this space. See Section 7.6.4, “MyISAM Table Optimization”.

- `Key`

  The key number.

- `Keyblocks used`

  What percentage of the keyblocks are used. When a table has just been reorganized with `myisamchk`, the values are very high (very near theoretical maximum).

- `Packed`

  MySQL tries to pack key values that have a common suffix. This can only be used for indexes on `CHAR` and `VARCHAR` columns. For long indexed strings that have similar leftmost parts, this can significantly reduce the space used. In the preceding example, the second key is 40 bytes long and a 97% reduction in space is achieved.

- `Max levels`

  How deep the B-tree for this key is. Large tables with long key values get high values.

- `Records`

  How many rows are in the table.

- `M.recordlength`

  The average row length. This is the exact row length for tables with fixed-length rows, because all rows have the same length.

- `Packed`

  MySQL strips spaces from the end of strings. The `Packed` value indicates the percentage of savings achieved by doing this.

- `Recordspace used`

  What percentage of the data file is used.

- `Empty space`

  What percentage of the data file is unused.

- `Blocks/Record`

  Average number of blocks per row (that is, how many links a fragmented row is composed of). This is always 1.0 for fixed-format tables. This value should stay as close to 1.0 as possible. If it gets too large, you can reorganize the table. See Section 7.6.4, “MyISAM Table Optimization”.

- `Recordblocks`

  How many blocks (links) are used. For fixed-format tables, this is the same as the number of rows.

- `Deleteblocks`

  How many blocks (links) are deleted.

- `Recorddata`

  How many bytes in the data file are used.

- `Deleted data`

  How many bytes in the data file are deleted (unused).

- `Lost space`

  If a row is updated to a shorter length, some space is lost. This is the sum of all such losses, in bytes.

- `Linkdata`

  When the dynamic table format is used, row fragments are linked with pointers (4 to 7 bytes each). `Linkdata` is the sum of the amount of storage used by all such pointers.

4.6.3.6 myisamchk Memory Usage

Memory allocation is important when you run `myisamchk`. `myisamchk` uses no more memory than its memory-related variables are set to. If you are going to use `myisamchk` on very large tables, you should first decide how much memory you want it to use. The default is to use only about 3MB to perform repairs. By using larger values, you can get `myisamchk` to operate faster. For example, if you have more than 512MB RAM available, you could use options such as these (in addition to any other options you might specify):

```
shell> myisamchk --myisam_sort_buffer_size=256M \
           --key_buffer_size=512M \
           --read_buffer_size=64M \
           --write_buffer_size=64M ...
```

Using `--myisam_sort_buffer_size=16M` is probably enough for most cases.

Be aware that `myisamchk` uses temporary files in `TMPDIR`. If `TMPDIR` points to a memory file system, out of memory errors can easily occur. If this happens, run `myisamchk` with the `--tmpdir=dir_name` option to specify a directory located on a file system that has more space.

When performing repair operations, myisamchk also needs a lot of disk space:

- Twice the size of the data file (the original file and a copy). This space is not needed if you do a repair with --quick; in this case, only the index file is re-created. *This space must be available on the same file system as the original data file*, as the copy is created in the same directory as the original.
- Space for the new index file that replaces the old one. The old index file is truncated at the start of the repair operation, so you usually ignore this space. This space must be available on the same file system as the original data file.
- When using --recover or --sort-recover (but not when using --safe-recover), you need space on disk for sorting. This space is allocated in the temporary directory (specified by TMPDIR or --tmpdir=*dir_name*). The following formula yields the amount of space required:

  ```
  (largest_key + row_pointer_length) * number_of_rows * 2
  ```

  You can check the length of the keys and the *row_pointer_length* with myisamchk -dv *tbl_name* (see Section 4.6.3.5, "Obtaining Table Information with myisamchk"). The *row_pointer_length* and *number_of_rows* values are the Datafile pointer and Data records values in the table description. To determine the *largest_key* value, check the Key lines in the table description. The Len column indicates the number of bytes for each key part. For a multiple-column index, the key size is the sum of the Len values for all key parts.

If you have a problem with disk space during repair, you can try --safe-recover instead of --recover.

4.6.4 myisamlog — Display MyISAM Log File Contents

myisamlog processes the contents of a MyISAM log file. To create such a file, start the server with a --log-isam=log_file option.

Invoke myisamlog like this:

```
shell> myisamlog [options] [file_name [tbl_name] ...]
```

The default operation is update (-u). If a recovery is done (-r), all writes and possibly updates and deletes are done and errors are only counted. The default log file name is myisam.log if no *log_file* argument is given. If tables are named on the command line, only those tables are updated.

myisamlog supports the following options:

- -?, -I

  Display a help message and exit.

- -c *N*

  Execute only *N* commands.

- -f *N*

  Specify the maximum number of open files.

- -F *filepath/*

  Specify the file path with a trailing slash.

- -i

  Display extra information before exiting.

- -o *offset*

Specify the starting offset.

- `-p N`

  Remove *N* components from path.

- `-r`

  Perform a recovery operation.

- `-R record_pos_file record_pos`

  Specify record position file and record position.

- `-u`

  Perform an update operation.

- `-v`

  Verbose mode. Print more output about what the program does. This option can be given multiple times to produce more and more output.

- `-w write_file`

  Specify the write file.

- `-V`

  Display version information.

4.6.5 myisampack — Generate Compressed, Read-Only MyISAM Tables

The `myisampack` utility compresses `MyISAM` tables. `myisampack` works by compressing each column in the table separately. Usually, `myisampack` packs the data file 40% to 70%.

When the table is used later, the server reads into memory the information needed to decompress columns. This results in much better performance when accessing individual rows, because you only have to uncompress exactly one row.

MySQL uses `mmap()` when possible to perform memory mapping on compressed tables. If `mmap()` does not work, MySQL falls back to normal read/write file operations.

Please note the following:

- If the `mysqld` server was invoked with external locking disabled, it is not a good idea to invoke `myisampack` if the table might be updated by the server during the packing process. It is safest to compress tables with the server stopped.
- After packing a table, it becomes read only. This is generally intended (such as when accessing packed tables on a CD).
- `myisampack` does not support partitioned tables.

Invoke `myisampack` like this:

```
shell> myisampack [options] file_name ...
```

Each file name argument should be the name of an index (`.MYI`) file. If you are not in the database directory, you should specify the path name to the file. It is permissible to omit the `.MYI` extension.

After you compress a table with `myisampack`, use `myisamchk -rq` to rebuild its indexes. Section 4.6.3, “myisamchk — MyISAM Table-Maintenance Utility”.

myisampack supports the following options. It also reads option files and supports the options for processing them described at Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- --help, -?

  Display a help message and exit.

- --backup, -b

  Make a backup of each table's data file using the name *tbl_name*.OLD.

- --character-sets-dir=*dir_name*

  The directory where character sets are installed. See Section 10.15, “Character Set Configuration”.

- --debug[=*debug_options*], -# [*debug_options*]

  Write a debugging log. A typical *debug_options* string is d:t:o,*file_name*. The default is d:t:o.

  This option is available only if MySQL was built using WITH_DEBUG. MySQL release binaries provided by Oracle are *not* built using this option.

- --force, -f

  Produce a packed table even if it becomes larger than the original or if the intermediate file from an earlier invocation of myisampack exists. (myisampack creates an intermediate file named *tbl_name*.TMD in the database directory while it compresses the table. If you kill myisampack, the .TMD file might not be deleted.) Normally, myisampack exits with an error if it finds that *tbl_name*.TMD exists. With --force, myisampack packs the table anyway.

- --join=*big_tbl_name*, -j *big_tbl_name*

  Join all tables named on the command line into a single packed table *big_tbl_name*. All tables that are to be combined *must* have identical structure (same column names and types, same indexes, and so forth).

  big_tbl_name must not exist prior to the join operation. All source tables named on the command line to be merged into *big_tbl_name* must exist. The source tables are read for the join operation but not modified.

- --silent, -s

  Silent mode. Write output only when errors occur.

- --test, -t

  Do not actually pack the table, just test packing it.

- --tmpdir=*dir_name*, -T *dir_name*

  Use the named directory as the location where myisampack creates temporary files.

- --verbose, -v

  Verbose mode. Write information about the progress of the packing operation and its result.

- --version, -V

  Display version information and exit.

- --wait, -w

Wait and retry if the table is in use. If the `mysqld` server was invoked with external locking disabled, it is not a good idea to invoke `myisampack` if the table might be updated by the server during the packing process.

The following sequence of commands illustrates a typical table compression session:

```
shell> ls -l station.*
-rw-rw-r--   1 jones    my         994128 Apr 17 19:00 station.MYD
-rw-rw-r--   1 jones    my          53248 Apr 17 19:00 station.MYI
-rw-rw-r--   1 jones    my           5767 Apr 17 19:00 station.frm

shell> myisamchk -dvv station

MyISAM file:     station
Isam-version:  2
Creation time: 1996-03-13 10:08:58
Recover time:  1997-02-02  3:06:43
Data records:              1192  Deleted blocks:              0
Datafile parts:            1192  Deleted data:                0
Datafile pointer (bytes):     2  Keyfile pointer (bytes):     2
Max datafile length:   54657023  Max keyfile length:   33554431
Recordlength:               834
Record format: Fixed length

table description:
Key Start Len Index   Type                 Root  Blocksize    Rec/key
1   2     4   unique  unsigned long        1024       1024          1
2   32    30  multip. text                10240       1024          1

Field Start Length Type
1     1     1
2     2     4
3     6     4
4     10    1
5     11    20
6     31    1
7     32    30
8     62    35
9     97    35
10    132   35
11    167   4
12    171   16
13    187   35
14    222   4
15    226   16
16    242   20
17    262   20
18    282   20
19    302   30
20    332   4
21    336   4
22    340   1
23    341   8
24    349   8
25    357   8
26    365   2
27    367   2
28    369   4
29    373   4
30    377   1
31    378   2
32    380   8
33    388   4
34    392   4
35    396   4
36    400   4
37    404   1
38    405   4
39    409   4
40    413   4
```

```
41    417   4
42    421   4
43    425   4
44    429   20
45    449   30
46    479   1
47    480   1
48    481   79
49    560   79
50    639   79
51    718   79
52    797   8
53    805   1
54    806   1
55    807   20
56    827   4
57    831   4

shell> myisampack station.MYI
Compressing station.MYI: (1192 records)
- Calculating statistics

normal:     20  empty-space:   16  empty-zero:     12  empty-fill:  11
pre-space:   0  end-space:     12  table-lookups:   5  zero:         7
Original trees:  57  After join: 17
- Compressing file
87.14%
Remember to run myisamchk -rq on compressed tables

shell> myisamchk -rq station
- check record delete-chain
- recovering (with sort) MyISAM-table 'station'
Data records: 1192
- Fixing index 1
- Fixing index 2

shell> mysqladmin -uroot flush-tables

shell> ls -l station.*
-rw-rw-r--   1 jones    my         127874 Apr 17 19:00 station.MYD
-rw-rw-r--   1 jones    my          55296 Apr 17 19:04 station.MYI
-rw-rw-r--   1 jones    my           5767 Apr 17 19:00 station.frm

shell> myisamchk -dvv station

MyISAM file:     station
Isam-version:  2
Creation time: 1996-03-13 10:08:58
Recover time:  1997-04-17 19:04:26
Data records:               1192  Deleted blocks:              0
Datafile parts:             1192  Deleted data:                0
Datafile pointer (bytes):      3  Keyfile pointer (bytes):     1
Max datafile length:    16777215  Max keyfile length:     131071
Recordlength:               834
Record format: Compressed

table description:
Key Start Len Index   Type                 Root  Blocksize    Rec/key
1   2     4   unique  unsigned long       10240       1024          1
2   32    30  multip. text                54272       1024          1

Field Start Length Type                          Huff tree  Bits
1     1     1      constant                            1     0
2     2     4      zerofill(1)                         2     9
3     6     4      no zeros, zerofill(1)               2     9
4     10    1                                          3     9
5     11    20     table-lookup                        4     0
6     31    1                                          3     9
7     32    30     no endspace, not_always             5     9
8     62    35     no endspace, not_always, no empty   6     9
9     97    35     no empty                            7     9
10    132   35     no endspace, not_always, no empty   6     9
```

```
11    167   4      zerofill(1)                          2     9
12    171   16     no endspace, not_always, no empty    5     9
13    187   35     no endspace, not_always, no empty    6     9
14    222   4      zerofill(1)                          2     9
15    226   16     no endspace, not_always, no empty    5     9
16    242   20     no endspace, not_always              8     9
17    262   20     no endspace, no empty                8     9
18    282   20     no endspace, no empty                5     9
19    302   30     no endspace, no empty                6     9
20    332   4      always zero                          2     9
21    336   4      always zero                          2     9
22    340   1                                           3     9
23    341   8      table-lookup                         9     0
24    349   8      table-lookup                        10     0
25    357   8      always zero                          2     9
26    365   2                                           2     9
27    367   2      no zeros, zerofill(1)                2     9
28    369   4      no zeros, zerofill(1)                2     9
29    373   4      table-lookup                        11     0
30    377   1                                           3     9
31    378   2      no zeros, zerofill(1)                2     9
32    380   8      no zeros                             2     9
33    388   4      always zero                          2     9
34    392   4      table-lookup                        12     0
35    396   4      no zeros, zerofill(1)               13     9
36    400   4      no zeros, zerofill(1)                2     9
37    404   1                                           2     9
38    405   4      no zeros                             2     9
39    409   4      always zero                          2     9
40    413   4      no zeros                             2     9
41    417   4      always zero                          2     9
42    421   4      no zeros                             2     9
43    425   4      always zero                          2     9
44    429   20     no empty                             3     9
45    449   30     no empty                             3     9
46    479   1                                          14     4
47    480   1                                          14     4
48    481   79     no endspace, no empty               15     9
49    560   79     no empty                             2     9
50    639   79     no empty                             2     9
51    718   79     no endspace                         16     9
52    797   8      no empty                             2     9
53    805   1                                          17     1
54    806   1                                           3     9
55    807   20     no empty                             3     9
56    827   4      no zeros, zerofill(2)                2     9
57    831   4      no zeros, zerofill(1)                2     9
```

`myisampack` displays the following kinds of information:

- `normal`

  The number of columns for which no extra packing is used.

- `empty-space`

  The number of columns containing values that are only spaces. These occupy one bit.

- `empty-zero`

  The number of columns containing values that are only binary zeros. These occupy one bit.

- `empty-fill`

  The number of integer columns that do not occupy the full byte range of their type. These are changed to a smaller type. For example, a `BIGINT` column (eight bytes) can be stored as a `TINYINT` column (one byte) if all its values are in the range from `-128` to `127`.

- `pre-space`

The number of decimal columns that are stored with leading spaces. In this case, each value contains a count for the number of leading spaces.

- `end-space`

  The number of columns that have a lot of trailing spaces. In this case, each value contains a count for the number of trailing spaces.

- `table-lookup`

  The column had only a small number of different values, which were converted to an `ENUM` before Huffman compression.

- `zero`

  The number of columns for which all values are zero.

- `Original trees`

  The initial number of Huffman trees.

- `After join`

  The number of distinct Huffman trees left after joining trees to save some header space.

After a table has been compressed, the `Field` lines displayed by `myisamchk -dvv` include additional information about each column:

- `Type`

  The data type. The value may contain any of the following descriptors:

  - `constant`

    All rows have the same value.

  - `no endspace`

    Do not store endspace.

  - `no endspace, not_always`

    Do not store endspace and do not do endspace compression for all values.

  - `no endspace, no empty`

    Do not store endspace. Do not store empty values.

  - `table-lookup`

    The column was converted to an `ENUM`.

  - `zerofill(N)`

    The most significant *`N`* bytes in the value are always 0 and are not stored.

  - `no zeros`

    Do not store zeros.

  - `always zero`

    Zero values are stored using one bit.

- `Huff tree`

  The number of the Huffman tree associated with the column.

- `Bits`

  The number of bits used in the Huffman tree.

After you run `myisampack`, use `myisamchk` to re-create any indexes. At this time, you can also sort the index blocks and create statistics needed for the MySQL optimizer to work more efficiently:

```
shell> myisamchk -rq --sort-index --analyze tbl_name.MYI
```

After you have installed the packed table into the MySQL database directory, you should execute `mysqladmin flush-tables` to force `mysqld` to start using the new table.

To unpack a packed table, use the `--unpack` option to `myisamchk`.

4.6.6 `mysql_config_editor` — MySQL Configuration Utility

The `mysql_config_editor` utility enables you to store authentication credentials in an obfuscated login path file named `.mylogin.cnf`. The file location is the `%APPDATA%\MySQL` directory on Windows and the current user's home directory on non-Windows systems. The file can be read later by MySQL client programs to obtain authentication credentials for connecting to MySQL Server.

The unobfuscated format of the `.mylogin.cnf` login path file consists of option groups, similar to other option files. Each option group in `.mylogin.cnf` is called a “login path,” which is a group that permits only certain options: `host`, `user`, `password`, `port` and `socket`. Think of a login path option group as a set of options that specify which MySQL server to connect to and which account to authenticate as. Here is an unobfuscated example:

```
[client]
user = mydefaultname
password = mydefaultpass
host = 127.0.0.1
[mypath]
user = myothername
password = myotherpass
host = localhost
```

When you invoke a client program to connect to the server, the client uses `.mylogin.cnf` in conjunction with other option files. Its precedence is higher than other option files, but less than options specified explicitly on the client command line. For information about the order in which option files are used, see Section 4.2.2.2, “Using Option Files”.

To specify an alternate login path file name, set the `MYSQL_TEST_LOGIN_FILE` environment variable. This variable is recognized by `mysql_config_editor`, by standard MySQL clients (`mysql`, `mysqladmin`, and so forth), and by the `mysql-test-run.pl` testing utility.

Programs use groups in the login path file as follows:

- `mysql_config_editor` operates on the `client` login path by default if you specify no `--login-path=`*`name`* option to indicate explicitly which login path to use.

- Without a `--login-path` option, client programs read the same option groups from the login path file that they read from other option files. Consider this command:

  ```
  shell> mysql
  ```

  By default, the `mysql` client reads the `[client]` and `[mysql]` groups from other option files, so it reads them from the login path file as well.

- With a `--login-path` option, client programs additionally read the named login path from the login path file. The option groups read from other option files remain the same. Consider this command:

```
shell> mysql --login-path=mypath
```

The mysql client reads [client] and [mysql] from other option files, and [client], [mysql], and [mypath] from the login path file.

- Client programs read the login path file even when the --no-defaults option is used. This permits passwords to be specified in a safer way than on the command line even if --no-defaults is present.

mysql_config_editor obfuscates the .mylogin.cnf file so it cannot be read as cleartext, and its contents when unobfuscated by client programs are used only in memory. In this way, passwords can be stored in a file in non-cleartext format and used later without ever needing to be exposed on the command line or in an environment variable. mysql_config_editor provides a print command for displaying the login path file contents, but even in this case, password values are masked so as never to appear in a way that other users can see them.

The obfuscation used by mysql_config_editor prevents passwords from appearing in .mylogin.cnf as cleartext and provides a measure of security by preventing inadvertent password exposure. For example, if you display a regular unobfuscated my.cnf option file on the screen, any passwords it contains are visible for anyone to see. With .mylogin.cnf, that is not true. But the obfuscation used does not deter a determined attacker and you should not consider it unbreakable. A user who can gain system administration privileges on your machine to access your files could unobfuscate the .mylogin.cnf file with some effort.

The login path file must be readable and writable to the current user, and inaccessible to other users. Otherwise, mysql_config_editor ignores it, and client programs do not use it, either.

Invoke mysql_config_editor like this:

```
shell> mysql_config_editor [program_options] command [command_options]
```

If the login path file does not exist, mysql_config_editor creates it.

Command arguments are given as follows:

- *program_options* consists of general mysql_config_editor options.
- command indicates what action to perform on the .mylogin.cnf login path file. For example, set writes a login path to the file, remove removes a login path, and print displays login path contents.
- *command_options* indicates any additional options specific to the command, such as the login path name and the values to use in the login path.

The position of the command name within the set of program arguments is significant. For example, these command lines have the same arguments, but produce different results:

```
shell> mysql_config_editor --help set
shell> mysql_config_editor set --help
```

The first command line displays a general mysql_config_editor help message, and ignores the set command. The second command line displays a help message specific to the set command.

Suppose that you want to establish a client login path that defines your default connection parameters, and an additional login path named remote for connecting to the MySQL server the host remote.example.com. You want to log in as follows:

- By default, to the local server with a user name and password of localuser and localpass
- To the remote server with a user name and password of remoteuser and remotepass

To set up the login paths in the .mylogin.cnf file, use the following set commands. Enter each command on a single line, and enter the appropriate passwords when prompted:

```
shell> mysql_config_editor set --login-path=client
         --host=localhost --user=localuser --password
```

```
Enter password: enter password "localpass" here
shell> mysql_config_editor set --login-path=remote
         --host=remote.example.com --user=remoteuser --password
Enter password: enter password "remotepass" here
```

mysql_config_editor uses the client login path by default, so the --login-path=client option can be omitted from the first command without changing its effect.

To see what mysql_config_editor writes to the .mylogin.cnf file, use the print command:

```
shell> mysql_config_editor print --all
[client]
user = localuser
password = *****
host = localhost
[remote]
user = remoteuser
password = *****
host = remote.example.com
```

The print command displays each login path as a set of lines beginning with a group header indicating the login path name in square brackets, followed by the option values for the login path. Password values are masked and do not appear as cleartext.

If you do not specify --all to display all login paths or --login-path=*name* to display a named login path, the print command displays the client login path by default, if there is one.

As shown by the preceding example, the login path file can contain multiple login paths. In this way, mysql_config_editor makes it easy to set up multiple “personalities” for connecting to different MySQL servers, or for connecting to a given server using different accounts. Any of these can be selected by name later using the --login-path option when you invoke a client program. For example, to connect to the remote server, use this command:

```
shell> mysql --login-path=remote
```

Here, mysql reads the [client] and [mysql] option groups from other option files, and the [client], [mysql], and [remote] groups from the login path file.

To connect to the local server, use this command:

```
shell> mysql --login-path=client
```

Because mysql reads the client and mysql login paths by default, the --login-path option does not add anything in this case. That command is equivalent to this one:

```
shell> mysql
```

Options read from the login path file take precedence over options read from other option files. Options read from login path groups appearing later in the login path file take precedence over options read from groups appearing earlier in the file.

mysql_config_editor adds login paths to the login path file in the order you create them, so you should create more general login paths first and more specific paths later. If you need to move a login path within the file, you can remove it, then recreate it to add it to the end. For example, a client login path is more general because it is read by all client programs, whereas a mysqldump login path is read only by mysqldump. Options specified later override options specified earlier, so putting the login paths in the order client, mysqldump enables mysqldump-specific options to override client options.

When you use the set command with mysql_config_editor to create a login path, you need not specify all possible option values (host name, user name, password, port, socket). Only those values given are written to the path. Any missing values required later can be specified when you invoke a client path to connect to the MySQL server, either in other option files or on the command line. Any options specified on the command line override those specified in the login path file or other option files. For example, if the credentials in the remote login path also apply for the host remote2.example.com, connect to the server on that host like this:

```
shell> mysql --login-path=remote --host=remote2.example.com
```

mysql_config_editor General Options

mysql_config_editor supports the following general options, which may be used preceding any command named on the command line. For descriptions of command-specific options, see mysql_config_editor Commands and Command-Specific Options.

Table 4.18 mysql_config_editor General Options

Option Name	Description
--debug	Write debugging log
--help	Display help message and exit
--verbose	Verbose mode
--version	Display version information and exit

- `--help`, `-?`

  Display a general help message and exit.

  To see a command-specific help message, invoke mysql_config_editor as follows, where *command* is a command other than `help`:

  ```
  shell> mysql_config_editor command --help
  ```

- `--debug[=debug_options]`, `-# debug_options`

  Write a debugging log. A typical *debug_options* string is `d:t:o,file_name`. The default is `d:t:o,/tmp/mysql_config_editor.trace`.

  This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are *not* built using this option.

- `--verbose`, `-v`

  Verbose mode. Print more information about what the program does. This option may be helpful in diagnosing problems if an operation does not have the effect you expect.

- `--version`, `-V`

  Display version information and exit.

mysql_config_editor Commands and Command-Specific Options

This section describes the permitted mysql_config_editor commands, and, for each one, the command-specific options permitted following the command name on the command line.

In addition, mysql_config_editor supports general options that can be used preceding any command. For descriptions of these options, see mysql_config_editor General Options.

mysql_config_editor supports these commands:

- `help`

  Display a general help message and exit. This command takes no following options.

  To see a command-specific help message, invoke mysql_config_editor as follows, where *command* is a command other than `help`:

  ```
  shell> mysql_config_editor command --help
  ```

- `print [options]`

Print the contents of the login path file in unobfuscated form, with the exception that passwords are displayed as `*****`.

The default login path name is `client` if no login path is named. If both `--all` and `--login-path` are given, `--all` takes precedence.

The `print` command permits these options following the command name:

- `--help`, `-?`

  Display a help message for the `print` command and exit.

  To see a general help message, use `mysql_config_editor --help`.

- `--all`

  Print the contents of all login paths in the login path file.

- `--login-path=`*`name`*, `-G` *`name`*

  Print the contents of the named login path.

- `remove [`*`options`*`]`

  Remove a login path from the login path file, or modify a login path by removing options from it.

  This command removes from the login path only such options as are specified with the `--host`, `--password`, `--port`, `--socket`, and `--user` options. If none of those options are given, `remove`

removes the entire login path. For example, this command removes only the `user` option from the `mypath` login path rather than the entire `mypath` login path:

```
shell> mysql_config_editor remove --login-path=mypath --user
```

This command removes the entire `mypath` login path:

```
shell> mysql_config_editor remove --login-path=mypath
```

The `remove` command permits these options following the command name:

- `--help`, `-?`

  Display a help message for the `remove` command and exit.

  To see a general help message, use `mysql_config_editor --help`.

- `--host`, `-h`

  Remove the host name from the login path.

- `--login-path=`*`name`*, `-G` *`name`*

  The login path to remove or modify. The default login path name is `client` if this option is not given.

- `--password`, `-p`

  Remove the password from the login path.

- `--port`, `-P`

  Remove the TCP/IP port number from the login path. This option was added in MySQL 5.6.11.

- `--socket`, `-S`

  Remove the Unix socket file name from the login path. This option was added in MySQL 5.6.11.

- `--user`, `-u`

  Remove the user name from the login path.

- `--warn`, `-w`

  Warn and prompt the user for confirmation if the command attempts to remove the default login path (`client`) and `--login-path=client` was not specified. This option is enabled by default; use `--skip-warn` to disable it.

- `reset [`*`options`*`]`

Empty the contents of the login path file.

The `reset` command permits these options following the command name:

- `--help`, `-?`

  Display a help message for the `reset` command and exit.

  To see a general help message, use `mysql_config_editor --help`.

- `set [options]`

  Write a login path to the login path file.

  This command writes to the login path only such options as are specified with the `--host`, `--password`, `--port`, `--socket`, and `--user` options. If none of those options are given, `mysql_config_editor` writes the login path as an empty group.

  The `set` command permits these options following the command name:

  - `--help`, `-?`

    Display a help message for the `set` command and exit.

    To see a general help message, use `mysql_config_editor --help`.

  - `--host=host_name`, `-h host_name`

    The host name to write to the login path.

  - `--login-path=name`, `-G name`

    The login path to create. The default login path name is `client` if this option is not given.

  - `--password`, `-p`

    Prompt for a password to write to the login path. After `mysql_config_editor` displays the prompt, type the password and press Enter. To prevent other users from seeing the password, `mysql_config_editor` does not echo it.

    To specify an empty password, press Enter at the password prompt. The resulting login path written to the login path file includes a line like this:

    ```
    password =
    ```

  - `--port=port_num`, `-P port_num`

    The TCP/IP port number to write to the login path. This option was added in MySQL 5.6.11.

  - `--socket=file_name`, `-S file_name`

    The Unix socket file name to write to the login path. This option was added in MySQL 5.6.11.

  - `--user=user_name`, `-u user_name`

    The user name to write to the login path.

  - `--warn`, `-w`

    Warn and prompt the user for confirmation if the command attempts to overwrite an existing login path. This option is enabled by default; use `--skip-warn` to disable it.

4.6.7 `mysqlaccess` — Client for Checking Access Privileges

Note

This utility is deprecated in MySQL 5.6.17 and removed in MySQL 5.7

`mysqlaccess` is a diagnostic tool that Yves Carlier has provided for the MySQL distribution. It checks the access privileges for a host name, user name, and database combination. `mysqlaccess` checks access using only the `user` and `db` tables. It does not check table, column, or routine privileges specified in the `tables_priv`, `columns_priv`, or `procs_priv` tables.

Invoke mysqlaccess like this:

```
shell> mysqlaccess [host_name [user_name [db_name]]] [options]
```

When mysqlaccess runs, it loads and executes the contents of its configuration file, mysqlaccess.conf. mysqlaccess looks for the configuration file in these locations, in order:

- The directory named by the SYSCONFDIR option to CMake when MySQL was built. By default, this is the etc directory located under the compiled-in installation directory.
- The /etc directory.

mysqlaccess supports the options shown in the following table.

Table 4.19 mysqlaccess Options

Option Name	Description
--brief	Generate reports in single-line tabular format
--commit	Copy the new access privileges from the temporary tables to the original grant tables
--copy	Reload the temporary grant tables from original ones
--db	Specify the database name
--debug	Specify the debug level
--help	Display help message and exit
--host	Host on which MySQL server is located
--howto	Display some examples that show how to use mysqlaccess
--old_server	Assume that the server is an old MySQL server (prior to MySQL 3.21)
--password	Password to use when connecting to server
--plan	Display suggestions and ideas for future releases
--preview	Show the privilege differences after making changes to the temporary grant tables
--relnotes	Display release notes
--rhost	Host on which MySQL server is located
--rollback	Undo the most recent changes to the temporary grant tables
--spassword	Password to use when connecting to server as the superuser
--superuser	Specify the user name for connecting as the superuser
--table	Generate reports in table format
--user	MySQL user name to use when connecting to server
--version	Display version information and exit

- --help, -?

  Display a help message and exit.

- --brief, -b

  Generate reports in single-line tabular format.

- --commit

  Copy the new access privileges from the temporary tables to the original grant tables. The grant tables must be flushed for the new privileges to take effect. (For example, execute a mysqladmin reload command.)

- `--copy`

  Reload the temporary grant tables from original ones.

- `--db=db_name`, `-d db_name`

  Specify the database name.

- `--debug=N`

  Specify the debug level. *`N`* can be an integer from 0 to 3.

  This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are *not* built using this option.

- `--host=host_name`, `-h host_name`

  The host name to use in the access privileges.

- `--howto`

  Display some examples that show how to use `mysqlaccess`.

- `--old_server`

  Assume that the server is an old MySQL server (before MySQL 3.21) that does not yet know how to handle full `WHERE` clauses.

- `--password[=password]`, `-p[password]`

  The password to use when connecting to the server. If you omit the *`password`* value following the `--password` or `-p` option on the command line, `mysqlaccess` prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

- `--plan`

  Display suggestions and ideas for future releases.

- `--preview`

  Show the privilege differences after making changes to the temporary grant tables.

- `--relnotes`

  Display the release notes.

- `--rhost=host_name`, `-H host_name`

  Connect to the MySQL server on the given host.

- `--rollback`

  Undo the most recent changes to the temporary grant tables.

- `--spassword[=password]`, `-P[password]`

  The password to use when connecting to the server as the superuser. If you omit the *`password`* value following the `--spassword` or `-p` option on the command line, `mysqlaccess` prompts for one.

  Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

- `--superuser=user_name`, `-U user_name`

  Specify the user name for connecting as the superuser.

- `--table`, `-t`

  Generate reports in table format.

- `--user=user_name`, `-u user_name`

  The user name to use in the access privileges.

- `--version`, `-v`

  Display version information and exit.

If your MySQL distribution is installed in some nonstandard location, you must change the location where `mysqlaccess` expects to find the `mysql` client. Edit the `mysqlaccess` script at approximately line 18. Search for a line that looks like this:

```
$MYSQL     = '/usr/local/bin/mysql';    # path to mysql executable
```

Change the path to reflect the location where `mysql` actually is stored on your system. If you do not do this, a `Broken pipe` error occurs when you run `mysqlaccess`.

4.6.8 mysqlbinlog — Utility for Processing Binary Log Files

The server's binary log consists of files containing “events” that describe modifications to database contents. The server writes these files in binary format. To display their contents in text format, use the `mysqlbinlog` utility. You can also use `mysqlbinlog` to display the contents of relay log files written by a replica server in a replication setup because relay logs have the same format as binary logs. The binary log and relay log are discussed further in Section 5.4.4, “The Binary Log”, and Section 17.2.2, “Relay Log and Replication Metadata Repositories”.

Invoke `mysqlbinlog` like this:

```
shell> mysqlbinlog [options] log_file ...
```

For example, to display the contents of the binary log file named `binlog.000003`, use this command:

```
shell> mysqlbinlog binlog.0000003
```

The output includes events contained in `binlog.000003`. For statement-based logging, event information includes the SQL statement, the ID of the server on which it was executed, the timestamp when the statement was executed, how much time it took, and so forth. For row-based logging, the event indicates a row change rather than an SQL statement. See Section 17.1.2, “Replication Formats”, for information about logging modes.

Events are preceded by header comments that provide additional information. For example:

```
# at 141
#100309  9:28:36 server id 123  end_log_pos 245
  Query thread_id=3350  exec_time=11  error_code=0
```

In the first line, the number following `at` indicates the file offset, or starting position, of the event in the binary log file.

The second line starts with a date and time indicating when the statement started on the server where the event originated. For replication, this timestamp is propagated to replica servers. `server id` is the `server_id` value of the server where the event originated. `end_log_pos` indicates where the next event starts (that is, it is the end position of the current event + 1). `thread_id` indicates which thread executed the event. `exec_time` is the time spent executing the event, on a source server. On a replica, it is the difference of the end execution time on the replica minus the beginning execution time on the source. The difference serves as an indicator of how much replication lags behind the source. `error_code` indicates the result from executing the event. Zero means that no error occurred.

Note

When using event groups, the file offsets of events may be grouped together and the comments of events may be grouped together. Do not mistake these grouped events for blank file offsets.

The output from `mysqlbinlog` can be re-executed (for example, by using it as input to `mysql`) to redo the statements in the log. This is useful for recovery operations after an unexpected server exit. For other usage examples, see the discussion later in this section and in Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”.

Normally, you use `mysqlbinlog` to read binary log files directly and apply them to the local MySQL server. It is also possible to read binary logs from a remote server by using the `--read-from-remote-server` option. To read remote binary logs, the connection parameter options can be given to indicate how to connect to the server. These options are `--host`, `--password`, `--port`, `--protocol`, `--socket`, and `--user`.

When running `mysqlbinlog` against a large binary log, be careful that the filesystem has enough space for the resulting files. To configure the directory that `mysqlbinlog` uses for temporary files, use the `TMPDIR` environment variable.

In MySQL 5.6.10 and later, `mysqlbinlog` sets the value of `pseudo_slave_mode` to true before executing any SQL statements.

`mysqlbinlog` supports the following options, which can be specified on the command line or in the `[mysqlbinlog]` and `[client]` groups of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.20 mysqlbinlog Options

Option Name	Description	Introduced
--base64-output	Print binary log entries using base-64 encoding	
--bind-address	Use specified network interface to connect to MySQL Server	
--binlog-row-event-max-size	Binary log max event size	
--character-sets-dir	Directory where character sets are installed	
--connection-server-id	Used for testing and debugging. See text for applicable default values and other particulars	5.6.20
--database	List entries for just this database	
--debug	Write debugging log	
--debug-check	Print debugging information when program exits	
--debug-info	Print debugging information, memory, and CPU statistics when program exits	
--default-auth	Authentication plugin to use	
--defaults-extra-file	Read named option file in addition to usual option files	
--defaults-file	Read only named option file	
--defaults-group-suffix	Option group suffix value	
--disable-log-bin	Disable binary logging	
--exclude-gtids	Do not show any of the groups in the GTID set provided	
--force-if-open	Read binary log files even if open or not closed properly	
--force-read	If mysqlbinlog reads a binary log event that it does not recognize, it prints a warning	
--help	Display help message and exit	

Option Name	Description	Introduced
--hexdump	Display a hex dump of the log in comments	
--host	Host on which MySQL server is located	
--include-gtids	Show only the groups in the GTID set provided	
--local-load	Prepare local temporary files for LOAD DATA in the specified directory	
--login-path	Read login path options from .mylogin.cnf	
--no-defaults	Read no option files	
--offset	Skip the first N entries in the log	
open_files_limit	Specify the number of open file descriptors to reserve	
--password	Password to use when connecting to server	
--plugin-dir	Directory where plugins are installed	
--port	TCP/IP port number for connection	
--print-defaults	Print default options	
--protocol	Transport protocol to use	
--raw	Write events in raw (binary) format to output files	
--read-from-remote-master	Read the binary log from a MySQL master rather than reading a local log file	
--read-from-remote-server	Read binary log from MySQL server rather than local log file	
--result-file	Direct output to named file	
--secure-auth	Do not send passwords to server in old (pre-4.1) format	5.6.17
--server-id	Extract only those events created by the server having the given server ID	
--server-id-bits	Tell mysqlbinlog how to interpret server IDs in binary log when log was written by a mysqld having its server-id-bits set to less than the maximum; supported only by MySQL Cluster version of mysqlbinlog	
--set-charset	Add a SET NAMES charset_name statement to the output	
--shared-memory-base-name	Shared-memory name for shared-memory connections (Windows only)	
--short-form	Display only the statements contained in the log	
--skip-gtids	Do not print any GTIDs; use this when writing a dump file from binary logs containing GTIDs	
--socket	Unix socket file or Windows named pipe to use	
--ssl-crl	File that contains certificate revocation lists	
--ssl-crlpath	Directory that contains certificate revocation-list files	
--ssl-mode	Desired security state of connection to server	5.6.30
--start-datetime	Read binary log from first event with timestamp equal to or later than datetime argument	
--start-position	Decode binary log from first event with position equal to or greater than argument	
--stop-datetime	Stop reading binary log at first event with timestamp equal to or greater than datetime argument	

Option Name	Description	Introduced
--stop-never	Stay connected to server after reading last binary log file	
--stop-never-slave-server-id	Slave server ID to report when connecting to server	
--stop-position	Stop decoding binary log at first event with position equal to or greater than argument	
--to-last-log	Do not stop at the end of requested binary log from a MySQL server, but rather continue printing to end of last binary log	
--user	MySQL user name to use when connecting to server	
--verbose	Reconstruct row events as SQL statements	
--verify-binlog-checksum	Verify checksums in binary log	
--version	Display version information and exit	

- `--help`, `-?`

  Display a help message and exit.

- `--base64-output=value`

  This option determines when events should be displayed encoded as base-64 strings using `BINLOG` statements. The option has these permissible values (not case-sensitive):

  - `AUTO` ("automatic") or `UNSPEC` ("unspecified") displays `BINLOG` statements automatically when necessary (that is, for format description events and row events). If no `--base64-output` option is given, the effect is the same as `--base64-output=AUTO`.

    Note

    Automatic `BINLOG` display is the only safe behavior if you intend to use the output of `mysqlbinlog` to re-execute binary log file contents. The other option values are intended only for debugging or testing purposes because they may produce output that does not include all events in executable form.

  - `NEVER` causes `BINLOG` statements not to be displayed. `mysqlbinlog` exits with an error if a row event is found that must be displayed using `BINLOG`.

  - `DECODE-ROWS` specifies to `mysqlbinlog` that you intend for row events to be decoded and displayed as commented SQL statements by also specifying the `--verbose` option. Like `NEVER`, `DECODE-ROWS` suppresses display of `BINLOG` statements, but unlike `NEVER`, it does not exit with an error if a row event is found.

  For examples that show the effect of `--base64-output` and `--verbose` on row event output, see Section 4.6.8.2, “mysqlbinlog Row Event Display”.

- `--bind-address=ip_address`

  On a computer having multiple network interfaces, use this option to select which interface to use for connecting to the MySQL server.

- `--binlog-row-event-max-size=N`

Command-Line Format	`--binlog-row-event-max-size=#`
Type	Numeric
Default Value	`4294967040`
Minimum Value	`256`

Maximum Value	18446744073709547520

Specify the maximum size of a row-based binary log event, in bytes. Rows are grouped into events smaller than this size if possible. The value should be a multiple of 256. The default is 4GB.

- `--character-sets-dir=dir_name`

  The directory where character sets are installed. See Section 10.15, “Character Set Configuration”.

- `--connection-server-id=server_id`

  This option is used to test a MySQL server for support of the `BINLOG_DUMP_NON_BLOCK` connection flag, which was inadvertently removed in MySQL 5.6.5, and restored in MySQL 5.6.20 (Bug #18000079, Bug #71178). It is not required for normal operation.

  The effective default and minimum values for this option depend on whether `mysqlbinlog` is run in blocking mode or non-blocking mode. When `mysqlbinlog` is run in blocking mode, the default (and minimum) value is 1; when run in non-blocking mode, the default (and minimum) value is 0.

  This option was added in MySQL 5.6.20.

- `--database=db_name`, `-d db_name`

  This option causes `mysqlbinlog` to output entries from the binary log (local log only) that occur while *`db_name`* is been selected as the default database by `USE`.

  The `--database` option for `mysqlbinlog` is similar to the `--binlog-do-db` option for `mysqld`, but can be used to specify only one database. If `--database` is given multiple times, only the last instance is used.

  The effects of this option depend on whether the statement-based or row-based logging format is in use, in the same way that the effects of `--binlog-do-db` depend on whether statement-based or row-based logging is in use.

  Statement-based logging. The `--database` option works as follows:

  - While *`db_name`* is the default database, statements are output whether they modify tables in *`db_name`* or a different database.
  - Unless *`db_name`* is selected as the default database, statements are not output, even if they modify tables in *`db_name`*.
  - There is an exception for `CREATE DATABASE`, `ALTER DATABASE`, and `DROP DATABASE`. The database being *created, altered, or dropped* is considered to be the default database when determining whether to output the statement.

  Suppose that the binary log was created by executing these statements using statement-based-logging:

```
INSERT INTO test.t1 (i) VALUES(100);
INSERT INTO db2.t2 (j)  VALUES(200);
USE test;
INSERT INTO test.t1 (i) VALUES(101);
INSERT INTO t1 (i)      VALUES(102);
INSERT INTO db2.t2 (j)  VALUES(201);
USE db2;
INSERT INTO test.t1 (i) VALUES(103);
INSERT INTO db2.t2 (j)  VALUES(202);
INSERT INTO t2 (j)      VALUES(203);
```

  `mysqlbinlog --database=test` does not output the first two `INSERT` statements because there is no default database. It outputs the three `INSERT` statements following `USE test`, but not the three `INSERT` statements following `USE db2`.

`mysqlbinlog --database=db2` does not output the first two `INSERT` statements because there is no default database. It does not output the three `INSERT` statements following `USE test`, but does output the three `INSERT` statements following `USE db2`.

Row-based logging. `mysqlbinlog` outputs only entries that change tables belonging to *`db_name`*. The default database has no effect on this. Suppose that the binary log just described was created using row-based logging rather than statement-based logging. `mysqlbinlog --database=test` outputs only those entries that modify `t1` in the test database, regardless of whether `USE` was issued or what the default database is.

If a server is running with `binlog_format` set to `MIXED` and you want it to be possible to use `mysqlbinlog` with the `--database` option, you must ensure that tables that are modified are in the database selected by `USE`. (In particular, no cross-database updates should be used.)

- `--debug[=debug_options]`, `-# [debug_options]`

  Write a debugging log. A typical *`debug_options`* string is `d:t:o,file_name`. The default is `d:t:o,/tmp/mysqlbinlog.trace`.

  This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are *not* built using this option.

- `--debug-check`

  Print some debugging information when the program exits.

  This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are *not* built using this option.

- `--debug-info`

  Print debugging information and memory and CPU usage statistics when the program exits.

  This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are *not* built using this option.

- `--default-auth=plugin`

  A hint about which client-side authentication plugin to use. See Section 6.2.11, “Pluggable Authentication”.

- `--defaults-extra-file=file_name`

  Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=file_name`

  Use only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

  Exception: Even with `--defaults-file`, client programs read `.mylogin.cnf`.

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of *`str`*. For example, `mysqlbinlog` normally reads the `[client]` and `[mysqlbinlog]` groups. If the `--defaults-group-suffix=_other` option is given, `mysqlbinlog` also reads the `[client_other]` and `[mysqlbinlog_other]` groups.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--disable-log-bin`, `-D`

  Disable binary logging. This is useful for avoiding an endless loop if you use the `--to-last-log` option and are sending the output to the same MySQL server. This option also is useful when restoring after an unexpected exit to avoid duplication of the statements you have logged.

  This option causes `mysqlbinlog` to include a `SET sql_log_bin = 0` statement in its output to disable binary logging of the remaining output. Manipulating the session value of the `sql_log_bin` system variable is a restricted operation, so this option requires that you have privileges sufficient to set restricted session variables. See Section 5.1.8.1, “System Variable Privileges”.

- `--exclude-gtids=gtid_set`

  Do not display any of the groups listed in the *`gtid_set`*.

- `--force-if-open`, `-F`

  Read binary log files even if they are open or were not closed properly.

- `--force-read`, `-f`

  With this option, if `mysqlbinlog` reads a binary log event that it does not recognize, it prints a warning, ignores the event, and continues. Without this option, `mysqlbinlog` stops if it reads such an event.

- `--hexdump`, `-H`

  Display a hex dump of the log in comments, as described in Section 4.6.8.1, “mysqlbinlog Hex Dump Format”. The hex output can be helpful for replication debugging.

- `--host=host_name`, `-h host_name`

  Get the binary log from the MySQL server on the given host.

- `--include-gtids=gtid_set`

  Display only the groups listed in the *`gtid_set`*.

- `--local-load=dir_name`, `-l dir_name`

  For data loading operations corresponding to `LOAD DATA` statements, `mysqlbinlog` extracts the files from the binary log events, writes them as temporary ffiles to the local file system, and writes `LOAD DATA LOCAL` statements to cause the files to be loaded. By default, `mysqlbinlog` writes these temporary files to an operating system-specific directory. The `--local-load` option can be used to explicitly specify the directory where `mysqlbinlog` should prepare local temporary files.

  Important

  These temporary files are not automatically removed by `mysqlbinlog` or any other MySQL program.

- `--login-path=name`

Read options from the named login path in the .mylogin.cnf login path file. A "login path" is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the mysql_config_editor utility. See Section 4.6.6, "mysql_config_editor — MySQL Configuration Utility".

For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- --no-defaults

  Do not read any option files. If program startup fails due to reading unknown options from an option file, --no-defaults can be used to prevent them from being read.

  The exception is that the .mylogin.cnf file, if it exists, is read in all cases. This permits passwords to be specified in a safer way than on the command line even when --no-defaults is used. (.mylogin.cnf is created by the mysql_config_editor utility. See Section 4.6.6, "mysql_config_editor — MySQL Configuration Utility".)

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- --offset=*N*, -o *N*

  Skip the first *N* entries in the log.

- --open-files-limit=*N*

  Specify the number of open file descriptors to reserve.

- --password[=*password*], -p[*password*]

  The password of the MySQL account used for connecting to the server. The password value is optional. If not given, mysqlbinlog prompts for one. If given, there must be *no space* between --password= or -p and the password following it. If no password option is specified, the default is to send no password.

  Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, "End-User Guidelines for Password Security".

  To explicitly specify that there is no password and that mysqlbinlog should not prompt for one, use the --skip-password option.

- --plugin-dir=*dir_name*

  The directory in which to look for plugins. Specify this option if the --default-auth option is used to specify an authentication plugin but mysqlbinlog does not find it. See Section 6.2.11, "Pluggable Authentication".

- --port=*port_num*, -P *port_num*

  The TCP/IP port number to use for connecting to a remote server.

- --print-defaults

  Print the program name and all options that it gets from option files.

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- --protocol={TCP|SOCKET|PIPE|MEMORY}

The transport protocol to use for connecting to the server. It is useful when the other connection parameters normally result in use of a protocol other than the one you want. For details on the permissible values, see Section 4.2.5, “Connection Transport Protocols”.

- `--raw`

  By default, `mysqlbinlog` reads binary log files and writes events in text format. The `--raw` option tells `mysqlbinlog` to write them in their original binary format. Its use requires that `--read-from-remote-server` also be used because the files are requested from a server. `mysqlbinlog` writes one output file for each file read from the server. The `--raw` option can be used to make a backup of a server's binary log. With the `--stop-never` option, the backup is “live” because `mysqlbinlog` stays connected to the server. By default, output files are written in the current directory with the same names as the original log files. Output file names can be modified using the `--result-file` option. For more information, see Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”.

- `--read-from-remote-master=`*`type`*

  Read binary logs from a MySQL server with the `COM_BINLOG_DUMP` or `COM_BINLOG_DUMP_GTID` commands by setting the option value to either `BINLOG-DUMP-NON-GTIDS` or `BINLOG-DUMP-GTIDS`, respectively. If `--read-from-remote-master=BINLOG-DUMP-GTIDS` is combined with `--exclude-gtids`, transactions can be filtered out on the source, avoiding unnecessary network traffic.

  The connection parameter options are used with this option or the `--read-from-remote-server` option. These options are `--host`, `--password`, `--port`, `--protocol`, `--socket`, and `--user`. If neither of the remote options is specified, the connection parameter options are ignored.

  The `REPLICATION SLAVE` privilege is required to use this option.

- `--read-from-remote-server`, `-R`

  Read the binary log from a MySQL server rather than reading a local log file. This option requires that the remote server be running. It works only for binary log files on the remote server, not relay log files.

  The connection parameter options are used with this option or the `--read-from-remote-master` option. These options are `--host`, `--password`, `--port`, `--protocol`, `--socket`, and `--user`. If neither of the remote options is specified, the connection parameter options are ignored.

  The `REPLICATION SLAVE` privilege is required to use this option.

  This option is like `--read-from-remote-master=BINLOG-DUMP-NON-GTIDS`.

- `--result-file=`*`name`*, `-r` *`name`*

  Without the `--raw` option, this option indicates the file to which `mysqlbinlog` writes text output. With `--raw`, `mysqlbinlog` writes one binary output file for each log file transferred from the server, writing them by default in the current directory using the same names as the original log file. In this case, the `--result-file` option value is treated as a prefix that modifies output file names.

- `--secure-auth`

  Do not send passwords to the server in old (pre-4.1) format. This prevents connections except for servers that use the newer password format. This option is enabled by default; use `--skip-secure-auth` to disable it. This option was added in MySQL 5.6.17.

Note

Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be

avoided. Pre-4.1 passwords are deprecated; expect support for them to be removed in a future MySQL release. For account upgrade instructions, see Section 6.4.1.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”.

Note

This option is deprecated; expect it to be removed in a future release. As of MySQL 5.7.5, it is always enabled and attempting to disable it produces an error.

- `--server-id=id`

  Display only those events created by the server having the given server ID.

- `--server-id-bits=N`

  Use only the first `N` bits of the `server_id` to identify the server. If the binary log was written by a `mysqld` with server-id-bits set to less than 32 and user data stored in the most significant bit, running `mysqlbinlog` with `--server-id-bits` set to 32 enables this data to be seen.

  This option is supported only by the versions of `mysqlbinlog` supplied with the NDB Cluster distribution, or built from the NDB Cluster sources.

- `--set-charset=charset_name`

  Add a `SET NAMES charset_name` statement to the output to specify the character set to be used for processing log files.

- `--shared-memory-base-name=name`

  On Windows, the shared-memory name to use for connections made using shared memory to a local server. The default value is `MYSQL`. The shared-memory name is case-sensitive.

  This option applies only if the server was started with the `shared_memory` system variable enabled to support shared-memory connections.

- `--short-form`, `-s`

  Display only the statements contained in the log, without any extra information or row-based events. This is for testing only, and should not be used in production systems.

- `--skip-gtids[=(true|false)]`

  Do not display any GTIDs in the output. This is needed when writing to a dump file from one or more binary logs containing GTIDs, as shown in this example:

```
shell> mysqlbinlog --skip-gtids binlog.000001 >  /tmp/dump.sql
shell> mysqlbinlog --skip-gtids binlog.000002 >> /tmp/dump.sql
shell> mysql -u root -p -e "source /tmp/dump.sql"
```

  The use of this option is otherwise not normally recommended in production.

- `--socket=path`, `-S path`

  For connections to `localhost`, the Unix socket file to use, or, on Windows, the name of the named pipe to use.

  On Windows, this option applies only if the server was started with the `named_pipe` system variable enabled to support named-pipe connections. In addition, the user making the connection must be a member of the Windows group specified by the `named_pipe_full_access_group` system variable.

- `--ssl*`

  Options that begin with `--ssl` specify whether to connect to the server using SSL and indicate where to find SSL keys and certificates. See Command Options for Encrypted Connections.

- `--start-datetime=datetime`

  Start reading the binary log at the first event having a timestamp equal to or later than the *`datetime`* argument. The *`datetime`* value is relative to the local time zone on the machine where you run `mysqlbinlog`. The value should be in a format accepted for the `DATETIME` or `TIMESTAMP` data types. For example:

  ```
  shell> mysqlbinlog --start-datetime="2005-12-25 11:25:56" binlog.000003
  ```

  This option is useful for point-in-time recovery. See Section 7.3, “Example Backup and Recovery Strategy”.

- `--start-position=N`, `-j N`

  Start reading the binary log at the first event having a position equal to or greater than *`N`*. This option applies to the first log file named on the command line.

  This option is useful for point-in-time recovery. See Section 7.3, “Example Backup and Recovery Strategy”.

- `--stop-datetime=datetime`

  Stop reading the binary log at the first event having a timestamp equal to or later than the *`datetime`* argument. This option is useful for point-in-time recovery. See the description of the `--start-datetime` option for information about the *`datetime`* value.

  This option is useful for point-in-time recovery. See Section 7.3, “Example Backup and Recovery Strategy”.

- `--stop-never`

  This option is used with `--read-from-remote-server`. It tells `mysqlbinlog` to remain connected to the server. Otherwise `mysqlbinlog` exits when the last log file has been transferred from the server. `--stop-never` implies `--to-last-log`, so only the first log file to transfer need be named on the command line.

  `--stop-never` is commonly used with `--raw` to make a live binary log backup, but also can be used without `--raw` to maintain a continuous text display of log events as the server generates them.

- `--stop-never-slave-server-id=id`

  With `--stop-never`, `mysqlbinlog` reports a server ID of 65535 when it connects to the server. `--stop-never-slave-server-id` explicitly specifies the server ID to report. It can be used to avoid a conflict with the ID of a replica server or another `mysqlbinlog` process. See Section 4.6.8.4, “Specifying the mysqlbinlog Server ID”.

- `--stop-position=N`

  Stop reading the binary log at the first event having a position equal to or greater than *`N`*. This option applies to the last log file named on the command line.

  This option is useful for point-in-time recovery. See Section 7.3, “Example Backup and Recovery Strategy”.

- `--to-last-log`, `-t`

  Do not stop at the end of the requested binary log from a MySQL server, but rather continue printing until the end of the last binary log. If you send the output to the same MySQL server, this may lead to an endless loop. This option requires `--read-from-remote-server`.

- `--user=user_name`, `-u user_name`

  The user name of the MySQL account to use when connecting to a remote server.

- `--verbose`, `-v`

  Reconstruct row events and display them as commented SQL statements. If this option is given twice (by passing in either "-vv" or "--verbose --verbose"), the output includes comments to indicate column data types and some metadata, and row query log events if so configured.

  For examples that show the effect of `--base64-output` and `--verbose` on row event output, see Section 4.6.8.2, “mysqlbinlog Row Event Display”.

- `--verify-binlog-checksum`, `-c`

  Verify checksums in binary log files.

- `--version`, `-V`

  Display version information and exit.

  Prior to MySQL 5.6.11, the `mysqlbinlog` version number shown was 3.3. In MySQL 5.6.11 and later, this is 3.4. (Bug #15894381, Bug #67643)

You can pipe the output of `mysqlbinlog` into the `mysql` client to execute the events contained in the binary log. This technique is used to recover from an unexpected exit when you have an old backup (see Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”). For example:

```
shell> mysqlbinlog binlog.000001 | mysql -u root -p
```

Or:

```
shell> mysqlbinlog binlog.[0-9]* | mysql -u root -p
```

If the statements produced by `mysqlbinlog` may contain `BLOB` values, these may cause problems when `mysql` processes them. In this case, invoke `mysql` with the `--binary-mode` option.

You can also redirect the output of `mysqlbinlog` to a text file instead, if you need to modify the statement log first (for example, to remove statements that you do not want to execute for some reason). After editing the file, execute the statements that it contains by using it as input to the `mysql` program:

```
shell> mysqlbinlog binlog.000001 > tmpfile
shell> ... edit tmpfile ...
shell> mysql -u root -p < tmpfile
```

When `mysqlbinlog` is invoked with the `--start-position` option, it displays only those events with an offset in the binary log greater than or equal to a given position (the given position must match the start of one event). It also has options to stop and start when it sees an event with a given date and time. This enables you to perform point-in-time recovery using the `--stop-datetime` option (to be able to say, for example, “roll forward my databases to how they were today at 10:30 a.m.”).

If you have more than one binary log to execute on the MySQL server, the safe method is to process them all using a single connection to the server. Here is an example that demonstrates what may be *unsafe*:

```
shell> mysqlbinlog binlog.000001 | mysql -u root -p # DANGER!!
shell> mysqlbinlog binlog.000002 | mysql -u root -p # DANGER!!
```

Processing binary logs this way using multiple connections to the server causes problems if the first log file contains a CREATE TEMPORARY TABLE statement and the second log contains a statement that uses the temporary table. When the first mysql process terminates, the server drops the temporary table. When the second mysql process attempts to use the table, the server reports “unknown table.”

To avoid problems like this, use a *single* mysql process to execute the contents of all binary logs that you want to process. Here is one way to do so:

```
shell> mysqlbinlog binlog.000001 binlog.000002 | mysql -u root -p
```

Another approach is to write all the logs to a single file and then process the file:

```
shell> mysqlbinlog binlog.000001 >  /tmp/statements.sql
shell> mysqlbinlog binlog.000002 >> /tmp/statements.sql
shell> mysql -u root -p -e "source /tmp/statements.sql"
```

mysqlbinlog can produce output that reproduces a LOAD DATA operation without the original data file. mysqlbinlog copies the data to a temporary file and writes a LOAD DATA LOCAL statement that refers to the file. The default location of the directory where these files are written is system-specific. To specify a directory explicitly, use the --local-load option.

Because mysqlbinlog converts LOAD DATA statements to LOAD DATA LOCAL statements (that is, it adds LOCAL), both the client and the server that you use to process the statements must be configured with the LOCAL capability enabled. See Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”.

Warning

The temporary files created for LOAD DATA LOCAL statements are *not* automatically deleted because they are needed until you actually execute those statements. You should delete the temporary files yourself after you no longer need the statement log. The files can be found in the temporary file directory and have names like *original_file_name-#-#*.

4.6.8.1 mysqlbinlog Hex Dump Format

The --hexdump option causes mysqlbinlog to produce a hex dump of the binary log contents:

```
shell> mysqlbinlog --hexdump master-bin.000001
```

The hex output consists of comment lines beginning with #, so the output might look like this for the preceding command:

```
/*!40019 SET @@SESSION.max_insert_delayed_threads=0*/;
/*!50003 SET @OLD_COMPLETION_TYPE=@@COMPLETION_TYPE,COMPLETION_TYPE=0*/;
# at 4
#051024 17:24:13 server id 1  end_log_pos 98
# Position  Timestamp   Type   Master ID        Size      Master Pos    Flags
# 00000004 9d fc 5c 43   0f   01 00 00 00   5e 00 00 00   62 00 00 00   00 00
# 00000017 04 00 35 2e 30 2e 31 35  2d 64 65 62 75 67 2d 6c |..5.0.15.debug.l|
# 00000027 6f 67 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |og..............|
# 00000037 00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00 |................|
# 00000047 00 00 00 00 9d fc 5c 43  13 38 0d 00 08 00 12 00 |.......C.8......|
# 00000057 04 04 04 04 12 00 00 4b  00 04 1a               |.......K...|
#       Start: binlog v 4, server v 5.0.15-debug-log created 051024 17:24:13
#       at startup
ROLLBACK;
```

Hex dump output currently contains the elements in the following list. This format is subject to change. For more information about binary log format, see MySQL Internals: The Binary Log.

- Position: The byte position within the log file.
- Timestamp: The event timestamp. In the example shown, '9d fc 5c 43' is the representation of '051024 17:24:13' in hexadecimal.

- `Type`: The event type code.
- `Master ID`: The server ID of the source that created the event.
- `Size`: The size in bytes of the event.
- `Master Pos`: The position of the next event in the original source log file.
- `Flags`: Event flag values.

4.6.8.2 mysqlbinlog Row Event Display

The following examples illustrate how `mysqlbinlog` displays row events that specify data modifications. These correspond to events with the `WRITE_ROWS_EVENT`, `UPDATE_ROWS_EVENT`, and `DELETE_ROWS_EVENT` type codes. The `--base64-output=DECODE-ROWS` and `--verbose` options may be used to affect row event output.

Suppose that the server is using row-based binary logging and that you execute the following sequence of statements:

```
CREATE TABLE t
(
  id   INT NOT NULL,
  name VARCHAR(20) NOT NULL,
  date DATE NULL
) ENGINE = InnoDB;

START TRANSACTION;
INSERT INTO t VALUES(1, 'apple', NULL);
UPDATE t SET name = 'pear', date = '2009-01-01' WHERE id = 1;
DELETE FROM t WHERE id = 1;
COMMIT;
```

By default, `mysqlbinlog` displays row events encoded as base-64 strings using `BINLOG` statements. Omitting extraneous lines, the output for the row events produced by the preceding statement sequence looks like this:

```
shell> mysqlbinlog log_file
...
# at 218
#080828 15:03:08 server id 1  end_log_pos 258   Write_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAANoAAAAAABEAAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBcBAAAAKAAAAAIBAAAQABEAAAAAAAAEAA//8AQAAAAVhcHBsZQ==
'/*!*/;
...
# at 302
#080828 15:03:08 server id 1  end_log_pos 356   Update_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAAC4BAAAAABEAAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBgBAAAANgAAAGQBAAAQABEAAAAAAAAEAA////AEAAAAFYXBwbGX4AQAAAARwZWFyIbIP
'/*!*/;
...
# at 400
#080828 15:03:08 server id 1  end_log_pos 442   Delete_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAAJABAAAAABEAAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBkBAAAAKgAAALoBAAAQABEAAAAAAAAEAA//4AQAAAARwZWFyIbIP
'/*!*/;
```

To see the row events as comments in the form of “pseudo-SQL” statements, run `mysqlbinlog` with the `--verbose` or `-v` option. The output contains lines beginning with `###`:

```
shell> mysqlbinlog -v log_file
...
# at 218
```

```
#080828 15:03:08 server id 1  end_log_pos 258   Write_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAANoAAAAAABEAAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBcBAAAAKAAAAAIBAAAQABEAAAAAAAEAA//8AQAAAAVhcHBsZQ==
'/*!*/;
### INSERT INTO test.t
### SET
###   @1=1
###   @2='apple'
###   @3=NULL
...
# at 302
#080828 15:03:08 server id 1  end_log_pos 356   Update_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAAC4BAAAAABEAAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBgBAAAANgAAAGQBAAAQABEAAAAAAAEAA////AEAAAAFYXBwbGX4AQAAAARwZWFyIbIP
'/*!*/;
### UPDATE test.t
### WHERE
###   @1=1
###   @2='apple'
###   @3=NULL
### SET
###   @1=1
###   @2='pear'
###   @3='2009:01:01'
...
# at 400
#080828 15:03:08 server id 1  end_log_pos 442   Delete_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAAJABAAAAABEAAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBkBAAAAKgAAALoBAAAQABEAAAAAAAEAA//4AQAAAARwZWFyIbIP
'/*!*/;
### DELETE FROM test.t
### WHERE
###   @1=1
###   @2='pear'
###   @3='2009:01:01'
```

Specify `--verbose` or `-v` twice to also display data types and some metadata for each column. The output contains an additional comment following each column change:

```
shell> mysqlbinlog -vv log_file
...
# at 218
#080828 15:03:08 server id 1  end_log_pos 258   Write_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAANoAAAAAABEAAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBcBAAAAKAAAAAIBAAAQABEAAAAAAAEAA//8AQAAAAVhcHBsZQ==
'/*!*/;
### INSERT INTO test.t
### SET
###   @1=1 /* INT meta=0 nullable=0 is_null=0 */
###   @2='apple' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
###   @3=NULL /* VARSTRING(20) meta=0 nullable=1 is_null=1 */
...
# at 302
#080828 15:03:08 server id 1  end_log_pos 356   Update_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAAC4BAAAAABEAAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBgBAAAANgAAAGQBAAAQABEAAAAAAAEAA////AEAAAAFYXBwbGX4AQAAAARwZWFyIbIP
'/*!*/;
### UPDATE test.t
### WHERE
###   @1=1 /* INT meta=0 nullable=0 is_null=0 */
###   @2='apple' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
###   @3=NULL /* VARSTRING(20) meta=0 nullable=1 is_null=1 */
```

```
### SET
###   @1=1 /* INT meta=0 nullable=0 is_null=0 */
###   @2='pear' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
###   @3='2009:01:01' /* DATE meta=0 nullable=1 is_null=0 */
...
# at 400
#080828 15:03:08 server id 1  end_log_pos 442   Delete_rows: table id 17 flags: STMT_END_F

BINLOG '
fAS3SBMBAAAALAAAAJABAAAAABEAAAAAAAAAABHRlc3QAAXQAAwMPCgIUAAQ=
fAS3SBkBAAAAKgAAALoBAAAQABEAAAAAAAAEAA//4AQAAAARwZWFyIbIP
'/*!*/;
### DELETE FROM test.t
### WHERE
###   @1=1 /* INT meta=0 nullable=0 is_null=0 */
###   @2='pear' /* VARSTRING(20) meta=20 nullable=0 is_null=0 */
###   @3='2009:01:01' /* DATE meta=0 nullable=1 is_null=0 */
```

You can tell `mysqlbinlog` to suppress the `BINLOG` statements for row events by using the `--base64-output=DECODE-ROWS` option. This is similar to `--base64-output=NEVER` but does not exit with an error if a row event is found. The combination of `--base64-output=DECODE-ROWS` and `--verbose` provides a convenient way to see row events only as SQL statements:

```
shell> mysqlbinlog -v --base64-output=DECODE-ROWS log_file
...
# at 218
#080828 15:03:08 server id 1  end_log_pos 258   Write_rows: table id 17 flags: STMT_END_F
### INSERT INTO test.t
### SET
###   @1=1
###   @2='apple'
###   @3=NULL
...
# at 302
#080828 15:03:08 server id 1  end_log_pos 356   Update_rows: table id 17 flags: STMT_END_F
### UPDATE test.t
### WHERE
###   @1=1
###   @2='apple'
###   @3=NULL
### SET
###   @1=1
###   @2='pear'
###   @3='2009:01:01'
...
# at 400
#080828 15:03:08 server id 1  end_log_pos 442   Delete_rows: table id 17 flags: STMT_END_F
### DELETE FROM test.t
### WHERE
###   @1=1
###   @2='pear'
###   @3='2009:01:01'
```

Note

You should not suppress `BINLOG` statements if you intend to re-execute `mysqlbinlog` output.

The SQL statements produced by `--verbose` for row events are much more readable than the corresponding `BINLOG` statements. However, they do not correspond exactly to the original SQL statements that generated the events. The following limitations apply:

- The original column names are lost and replaced by *`@N`*, where *`N`* is a column number.
- Character set information is not available in the binary log, which affects string column display:
  - There is no distinction made between corresponding binary and nonbinary string types (`BINARY` and `CHAR`, `VARBINARY` and `VARCHAR`, `BLOB` and `TEXT`). The output uses a data type of `STRING` for fixed-length strings and `VARSTRING` for variable-length strings.

- For multibyte character sets, the maximum number of bytes per character is not present in the binary log, so the length for string types is displayed in bytes rather than in characters. For example, `STRING(4)` is used as the data type for values from either of these column types:

```
CHAR(4) CHARACTER SET latin1
CHAR(2) CHARACTER SET ucs2
```

- Due to the storage format for events of type `UPDATE_ROWS_EVENT`, `UPDATE` statements are displayed with the `WHERE` clause preceding the `SET` clause.

Proper interpretation of row events requires the information from the format description event at the beginning of the binary log. Because `mysqlbinlog` does not know in advance whether the rest of the log contains row events, by default it displays the format description event using a `BINLOG` statement in the initial part of the output.

If the binary log is known not to contain any events requiring a `BINLOG` statement (that is, no row events), the `--base64-output=NEVER` option can be used to prevent this header from being written.

4.6.8.3 Using mysqlbinlog to Back Up Binary Log Files

By default, `mysqlbinlog` reads binary log files and displays their contents in text format. This enables you to examine events within the files more easily and to re-execute them (for example, by using the output as input to `mysql`). `mysqlbinlog` can read log files directly from the local file system, or, with the `--read-from-remote-server` option, it can connect to a server and request binary log contents from that server. `mysqlbinlog` writes text output to its standard output, or to the file named as the value of the `--result-file=`*`file_name`* option if that option is given.

- mysqlbinlog Backup Capabilities
- mysqlbinlog Backup Options
- Static and Live Backups
- Output File Naming
- Example: mysqldump + mysqlbinlog for Backup and Restore
- mysqlbinlog Backup Restrictions

mysqlbinlog Backup Capabilities

`mysqlbinlog` can read binary log files and write new files containing the same content—that is, in binary format rather than text format. This capability enables you to easily back up a binary log in its original format. `mysqlbinlog` can make a static backup, backing up a set of log files and stopping when the end of the last file is reached. It can also make a continuous (“live”) backup, staying connected to the server when it reaches the end of the last log file and continuing to copy new events as they are generated. In continuous-backup operation, `mysqlbinlog` runs until the connection ends (for example, when the server exits) or `mysqlbinlog` is forcibly terminated. When the connection ends, `mysqlbinlog` does not wait and retry the connection, unlike a replica server. To continue a live backup after the server has been restarted, you must also restart `mysqlbinlog`.

mysqlbinlog Backup Options

Binary log backup requires that you invoke `mysqlbinlog` with two options at minimum:

- The `--read-from-remote-server` (or `-R`) option tells `mysqlbinlog` to connect to a server and request its binary log. (This is similar to a replica server connecting to its source server.)
- The `--raw` option tells `mysqlbinlog` to write raw (binary) output, not text output.

Along with `--read-from-remote-server`, it is common to specify other options: `--host` indicates where the server is running, and you may also need to specify connection options such as `--user` and `--password`.

Several other options are useful in conjunction with `--raw`:

- `--stop-never`: Stay connected to the server after reaching the end of the last log file and continue to read new events.
- `--stop-never-slave-server-id=id`: The server ID that `mysqlbinlog` reports to the server when `--stop-never` is used. The default is 65535. This can be used to avoid a conflict with the ID of a replica server or another `mysqlbinlog` process. See Section 4.6.8.4, "Specifying the mysqlbinlog Server ID".
- `--result-file`: A prefix for output file names, as described later.

Static and Live Backups

To back up a server's binary log files with `mysqlbinlog`, you must specify file names that actually exist on the server. If you do not know the names, connect to the server and use the `SHOW BINARY LOGS` statement to see the current names. Suppose that the statement produces this output:

```
mysql> SHOW BINARY LOGS;
+---------------+-----------+
| Log_name      | File_size |
+---------------+-----------+
| binlog.000130 |     27459 |
| binlog.000131 |     13719 |
| binlog.000132 |     43268 |
+---------------+-----------+
```

With that information, you can use `mysqlbinlog` to back up the binary log to the current directory as follows (enter each command on a single line):

- To make a static backup of `binlog.000130` through `binlog.000132`, use either of these commands:

  ```
  mysqlbinlog --read-from-remote-server --host=host_name --raw
    binlog.000130 binlog.000131 binlog.000132

  mysqlbinlog --read-from-remote-server --host=host_name --raw
    --to-last-log binlog.000130
  ```

  The first command specifies every file name explicitly. The second names only the first file and uses `--to-last-log` to read through the last. A difference between these commands is that if the server happens to open `binlog.000133` before `mysqlbinlog` reaches the end of `binlog.000132`, the first command does not read it, but the second command does.

- To make a live backup in which `mysqlbinlog` starts with `binlog.000130` to copy existing log files, then stays connected to copy new events as the server generates them:

  ```
  mysqlbinlog --read-from-remote-server --host=host_name --raw
    --stop-never binlog.000130
  ```

  With `--stop-never`, it is not necessary to specify `--to-last-log` to read to the last log file because that option is implied.

Output File Naming

Without `--raw`, `mysqlbinlog` produces text output and the `--result-file` option, if given, specifies the name of the single file to which all output is written. With `--raw`, `mysqlbinlog` writes one binary output file for each log file transferred from the server. By default, `mysqlbinlog` writes the files in the current directory with the same names as the original log files. To modify the output file names, use the `--result-file` option. In conjunction with `--raw`, the `--result-file` option value is treated as a prefix that modifies the output file names.

Suppose that a server currently has binary log files named `binlog.000999` and up. If you use `mysqlbinlog --raw` to back up the files, the `--result-file` option produces output file names as

shown in the following table. You can write the files to a specific directory by beginning the `--result-file` value with the directory path. If the `--result-file` value consists only of a directory name, the value must end with the pathname separator character. Output files are overwritten if they exist.

`--result-file` Option	**Output File Names**
`--result-file=x`	`xbinlog.000999` and up
`--result-file=/tmp/`	`/tmp/binlog.000999` and up
`--result-file=/tmp/x`	`/tmp/xbinlog.000999` and up

Example: mysqldump + mysqlbinlog for Backup and Restore

The following example describes a simple scenario that shows how to use `mysqldump` and `mysqlbinlog` together to back up a server's data and binary log, and how to use the backup to restore the server if data loss occurs. The example assumes that the server is running on host *`host_name`* and its first binary log file is named `binlog.000999`. Enter each command on a single line.

Use `mysqlbinlog` to make a continuous backup of the binary log:

```
mysqlbinlog --read-from-remote-server --host=host_name --raw
  --stop-never binlog.000999
```

Use `mysqldump` to create a dump file as a snapshot of the server's data. Use `--all-databases`, `--events`, and `--routines` to back up all data, and `--master-data=2` to include the current binary log coordinates in the dump file.

```
mysqldump --host=host_name --all-databases --events --routines --master-data=2> dump_file
```

Execute the `mysqldump` command periodically to create newer snapshots as desired.

If data loss occurs (for example, if the server unexpectedly exits), use the most recent dump file to restore the data:

```
mysql --host=host_name -u root -p < dump_file
```

Then use the binary log backup to re-execute events that were written after the coordinates listed in the dump file. Suppose that the coordinates in the file look like this:

```
-- CHANGE MASTER TO MASTER_LOG_FILE='binlog.001002', MASTER_LOG_POS=27284;
```

If the most recent backed-up log file is named `binlog.001004`, re-execute the log events like this:

```
mysqlbinlog --start-position=27284 binlog.001002 binlog.001003 binlog.001004
  | mysql --host=host_name -u root -p
```

You might find it easier to copy the backup files (dump file and binary log files) to the server host to make it easier to perform the restore operation, or if MySQL does not allow remote `root` access.

mysqlbinlog Backup Restrictions

Binary log backups with `mysqlbinlog` are subject to these restrictions:

- `mysqlbinlog` does not automatically reconnect to the MySQL server if the connection is lost (for example, if a server restart occurs or there is a network outage).
- Prior to MySQL 5.6.37, `mysqlbinlog` does not get all events as they are committed, even if the server is configured with `sync_binlog=1`. This means that some of the most recent events may be missing. To ensure that `mysqlbinlog` sees the most recent events, flush the binary log on the server that you are backing up.
- The delay for a backup is similar to the delay for a replica.

4.6.8.4 Specifying the mysqlbinlog Server ID

When invoked with the `--read-from-remote-server` option, `mysqlbinlog` connects to a MySQL server, specifies a server ID to identify itself, and requests binary log files from the server. You can use `mysqlbinlog` to request log files from a server in several ways:

- Specify an explicitly named set of files: For each file, `mysqlbinlog` connects and issues a `Binlog dump` command. The server sends the file and disconnects. There is one connection per file.
- Specify the beginning file and `--to-last-log`: `mysqlbinlog` connects and issues a `Binlog dump` command for all files. The server sends all files and disconnects.
- Specify the beginning file and `--stop-never` (which implies `--to-last-log`): `mysqlbinlog` connects and issues a `Binlog dump` command for all files. The server sends all files, but does not disconnect after sending the last one.

With `--read-from-remote-server` only, `mysqlbinlog` connects using a server ID of 0, which tells the server to disconnect after sending the last requested log file.

With `--read-from-remote-server` and `--stop-never`, `mysqlbinlog` connects using a nonzero server ID, so the server does not disconnect after sending the last log file. The server ID is 65535 by default, but this can be changed with `--stop-never-slave-server-id`.

Thus, for the first two ways of requesting files, the server disconnects because `mysqlbinlog` specifies a server ID of 0. It does not disconnect if `--stop-never` is given because `mysqlbinlog` specifies a nonzero server ID.

4.6.9 `mysqldumpslow` — Summarize Slow Query Log Files

The MySQL slow query log contains information about queries that take a long time to execute (see Section 5.4.5, “The Slow Query Log”). `mysqldumpslow` parses MySQL slow query log files and summarizes their contents.

Normally, `mysqldumpslow` groups queries that are similar except for the particular values of number and string data values. It “abstracts” these values to `N` and `'S'` when displaying summary output. To modify value abstracting behavior, use the `-a` and `-n` options.

Invoke `mysqldumpslow` like this:

```
shell> mysqldumpslow [options] [log_file ...]
```

Example of usage:

```
shell> mysqldumpslow

Reading mysql slow query log from /usr/local/mysql/data/mysqld56-slow.log
Count: 1  Time=4.32s (4s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@localhost
 insert into t2 select * from t1

Count: 3  Time=2.53s (7s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@localhost
 insert into t2 select * from t1 limit N

Count: 3  Time=2.13s (6s)  Lock=0.00s (0s)  Rows=0.0 (0), root[root]@localhost
 insert into t1 select * from t1
```

`mysqldumpslow` supports the following options.

Table 4.21 mysqldumpslow Options

Option Name	Description
-a	Do not abstract all numbers to N and strings to 'S'

Option Name	Description
-n	Abstract numbers with at least the specified digits
--debug	Write debugging information
-g	Only consider statements that match the pattern
--help	Display help message and exit
-h	Host name of the server in the log file name
-i	Name of the server instance
-l	Do not subtract lock time from total time
-r	Reverse the sort order
-s	How to sort output
-t	Display only first num queries
--verbose	Verbose mode

- `--help`

  Display a help message and exit.

- `-a`

  Do not abstract all numbers to `N` and strings to `'S'`.

- `--debug`, `-d`

  Run in debug mode.

  This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are *not* built using this option.

- `-g` *`pattern`*

  Consider only queries that match the (`grep`-style) pattern.

- `-h` *`host_name`*

  Host name of MySQL server for `*-slow.log` file name. The value can contain a wildcard. The default is `*` (match all).

- `-i` *`name`*

  Name of server instance (if using `mysql.server` startup script).

- `-l`

  Do not subtract lock time from total time.

- `-n` *`N`*

  Abstract numbers with at least *`N`* digits within names.

- `-r`

  Reverse the sort order.

- `-s` *`sort_type`*

  How to sort the output. The value of *`sort_type`* should be chosen from the following list:

  - `t`, `at`: Sort by query time or average query time

- l, al: Sort by lock time or average lock time
- r, ar: Sort by rows sent or average rows sent
- c: Sort by count

By default, mysqldumpslow sorts by average query time (equivalent to -s at).

- -t N

Display only the first N queries in the output.

- --verbose, -v

Verbose mode. Print more information about what the program does.

4.6.10 mysqlhotcopy — A Database Backup Program

Note

This utility is deprecated in MySQL 5.6.20 and removed in MySQL 5.7

mysqlhotcopy is a Perl script that was originally written and contributed by Tim Bunce. It uses FLUSH TABLES, LOCK TABLES, and cp or scp to make a database backup. It is a fast way to make a backup of the database or single tables, but it can be run only on the same machine where the database directories are located. mysqlhotcopy works only for backing up MyISAM and ARCHIVE tables. It runs on Unix.

To use mysqlhotcopy, you must have read access to the files for the tables that you are backing up, the SELECT privilege for those tables, the RELOAD privilege (to be able to execute FLUSH TABLES), and the LOCK TABLES privilege (to be able to lock the tables).

```
shell> mysqlhotcopy db_name [/path/to/new_directory]
```

```
shell> mysqlhotcopy db_name_1 ... db_name_n /path/to/new_directory
```

Back up tables in the given database that match a regular expression:

```
shell> mysqlhotcopy db_name./regex/
```

The regular expression for the table name can be negated by prefixing it with a tilde (~):

```
shell> mysqlhotcopy db_name./~regex/
```

mysqlhotcopy supports the following options, which can be specified on the command line or in the [mysqlhotcopy] and [client] groups of an option file. For information about option files used by MySQL programs, see Section 4.2.2.2, “Using Option Files”.

Table 4.22 mysqlhotcopy Options

Option Name	Description
--addtodest	Do not rename target directory (if it exists); merely add files to it
--allowold	Do not abort if a target exists; rename it by adding an _old suffix
--checkpoint	Insert checkpoint entries
--chroot	Base directory of the chroot jail in which mysqld operates
--debug	Write debugging log
--dryrun	Report actions without performing them
--flushlog	Flush logs after all tables are locked

Option Name	Description
--help	Display help message and exit
--host	Host on which MySQL server is located
--keepold	Do not delete previous (renamed) target when done
--method	The method for copying files
--noindices	Do not include full index files in the backup
--old_server	Connect to server that does not support FLUSH TABLES tbl_list WITH READ LOCK
--password	Password to use when connecting to server
--port	TCP/IP port number for connection
--quiet	Be silent except for errors
--regexp	Copy all databases with names that match the given regular expression
--resetmaster	Reset the binary log after locking all the tables
--resetslave	Reset the master.info file after locking all the tables
--socket	Unix socket file or Windows named pipe to use
--tmpdir	The temporary directory
--user	MySQL user name to use when connecting to server

- `--help`, `-?`

  Display a help message and exit.

- `--addtodest`

  Do not rename target directory (if it exists); merely add files to it.

- `--allowold`

  Do not abort if a target exists; rename it by adding an `_old` suffix.

- `--checkpoint=`*`db_name.tbl_name`*

  Insert checkpoint entries into the specified database *`db_name`* and table *`tbl_name`*.

- `--chroot=`*`dir_name`*

  Base directory of the `chroot` jail in which `mysqld` operates. The *`dir_name`* value should match that of the `--chroot` option given to `mysqld`.

- `--debug`

  Enable debug output.

  This option is available only if MySQL was built using `WITH_DEBUG`. MySQL release binaries provided by Oracle are *not* built using this option.

- `--dryrun`, `-n`

  Report actions without performing them.

- `--flushlog`

  Flush logs after all tables are locked.

- `--host=`*`host_name`*, `-h` *`host_name`*

The host name of the local host to use for making a TCP/IP connection to the local server. By default, the connection is made to `localhost` using a Unix socket file.

- `--keepold`

  Do not delete previous (renamed) target when done.

- `--method=command`

  The method for copying files (`cp` or `scp`). The default is `cp`.

- `--noindices`

  Do not include full index files for `MyISAM` tables in the backup. This makes the backup smaller and faster. The indexes for reloaded tables can be reconstructed later with `myisamchk -rq`.

- `--password=password`, `-ppassword`

  The password to use when connecting to the server. The password value is not optional for this option, unlike for other MySQL programs.

  Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

- `--port=port_num`, `-P port_num`

  The TCP/IP port number to use when connecting to the local server.

- `--old_server`

  In MySQL 5.6, `mysqlhotcopy` uses `FLUSH TABLES tbl_list WITH READ LOCK` to flush and lock tables. Use the `--old_server` option if the server is older than 5.5.3, which is when that statement was introduced.

- `--quiet`, `-q`

  Be silent except for errors.

- `--record_log_pos=db_name.tbl_name`

  Record source and replica status in the specified database *`db_name`* and table *`tbl_name`*.

- `--regexp=expr`

  Copy all databases with names that match the given regular expression.

- `--resetmaster`

  Reset the binary log after locking all the tables.

- `--resetslave`

  Reset the source info repository file or table after locking all the tables.

- `--socket=path`, `-S path`

  The Unix socket file to use for connections to `localhost`.

- `--suffix=str`

  The suffix to use for names of copied databases.

- `--tmpdir=dir_name`

  The temporary directory. The default is `/tmp`.

- `--user=user_name`, `-u user_name`

  The MySQL user name to use when connecting to the server.

Use `perldoc` for additional `mysqlhotcopy` documentation, including information about the structure of the tables needed for the `--checkpoint` and `--record_log_pos` options:

```
shell> perldoc mysqlhotcopy
```

4.6.11 mysql_convert_table_format — Convert Tables to Use a Given Storage Engine

Note

This utility is deprecated in MySQL 5.6.17 and removed in MySQL 5.7

`mysql_convert_table_format` converts the tables in a database to use a particular storage engine (`MyISAM` by default). `mysql_convert_table_format` is written in Perl and requires that the `DBI` and `DBD::mysql` Perl modules be installed (see Section 2.13, “Perl Installation Notes”).

Invoke `mysql_convert_table_format` like this:

```
shell> mysql_convert_table_format [options]db_name
```

The *db_name* argument indicates the database containing the tables to be converted.

`mysql_convert_table_format` supports the options described in the following list.

- `--help`

  Display a help message and exit.

- `--force`

  Continue even if errors occur.

- `--host=host_name`

  Connect to the MySQL server on the given host.

- `--password=password`

  The password to use when connecting to the server. The password value is not optional for this option, unlike for other MySQL programs.

  Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

- `--port=port_num`

  The TCP/IP port number to use for the connection.

- `--socket=path`

  For connections to `localhost`, the Unix socket file to use.

- `--type=engine_name`

Specify the storage engine that the tables should be converted to use. The default is `MyISAM` if this option is not given.

- `--user=user_name`

  The MySQL user name to use when connecting to the server.

- `--verbose`

  Verbose mode. Print more information about what the program does.

- `--version`

  Display version information and exit.

4.6.12 mysql_find_rows — Extract SQL Statements from Files

Note

This utility is deprecated in MySQL 5.6.17 and removed in MySQL 5.7

`mysql_find_rows` reads files containing SQL statements and extracts statements that match a given regular expression or that contain `USE db_name` or `SET` statements. The utility expects statements to be terminated with semicolon (`;`) characters.

Invoke `mysql_find_rows` like this:

```
shell> mysql_find_rows [options] [file_name ...]
```

Each *`file_name`* argument should be the name of file containing SQL statements. If no file names are given, `mysql_find_rows` reads the standard input.

Examples:

```
mysql_find_rows --regexp=problem_table --rows=20 < update.log
mysql_find_rows --regexp=problem_table  update-log.1 update-log.2
```

`mysql_find_rows` supports the following options:

- `--help`, `--Information`

  Display a help message and exit.

- `--regexp=pattern`

  Display queries that match the pattern.

- `--rows=N`

  Quit after displaying *`N`* queries.

- `--skip-use-db`

  Do not include `USE db_name` statements in the output.

- `--start_row=N`

  Start output from this row.

4.6.13 mysql_fix_extensions — Normalize Table File Name Extensions

Note

This utility is deprecated in MySQL 5.6.17 and removed in MySQL 5.7

`mysql_fix_extensions` converts the extensions for `MyISAM` (or `ISAM`) table files to their canonical forms. It looks for files with extensions matching any lettercase variant of `.frm`, `.myd`, `.myi`, `.isd`, and `.ism` and renames them to have extensions of `.frm`, `.MYD`, `.MYI`, `.ISD`, and `.ISM`, respectively. This can be useful after transferring the files from a system with case-insensitive file names (such as Windows) to a system with case-sensitive file names.

Invoke `mysql_fix_extensions` like this, where *`data_dir`* is the path name to the MySQL data directory.

```
shell> mysql_fix_extensions data_dir
```

4.6.14 `mysql_setpermission` — Interactively Set Permissions in Grant Tables

Note

This utility is deprecated in MySQL 5.6.17 and removed in MySQL 5.7

`mysql_setpermission` is a Perl script that was originally written and contributed by Luuk de Boer. It interactively sets permissions in the MySQL grant tables. `mysql_setpermission` is written in Perl and requires that the `DBI` and `DBD::mysql` Perl modules be installed (see Section 2.13, “Perl Installation Notes”).

Invoke `mysql_setpermission` like this:

```
shell> mysql_setpermission [options]
```

`options` should be either `--help` to display the help message, or options that indicate how to connect to the MySQL server. The account used when you connect determines which permissions you have when attempting to modify existing permissions in the grant tables.

`mysql_setpermissions` also reads options from the `[client]` and `[perl]` groups in the `.my.cnf` file in your home directory, if the file exists.

`mysql_setpermission` supports the following options:

- `--help`

  Display a help message and exit.

- `--host=`*`host_name`*

  Connect to the MySQL server on the given host.

- `--password=`*`password`*

  The password to use when connecting to the server. The password value is not optional for this option, unlike for other MySQL programs.

  Specifying a password on the command line should be considered insecure. To avoid giving the password on the command line, use an option file. See Section 6.1.2.1, “End-User Guidelines for Password Security”.

- `--port=`*`port_num`*

  The TCP/IP port number to use for the connection.

- `--socket=path`

  For connections to `localhost`, the Unix socket file to use.

- `--user=user_name`

  The MySQL user name to use when connecting to the server.

4.6.15 mysql_waitpid — Kill Process and Wait for Its Termination

`mysql_waitpid` signals a process to terminate and waits for the process to exit. It uses the `kill()` system call and Unix signals, so it runs on Unix and Unix-like systems.

This program is deprecated as of MySQL 5.6.19 and is removed in MySQL 5.7.

Invoke `mysql_waitpid` like this:

```
shell> mysql_waitpid [options] pid wait_time
```

`mysql_waitpid` sends signal 0 to the process identified by *`pid`* and waits up to *`wait_time`* seconds for the process to terminate. *`pid`* and *`wait_time`* must be positive integers.

If process termination occurs within the wait time or the process does not exist, `mysql_waitpid` returns 0. Otherwise, it returns 1.

If the `kill()` system call cannot handle signal 0, `mysql_waitpid()` uses signal 1 instead.

`mysql_waitpid` supports the following options:

- `--help`, `-?`, `-I`

  Display a help message and exit.

- `--verbose`, `-v`

  Verbose mode. Display a warning if signal 0 could not be used and signal 1 is used instead.

- `--version`, `-V`

  Display version information and exit.

4.6.16 mysql_zap — Kill Processes That Match a Pattern

`mysql_zap` kills processes that match a pattern. It uses the `ps` command and Unix signals, so it runs on Unix and Unix-like systems.

This program is deprecated as of MySQL 5.6.19 and is removed in MySQL 5.7.

Invoke `mysql_zap` like this:

```
shell> mysql_zap [-signal] [-?Ift] pattern
```

A process matches if its output line from the `ps` command contains the pattern. By default, `mysql_zap` asks for confirmation for each process. Respond `y` to kill the process, or `q` to exit `mysql_zap`. For any other response, `mysql_zap` does not attempt to kill the process.

If the `-signal` option is given, it specifies the name or number of the signal to send to each process. Otherwise, `mysql_zap` tries first with `TERM` (signal 15) and then with `KILL` (signal 9).

`mysql_zap` supports the following additional options:

- `--help`, `-?`, `-I`

  Display a help message and exit.

- `-f`

  Force mode. `mysql_zap` attempts to kill each process without confirmation.

- `-t`

  Test mode. Display information about each process but do not kill it.

4.7 Program Development Utilities

This section describes some utilities that you may find useful when developing MySQL programs.

In shell scripts, you can use the `my_print_defaults` program to parse option files and see what options would be used by a given program. The following example shows the output that `my_print_defaults` might produce when asked to show the options found in the `[client]` and `[mysql]` groups:

```
shell> my_print_defaults client mysql
--port=3306
--socket=/tmp/mysql.sock
--no-auto-rehash
```

Note

For developers: Option file handling is implemented in the C client library simply by processing all options in the appropriate group or groups before any command-line arguments. This works well for programs that use the last instance of an option that is specified multiple times. If you have a C or C++ program that handles multiply specified options this way but that doesn't read option files, you need add only two lines to give it that capability. Check the source code of any of the standard MySQL clients to see how to do this.

Several other language interfaces to MySQL are based on the C client library, and some of them provide a way to access option file contents. These include Perl and Python. For details, see the documentation for your preferred interface.

4.7.1 `msql2mysql` — Convert mSQL Programs for Use with MySQL

Note

This utility is deprecated in MySQL 5.6.17 and removed in MySQL 5.7

Initially, the MySQL C API was developed to be very similar to that for the mSQL database system. Because of this, mSQL programs often can be converted relatively easily for use with MySQL by changing the names of the C API functions.

The `msql2mysql` utility performs the conversion of mSQL C API function calls to their MySQL equivalents. `msql2mysql` converts the input file in place, so make a copy of the original before converting it. For example, use `msql2mysql` like this:

```
shell> cp client-prog.c client-prog.c.orig
shell> msql2mysql client-prog.c
client-prog.c converted
```

Then examine `client-prog.c` and make any post-conversion revisions that may be necessary.

`msql2mysql` uses the `replace` utility to make the function name substitutions. See Section 4.8.2, “`replace` — A String-Replacement Utility”.

4.7.2 `mysql_config` — Display Options for Compiling Clients

mysql_config provides you with useful information for compiling your MySQL client and connecting it to MySQL. It is a shell script, so it is available only on Unix and Unix-like systems.

mysql_config supports the following options.

- --cflags

  C Compiler flags to find include files and critical compiler flags and defines used when compiling the libmysqlclient library. The options returned are tied to the specific compiler that was used when the library was created and might clash with the settings for your own compiler. Use --include for more portable options that contain only include paths.

- --cxxflags

  Like --cflags, but for C++ compiler flags. This option was added in MySQL 5.6.4.

- --include

  Compiler options to find MySQL include files.

- --libmysqld-libs, --embedded-libs, --embedded

  Libraries and options required to link with libmysqld, the MySQL embedded server.

- --libs

  Libraries and options required to link with the MySQL client library.

- --libs_r

  Libraries and options required to link with the thread-safe MySQL client library. In MySQL 5.6, all client libraries are thread-safe, so this option need not be used. The --libs option can be used in all cases.

- --plugindir

  The default plugin directory path name, defined when configuring MySQL.

- --port

  The default TCP/IP port number, defined when configuring MySQL.

- --socket

  The default Unix socket file, defined when configuring MySQL.

- --variable=*var_name*

  Display the value of the named configuration variable. Permitted *var_name* values are pkgincludedir (the header file directory), pkglibdir (the library directory), and plugindir (the plugin directory).

- --version

  Version number for the MySQL distribution.

If you invoke mysql_config with no options, it displays a list of all options that it supports, and their values:

```
shell> mysql_config
Usage: /usr/local/mysql/bin/mysql_config [options]
Options:
  --cflags         [-I/usr/local/mysql/include/mysql -mcpu=pentiumpro]
  --cxxflags       [-I/usr/local/mysql/include/mysql -mcpu=pentiumpro]
```

```
  --include        [-I/usr/local/mysql/include/mysql]
  --libs           [-L/usr/local/mysql/lib/mysql -lmysqlclient
                    -lpthread -lm -lrt -lssl -lcrypto -ldl]
  --libs_r         [-L/usr/local/mysql/lib/mysql -lmysqlclient_r
                    -lpthread -lm -lrt -lssl -lcrypto -ldl]
  --plugindir      [/usr/local/mysql/lib/plugin]
  --socket         [/tmp/mysql.sock]
  --port           [3306]
  --version        [5.6.11]
  --libmysqld-libs [-L/usr/local/mysql/lib/mysql -lmysqld
                    -lpthread -lm -lrt -lssl -lcrypto -ldl -lcrypt]
  --variable=VAR   VAR is one of:
          pkgincludedir [/usr/local/mysql/include]
          pkglibdir     [/usr/local/mysql/lib]
          plugindir     [/usr/local/mysql/lib/plugin]
```

You can use `mysql_config` within a command line using backticks to include the output that it produces for particular options. For example, to compile and link a MySQL client program, use `mysql_config` as follows:

```
gcc -c `mysql_config --cflags` progname.c
gcc -o progname progname.o `mysql_config --libs`
```

4.7.3 `my_print_defaults` — Display Options from Option Files

`my_print_defaults` displays the options that are present in option groups of option files. The output indicates which options are used by programs that read the specified option groups. For example, the `mysqlcheck` program reads the `[mysqlcheck]` and `[client]` option groups. To see what options are present in those groups in the standard option files, invoke `my_print_defaults` like this:

```
shell> my_print_defaults mysqlcheck client
--user=myusername
--password=password
--host=localhost
```

The output consists of options, one per line, in the form that they would be specified on the command line.

`my_print_defaults` supports the following options.

- `--help`, `-?`

  Display a help message and exit.

- `--config-file=file_name`, `--defaults-file=file_name`, `-c file_name`

  Read only the given option file.

- `--debug=debug_options`, `-# debug_options`

  Write a debugging log. A typical `debug_options` string is `d:t:o,file_name`. The default is `d:t:o,/tmp/my_print_defaults.trace`.

- `--defaults-extra-file=file_name`, `--extra-file=file_name`, `-e file_name`

  Read this option file after the global option file but (on Unix) before the user option file.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-group-suffix=suffix`, `-g suffix`

  In addition to the groups named on the command line, read groups that have the given suffix.

  For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--login-path=name`, `-l name`

  Read options from the named login path in the `.mylogin.cnf` login path file. A "login path" is an option group containing options that specify which MySQL server to connect to and which account to authenticate as. To create or modify a login path file, use the `mysql_config_editor` utility. See Section 4.6.6, "`mysql_config_editor` — MySQL Configuration Utility". This option was added in MySQL 5.6.6.

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- `--no-defaults`, `-n`

  Return an empty string.

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- `--show`, `-s`

  As of MySQL 5.6.25, `my_print_defaults` masks passwords by default. Use this option to display passwords in cleartext.

- `--verbose`, `-v`

  Verbose mode. Print more information about what the program does.

- `--version`, `-V`

  Display version information and exit.

4.7.4 `resolve_stack_dump` — Resolve Numeric Stack Trace Dump to Symbols

`resolve_stack_dump` resolves a numeric stack dump to symbols.

Invoke `resolve_stack_dump` like this:

```
shell> resolve_stack_dump [options] symbols_file [numeric_dump_file]
```

The symbols file should include the output from the `nm --numeric-sort mysqld` command. The numeric dump file should contain a numeric stack track from `mysqld`. If no numeric dump file is named on the command line, the stack trace is read from the standard input.

`resolve_stack_dump` supports the following options.

- `--help`, `-h`

  Display a help message and exit.

- `--numeric-dump-file=file_name`, `-n file_name`

  Read the stack trace from the given file.

- `--symbols-file=file_name`, `-s file_name`

  Use the given symbols file.

- `--version`, `-V`

  Display version information and exit.

For more information, see Section 5.8.1.5, "Using a Stack Trace".

4.8 Miscellaneous Programs

4.8.1 `perror` — Display MySQL Error Message Information

For most system errors, MySQL displays, in addition to an internal text message, the system error code in one of the following styles:

```
message ... (errno: #)
message ... (Errcode: #)
```

You can find out what the error code means by examining the documentation for your system or by using the `perror` utility.

`perror` prints a description for a system error code or for a storage engine (table handler) error code.

Invoke `perror` like this:

```
shell> perror [options] errorcode ...
```

Examples:

```
shell> perror 1231
MySQL error code 1231 (ER_WRONG_VALUE_FOR_VAR): Variable '%-.64s' can't
be set to the value of '%-.200s'
```

```
shell> perror 13 64
OS error code  13:  Permission denied
OS error code  64:  Machine is not on the network
```

To obtain the error message for an NDB Cluster error code, invoke `perror` with the `--ndb` option:

```
shell> perror --ndb errorcode
```

The meaning of system error messages may be dependent on your operating system. A given error code may mean different things on different operating systems.

`perror` supports the following options.

- `--help`, `--info`, `-I`, `-?`

  Display a help message and exit.

- `--ndb`

  Print the error message for an NDB Cluster error code.

- `--silent`, `-s`

  Silent mode. Print only the error message.

- `--verbose`, `-v`

  Verbose mode. Print error code and message. This is the default behavior.

- `--version`, `-V`

  Display version information and exit.

4.8.2 `replace` — A String-Replacement Utility

The `replace` utility program changes strings in place in files or on the standard input.

Invoke `replace` in one of the following ways:

```
shell> replace from to [from to] ... -- file_name [file_name] ...
shell> replace from to [from to] ... < file_name
```

from represents a string to look for and *to* represents its replacement. There can be one or more pairs of strings.

Use the `--` option to indicate where the string-replacement list ends and the file names begin. In this case, any file named on the command line is modified in place, so you may want to make a copy of the original before converting it. *replace* prints a message indicating which of the input files it actually modifies.

If the `--` option is not given, `replace` reads the standard input and writes to the standard output.

`replace` uses a finite state machine to match longer strings first. It can be used to swap strings. For example, the following command swaps `a` and `b` in the given files, `file1` and `file2`:

```
shell> replace a b b a -- file1 file2 ...
```

The `replace` program is used by `msql2mysql`. See Section 4.7.1, “`msql2mysql` — Convert mSQL Programs for Use with MySQL”.

`replace` supports the following options.

- `-?`, `-I`

  Display a help message and exit.

- *`-#debug_options`*

  Enable debugging.

- `-s`

  Silent mode. Print less information what the program does.

- `-v`

  Verbose mode. Print more information about what the program does.

- `-V`

  Display version information and exit.

4.8.3 `resolveip` — Resolve Host name to IP Address or Vice Versa

The `resolveip` utility resolves host names to IP addresses and vice versa.

Invoke `resolveip` like this:

```
shell> resolveip [options] {host_name|ip-addr} ...
```

`resolveip` supports the following options.

- `--help`, `--info`, `-?`, `-I`

  Display a help message and exit.

- `--silent`, `-s`

  Silent mode. Produce less output.

- `--version`, `-V`

Display version information and exit.

4.9 Environment Variables

This section lists environment variables that are used directly or indirectly by MySQL. Most of these can also be found in other places in this manual.

Options on the command line take precedence over values specified in option files and environment variables, and values in option files take precedence over values in environment variables. In many cases, it is preferable to use an option file instead of environment variables to modify the behavior of MySQL. See Section 4.2.2.2, “Using Option Files”.

Variable	Description
`AUTHENTICATION_PAM_LOG`	PAM authentication plugin debug logging settings.
`CC`	The name of your C compiler (for running `CMake`).
`CXX`	The name of your C++ compiler (for running `CMake`).
`CC`	The name of your C compiler (for running `CMake`).
`DBI_USER`	The default user name for Perl DBI.
`DBI_TRACE`	Trace options for Perl DBI.
`HOME`	The default path for the `mysql` history file is `$HOME/.mysql_history`.
`LD_RUN_PATH`	Used to specify the location of `libmysqlclient.so`.
`LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN`	Enable `mysql_clear_password` authentication plugin; see Section 6.4.1.5, “Client-Side Cleartext Pluggable Authentication”.
`LIBMYSQL_PLUGIN_DIR`	Directory in which to look for client plugins.
`LIBMYSQL_PLUGINS`	Client plugins to preload.
`MYSQL_DEBUG`	Debug trace options when debugging.
`MYSQL_GROUP_SUFFIX`	Option group suffix value (like specifying `--defaults-group-suffix`).
`MYSQL_HISTFILE`	The path to the `mysql` history file. If this variable is set, its value overrides the default for `$HOME/.mysql_history`.
`MYSQL_HISTIGNORE`	Patterns specifying statements not to log to `$HOME/.mysql_history`.
`MYSQL_HOME`	The path to the directory in which the server-specific `my.cnf` file resides.
`MYSQL_HOST`	The default host name used by the `mysql` command-line client.
`MYSQL_OPENSSL_UDF_DH_BITS_THRESHOLD`	Maximum key length for `create_dh_parameters()`. See Section 6.5.2, “MySQL Enterprise Encryption Usage and Examples”.
`MYSQL_OPENSSL_UDF_DSA_BITS_THRESHOLD`	Maximum DSA key length for `create_asymmetric_priv_key()`. See Section 6.5.2, “MySQL Enterprise Encryption Usage and Examples”.
`MYSQL_OPENSSL_UDF_RSA_BITS_THRESHOLD`	Maximum RSA key length for `create_asymmetric_priv_key()`. See Section 6.5.2, “MySQL Enterprise Encryption Usage and Examples”.
`MYSQL_PS1`	The command prompt to use in the `mysql` command-line client.

Variable	Description
MYSQL_PWD	The default password when connecting to mysqld. Using this is insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”.
MYSQL_TCP_PORT	The default TCP/IP port number.
MYSQL_TEST_LOGIN_FILE	The name of the .mylogin.cnf login path file.
MYSQL_UNIX_PORT	The default Unix socket file name; used for connections to localhost.
PATH	Used by the shell to find MySQL programs.
TMPDIR	The directory in which temporary files are created.
TZ	This should be set to your local time zone. See Section B.3.3.7, “Time Zone Problems”.
UMASK	The user-file creation mode when creating files. See note following table.
UMASK_DIR	The user-directory creation mode when creating directories. See note following table.
USER	The default user name on Windows when connecting to mysqld.

For information about the mysql history file, see Section 4.5.1.3, “mysql Client Logging”.

MYSQL_TEST_LOGIN_FILE is the path name of the login path file (the file created by mysql_config_editor). If not set, the default value is %APPDATA%\MySQL\.mylogin.cnf directory on Windows and $HOME/.mylogin.cnf on non-Windows systems. See Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”.

The default UMASK and UMASK_DIR values are 0660 and 0700, respectively. MySQL assumes that the value for UMASK or UMASK_DIR is in octal if it starts with a zero. For example, setting UMASK=0600 is equivalent to UMASK=384 because 0600 octal is 384 decimal.

The UMASK and UMASK_DIR variables, despite their names, are used as modes, not masks:

- If UMASK is set, mysqld uses ($UMASK | 0600) as the mode for file creation, so that newly created files have a mode in the range from 0600 to 0666 (all values octal).
- If UMASK_DIR is set, mysqld uses ($UMASK_DIR | 0700) as the base mode for directory creation, which then is AND-ed with ~(~$UMASK & 0666), so that newly created directories have a mode in the range from 0700 to 0777 (all values octal). The AND operation may remove read and write permissions from the directory mode, but not execute permissions.

See also Section B.3.3.1, “Problems with File Permissions”.

4.10 Unix Signal Handling in MySQL

On Unix and Unix-like systems, a process can be the recipient of signals sent to it by the root system account or the system account that owns the process. Signals can be sent using the kill command. Some command interpreters associate certain key sequences with signals, such as **Control+C** to send a SIGINT signal. This section describes how the MySQL server and client programs respond to signals.

- Server Response to Signals
- Client Response to Signals

Server Response to Signals

`mysqld` responds to signals as follows:

- `SIGTERM` causes the server to shut down.
- `SIGHUP` causes the server to reload the grant tables and to flush tables, logs, the thread cache, and the host cache. These actions are like various forms of the `FLUSH` statement. Sending the signal enables the flush operations to be performed without having to connect to the server, which requires a MySQL account that has privileges sufficient for those operations. The server also writes a status report to the error log that has this format:

```
Status information:

Current dir: /var/mysql/data/
Running threads: 4  Stack size: 262144
Current locks:
lock: 0x7f742c02c0e0:

lock: 0x2cee2a20:
:
lock: 0x207a080:

Key caches:
default
Buffer_size:       8388608
Block_size:           1024
Division_limit:        100
Age_limit:             300
blocks used:             4
not flushed:             0
w_requests:              0
writes:                  0
r_requests:              8
reads:                   4

handler status:
read_key:           13
read_next:           4
read_rnd             0
read_first:         13
write:               1
delete               0
update:              0

Table status:
Opened tables:        121
Open tables:          114
Open files:            18
Open streams:           0

Memory status:
<malloc version="1">
<heap nr="0">
<sizes>
  <size from="17" to="32" total="32" count="1"/>
  <size from="33" to="48" total="96" count="2"/>
  <size from="33" to="33" total="33" count="1"/>
  <size from="97" to="97" total="6014" count="62"/>
  <size from="113" to="113" total="904" count="8"/>
  <size from="193" to="193" total="193" count="1"/>
  <size from="241" to="241" total="241" count="1"/>
  <size from="609" to="609" total="609" count="1"/>
  <size from="16369" to="16369" total="49107" count="3"/>
  <size from="24529" to="24529" total="98116" count="4"/>
  <size from="32689" to="32689" total="32689" count="1"/>
  <unsorted from="241" to="7505" total="7746" count="2"/>
</sizes>
<total type="fast" count="3" size="128"/>
<total type="rest" count="84" size="195652"/>
<system type="current" size="690774016"/>
<system type="max" size="690774016"/>
<aspace type="total" size="690774016"/>
```

```
<aspace type="mprotect" size="690774016"/>
</heap>
:
<total type="fast" count="85" size="5520"/>
<total type="rest" count="116" size="316820"/>
<total type="mmap" count="82" size="939954176"/>
<system type="current" size="695717888"/>
<system type="max" size="695717888"/>
<aspace type="total" size="695717888"/>
<aspace type="mprotect" size="695717888"/>
</malloc>

Events status:
LLA = Last Locked At  LUA = Last Unlocked At
WOC = Waiting On Condition  DL = Data Locked

Event scheduler status:
State      : INITIALIZED
Thread id  : 0
LLA        : n/a:0
LUA        : n/a:0
WOC        : NO
Workers    : 0
Executed   : 0
Data locked: NO

Event queue status:
Element count   : 0
Data locked     : NO
Attempting lock : NO
LLA             : init_queue:95
LUA             : init_queue:103
WOC             : NO
Next activation : never
```

- `SIGINT` normally is ignored by the server. Starting the server with the `--gdb` option installs an interrupt handler for `SIGINT` for debugging purposes. See Section 5.8.1.4, “Debugging mysqld under gdb”.

Client Response to Signals

MySQL client programs respond to signals as follows:

- The `mysql` client interprets `SIGINT` (typically the result of typing **Control+C**) as instruction to interrupt the current statement if there is one, or to cancel any partial input line otherwise. This behavior can be disabled using the `--sigint-ignore` option to ignore `SIGINT` signals.
- Client programs that use the MySQL client library block `SIGPIPE` signals by default. These variations are possible:
  - Client can install their own `SIGPIPE` handler to override the default behavior. See Writing C API Threaded Client Programs.
  - Clients can prevent installation of `SIGPIPE` handlers by specifying the `CLIENT_IGNORE_SIGPIPE` option to `mysql_real_connect()` at connect time. See mysql_real_connect().

Chapter 5 MySQL Server Administration

Table of Contents

5.1 The MySQL Server 462
5.1.1 Configuring the Server 462
5.1.2 Server Configuration Defaults 464
5.1.3 Server Option, System Variable, and Status Variable Reference 466
5.1.4 Server System Variable Reference 496
5.1.5 Server Status Variable Reference 510
5.1.6 Server Command Options 522
5.1.7 Server System Variables 545
5.1.8 Using System Variables 637
5.1.9 Server Status Variables 650
5.1.10 Server SQL Modes 668
5.1.11 Connection Management 676
5.1.12 IPv6 Support 681
5.1.13 MySQL Server Time Zone Support 685
5.1.14 Server-Side Help Support 690
5.1.15 The Server Shutdown Process 690
5.2 The MySQL Data Directory 692
5.3 The mysql System Database 692
5.4 MySQL Server Logs 694
5.4.1 Selecting General Query Log and Slow Query Log Output Destinations 695
5.4.2 The Error Log 697
5.4.3 The General Query Log 699
5.4.4 The Binary Log 701
5.4.5 The Slow Query Log 712
5.4.6 The DDL Log 714
5.4.7 Server Log Maintenance 715
5.5 MySQL Server Plugins 716
5.5.1 Installing and Uninstalling Plugins 717
5.5.2 Obtaining Server Plugin Information 721
5.5.3 MySQL Enterprise Thread Pool 722
5.5.4 MySQL Plugin Services 728
5.6 MySQL Server User-Defined Functions 728
5.6.1 Installing and Uninstalling User-Defined Functions 728
5.6.2 Obtaining User-Defined Function Information 729
5.7 Running Multiple MySQL Instances on One Machine 729
5.7.1 Setting Up Multiple Data Directories 731
5.7.2 Running Multiple MySQL Instances on Windows 732
5.7.3 Running Multiple MySQL Instances on Unix 735
5.7.4 Using Client Programs in a Multiple-Server Environment 736
5.8 Debugging MySQL 736
5.8.1 Debugging a MySQL Server 736
5.8.2 Debugging a MySQL Client 743
5.8.3 The DBUG Package 743
5.8.4 Tracing mysqld Using DTrace 746

MySQL Server (`mysqld`) is the main program that does most of the work in a MySQL installation. This chapter provides an overview of MySQL Server and covers general server administration:

- Server configuration
- The data directory, particularly the `mysql` system database
- The server log files

- Management of multiple servers on a single machine

For additional information on administrative topics, see also:

- Chapter 6, *Security*
- Section 14.8, “InnoDB Configuration”
- Chapter 7, *Backup and Recovery*
- Chapter 17, *Replication*
- Chapter 19, *Partitioning*

5.1 The MySQL Server

`mysqld` is the MySQL server. The following discussion covers these MySQL server configuration topics:

- Startup options that the server supports. You can specify these options on the command line, through configuration files, or both.
- Server system variables. These variables reflect the current state and values of the startup options, some of which can be modified while the server is running.
- Server status variables. These variables contain counters and statistics about runtime operation.
- How to set the server SQL mode. This setting modifies certain aspects of SQL syntax and semantics, for example for compatibility with code from other database systems, or to control the error handling for particular situations.
- How the server manages client connections.
- Configuring and using IPv6 support.
- Configuring and using time zone support.
- Server-side help capabilities.
- The server shutdown process. There are performance and reliability considerations depending on the type of table (transactional or nontransactional) and whether you use replication.

Note

Not all storage engines are supported by all MySQL server binaries and configurations. To find out how to determine which storage engines your MySQL server installation supports, see Section 13.7.5.17, “SHOW ENGINES Statement”.

5.1.1 Configuring the Server

The MySQL server, `mysqld`, has many command options and system variables that can be set at startup to configure its operation. To determine the default command option and system variable values used by the server, execute this command:

```
shell> mysqld --verbose --help
```

The command produces a list of all `mysqld` options and configurable system variables. Its output includes the default option and variable values and looks something like this:

```
abort-slave-event-count           0
allow-suspicious-udfs             FALSE
archive                           ON
auto-increment-increment          1
```

```
auto-increment-offset             1
autocommit                        TRUE
automatic-sp-privileges           TRUE
back-log                          80
basedir                           /home/jon/bin/mysql-5.6/
...
tmpdir                            /tmp
transaction-alloc-block-size      8192
transaction-isolation             REPEATABLE-READ
transaction-prealloc-size         4096
transaction-read-only             FALSE
updatable-views-with-limit        YES
verbose                           TRUE
wait-timeout                      28800
```

To see the current system variable values actually used by the server as it runs, connect to it and execute this statement:

```
mysql> SHOW VARIABLES;
```

To see some statistical and status indicators for a running server, execute this statement:

```
mysql> SHOW STATUS;
```

System variable and status information also is available using the `mysqladmin` command:

```
shell> mysqladmin variables
shell> mysqladmin extended-status
```

For a full description of all command options, system variables, and status variables, see these sections:

- Section 5.1.6, “Server Command Options”
- Section 5.1.7, “Server System Variables”
- Section 5.1.9, “Server Status Variables”

More detailed monitoring information is available from the Performance Schema; see Chapter 22, *MySQL Performance Schema*.

MySQL uses algorithms that are very scalable, so you can usually run with very little memory. However, normally better performance results from giving MySQL more memory.

When tuning a MySQL server, the two most important variables to configure are `key_buffer_size` and `table_open_cache`. You should first feel confident that you have these set appropriately before trying to change any other variables.

The following examples indicate some typical variable values for different runtime configurations.

- If you have at least 1-2GB of memory and many tables and want maximum performance with a moderate number of clients, use something like this:

  ```
  shell> mysqld_safe --key_buffer_size=384M --table_open_cache=4000 \
             --sort_buffer_size=4M --read_buffer_size=1M &
  ```

- If you have only 256MB of memory and only a few tables, but you still do a lot of sorting, you can use something like this:

  ```
  shell> mysqld_safe --key_buffer_size=64M --sort_buffer_size=1M
  ```

  If there are very many simultaneous connections, swapping problems may occur unless `mysqld` has been configured to use very little memory for each connection. `mysqld` performs better if you have enough memory for all connections.

- With little memory and lots of connections, use something like this:

  ```
  shell> mysqld_safe --key_buffer_size=512K --sort_buffer_size=100K \
  ```

```
          --read_buffer_size=100K &
```

Or even this:

```
shell> mysqld_safe --key_buffer_size=512K --sort_buffer_size=16K \
          --table_open_cache=32 --read_buffer_size=8K \
          --net_buffer_length=1K &
```

If you are performing `GROUP BY` or `ORDER BY` operations on tables that are much larger than your available memory, increase the value of `read_rnd_buffer_size` to speed up the reading of rows following sorting operations.

You can make use of the example option files included with your MySQL distribution; see Section 5.1.2, “Server Configuration Defaults”.

If you specify an option on the command line for `mysqld` or `mysqld_safe`, it remains in effect only for that invocation of the server. To use the option every time the server runs, put it in an option file. See Section 4.2.2.2, “Using Option Files”.

5.1.2 Server Configuration Defaults

The MySQL server has many operating parameters, which you can change at server startup using command-line options or configuration files (option files). It is also possible to change many parameters at runtime. For general instructions on setting parameters at startup or runtime, see Section 5.1.6, “Server Command Options”, and Section 5.1.7, “Server System Variables”.

5.1.2.1 Changes to Server Defaults

In MySQL 5.6, several MySQL Server parameter defaults differ from the defaults in previous releases. The motivation for these changes is to provide better out-of-box performance and to reduce the need for database administrators to change settings manually.

In some cases, a parameter has a different fixed default value. In other cases, the server autosizes a parameter at startup using a formula based on other related parameters or server host configuration, rather than using a fixed value. For example, the setting for `back_log` is its previous default of 50, adjusted up by an amount proportional to the value of `max_connections`. The idea behind autosizing is that when the server has information available to make a decision about a parameter setting likely to be better than a fixed default, it does so.

The following table summarizes changes to defaults. The Version column indicates when each default changed. For variables that are autosized, the main variable description provides additional detail about the sizing algorithm. See Section 5.1.7, “Server System Variables”, and Section 14.14, “InnoDB Startup Options and System Variables”. Any of these default settings can be overridden by specifying an explicit value at server startup.

Table 5.1 Changes to Server Defaults in MySQL 5.6

Parameter	Old Default	New Default	Version
`back_log`	50	Autosized using `max_connections`	5.6.6
`binlog_checksum`	`NONE`	`CRC32`	5.6.6
`--binlog-row-event-max-size`	1024	8192	5.6.6
`flush_time`	1800 (on Windows)	0	5.6.6
`host_cache_size`	128	Autosized using `max_connections`	5.6.8
`innodb_autoextend_increment`	8	64	5.6.6
`innodb_buffer_pool_instances`	1	8 (platform dependent)	5.6.6
`innodb_concurrency_tickets`	500	5000	5.6.6

Parameter	Old Default	New Default	Version
innodb_data_file_path	ibdata1:10M:autoextend	ibdata1:12M:autoextend	5.6.7
innodb_file_per_table	0	1	5.6.6
innodb_log_file_size	5MB	48MB	5.6.8
innodb_old_blocks_time	0	1000	5.6.6
innodb_open_files	300	Autosized using innodb_file_per_table, table_open_cache	5.6.6
innodb_stats_on_metadata	ON	OFF	5.6.6
join_buffer_size	128KB	256KB	5.6.6
max_allowed_packet	1MB	4MB	5.6.6
max_connect_errors	10	100	5.6.6
open_files_limit	0	Autosized using max_connections	5.6.8
performance_schema	OFF	ON	5.6.6
performance_schema_events_waits_history_long_size	10000	Autosized	5.6.6
performance_schema_events_waits_history_size	10	Autosized	5.6.6
performance_schema_max_cond_instances	1000	Autosized	5.6.6
performance_schema_max_file_instances	10000	Autosized	5.6.6
performance_schema_max_mutex_instances	1000000	Autosized	5.6.6
performance_schema_max_rwlock_instances	1000000	Autosized	5.6.6
performance_schema_max_table_handles	100000	Autosized	5.6.6
performance_schema_max_table_instances	50000	Autosized	5.6.6
performance_schema_max_thread_instances	1000	Autosized	5.6.6
query_cache_size	0	1M	5.6.8
query_cache_type	ON	OFF	5.6.8
secure_auth	OFF	ON	5.6.7
sql_mode	" (empty string)	NO_ENGINE_SUBSTITUTION	5.6.6
sync_master_info	0	10000	5.6.6
sync_relay_log	0	10000	5.6.6
sync_relay_log_info	0	10000	5.6.6
table_definition_cache	400	Autosized using table_open_cache	5.6.8
table_open_cache	400	2000	5.6.8
thread_cache_size	0	Autosized using max_connections	5.6.8

In MySQL 5.6.6, the default for innodb_checksum_algorithm was changed from INNODB to CRC32. For compatibility reasons, the default was returned in 5.6.7 to INNODB.

5.1.2.2 Using a Sample Default Server Configuration File

On Unix platforms, mysql_install_db creates a default option file named my.cnf in the base installation directory. This file is created from a template included in the distribution package named my-default.cnf. You can find the template in or under the base installation directory. When started using mysqld_safe, the server uses my.cnf file by default. If my.cnf already exists, mysql_install_db assumes it to be in use and writes a new file named my-new.cnf instead.

With one exception, the settings in the default option file are commented and have no effect. The exception is that the file changes the `sql_mode` system variable from its default of `NO_ENGINE_SUBSTITUTION` to also include `STRICT_TRANS_TABLES`:

```
sql_mode=NO_ENGINE_SUBSTITUTION,STRICT_TRANS_TABLES
```

This setting produces a server configuration that results in errors rather than warnings for bad data in operations that modify transactional tables. See Section 5.1.10, “Server SQL Modes”.

The `my-default.cnf` template replaces the older sample option files formerly supplied with MySQL distributions (`my-small.cnf`, `my-medium.cnf`, `my-large.cnf`, and `my-huge.cnf`).

On Windows, MySQL Installer interacts with the user and creates a file named `my.ini` in the base installation directory as the default option file. If you install on Windows from a Zip archive, you can copy the `my-default.ini` template file in the base installation directory to `my.ini` and use the latter as the default option file.

Note

On Windows, the `.ini` or `.cnf` option file extension might not be displayed.

On any platform, after completing the installation process, you can edit the default option file at any time to modify the parameters used by the server. For example, to use a parameter setting in the file that is commented with a `#` character at the beginning of the line, remove the `#`, and modify the parameter value if necessary. To disable a To disable a setting, either add a `#` to the beginning of the line or remove it.

For additional information about option file format and syntax, see Section 4.2.2.2, “Using Option Files”.

5.1.3 Server Option, System Variable, and Status Variable Reference

The following table lists all command-line options, system variables, and status variables applicable within `mysqld`.

The table lists command-line options (Cmd-line), options valid in configuration files (Option file), server system variables (System Var), and status variables (Status var) in one unified list, with an indication of where each option or variable is valid. If a server option set on the command line or in an option file differs from the name of the corresponding system variable, the variable name is noted immediately below the corresponding option. For system and status variables, the scope of the variable (Var Scope) is Global, Session, or both. Please see the corresponding item descriptions for details on setting and using the options and variables. Where appropriate, direct links to further information about the items are provided.

For a version of this table that is specific to NDB Cluster, see Section 18.3.2.5, “NDB Cluster mysqld Option and Variable Reference”.

Table 5.2 Command-Line Option, System Variable, and Status Variable Summary

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
abort-slave-event-count	Yes	Yes				
Aborted_clients				Yes	Global	No
Aborted_connects				Yes	Global	No
allow-suspicious-udfs	Yes	Yes				
ansi	Yes	Yes				
audit-log	Yes	Yes				
audit_log_buffer_size	Yes	Yes	Yes		Global	No
audit_log_connection_policy	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
audit_log_current_session			Yes		Both	No
Audit_log_current_size				Yes	Global	No
Audit_log_event_max_drop_size				Yes	Global	No
Audit_log_events				Yes	Global	No
Audit_log_events_filtered				Yes	Global	No
Audit_log_events_lost				Yes	Global	No
Audit_log_events_written				Yes	Global	No
audit_log_exclude_accounts	Yes	Yes	Yes		Global	Yes
audit_log_file	Yes	Yes	Yes		Global	No
audit_log_flush			Yes		Global	Yes
audit_log_format	Yes	Yes	Yes		Global	No
audit_log_include_accounts	Yes	Yes	Yes		Global	Yes
audit_log_policy	Yes	Yes	Yes		Global	Va
audit_log_rotate_on_size	Yes	Yes	Yes		Global	Yes
audit_log_statement_policy	Yes	Yes	Yes		Global	Yes
audit_log_strategy	Yes	Yes	Yes		Global	No
Audit_log_total_size				Yes	Global	No
Audit_log_write_waits				Yes	Global	No
authentication_windows_log_level	Yes	Yes	Yes		Global	No
authentication_windows_use_principal_name	Yes	Yes	Yes		Global	No
auto_increment_increment	Yes	Yes	Yes		Both	Yes
auto_increment_offset	Yes	Yes	Yes		Both	Yes
autocommit	Yes	Yes	Yes		Both	Yes
automatic_sp_privileges	Yes	Yes	Yes		Global	Yes
avoid_temporal_upgrade	Yes	Yes	Yes		Global	Yes
back_log	Yes	Yes	Yes		Global	No
basedir	Yes	Yes	Yes		Global	No
big_tables	Yes	Yes	Yes		Both	Yes
bind_address	Yes	Yes	Yes		Global	No
Binlog_cache_disk_use				Yes	Global	No
binlog_cache_size	Yes	Yes	Yes		Global	Yes
Binlog_cache_use				Yes	Global	No
binlog-checksum	Yes	Yes				
binlog_checksum	Yes	Yes	Yes		Global	Yes
binlog_direct_non_transactional_updates	Yes	Yes	Yes		Both	Yes
binlog-do-db	Yes	Yes				
binlog_error_action	Yes	Yes	Yes		Global	Yes
binlog_format	Yes	Yes	Yes		Both	Yes
binlog_gtid_simple_recovery	Yes	Yes	Yes		Global	No
binlog-ignore-db	Yes	Yes				
binlog_max_flush_queue_time	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
binlog_order_commits	Yes	Yes	Yes		Global	Yes
binlog-row-event-max-size	Yes	Yes				
binlog_row_image	Yes	Yes	Yes		Both	Yes
binlog_rows_query_log_events	Yes	Yes	Yes		Both	Yes
Binlog_stmt_cache_disk_use				Yes	Global	No
binlog_stmt_cache_size	Yes	Yes	Yes		Global	Yes
Binlog_stmt_cache_use				Yes	Global	No
binlogging_impossible_mode	Yes	Yes	Yes		Both	Yes
block_encryption_mode	Yes	Yes	Yes		Both	Yes
bootstrap	Yes	Yes				
bulk_insert_buffer_size	Yes	Yes	Yes		Both	Yes
Bytes_received				Yes	Both	No
Bytes_sent				Yes	Both	No
character_set_client			Yes		Both	Yes
character-set-client-handshake	Yes	Yes				
character_set_connection			Yes		Both	Yes
character_set_database (note 1)			Yes		Both	Yes
character_set_filesystem	Yes	Yes	Yes		Both	Yes
character_set_results			Yes		Both	Yes
character_set_server	Yes	Yes	Yes		Both	Yes
character_set_system			Yes		Global	No
character_sets_dir	Yes	Yes	Yes		Global	No
chroot	Yes	Yes				
collation_connection			Yes		Both	Yes
collation_database (note 1)			Yes		Both	Yes
collation_server	Yes	Yes	Yes		Both	Yes
Com_admin_commands				Yes	Both	No
Com_alter_db				Yes	Both	No
Com_alter_db_upgrade				Yes	Both	No
Com_alter_event				Yes	Both	No
Com_alter_function				Yes	Both	No
Com_alter_procedure				Yes	Both	No
Com_alter_server				Yes	Both	No
Com_alter_table				Yes	Both	No
Com_alter_tablespace				Yes	Both	No
Com_alter_user				Yes	Both	No
Com_analyze				Yes	Both	No
Com_assign_to_keycache				Yes	Both	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
Com_begin				Yes	Both	No
Com_binlog				Yes	Both	No
Com_call_procedure				Yes	Both	No
Com_change_db				Yes	Both	No
Com_change_master				Yes	Both	No
Com_check				Yes	Both	No
Com_checksum				Yes	Both	No
Com_commit				Yes	Both	No
Com_create_db				Yes	Both	No
Com_create_event				Yes	Both	No
Com_create_function				Yes	Both	No
Com_create_index				Yes	Both	No
Com_create_procedure				Yes	Both	No
Com_create_server				Yes	Both	No
Com_create_table				Yes	Both	No
Com_create_trigger				Yes	Both	No
Com_create_udf				Yes	Both	No
Com_create_user				Yes	Both	No
Com_create_view				Yes	Both	No
Com_dealloc_sql				Yes	Both	No
Com_delete				Yes	Both	No
Com_delete_multi				Yes	Both	No
Com_do				Yes	Both	No
Com_drop_db				Yes	Both	No
Com_drop_event				Yes	Both	No
Com_drop_function				Yes	Both	No
Com_drop_index				Yes	Both	No
Com_drop_procedure				Yes	Both	No
Com_drop_server				Yes	Both	No
Com_drop_table				Yes	Both	No
Com_drop_trigger				Yes	Both	No
Com_drop_user				Yes	Both	No
Com_drop_view				Yes	Both	No
Com_empty_query				Yes	Both	No
Com_execute_sql				Yes	Both	No
Com_flush				Yes	Both	No
Com_get_diagnostics				Yes	Both	No
Com_grant				Yes	Both	No
Com_ha_close				Yes	Both	No
Com_ha_open				Yes	Both	No
Com_ha_read				Yes	Both	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
Com_help				Yes	Both	No
Com_insert				Yes	Both	No
Com_insert_select				Yes	Both	No
Com_install_plugin				Yes	Both	No
Com_kill				Yes	Both	No
Com_load				Yes	Both	No
Com_lock_tables				Yes	Both	No
Com_optimize				Yes	Both	No
Com_preload_keys				Yes	Both	No
Com_prepare_sql				Yes	Both	No
Com_purge				Yes	Both	No
Com_purge_before_date				Yes	Both	No
Com_release_savepoint				Yes	Both	No
Com_rename_table				Yes	Both	No
Com_rename_user				Yes	Both	No
Com_repair				Yes	Both	No
Com_replace				Yes	Both	No
Com_replace_select				Yes	Both	No
Com_reset				Yes	Both	No
Com_resignal				Yes	Both	No
Com_revoke				Yes	Both	No
Com_revoke_all				Yes	Both	No
Com_rollback				Yes	Both	No
Com_rollback_to_savepoint				Yes	Both	No
Com_savepoint				Yes	Both	No
Com_select				Yes	Both	No
Com_set_option				Yes	Both	No
Com_show_authors				Yes	Both	No
Com_show_binlog_events				Yes	Both	No
Com_show_binlogs				Yes	Both	No
Com_show_charsets				Yes	Both	No
Com_show_collations				Yes	Both	No
Com_show_contributors				Yes	Both	No
Com_show_create_db				Yes	Both	No
Com_show_create_event				Yes	Both	No
Com_show_create_func				Yes	Both	No
Com_show_create_proc				Yes	Both	No
Com_show_create_table				Yes	Both	No
Com_show_create_trigger				Yes	Both	No
Com_show_databases				Yes	Both	No
Com_show_engine_logs				Yes	Both	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
Com_show_engine_mutex				Yes	Both	No
Com_show_engine_status				Yes	Both	No
Com_show_errors				Yes	Both	No
Com_show_events				Yes	Both	No
Com_show_fields				Yes	Both	No
Com_show_function_code				Yes	Both	No
Com_show_function_status				Yes	Both	No
Com_show_grants				Yes	Both	No
Com_show_keys				Yes	Both	No
Com_show_master_status				Yes	Both	No
Com_show_ndb_status				Yes	Both	No
Com_show_open_tables				Yes	Both	No
Com_show_plugins				Yes	Both	No
Com_show_privileges				Yes	Both	No
Com_show_procedure_code				Yes	Both	No
Com_show_procedure_status				Yes	Both	No
Com_show_processlist				Yes	Both	No
Com_show_profile				Yes	Both	No
Com_show_profiles				Yes	Both	No
Com_show_relaylog_events				Yes	Both	No
Com_show_slave_hosts				Yes	Both	No
Com_show_slave_status				Yes	Both	No
Com_show_status				Yes	Both	No
Com_show_storage_engines				Yes	Both	No
Com_show_table_status				Yes	Both	No
Com_show_tables				Yes	Both	No
Com_show_triggers				Yes	Both	No
Com_show_variables				Yes	Both	No
Com_show_warnings				Yes	Both	No
Com_signal				Yes	Both	No
Com_slave_start				Yes	Both	No
Com_slave_stop				Yes	Both	No
Com_stmt_close				Yes	Both	No
Com_stmt_execute				Yes	Both	No
Com_stmt_fetch				Yes	Both	No
Com_stmt_prepare				Yes	Both	No
Com_stmt_reprepare				Yes	Both	No
Com_stmt_reset				Yes	Both	No
Com_stmt_send_long_data				Yes	Both	No
Com_truncate				Yes	Both	No
Com_uninstall_plugin				Yes	Both	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
Com_unlock_tables				Yes	Both	No
Com_update				Yes	Both	No
Com_update_multi				Yes	Both	No
Com_xa_commit				Yes	Both	No
Com_xa_end				Yes	Both	No
Com_xa_prepare				Yes	Both	No
Com_xa_recover				Yes	Both	No
Com_xa_rollback				Yes	Both	No
Com_xa_start				Yes	Both	No
completion_type	Yes	Yes	Yes		Both	Yes
Compression				Yes	Session	No
concurrent_insert	Yes	Yes	Yes		Global	Yes
connect_timeout	Yes	Yes	Yes		Global	Yes
Connection_control_delay_generated				Yes	Global	No
connection_control_failed_connections_threshold	Yes	Yes	Yes		Global	Yes
connection_control_max_connection_delay	Yes	Yes	Yes		Global	Yes
connection_control_min_connection_delay	Yes	Yes	Yes		Global	Yes
Connection_errors_accept				Yes	Global	No
Connection_errors_internal				Yes	Global	No
Connection_errors_max_connections				Yes	Global	No
Connection_errors_peer_address				Yes	Global	No
Connection_errors_select				Yes	Global	No
Connection_errors_tcpwrap				Yes	Global	No
Connections				Yes	Global	No
console	Yes	Yes				
core-file	Yes	Yes				
core_file			Yes		Global	No
create_old_temporals	Yes	Yes	Yes		Global	No
Created_tmp_disk_tables				Yes	Both	No
Created_tmp_files				Yes	Global	No
Created_tmp_tables				Yes	Both	No
daemon_memcached_enable_binlog	Yes	Yes	Yes		Global	No
daemon_memcached_engine_lib_name	Yes	Yes	Yes		Global	No
daemon_memcached_engine_lib_path	Yes	Yes	Yes		Global	No
daemon_memcached_option	Yes	Yes	Yes		Global	No
daemon_memcached_r_batch_size	Yes	Yes	Yes		Global	No
daemon_memcached_w_batch_size	Yes	Yes	Yes		Global	No
datadir	Yes	Yes	Yes		Global	No
date_format			Yes		Global	No
datetime_format			Yes		Global	No
debug	Yes	Yes	Yes		Both	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
debug_sync			Yes		Session	Yes
debug-sync-timeout	Yes	Yes				
default-authentication-plugin	Yes	Yes				
default_storage_engine	Yes	Yes	Yes		Both	Yes
default-time-zone	Yes	Yes				
default_tmp_storage_engine	Yes	Yes	Yes		Both	Yes
default_week_format	Yes	Yes	Yes		Both	Yes
defaults-extra-file	Yes					
defaults-file	Yes					
defaults-group-suffix	Yes					
delay_key_write	Yes	Yes	Yes		Global	Yes
Delayed_errors				Yes	Global	No
delayed_insert_limit	Yes	Yes	Yes		Global	Yes
Delayed_insert_threads				Yes	Global	No
delayed_insert_timeout	Yes	Yes	Yes		Global	Yes
delayed_queue_size	Yes	Yes	Yes		Global	Yes
Delayed_writes				Yes	Global	No
des-key-file	Yes	Yes				
disconnect_on_expired_password	Yes	Yes	Yes		Global	No
disconnect-slave-event-count	Yes	Yes				
div_precision_increment	Yes	Yes	Yes		Both	Yes
end_markers_in_json	Yes	Yes	Yes		Both	Yes
enforce_gtid_consistency	Yes	Yes	Yes		Global	No
eq_range_index_dive_limit	Yes	Yes	Yes		Both	Yes
error_count			Yes		Session	No
event_scheduler	Yes	Yes	Yes		Global	Yes
exit-info	Yes	Yes				
expire_logs_days	Yes	Yes	Yes		Global	Yes
explicit_defaults_for_timestamp	Yes	Yes	Yes		Both	Yes
external-locking	Yes	Yes				
- *Variable*: skip_external_locking						
external_user			Yes		Session	No
federated	Yes	Yes				
Firewall_access_denied				Yes	Global	No
Firewall_access_granted				Yes	Global	No
Firewall_cached_entries				Yes	Global	No
flush	Yes	Yes	Yes		Global	Yes
Flush_commands				Yes	Global	No
flush_time	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
foreign_key_checks			Yes		Both	Yes
ft_boolean_syntax	Yes	Yes	Yes		Global	Yes
ft_max_word_len	Yes	Yes	Yes		Global	No
ft_min_word_len	Yes	Yes	Yes		Global	No
ft_query_expansion_limit	Yes	Yes	Yes		Global	No
ft_stopword_file	Yes	Yes	Yes		Global	No
gdb	Yes	Yes				
general_log	Yes	Yes	Yes		Global	Yes
general_log_file	Yes	Yes	Yes		Global	Yes
group_concat_max_len	Yes	Yes	Yes		Both	Yes
gtid_executed			Yes		Both	No
gtid_mode	Yes	Yes	Yes		Global	No
gtid_next			Yes		Session	Yes
gtid_owned			Yes		Both	No
gtid_purged			Yes		Global	Yes
Handler_commit				Yes	Both	No
Handler_delete				Yes	Both	No
Handler_discover				Yes	Both	No
Handler_external_lock				Yes	Both	No
Handler_mrr_init				Yes	Both	No
Handler_prepare				Yes	Both	No
Handler_read_first				Yes	Both	No
Handler_read_key				Yes	Both	No
Handler_read_last				Yes	Both	No
Handler_read_next				Yes	Both	No
Handler_read_prev				Yes	Both	No
Handler_read_rnd				Yes	Both	No
Handler_read_rnd_next				Yes	Both	No
Handler_rollback				Yes	Both	No
Handler_savepoint				Yes	Both	No
Handler_savepoint_rollback				Yes	Both	No
Handler_update				Yes	Both	No
Handler_write				Yes	Both	No
have_compress			Yes		Global	No
have_crypt			Yes		Global	No
have_dynamic_loading			Yes		Global	No
have_geometry			Yes		Global	No
have_openssl			Yes		Global	No
have_profiling			Yes		Global	No
have_query_cache			Yes		Global	No
have_rtree_keys			Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
have_ssl			Yes		Global	No
have_symlink			Yes		Global	No
help	Yes	Yes				
host_cache_size	Yes	Yes	Yes		Global	Yes
hostname			Yes		Global	No
identity			Yes		Session	Yes
ignore_builtin_innodb	Yes	Yes	Yes		Global	No
ignore-db-dir	Yes	Yes				
ignore_db_dirs			Yes		Global	No
init_connect	Yes	Yes	Yes		Global	Yes
init_file	Yes	Yes	Yes		Global	No
init_slave	Yes	Yes	Yes		Global	Yes
innodb	Yes	Yes				
innodb_adaptive_flushing	Yes	Yes	Yes		Global	Yes
innodb_adaptive_flushing_lwm	Yes	Yes	Yes		Global	Yes
innodb_adaptive_hash_index	Yes	Yes	Yes		Global	Yes
innodb_adaptive_max_sleep_delay	Yes	Yes	Yes		Global	Yes
innodb_additional_mem_pool_size	Yes	Yes	Yes		Global	No
innodb_api_bk_commit_interval	Yes	Yes	Yes		Global	Yes
innodb_api_disable_rowlock	Yes	Yes	Yes		Global	No
innodb_api_enable_binlog	Yes	Yes	Yes		Global	No
innodb_api_enable_mdl	Yes	Yes	Yes		Global	No
innodb_api_trx_level	Yes	Yes	Yes		Global	Yes
innodb_autoextend_increment	Yes	Yes	Yes		Global	Yes
innodb_autoinc_lock_mode	Yes	Yes	Yes		Global	No
Innodb_available_undo_logs				Yes	Global	No
Innodb_buffer_pool_bytes_data				Yes	Global	No
Innodb_buffer_pool_bytes_dirty				Yes	Global	No
innodb_buffer_pool_dump_at_shutdown	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_dump_now	Yes	Yes	Yes		Global	Yes
Innodb_buffer_pool_dump_status				Yes	Global	No
innodb_buffer_pool_filename	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_instances	Yes	Yes	Yes		Global	No
innodb_buffer_pool_load_abort	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_load_at_startup	Yes	Yes	Yes		Global	No
innodb_buffer_pool_load_now	Yes	Yes	Yes		Global	Yes
Innodb_buffer_pool_load_status				Yes	Global	No
Innodb_buffer_pool_pages_data				Yes	Global	No
Innodb_buffer_pool_pages_dirty				Yes	Global	No
Innodb_buffer_pool_pages_flushed				Yes	Global	No
Innodb_buffer_pool_pages_free				Yes	Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
Innodb_buffer_pool_pages_latched				Yes	Global	No
Innodb_buffer_pool_pages_misc				Yes	Global	No
Innodb_buffer_pool_pages_total				Yes	Global	No
Innodb_buffer_pool_read_ahead				Yes	Global	No
Innodb_buffer_pool_read_ahead_evicted				Yes	Global	No
Innodb_buffer_pool_read_ahead_rnd				Yes	Global	No
Innodb_buffer_pool_read_requests				Yes	Global	No
Innodb_buffer_pool_reads				Yes	Global	No
innodb_buffer_pool_size	Yes	Yes	Yes		Global	No
Innodb_buffer_pool_wait_free				Yes	Global	No
Innodb_buffer_pool_write_requests				Yes	Global	No
innodb_change_buffer_max_size	Yes	Yes	Yes		Global	Yes
innodb_change_buffering	Yes	Yes	Yes		Global	Yes
innodb_change_buffering_debug	Yes	Yes	Yes		Global	Yes
innodb_checksum_algorithm	Yes	Yes	Yes		Global	Yes
innodb_checksums	Yes	Yes	Yes		Global	No
innodb_cmp_per_index_enabled	Yes	Yes	Yes		Global	Yes
innodb_commit_concurrency	Yes	Yes	Yes		Global	Yes
innodb_compression_failure_threshold_pct	Yes	Yes	Yes		Global	Yes
innodb_compression_level	Yes	Yes	Yes		Global	Yes
innodb_compression_pad_pct_max	Yes	Yes	Yes		Global	Yes
innodb_concurrency_tickets	Yes	Yes	Yes		Global	Yes
innodb_data_file_path	Yes	Yes	Yes		Global	No
Innodb_data_fsyncs				Yes	Global	No
innodb_data_home_dir	Yes	Yes	Yes		Global	No
Innodb_data_pending_fsyncs				Yes	Global	No
Innodb_data_pending_reads				Yes	Global	No
Innodb_data_pending_writes				Yes	Global	No
Innodb_data_read				Yes	Global	No
Innodb_data_reads				Yes	Global	No
Innodb_data_writes				Yes	Global	No
Innodb_data_written				Yes	Global	No
Innodb_dblwr_pages_written				Yes	Global	No
Innodb_dblwr_writes				Yes	Global	No
innodb_disable_sort_file_cache	Yes	Yes	Yes		Global	Yes
innodb_doublewrite	Yes	Yes	Yes		Global	No
innodb_fast_shutdown	Yes	Yes	Yes		Global	Yes
innodb_fil_make_page_dirty_debug	Yes	Yes	Yes		Global	Yes
innodb_file_format	Yes	Yes	Yes		Global	Yes
innodb_file_format_check	Yes	Yes	Yes		Global	No
innodb_file_format_max	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
innodb_file_per_table	Yes	Yes	Yes		Global	Yes
innodb_flush_log_at_timeout	Yes	Yes	Yes		Global	Yes
innodb_flush_log_at_trx_commit	Yes	Yes	Yes		Global	Yes
innodb_flush_method	Yes	Yes	Yes		Global	No
innodb_flush_neighbors	Yes	Yes	Yes		Global	Yes
innodb_flushing_avg_loops	Yes	Yes	Yes		Global	Yes
innodb_force_load_corrupted	Yes	Yes	Yes		Global	No
innodb_force_recovery	Yes	Yes	Yes		Global	No
innodb_ft_aux_table			Yes		Global	Yes
innodb_ft_cache_size	Yes	Yes	Yes		Global	No
innodb_ft_enable_diag_print	Yes	Yes	Yes		Global	Yes
innodb_ft_enable_stopword	Yes	Yes	Yes		Both	Yes
innodb_ft_max_token_size	Yes	Yes	Yes		Global	No
innodb_ft_min_token_size	Yes	Yes	Yes		Global	No
innodb_ft_num_word_optimize	Yes	Yes	Yes		Global	Yes
innodb_ft_result_cache_limit	Yes	Yes	Yes		Global	Yes
innodb_ft_server_stopword_table	Yes	Yes	Yes		Global	Yes
innodb_ft_sort_pll_degree	Yes	Yes	Yes		Global	No
innodb_ft_total_cache_size	Yes	Yes	Yes		Global	No
innodb_ft_user_stopword_table	Yes	Yes	Yes		Both	Yes
Innodb_have_atomic_builtins				Yes	Global	No
innodb_io_capacity	Yes	Yes	Yes		Global	Yes
innodb_io_capacity_max	Yes	Yes	Yes		Global	Yes
innodb_large_prefix	Yes	Yes	Yes		Global	Yes
innodb_limit_optimistic_insert_debug	Yes	Yes	Yes		Global	Yes
innodb_lock_wait_timeout	Yes	Yes	Yes		Both	Yes
innodb_locks_unsafe_for_binlog	Yes	Yes	Yes		Global	No
innodb_log_buffer_size	Yes	Yes	Yes		Global	No
innodb_log_checkpoint_now	Yes	Yes	Yes		Global	Yes
innodb_log_compressed_pages	Yes	Yes	Yes		Global	Yes
innodb_log_file_size	Yes	Yes	Yes		Global	No
innodb_log_files_in_group	Yes	Yes	Yes		Global	No
innodb_log_group_home_dir	Yes	Yes	Yes		Global	No
Innodb_log_waits				Yes	Global	No
Innodb_log_write_requests				Yes	Global	No
Innodb_log_writes				Yes	Global	No
innodb_lru_scan_depth	Yes	Yes	Yes		Global	Yes
innodb_max_dirty_pages_pct	Yes	Yes	Yes		Global	Yes
innodb_max_dirty_pages_pct_lwm	Yes	Yes	Yes		Global	Yes
innodb_max_purge_lag	Yes	Yes	Yes		Global	Yes
innodb_max_purge_lag_delay	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
innodb_mirrored_log_groups	Yes	Yes	Yes		Global	No
innodb_monitor_disable	Yes	Yes	Yes		Global	Yes
innodb_monitor_enable	Yes	Yes	Yes		Global	Yes
innodb_monitor_reset	Yes	Yes	Yes		Global	Yes
innodb_monitor_reset_all	Yes	Yes	Yes		Global	Yes
Innodb_num_open_files				Yes	Global	No
innodb_numa_interleave	Yes	Yes	Yes		Global	No
innodb_old_blocks_pct	Yes	Yes	Yes		Global	Yes
innodb_old_blocks_time	Yes	Yes	Yes		Global	Yes
innodb_online_alter_log_max_size	Yes	Yes	Yes		Global	Yes
innodb_open_files	Yes	Yes	Yes		Global	No
innodb_optimize_fulltext_only	Yes	Yes	Yes		Global	Yes
Innodb_os_log_fsyncs				Yes	Global	No
Innodb_os_log_pending_fsyncs				Yes	Global	No
Innodb_os_log_pending_writes				Yes	Global	No
Innodb_os_log_written				Yes	Global	No
Innodb_page_size				Yes	Global	No
innodb_page_size	Yes	Yes	Yes		Global	No
Innodb_pages_created				Yes	Global	No
Innodb_pages_read				Yes	Global	No
Innodb_pages_written				Yes	Global	No
innodb_print_all_deadlocks	Yes	Yes	Yes		Global	Yes
innodb_purge_batch_size	Yes	Yes	Yes		Global	Yes
innodb_purge_threads	Yes	Yes	Yes		Global	No
innodb_random_read_ahead	Yes	Yes	Yes		Global	Yes
innodb_read_ahead_threshold	Yes	Yes	Yes		Global	Yes
innodb_read_io_threads	Yes	Yes	Yes		Global	No
innodb_read_only	Yes	Yes	Yes		Global	No
innodb_replication_delay	Yes	Yes	Yes		Global	Yes
innodb_rollback_on_timeout	Yes	Yes	Yes		Global	No
innodb_rollback_segments	Yes	Yes	Yes		Global	Yes
Innodb_row_lock_current_waits				Yes	Global	No
Innodb_row_lock_time				Yes	Global	No
Innodb_row_lock_time_avg				Yes	Global	No
Innodb_row_lock_time_max				Yes	Global	No
Innodb_row_lock_waits				Yes	Global	No
Innodb_rows_deleted				Yes	Global	No
Innodb_rows_inserted				Yes	Global	No
Innodb_rows_read				Yes	Global	No
Innodb_rows_updated				Yes	Global	No
innodb_saved_page_number_debug	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
innodb_sort_buffer_size	Yes	Yes	Yes		Global	No
innodb_spin_wait_delay	Yes	Yes	Yes		Global	Yes
innodb_stats_auto_recalc	Yes	Yes	Yes		Global	Yes
innodb_stats_include_delete_marked	Yes	Yes	Yes		Global	Yes
innodb_stats_method	Yes	Yes	Yes		Global	Yes
innodb_stats_on_metadata	Yes	Yes	Yes		Global	Yes
innodb_stats_persistent	Yes	Yes	Yes		Global	Yes
innodb_stats_persistent_sample_pages	Yes	Yes	Yes		Global	Yes
innodb_stats_sample_pages	Yes	Yes	Yes		Global	Yes
innodb_stats_transient_sample_pages	Yes	Yes	Yes		Global	Yes
innodb-status-file	Yes	Yes				
innodb_status_output	Yes	Yes	Yes		Global	Yes
innodb_status_output_locks	Yes	Yes	Yes		Global	Yes
innodb_strict_mode	Yes	Yes	Yes		Both	Yes
innodb_support_xa	Yes	Yes	Yes		Both	Yes
innodb_sync_array_size	Yes	Yes	Yes		Global	No
innodb_sync_spin_loops	Yes	Yes	Yes		Global	Yes
innodb_table_locks	Yes	Yes	Yes		Both	Yes
innodb_thread_concurrency	Yes	Yes	Yes		Global	Yes
innodb_thread_sleep_delay	Yes	Yes	Yes		Global	Yes
innodb_tmpdir	Yes	Yes	Yes		Both	Yes
Innodb_truncated_status_writes				Yes	Global	No
innodb_trx_purge_view_update_only_debug	Yes	Yes	Yes		Global	Yes
innodb_trx_rseg_n_slots_debug	Yes	Yes	Yes		Global	Yes
innodb_undo_directory	Yes	Yes	Yes		Global	No
innodb_undo_logs	Yes	Yes	Yes		Global	Yes
innodb_undo_tablespaces	Yes	Yes	Yes		Global	No
innodb_use_native_aio	Yes	Yes	Yes		Global	No
innodb_use_sys_malloc	Yes	Yes	Yes		Global	No
innodb_version			Yes		Global	No
innodb_write_io_threads	Yes	Yes	Yes		Global	No
insert_id			Yes		Session	Yes
install	Yes					
install-manual	Yes					
interactive_timeout	Yes	Yes	Yes		Both	Yes
join_buffer_size	Yes	Yes	Yes		Both	Yes
keep_files_on_create	Yes	Yes	Yes		Both	Yes
Key_blocks_not_flushed				Yes	Global	No
Key_blocks_unused				Yes	Global	No
Key_blocks_used				Yes	Global	No
key_buffer_size	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
key_cache_age_threshold	Yes	Yes	Yes		Global	Yes
key_cache_block_size	Yes	Yes	Yes		Global	Yes
key_cache_division_limit	Yes	Yes	Yes		Global	Yes
Key_read_requests				Yes	Global	No
Key_reads				Yes	Global	No
Key_write_requests				Yes	Global	No
Key_writes				Yes	Global	No
language	Yes	Yes	Yes		Global	No
large_files_support			Yes		Global	No
large_page_size			Yes		Global	No
large_pages	Yes	Yes	Yes		Global	No
last_insert_id			Yes		Session	Yes
Last_query_cost				Yes	Session	No
Last_query_partial_plans				Yes	Session	No
lc_messages	Yes	Yes	Yes		Both	Yes
lc_messages_dir	Yes	Yes	Yes		Global	No
lc_time_names	Yes	Yes	Yes		Both	Yes
license			Yes		Global	No
local_infile	Yes	Yes	Yes		Global	Yes
local-service	Yes					
lock_wait_timeout	Yes	Yes	Yes		Both	Yes
locked_in_memory			Yes		Global	No
log-bin	Yes	Yes				
log_bin			Yes		Global	No
log_bin_basename			Yes		Global	No
log_bin_index	Yes	Yes	Yes		Global	No
log_bin_trust_function_creators	Yes	Yes	Yes		Global	Yes
log_bin_use_v1_row_events	Yes	Yes	Yes		Global	No
log_error	Yes	Yes	Yes		Global	No
log-isam	Yes	Yes				
log_output	Yes	Yes	Yes		Global	Yes
log_queries_not_using_indexes	Yes	Yes	Yes		Global	Yes
log-raw	Yes	Yes				
log-short-format	Yes	Yes				
log_slave_updates	Yes	Yes	Yes		Global	No
log-slow-admin-statements	Yes	Yes				
log_slow_admin_statements	Yes	Yes	Yes		Global	Yes
log-slow-slave-statements	Yes	Yes				
log_slow_slave_statements	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
log-tc	Yes	Yes				
log-tc-size	Yes	Yes				
log_throttle_queries_not_using_indexes	Yes	Yes	Yes		Global	Yes
log_warnings	Yes	Yes	Yes		Varies	Yes
long_query_time	Yes	Yes	Yes		Both	Yes
low_priority_updates	Yes	Yes	Yes		Both	Yes
lower_case_file_system			Yes		Global	No
lower_case_table_names	Yes	Yes	Yes		Global	No
master-info-file	Yes	Yes				
master_info_repository	Yes	Yes	Yes		Global	Yes
master-retry-count	Yes	Yes				
master_verify_checksum	Yes	Yes	Yes		Global	Yes
max_allowed_packet	Yes	Yes	Yes		Both	Yes
max_binlog_cache_size	Yes	Yes	Yes		Global	Yes
max-binlog-dump-events	Yes	Yes				
max_binlog_size	Yes	Yes	Yes		Global	Yes
max_binlog_stmt_cache_size	Yes	Yes	Yes		Global	Yes
max_connect_errors	Yes	Yes	Yes		Global	Yes
max_connections	Yes	Yes	Yes		Global	Yes
max_delayed_threads	Yes	Yes	Yes		Both	Yes
max_digest_length	Yes	Yes	Yes		Global	No
max_error_count	Yes	Yes	Yes		Both	Yes
max_heap_table_size	Yes	Yes	Yes		Both	Yes
max_insert_delayed_threads			Yes		Both	Yes
max_join_size	Yes	Yes	Yes		Both	Yes
max_length_for_sort_data	Yes	Yes	Yes		Both	Yes
max_prepared_stmt_count	Yes	Yes	Yes		Global	Yes
max_relay_log_size	Yes	Yes	Yes		Global	Yes
max_seeks_for_key	Yes	Yes	Yes		Both	Yes
max_sort_length	Yes	Yes	Yes		Both	Yes
max_sp_recursion_depth	Yes	Yes	Yes		Both	Yes
max_tmp_tables			Yes		Both	Yes
Max_used_connections				Yes	Global	No
max_user_connections	Yes	Yes	Yes		Both	Yes
max_write_lock_count	Yes	Yes	Yes		Global	Yes
memlock	Yes	Yes				
- *Variable*: locked_in_memory						
metadata_locks_cache_size	Yes	Yes	Yes		Global	No
metadata_locks_hash_instances	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
min_examined_row_limit	Yes	Yes	Yes		Both	Yes
multi_range_count	Yes	Yes	Yes		Both	Yes
myisam-block-size	Yes	Yes				
myisam_data_pointer_size	Yes	Yes	Yes		Global	Yes
myisam_max_sort_file_size	Yes	Yes	Yes		Global	Yes
myisam_mmap_size	Yes	Yes	Yes		Global	No
myisam_recover_options	Yes	Yes	Yes		Global	No
myisam_repair_threads	Yes	Yes	Yes		Both	Yes
myisam_sort_buffer_size	Yes	Yes	Yes		Both	Yes
myisam_stats_method	Yes	Yes	Yes		Both	Yes
myisam_use_mmap	Yes	Yes	Yes		Global	Yes
mysql_firewall_max_query_size	Yes	Yes	Yes		Global	No
mysql_firewall_mode	Yes	Yes	Yes		Global	Yes
mysql_firewall_trace	Yes	Yes	Yes		Global	Yes
named_pipe	Yes	Yes	Yes		Global	No
named_pipe_full_access_group	Yes	Yes	Yes		Global	No
Ndb_api_adaptive_send_deferred_count				Yes	Global	No
Ndb_api_adaptive_send_deferred_count_session				Yes	Global	No
Ndb_api_adaptive_send_deferred_count_slave				Yes	Global	No
Ndb_api_adaptive_send_forced_count				Yes	Global	No
Ndb_api_adaptive_send_forced_count_session				Yes	Global	No
Ndb_api_adaptive_send_forced_count_slave				Yes	Global	No
Ndb_api_adaptive_send_unforced_count				Yes	Global	No
Ndb_api_adaptive_send_unforced_count_session				Yes	Global	No
Ndb_api_adaptive_send_unforced_count_slave				Yes	Global	No
Ndb_api_bytes_received_count				Yes	Global	No
Ndb_api_bytes_received_count_session				Yes	Session	No
Ndb_api_bytes_received_count_slave				Yes	Global	No
Ndb_api_bytes_sent_count				Yes	Global	No
Ndb_api_bytes_sent_count_session				Yes	Session	No
Ndb_api_bytes_sent_count_slave				Yes	Global	No
Ndb_api_event_bytes_count				Yes	Global	No
Ndb_api_event_bytes_count_injector				Yes	Global	No
Ndb_api_event_data_count				Yes	Global	No
Ndb_api_event_data_count_injector				Yes	Global	No
Ndb_api_event_nondata_count				Yes	Global	No
Ndb_api_event_nondata_count_injector				Yes	Global	No
Ndb_api_pk_op_count				Yes	Global	No
Ndb_api_pk_op_count_session				Yes	Session	No
Ndb_api_pk_op_count_slave				Yes	Global	No
Ndb_api_pruned_scan_count				Yes	Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
Ndb_api_pruned_scan_count_session				Yes	Session	No
Ndb_api_pruned_scan_count_slave				Yes	Global	No
Ndb_api_range_scan_count				Yes	Global	No
Ndb_api_range_scan_count_session				Yes	Session	No
Ndb_api_range_scan_count_slave				Yes	Global	No
Ndb_api_read_row_count				Yes	Global	No
Ndb_api_read_row_count_session				Yes	Session	No
Ndb_api_read_row_count_slave				Yes	Global	No
Ndb_api_scan_batch_count				Yes	Global	No
Ndb_api_scan_batch_count_session				Yes	Session	No
Ndb_api_scan_batch_count_slave				Yes	Global	No
Ndb_api_table_scan_count				Yes	Global	No
Ndb_api_table_scan_count_session				Yes	Session	No
Ndb_api_table_scan_count_slave				Yes	Global	No
Ndb_api_trans_abort_count				Yes	Global	No
Ndb_api_trans_abort_count_session				Yes	Session	No
Ndb_api_trans_abort_count_slave				Yes	Global	No
Ndb_api_trans_close_count				Yes	Global	No
Ndb_api_trans_close_count_session				Yes	Session	No
Ndb_api_trans_close_count_slave				Yes	Global	No
Ndb_api_trans_commit_count				Yes	Global	No
Ndb_api_trans_commit_count_session				Yes	Session	No
Ndb_api_trans_commit_count_slave				Yes	Global	No
Ndb_api_trans_local_read_row_count				Yes	Global	No
Ndb_api_trans_local_read_row_count_session				Yes	Session	No
Ndb_api_trans_local_read_row_count_slave				Yes	Global	No
Ndb_api_trans_start_count				Yes	Global	No
Ndb_api_trans_start_count_session				Yes	Session	No
Ndb_api_trans_start_count_slave				Yes	Global	No
Ndb_api_uk_op_count				Yes	Global	No
Ndb_api_uk_op_count_session				Yes	Session	No
Ndb_api_uk_op_count_slave				Yes	Global	No
Ndb_api_wait_exec_complete_count				Yes	Global	No
Ndb_api_wait_exec_complete_count_session				Yes	Session	No
Ndb_api_wait_exec_complete_count_slave				Yes	Global	No
Ndb_api_wait_meta_request_count				Yes	Global	No
Ndb_api_wait_meta_request_count_session				Yes	Session	No
Ndb_api_wait_meta_request_count_slave				Yes	Global	No
Ndb_api_wait_nanos_count				Yes	Global	No
Ndb_api_wait_nanos_count_session				Yes	Session	No
Ndb_api_wait_nanos_count_slave				Yes	Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
Ndb_api_wait_scan_result_count				Yes	Global	No
Ndb_api_wait_scan_result_count_session				Yes	Session	No
Ndb_api_wait_scan_result_count_slave				Yes	Global	No
ndb_autoincrement_prefetch_sz	Yes	Yes	Yes		Both	Yes
ndb_batch_size	Yes	Yes	Yes		Global	No
ndb_blob_read_batch_bytes	Yes	Yes	Yes		Both	Yes
ndb_blob_write_batch_bytes	Yes	Yes	Yes		Both	Yes
ndb_cache_check_time	Yes	Yes	Yes		Global	Yes
ndb_clear_apply_status	Yes		Yes		Global	Yes
ndb_cluster_connection_pool	Yes	Yes	Yes		Global	No
Ndb_cluster_node_id				Yes	Global	No
Ndb_config_from_host				Yes	Both	No
Ndb_config_from_port				Yes	Both	No
Ndb_conflict_fn_epoch				Yes	Global	No
Ndb_conflict_fn_epoch_trans				Yes	Global	No
Ndb_conflict_fn_epoch2				Yes	Global	No
Ndb_conflict_fn_epoch2_trans				Yes	Global	No
Ndb_conflict_fn_max				Yes	Global	No
Ndb_conflict_fn_max_del_win				Yes	Global	No
Ndb_conflict_fn_old				Yes	Global	No
Ndb_conflict_last_conflict_epoch			Yes		Global	No
Ndb_conflict_last_stable_epoch				Yes	Global	No
Ndb_conflict_reflected_op_discard_count				Yes	Global	No
Ndb_conflict_reflected_op_prepare_count				Yes	Global	No
Ndb_conflict_refresh_op_count				Yes	Global	No
Ndb_conflict_trans_conflict_commit_count				Yes	Global	No
Ndb_conflict_trans_detect_iter_count				Yes	Global	No
Ndb_conflict_trans_reject_count				Yes	Global	No
Ndb_conflict_trans_row_conflict_count				Yes	Global	No
Ndb_conflict_trans_row_reject_count				Yes	Global	No
ndb-connectstring	Yes	Yes				
ndb_deferred_constraints	Yes	Yes	Yes		Both	Yes
ndb_deferred_constraints	Yes	Yes	Yes		Both	Yes
ndb_distribution	Yes	Yes	Yes		Global	Yes
ndb_distribution	Yes	Yes	Yes		Global	Yes
Ndb_epoch_delete_delete_count				Yes	Global	No
ndb_eventbuffer_free_percent	Yes	Yes	Yes		Global	Yes
ndb_eventbuffer_max_alloc	Yes	Yes	Yes		Global	Yes
Ndb_execute_count				Yes	Global	No
ndb_extra_logging	Yes	Yes	Yes		Global	Yes
ndb_force_send	Yes	Yes	Yes		Both	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
ndb_index_stat_cache_entries	Yes	Yes	Yes		Both	Yes
ndb_index_stat_enable	Yes	Yes	Yes		Both	Yes
ndb_index_stat_option	Yes	Yes	Yes		Both	Yes
ndb_index_stat_update_freq	Yes	Yes	Yes		Both	Yes
ndb_join_pushdown			Yes		Both	Yes
Ndb_last_commit_epoch_server				Yes	Global	No
Ndb_last_commit_epoch_session				Yes	Session	No
ndb_log_apply_status	Yes	Yes	Yes		Global	No
ndb_log_apply_status	Yes	Yes	Yes		Global	No
ndb_log_bin	Yes		Yes		Both	Yes
ndb_log_binlog_index	Yes		Yes		Global	Yes
ndb_log_empty_epochs	Yes	Yes	Yes		Global	Yes
ndb_log_empty_epochs	Yes	Yes	Yes		Global	Yes
ndb_log_empty_update	Yes	Yes	Yes		Global	Yes
ndb_log_empty_update	Yes	Yes	Yes		Global	Yes
ndb_log_exclusive_reads	Yes	Yes	Yes		Both	Yes
ndb_log_exclusive_reads	Yes	Yes	Yes		Both	Yes
ndb_log_fail_terminate	Yes	Yes	Yes		Global	No
ndb_log_orig	Yes	Yes	Yes		Global	No
ndb_log_orig	Yes	Yes	Yes		Global	No
ndb_log_transaction_id	Yes	Yes	Yes		Global	No
ndb_log_transaction_id			Yes		Global	No
ndb_log_update_as_write	Yes	Yes	Yes		Global	Yes
ndb_log_update_minimal	Yes	Yes	Yes		Global	Yes
ndb_log_updated_only	Yes	Yes	Yes		Global	Yes
ndb-mgmd-host	Yes	Yes				
ndb_nodeid	Yes	Yes		Yes	Global	No
Ndb_number_of_data_nodes				Yes	Global	No
ndb_optimization_delay	Yes	Yes	Yes		Global	Yes
ndb_optimized_node_selection	Yes	Yes	Yes		Global	No
Ndb_pruned_scan_count				Yes	Global	No
Ndb_pushed_queries_defined				Yes	Global	No
Ndb_pushed_queries_dropped				Yes	Global	No
Ndb_pushed_queries_executed				Yes	Global	No
Ndb_pushed_reads				Yes	Global	No
ndb_recv_thread_activation_threshold	Yes	Yes	Yes		Global	Yes
ndb_recv_thread_cpu_mask	Yes	Yes	Yes		Global	Yes
ndb_report_thresh_binlog_epoch_slip	Yes	Yes	Yes		Global	Yes
ndb_report_thresh_binlog_mem_usage	Yes	Yes	Yes		Global	Yes
Ndb_scan_count				Yes	Global	No
ndb_show_foreign_key_mock_tables	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
ndb_slave_conflict_role	Yes	Yes	Yes		Global	Yes
Ndb_slave_max_replicated_epoch			Yes		Global	No
ndb_table_no_logging			Yes		Session	Yes
ndb_table_temporary			Yes		Session	Yes
ndb-transid-mysql-connection-map	Yes					
ndb_use_copying_alter_table			Yes		Both	No
ndb_use_exact_count			Yes		Both	Yes
ndb_use_transactions	Yes	Yes	Yes		Both	Yes
ndb_version			Yes		Global	No
ndb_version_string			Yes		Global	No
ndb_wait_connected	Yes	Yes	Yes		Global	No
ndb_wait_setup	Yes	Yes	Yes		Global	No
ndbcluster	Yes	Yes				
ndbinfo_database			Yes		Global	No
ndbinfo_max_bytes	Yes		Yes		Both	Yes
ndbinfo_max_rows	Yes		Yes		Both	Yes
ndbinfo_offline			Yes		Global	Yes
ndbinfo_show_hidden	Yes		Yes		Both	Yes
ndbinfo_table_prefix			Yes		Global	No
ndbinfo_version			Yes		Global	No
net_buffer_length	Yes	Yes	Yes		Both	Yes
net_read_timeout	Yes	Yes	Yes		Both	Yes
net_retry_count	Yes	Yes	Yes		Both	Yes
net_write_timeout	Yes	Yes	Yes		Both	Yes
new	Yes	Yes	Yes		Both	Yes
no-defaults	Yes					
Not_flushed_delayed_rows				Yes	Global	No
old	Yes	Yes	Yes		Global	No
old_alter_table	Yes	Yes	Yes		Both	Yes
old_passwords	Yes	Yes	Yes		Both	Yes
old-style-user-limits	Yes	Yes				
Open_files				Yes	Global	No
open_files_limit	Yes	Yes	Yes		Global	No
Open_streams				Yes	Global	No
Open_table_definitions				Yes	Global	No
Open_tables				Yes	Both	No
Opened_files				Yes	Global	No
Opened_table_definitions				Yes	Both	No
Opened_tables				Yes	Both	No
optimizer_prune_level	Yes	Yes	Yes		Both	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
optimizer_search_depth	Yes	Yes	Yes		Both	Yes
optimizer_switch	Yes	Yes	Yes		Both	Yes
optimizer_trace	Yes	Yes	Yes		Both	Yes
optimizer_trace_features	Yes	Yes	Yes		Both	Yes
optimizer_trace_limit	Yes	Yes	Yes		Both	Yes
optimizer_trace_max_mem_size	Yes	Yes	Yes		Both	Yes
optimizer_trace_offset	Yes	Yes	Yes		Both	Yes
partition	Yes	Yes				
performance_schema	Yes	Yes	Yes		Global	No
Performance_schema_accounts_lost				Yes	Global	No
performance_schema_accounts_size	Yes	Yes	Yes		Global	No
Performance_schema_cond_classes_lost				Yes	Global	No
Performance_schema_cond_instances_lost				Yes	Global	No
performance-schema-consumer-events-stages-current	Yes	Yes				
performance-schema-consumer-events-stages-history	Yes	Yes				
performance-schema-consumer-events-stages-history-long	Yes	Yes				
performance-schema-consumer-events-statements-current	Yes	Yes				
performance-schema-consumer-events-statements-history	Yes	Yes				
performance-schema-consumer-events-statements-history-long	Yes	Yes				
performance-schema-consumer-events-waits-current	Yes	Yes				
performance-schema-consumer-events-waits-history	Yes	Yes				
performance-schema-consumer-events-waits-history-long	Yes	Yes				
performance-schema-	Yes	Yes				

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
consumer-global-instrumentation						
performance-schema-consumer-statements-digest	Yes	Yes				
performance-schema-consumer-thread-instrumentation	Yes	Yes				
Performance_schema_digest_lost				Yes	Global	No
performance_schema_digests_size	Yes	Yes	Yes		Global	No
performance_schema_events_stages_history_long_size	Yes	Yes	Yes		Global	No
performance_schema_events_stages_history_size	Yes	Yes	Yes		Global	No
performance_schema_events_statements_history_long_size	Yes	Yes	Yes		Global	No
performance_schema_events_statements_history_size	Yes	Yes	Yes		Global	No
performance_schema_events_waits_history_long_size	Yes	Yes	Yes		Global	No
performance_schema_events_waits_history_size	Yes	Yes	Yes		Global	No
Performance_schema_file_classes_lost				Yes	Global	No
Performance_schema_file_handles_lost				Yes	Global	No
Performance_schema_file_instances_lost				Yes	Global	No
Performance_schema_hosts_lost				Yes	Global	No
performance_schema_hosts_size	Yes	Yes	Yes		Global	No
performance-schema-instrument	Yes	Yes				
Performance_schema_locker_lost				Yes	Global	No
performance_schema_max_cond_classes	Yes	Yes	Yes		Global	No
performance_schema_max_cond_instances	Yes	Yes	Yes		Global	No
performance_schema_max_digest_length	Yes	Yes	Yes		Global	No
performance_schema_max_file_classes	Yes	Yes	Yes		Global	No
performance_schema_max_file_handles	Yes	Yes	Yes		Global	No
performance_schema_max_file_instances	Yes	Yes	Yes		Global	No
performance_schema_max_mutex_classes	Yes	Yes	Yes		Global	No
performance_schema_max_mutex_instances	Yes	Yes	Yes		Global	No
performance_schema_max_rwlock_classes	Yes	Yes	Yes		Global	No
performance_schema_max_rwlock_instances	Yes	Yes	Yes		Global	No
performance_schema_max_socket_classes	Yes	Yes	Yes		Global	No
performance_schema_max_socket_instances	Yes	Yes	Yes		Global	No
performance_schema_max_stage_classes	Yes	Yes	Yes		Global	No
performance_schema_max_statement_classes	Yes	Yes	Yes		Global	No
performance_schema_max_table_handles	Yes	Yes	Yes		Global	No
performance_schema_max_table_instances	Yes	Yes	Yes		Global	No
performance_schema_max_thread_classes	Yes	Yes	Yes		Global	No
performance_schema_max_thread_instances	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
Performance_schema_mutex_classes_lost				Yes	Global	No
Performance_schema_mutex_instances_lost				Yes	Global	No
Performance_schema_rwlock_classes_lost				Yes	Global	No
Performance_schema_rwlock_instances_lost				Yes	Global	No
Performance_schema_session_connect_attrs_lost				Yes	Global	No
performance_schema_session_connect_attrs_size	Yes	Yes	Yes		Global	No
performance_schema_setup_actors_size	Yes	Yes	Yes		Global	No
performance_schema_setup_objects_size	Yes	Yes	Yes		Global	No
Performance_schema_socket_classes_lost				Yes	Global	No
Performance_schema_socket_instances_lost				Yes	Global	No
Performance_schema_stage_classes_lost				Yes	Global	No
Performance_schema_statement_classes_lost				Yes	Global	No
Performance_schema_table_handles_lost				Yes	Global	No
Performance_schema_table_instances_lost				Yes	Global	No
Performance_schema_thread_classes_lost				Yes	Global	No
Performance_schema_thread_instances_lost				Yes	Global	No
Performance_schema_users_lost				Yes	Global	No
performance_schema_users_size	Yes	Yes	Yes		Global	No
pid_file	Yes	Yes	Yes		Global	No
plugin_dir	Yes	Yes	Yes		Global	No
plugin_load	Yes	Yes	Yes		Global	No
plugin_load_add	Yes	Yes	Yes		Global	No
plugin-xxx	Yes	Yes				
port	Yes	Yes	Yes		Global	No
port-open-timeout	Yes	Yes				
preload_buffer_size	Yes	Yes	Yes		Both	Yes
Prepared_stmt_count				Yes	Global	No
print-defaults	Yes					
profiling			Yes		Both	Yes
profiling_history_size	Yes	Yes	Yes		Both	Yes
protocol_version			Yes		Global	No
proxy_user			Yes		Session	No
pseudo_slave_mode			Yes		Session	Yes
pseudo_thread_id			Yes		Session	Yes
Qcache_free_blocks				Yes	Global	No
Qcache_free_memory				Yes	Global	No
Qcache_hits				Yes	Global	No
Qcache_inserts				Yes	Global	No
Qcache_lowmem_prunes				Yes	Global	No
Qcache_not_cached				Yes	Global	No
Qcache_queries_in_cache				Yes	Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
Qcache_total_blocks				Yes	Global	No
Queries				Yes	Both	No
query_alloc_block_size	Yes	Yes	Yes		Both	Yes
query_cache_limit	Yes	Yes	Yes		Global	Yes
query_cache_min_res_unit	Yes	Yes	Yes		Global	Yes
query_cache_size	Yes	Yes	Yes		Global	Yes
query_cache_type	Yes	Yes	Yes		Both	Yes
query_cache_wlock_invalidate	Yes	Yes	Yes		Both	Yes
query_prealloc_size	Yes	Yes	Yes		Both	Yes
Questions				Yes	Both	No
rand_seed1			Yes		Session	Yes
rand_seed2			Yes		Session	Yes
range_alloc_block_size	Yes	Yes	Yes		Both	Yes
read_buffer_size	Yes	Yes	Yes		Both	Yes
read_only	Yes	Yes	Yes		Global	Yes
read_rnd_buffer_size	Yes	Yes	Yes		Both	Yes
relay_log	Yes	Yes	Yes		Global	No
relay_log_basename			Yes		Global	No
relay_log_index	Yes	Yes	Yes		Global	No
relay_log_info_file	Yes	Yes	Yes		Global	No
relay_log_info_repository	Yes	Yes	Yes		Global	Yes
relay_log_purge	Yes	Yes	Yes		Global	Yes
relay_log_recovery	Yes	Yes	Yes		Global	Varies
relay_log_space_limit	Yes	Yes	Yes		Global	No
remove	Yes					
replicate-do-db	Yes	Yes				
replicate-do-table	Yes	Yes				
replicate-ignore-db	Yes	Yes				
replicate-ignore-table	Yes	Yes				
replicate-rewrite-db	Yes	Yes				
replicate-same-server-id	Yes	Yes				
replicate-wild-do-table	Yes	Yes				
replicate-wild-ignore-table	Yes	Yes				
report_host	Yes	Yes	Yes		Global	No
report_password	Yes	Yes	Yes		Global	No
report_port	Yes	Yes	Yes		Global	No
report_user	Yes	Yes	Yes		Global	No
Rpl_semi_sync_master_clients				Yes	Global	No
rpl_semi_sync_master_enabled	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
Rpl_semi_sync_master_net_avg_wait_time				Yes	Global	No
Rpl_semi_sync_master_net_wait_time				Yes	Global	No
Rpl_semi_sync_master_net_waits				Yes	Global	No
Rpl_semi_sync_master_no_times				Yes	Global	No
Rpl_semi_sync_master_no_tx				Yes	Global	No
Rpl_semi_sync_master_status				Yes	Global	No
Rpl_semi_sync_master_timefunc_failures				Yes	Global	No
rpl_semi_sync_master_timeout	Yes	Yes	Yes		Global	Yes
rpl_semi_sync_master_trace_level	Yes	Yes	Yes		Global	Yes
Rpl_semi_sync_master_tx_avg_wait_time				Yes	Global	No
Rpl_semi_sync_master_tx_wait_time				Yes	Global	No
Rpl_semi_sync_master_tx_waits				Yes	Global	No
rpl_semi_sync_master_wait_no_slave	Yes	Yes	Yes		Global	Yes
Rpl_semi_sync_master_wait_pos_backtraverse				Yes	Global	No
Rpl_semi_sync_master_wait_sessions				Yes	Global	No
Rpl_semi_sync_master_yes_tx				Yes	Global	No
rpl_semi_sync_slave_enabled	Yes	Yes	Yes		Global	Yes
Rpl_semi_sync_slave_status				Yes	Global	No
rpl_semi_sync_slave_trace_level	Yes	Yes	Yes		Global	Yes
rpl_stop_slave_timeout	Yes	Yes	Yes		Global	Yes
Rsa_public_key				Yes	Global	No
safe-user-create	Yes	Yes				
secure_auth	Yes	Yes	Yes		Global	Yes
secure_file_priv	Yes	Yes	Yes		Global	No
Select_full_join				Yes	Both	No
Select_full_range_join				Yes	Both	No
Select_range				Yes	Both	No
Select_range_check				Yes	Both	No
Select_scan				Yes	Both	No
server_id	Yes	Yes	Yes		Global	Yes
server_id_bits	Yes	Yes	Yes		Global	No
server_uuid			Yes		Global	No
sha256_password_private_key_path	Yes	Yes	Yes		Global	No
sha256_password_public_key_path	Yes	Yes	Yes		Global	No
shared_memory	Yes	Yes	Yes		Global	No
shared_memory_base_name	Yes	Yes	Yes		Global	No
show_old_temporals	Yes	Yes	Yes		Both	Yes
show-slave-auth-info	Yes	Yes				
simplified_binlog_gtid_recovery	Yes	Yes	Yes		Global	No
skip-character-set-client-handshake	Yes	Yes				

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
skip_external_locking	Yes	Yes	Yes		Global	No
skip-grant-tables	Yes	Yes				
skip-host-cache	Yes	Yes				
skip_name_resolve	Yes	Yes	Yes		Global	No
skip-ndbcluster	Yes	Yes				
skip_networking	Yes	Yes	Yes		Global	No
skip-new	Yes	Yes				
skip-partition	Yes	Yes				
skip_show_database	Yes	Yes	Yes		Global	No
skip-slave-start	Yes	Yes				
skip-ssl	Yes	Yes				
skip-stack-trace	Yes	Yes				
slave_allow_batching	Yes	Yes	Yes		Global	Yes
slave_checkpoint_group	Yes	Yes	Yes		Global	Yes
slave_checkpoint_period	Yes	Yes	Yes		Global	Yes
slave_compressed_protocol	Yes	Yes	Yes		Global	Yes
slave_exec_mode	Yes	Yes	Yes		Global	Yes
Slave_heartbeat_period				Yes	Global	No
Slave_last_heartbeat				Yes	Global	No
slave_load_tmpdir	Yes	Yes	Yes		Global	No
slave_max_allowed_packet	Yes	Yes	Yes		Global	Yes
slave_net_timeout	Yes	Yes	Yes		Global	Yes
Slave_open_temp_tables				Yes	Global	No
slave_parallel_workers	Yes	Yes	Yes		Global	Yes
slave_pending_jobs_size_max	Yes	Yes	Yes		Global	Yes
Slave_received_heartbeats				Yes	Global	No
Slave_retried_transactions				Yes	Global	No
Slave_rows_last_search_algorithm_used				Yes	Global	No
slave_rows_search_algorithms	Yes	Yes	Yes		Global	Yes
Slave_running				Yes	Global	No
slave_skip_errors	Yes	Yes	Yes		Global	No
slave-sql-verify-checksum	Yes	Yes				
slave_sql_verify_checksum	Yes	Yes	Yes		Global	Yes
slave_transaction_retries	Yes	Yes	Yes		Global	Yes
slave_type_conversions	Yes	Yes	Yes		Global	No
Slow_launch_threads				Yes	Both	No
slow_launch_time	Yes	Yes	Yes		Global	Yes
Slow_queries				Yes	Both	No
slow_query_log	Yes	Yes	Yes		Global	Yes
slow_query_log_file	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
slow-start-timeout	Yes	Yes				
socket	Yes	Yes	Yes		Global	No
sort_buffer_size	Yes	Yes	Yes		Both	Ye
Sort_merge_passes				Yes	Both	No
Sort_range				Yes	Both	No
Sort_rows				Yes	Both	No
Sort_scan				Yes	Both	No
sporadic-binlog-dump-fail	Yes	Yes				
sql_auto_is_null			Yes		Both	Ye
sql_big_selects			Yes		Both	Ye
sql_buffer_result			Yes		Both	Ye
sql_log_bin			Yes		Session	Ye
sql_log_off			Yes		Both	Ye
sql_mode	Yes	Yes	Yes		Both	Ye
sql_notes			Yes		Both	Ye
sql_quote_show_create			Yes		Both	Ye
sql_safe_updates			Yes		Both	Ye
sql_select_limit			Yes		Both	Ye
sql_slave_skip_counter			Yes		Global	Ye
sql_warnings			Yes		Both	Ye
ssl	Yes	Yes				
Ssl_accept_renegotiates				Yes	Global	No
Ssl_accepts				Yes	Global	No
ssl_ca	Yes	Yes	Yes		Global	No
Ssl_callback_cache_hits				Yes	Global	No
ssl_capath	Yes	Yes	Yes		Global	No
ssl_cert	Yes	Yes	Yes		Global	No
Ssl_cipher				Yes	Both	No
ssl_cipher	Yes	Yes	Yes		Global	No
Ssl_cipher_list				Yes	Both	No
Ssl_client_connects				Yes	Global	No
Ssl_connect_renegotiates				Yes	Global	No
ssl_crl	Yes	Yes	Yes		Global	No
ssl_crlpath	Yes	Yes	Yes		Global	No
Ssl_ctx_verify_depth				Yes	Global	No
Ssl_ctx_verify_mode				Yes	Global	No
Ssl_default_timeout				Yes	Both	No
Ssl_finished_accepts				Yes	Global	No
Ssl_finished_connects				Yes	Global	No
ssl_key	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
Ssl_server_not_after				Yes	Both	No
Ssl_server_not_before				Yes	Both	No
Ssl_session_cache_hits				Yes	Global	No
Ssl_session_cache_misses				Yes	Global	No
Ssl_session_cache_mode				Yes	Global	No
Ssl_session_cache_overflows				Yes	Global	No
Ssl_session_cache_size				Yes	Global	No
Ssl_session_cache_timeouts				Yes	Global	No
Ssl_sessions_reused				Yes	Both	No
Ssl_used_session_cache_entries				Yes	Global	No
Ssl_verify_depth				Yes	Both	No
Ssl_verify_mode				Yes	Both	No
Ssl_version				Yes	Both	No
standalone	Yes	Yes				
storage_engine			Yes		Both	Yes
stored_program_cache	Yes	Yes	Yes		Global	Yes
super-large-pages	Yes	Yes				
symbolic-links	Yes	Yes				
sync_binlog	Yes	Yes	Yes		Global	Yes
sync_frm	Yes	Yes	Yes		Global	Yes
sync_master_info	Yes	Yes	Yes		Global	Yes
sync_relay_log	Yes	Yes	Yes		Global	Yes
sync_relay_log_info	Yes	Yes	Yes		Global	Yes
sysdate-is-now	Yes	Yes				
system_time_zone			Yes		Global	No
table_definition_cache	Yes	Yes	Yes		Global	Yes
Table_locks_immediate				Yes	Global	No
Table_locks_waited				Yes	Global	No
table_open_cache	Yes	Yes	Yes		Global	Yes
Table_open_cache_hits				Yes	Both	No
table_open_cache_instances	Yes	Yes	Yes		Global	No
Table_open_cache_misses				Yes	Both	No
Table_open_cache_overflows				Yes	Both	No
tc-heuristic-recover	Yes	Yes				
Tc_log_max_pages_used				Yes	Global	No
Tc_log_page_size				Yes	Global	No
Tc_log_page_waits				Yes	Global	No
temp-pool	Yes	Yes				
thread_cache_size	Yes	Yes	Yes		Global	Yes
thread_concurrency	Yes	Yes	Yes		Global	No
thread_handling	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
thread_pool_algorithm	Yes	Yes	Yes		Global	No
thread_pool_high_priority_connection	Yes	Yes	Yes		Both	Yes
thread_pool_max_unused_threads	Yes	Yes	Yes		Global	Yes
thread_pool_prio_kickup_timer	Yes	Yes	Yes		Both	Yes
thread_pool_size	Yes	Yes	Yes		Global	No
thread_pool_stall_limit	Yes	Yes	Yes		Global	Yes
thread_stack	Yes	Yes	Yes		Global	No
Threads_cached				Yes	Global	No
Threads_connected				Yes	Global	No
Threads_created				Yes	Global	No
Threads_running				Yes	Global	No
time_format			Yes		Global	No
time_zone			Yes		Both	Yes
timed_mutexes	Yes	Yes	Yes		Global	Yes
timestamp			Yes		Session	Yes
tmp_table_size	Yes	Yes	Yes		Both	Yes
tmpdir	Yes	Yes	Yes		Global	No
transaction_alloc_block_size	Yes	Yes	Yes		Both	Yes
transaction_allow_batching			Yes		Session	Yes
transaction-isolation	Yes	Yes				
- *Variable*: tx_isolation						
transaction_prealloc_size	Yes	Yes	Yes		Both	Yes
transaction-read-only	Yes	Yes				
- *Variable*: tx_read_only						
tx_isolation			Yes		Both	Yes
tx_read_only			Yes		Both	Yes
unique_checks			Yes		Both	Yes
updatable_views_with_limit	Yes	Yes	Yes		Both	Yes
Uptime				Yes	Global	No
Uptime_since_flush_status				Yes	Global	No
user	Yes	Yes				
validate-password	Yes	Yes				
validate_password_dictionary_file	Yes	Yes	Yes		Global	Var
validate_password_dictionary_file_last_parsed				Yes	Global	No
validate_password_dictionary_file_words_count				Yes	Global	No
validate_password_length	Yes	Yes	Yes		Global	Yes
validate_password_mixed_case_count	Yes	Yes	Yes		Global	Yes
validate_password_number_count	Yes	Yes	Yes		Global	Yes
validate_password_policy	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
validate_password_special_char_count	Yes	Yes	Yes		Global	Yes
validate_user_plugins	Yes	Yes	Yes		Global	No
verbose	Yes	Yes				
version			Yes		Global	No
version_comment			Yes		Global	No
version_compile_machine			Yes		Global	No
version_compile_os			Yes		Global	No
wait_timeout	Yes	Yes	Yes		Both	Yes
warning_count			Yes		Session	No

Notes:

1. This option is dynamic, but should be set only by server. You should not set this variable manually.

5.1.4 Server System Variable Reference

The following table lists all system variables applicable within `mysqld`.

The table lists command-line options (Cmd-line), options valid in configuration files (Option file), server system variables (System Var), and status variables (Status var) in one unified list, with an indication of where each option or variable is valid. If a server option set on the command line or in an option file differs from the name of the corresponding system variable, the variable name is noted immediately below the corresponding option. The scope of the variable (Var Scope) is Global, Session, or both. Please see the corresponding item descriptions for details on setting and using the variables. Where appropriate, direct links to further information about the items are provided.

Table 5.3 System Variable Summary

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
audit_log_buffer_size	Yes	Yes	Yes	Global	No
audit_log_connection_policy	Yes	Yes	Yes	Global	Yes
audit_log_current_session			Yes	Both	No
audit_log_exclude_accounts	Yes	Yes	Yes	Global	Yes
audit_log_file	Yes	Yes	Yes	Global	No
audit_log_flush			Yes	Global	Yes
audit_log_format	Yes	Yes	Yes	Global	No
audit_log_include_accounts	Yes	Yes	Yes	Global	Yes
audit_log_policy	Yes	Yes	Yes	Global	Varies
audit_log_rotate_on_size	Yes	Yes	Yes	Global	Yes
audit_log_statement_policy	Yes	Yes	Yes	Global	Yes
audit_log_strategy	Yes	Yes	Yes	Global	No
authentication_windows_log_level	Yes	Yes	Yes	Global	No
authentication_windows_use_principal_name	Yes	Yes	Yes	Global	No
auto_increment_increment	Yes	Yes	Yes	Both	Yes
auto_increment_offset	Yes	Yes	Yes	Both	Yes
autocommit	Yes	Yes	Yes	Both	Yes
automatic_sp_privileges	Yes	Yes	Yes	Global	Yes
avoid_temporal_upgrade	Yes	Yes	Yes	Global	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
back_log	Yes	Yes	Yes	Global	No
basedir	Yes	Yes	Yes	Global	No
big_tables	Yes	Yes	Yes	Both	Yes
bind_address	Yes	Yes	Yes	Global	No
binlog_cache_size	Yes	Yes	Yes	Global	Yes
binlog_checksum	Yes	Yes	Yes	Global	Yes
binlog_direct_non_transactional_updates	Yes	Yes	Yes	Both	Yes
binlog_error_action	Yes	Yes	Yes	Global	Yes
binlog_format	Yes	Yes	Yes	Both	Yes
binlog_gtid_simple_recovery	Yes	Yes	Yes	Global	No
binlog_max_flush_queue_time	Yes	Yes	Yes	Global	Yes
binlog_order_commits	Yes	Yes	Yes	Global	Yes
binlog_row_image	Yes	Yes	Yes	Both	Yes
binlog_rows_query_log_events	Yes	Yes	Yes	Both	Yes
binlog_stmt_cache_size	Yes	Yes	Yes	Global	Yes
binlogging_impossible_mode	Yes	Yes	Yes	Both	Yes
block_encryption_mode	Yes	Yes	Yes	Both	Yes
bulk_insert_buffer_size	Yes	Yes	Yes	Both	Yes
character_set_client			Yes	Both	Yes
character_set_connection			Yes	Both	Yes
character_set_database (note 1)			Yes	Both	Yes
character_set_filesystem	Yes	Yes	Yes	Both	Yes
character_set_results			Yes	Both	Yes
character_set_server	Yes	Yes	Yes	Both	Yes
character_set_system			Yes	Global	No
character_sets_dir	Yes	Yes	Yes	Global	No
collation_connection			Yes	Both	Yes
collation_database (note 1)			Yes	Both	Yes
collation_server	Yes	Yes	Yes	Both	Yes
completion_type	Yes	Yes	Yes	Both	Yes
concurrent_insert	Yes	Yes	Yes	Global	Yes
connect_timeout	Yes	Yes	Yes	Global	Yes
connection_control_failed_connections_threshold	Yes	Yes	Yes	Global	Yes
connection_control_max_connection_delay	Yes	Yes	Yes	Global	Yes
connection_control_min_connection_delay	Yes	Yes	Yes	Global	Yes
core_file			Yes	Global	No
create_old_temporals	Yes	Yes	Yes	Global	No
daemon_memcached_enable_binlog	Yes	Yes	Yes	Global	No
daemon_memcached_engine_lib_name	Yes	Yes	Yes	Global	No

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
daemon_memcached_engine_lib_path	Yes	Yes	Yes	Global	No
daemon_memcached_option	Yes	Yes	Yes	Global	No
daemon_memcached_r_batch_size	Yes	Yes	Yes	Global	No
daemon_memcached_w_batch_size	Yes	Yes	Yes	Global	No
datadir	Yes	Yes	Yes	Global	No
date_format			Yes	Global	No
datetime_format			Yes	Global	No
debug	Yes	Yes	Yes	Both	Yes
debug_sync			Yes	Session	Yes
default_storage_engine	Yes	Yes	Yes	Both	Yes
default_tmp_storage_engine	Yes	Yes	Yes	Both	Yes
default_week_format	Yes	Yes	Yes	Both	Yes
delay_key_write	Yes	Yes	Yes	Global	Yes
delayed_insert_limit	Yes	Yes	Yes	Global	Yes
delayed_insert_timeout	Yes	Yes	Yes	Global	Yes
delayed_queue_size	Yes	Yes	Yes	Global	Yes
disconnect_on_expired_password	Yes	Yes	Yes	Global	No
div_precision_increment	Yes	Yes	Yes	Both	Yes
end_markers_in_json	Yes	Yes	Yes	Both	Yes
enforce_gtid_consistency	Yes	Yes	Yes	Global	No
eq_range_index_dive_limit	Yes	Yes	Yes	Both	Yes
error_count			Yes	Session	No
event_scheduler	Yes	Yes	Yes	Global	Yes
expire_logs_days	Yes	Yes	Yes	Global	Yes
explicit_defaults_for_timestamp	Yes	Yes	Yes	Both	Yes
external_user			Yes	Session	No
flush	Yes	Yes	Yes	Global	Yes
flush_time	Yes	Yes	Yes	Global	Yes
foreign_key_checks			Yes	Both	Yes
ft_boolean_syntax	Yes	Yes	Yes	Global	Yes
ft_max_word_len	Yes	Yes	Yes	Global	No
ft_min_word_len	Yes	Yes	Yes	Global	No
ft_query_expansion_limit	Yes	Yes	Yes	Global	No
ft_stopword_file	Yes	Yes	Yes	Global	No
general_log	Yes	Yes	Yes	Global	Yes
general_log_file	Yes	Yes	Yes	Global	Yes
group_concat_max_len	Yes	Yes	Yes	Both	Yes
gtid_executed			Yes	Both	No
gtid_mode	Yes	Yes	Yes	Global	No
gtid_next			Yes	Session	Yes
gtid_owned			Yes	Both	No

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
gtid_purged			Yes	Global	Yes
have_compress			Yes	Global	No
have_crypt			Yes	Global	No
have_dynamic_loading			Yes	Global	No
have_geometry			Yes	Global	No
have_openssl			Yes	Global	No
have_profiling			Yes	Global	No
have_query_cache			Yes	Global	No
have_rtree_keys			Yes	Global	No
have_ssl			Yes	Global	No
have_symlink			Yes	Global	No
host_cache_size	Yes	Yes	Yes	Global	Yes
hostname			Yes	Global	No
identity			Yes	Session	Yes
ignore_builtin_innodb	Yes	Yes	Yes	Global	No
ignore_db_dirs			Yes	Global	No
init_connect	Yes	Yes	Yes	Global	Yes
init_file	Yes	Yes	Yes	Global	No
init_slave	Yes	Yes	Yes	Global	Yes
innodb_adaptive_flushing	Yes	Yes	Yes	Global	Yes
innodb_adaptive_flushing_lwm	Yes	Yes	Yes	Global	Yes
innodb_adaptive_hash_index	Yes	Yes	Yes	Global	Yes
innodb_adaptive_max_sleep_delay	Yes	Yes	Yes	Global	Yes
innodb_additional_mem_pool_size	Yes	Yes	Yes	Global	No
innodb_api_bk_commit_interval	Yes	Yes	Yes	Global	Yes
innodb_api_disable_rowlock	Yes	Yes	Yes	Global	No
innodb_api_enable_binlog	Yes	Yes	Yes	Global	No
innodb_api_enable_mdl	Yes	Yes	Yes	Global	No
innodb_api_trx_level	Yes	Yes	Yes	Global	Yes
innodb_autoextend_increment	Yes	Yes	Yes	Global	Yes
innodb_autoinc_lock_mode	Yes	Yes	Yes	Global	No
innodb_buffer_pool_dump_at_shutdown	Yes	Yes	Yes	Global	Yes
innodb_buffer_pool_dump_now	Yes	Yes	Yes	Global	Yes
innodb_buffer_pool_filename	Yes	Yes	Yes	Global	Yes
innodb_buffer_pool_instances	Yes	Yes	Yes	Global	No
innodb_buffer_pool_load_abort	Yes	Yes	Yes	Global	Yes
innodb_buffer_pool_load_at_startup	Yes	Yes	Yes	Global	No
innodb_buffer_pool_load_now	Yes	Yes	Yes	Global	Yes
innodb_buffer_pool_size	Yes	Yes	Yes	Global	No
innodb_change_buffer_max_size	Yes	Yes	Yes	Global	Yes
innodb_change_buffering	Yes	Yes	Yes	Global	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
innodb_change_buffering_debug	Yes	Yes	Yes	Global	Yes
innodb_checksum_algorithm	Yes	Yes	Yes	Global	Yes
innodb_checksums	Yes	Yes	Yes	Global	No
innodb_cmp_per_index_enabled	Yes	Yes	Yes	Global	Yes
innodb_commit_concurrency	Yes	Yes	Yes	Global	Yes
innodb_compression_failure_threshold_pct	Yes	Yes	Yes	Global	Yes
innodb_compression_level	Yes	Yes	Yes	Global	Yes
innodb_compression_pad_pct_max	Yes	Yes	Yes	Global	Yes
innodb_concurrency_tickets	Yes	Yes	Yes	Global	Yes
innodb_data_file_path	Yes	Yes	Yes	Global	No
innodb_data_home_dir	Yes	Yes	Yes	Global	No
innodb_disable_sort_file_cache	Yes	Yes	Yes	Global	Yes
innodb_doublewrite	Yes	Yes	Yes	Global	No
innodb_fast_shutdown	Yes	Yes	Yes	Global	Yes
innodb_fil_make_page_dirty_debug	Yes	Yes	Yes	Global	Yes
innodb_file_format	Yes	Yes	Yes	Global	Yes
innodb_file_format_check	Yes	Yes	Yes	Global	No
innodb_file_format_max	Yes	Yes	Yes	Global	Yes
innodb_file_per_table	Yes	Yes	Yes	Global	Yes
innodb_flush_log_at_timeout	Yes	Yes	Yes	Global	Yes
innodb_flush_log_at_trx_commit	Yes	Yes	Yes	Global	Yes
innodb_flush_method	Yes	Yes	Yes	Global	No
innodb_flush_neighbors	Yes	Yes	Yes	Global	Yes
innodb_flushing_avg_loops	Yes	Yes	Yes	Global	Yes
innodb_force_load_corrupted	Yes	Yes	Yes	Global	No
innodb_force_recovery	Yes	Yes	Yes	Global	No
innodb_ft_aux_table			Yes	Global	Yes
innodb_ft_cache_size	Yes	Yes	Yes	Global	No
innodb_ft_enable_diag_print	Yes	Yes	Yes	Global	Yes
innodb_ft_enable_stopword	Yes	Yes	Yes	Both	Yes
innodb_ft_max_token_size	Yes	Yes	Yes	Global	No
innodb_ft_min_token_size	Yes	Yes	Yes	Global	No
innodb_ft_num_word_optimize	Yes	Yes	Yes	Global	Yes
innodb_ft_result_cache_limit	Yes	Yes	Yes	Global	Yes
innodb_ft_server_stopword_table	Yes	Yes	Yes	Global	Yes
innodb_ft_sort_pll_degree	Yes	Yes	Yes	Global	No
innodb_ft_total_cache_size	Yes	Yes	Yes	Global	No
innodb_ft_user_stopword_table	Yes	Yes	Yes	Both	Yes
innodb_io_capacity	Yes	Yes	Yes	Global	Yes
innodb_io_capacity_max	Yes	Yes	Yes	Global	Yes
innodb_large_prefix	Yes	Yes	Yes	Global	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
innodb_limit_optimistic_insert_debug	Yes	Yes	Yes	Global	Yes
innodb_lock_wait_timeout	Yes	Yes	Yes	Both	Yes
innodb_locks_unsafe_for_binlog	Yes	Yes	Yes	Global	No
innodb_log_buffer_size	Yes	Yes	Yes	Global	No
innodb_log_checkpoint_now	Yes	Yes	Yes	Global	Yes
innodb_log_compressed_pages	Yes	Yes	Yes	Global	Yes
innodb_log_file_size	Yes	Yes	Yes	Global	No
innodb_log_files_in_group	Yes	Yes	Yes	Global	No
innodb_log_group_home_dir	Yes	Yes	Yes	Global	No
innodb_lru_scan_depth	Yes	Yes	Yes	Global	Yes
innodb_max_dirty_pages_pct	Yes	Yes	Yes	Global	Yes
innodb_max_dirty_pages_pct_lwm	Yes	Yes	Yes	Global	Yes
innodb_max_purge_lag	Yes	Yes	Yes	Global	Yes
innodb_max_purge_lag_delay	Yes	Yes	Yes	Global	Yes
innodb_mirrored_log_groups	Yes	Yes	Yes	Global	No
innodb_monitor_disable	Yes	Yes	Yes	Global	Yes
innodb_monitor_enable	Yes	Yes	Yes	Global	Yes
innodb_monitor_reset	Yes	Yes	Yes	Global	Yes
innodb_monitor_reset_all	Yes	Yes	Yes	Global	Yes
innodb_numa_interleave	Yes	Yes	Yes	Global	No
innodb_old_blocks_pct	Yes	Yes	Yes	Global	Yes
innodb_old_blocks_time	Yes	Yes	Yes	Global	Yes
innodb_online_alter_log_max_size	Yes	Yes	Yes	Global	Yes
innodb_open_files	Yes	Yes	Yes	Global	No
innodb_optimize_fulltext_only	Yes	Yes	Yes	Global	Yes
innodb_page_size	Yes	Yes	Yes	Global	No
innodb_print_all_deadlocks	Yes	Yes	Yes	Global	Yes
innodb_purge_batch_size	Yes	Yes	Yes	Global	Yes
innodb_purge_threads	Yes	Yes	Yes	Global	No
innodb_random_read_ahead	Yes	Yes	Yes	Global	Yes
innodb_read_ahead_threshold	Yes	Yes	Yes	Global	Yes
innodb_read_io_threads	Yes	Yes	Yes	Global	No
innodb_read_only	Yes	Yes	Yes	Global	No
innodb_replication_delay	Yes	Yes	Yes	Global	Yes
innodb_rollback_on_timeout	Yes	Yes	Yes	Global	No
innodb_rollback_segments	Yes	Yes	Yes	Global	Yes
innodb_saved_page_number_debug	Yes	Yes	Yes	Global	Yes
innodb_sort_buffer_size	Yes	Yes	Yes	Global	No
innodb_spin_wait_delay	Yes	Yes	Yes	Global	Yes
innodb_stats_auto_recalc	Yes	Yes	Yes	Global	Yes
innodb_stats_include_delete_marked	Yes	Yes	Yes	Global	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
innodb_stats_method	Yes	Yes	Yes	Global	Yes
innodb_stats_on_metadata	Yes	Yes	Yes	Global	Yes
innodb_stats_persistent	Yes	Yes	Yes	Global	Yes
innodb_stats_persistent_sample_pages	Yes	Yes	Yes	Global	Yes
innodb_stats_sample_pages	Yes	Yes	Yes	Global	Yes
innodb_stats_transient_sample_pages	Yes	Yes	Yes	Global	Yes
innodb_status_output	Yes	Yes	Yes	Global	Yes
innodb_status_output_locks	Yes	Yes	Yes	Global	Yes
innodb_strict_mode	Yes	Yes	Yes	Both	Yes
innodb_support_xa	Yes	Yes	Yes	Both	Yes
innodb_sync_array_size	Yes	Yes	Yes	Global	No
innodb_sync_spin_loops	Yes	Yes	Yes	Global	Yes
innodb_table_locks	Yes	Yes	Yes	Both	Yes
innodb_thread_concurrency	Yes	Yes	Yes	Global	Yes
innodb_thread_sleep_delay	Yes	Yes	Yes	Global	Yes
innodb_tmpdir	Yes	Yes	Yes	Both	Yes
innodb_trx_purge_view_update_only_debug	Yes	Yes	Yes	Global	Yes
innodb_trx_rseg_n_slots_debug	Yes	Yes	Yes	Global	Yes
innodb_undo_directory	Yes	Yes	Yes	Global	No
innodb_undo_logs	Yes	Yes	Yes	Global	Yes
innodb_undo_tablespaces	Yes	Yes	Yes	Global	No
innodb_use_native_aio	Yes	Yes	Yes	Global	No
innodb_use_sys_malloc	Yes	Yes	Yes	Global	No
innodb_version			Yes	Global	No
innodb_write_io_threads	Yes	Yes	Yes	Global	No
insert_id			Yes	Session	Yes
interactive_timeout	Yes	Yes	Yes	Both	Yes
join_buffer_size	Yes	Yes	Yes	Both	Yes
keep_files_on_create	Yes	Yes	Yes	Both	Yes
key_buffer_size	Yes	Yes	Yes	Global	Yes
key_cache_age_threshold	Yes	Yes	Yes	Global	Yes
key_cache_block_size	Yes	Yes	Yes	Global	Yes
key_cache_division_limit	Yes	Yes	Yes	Global	Yes
language	Yes	Yes	Yes	Global	No
large_files_support			Yes	Global	No
large_page_size			Yes	Global	No
large_pages	Yes	Yes	Yes	Global	No
last_insert_id			Yes	Session	Yes
lc_messages	Yes	Yes	Yes	Both	Yes
lc_messages_dir	Yes	Yes	Yes	Global	No
lc_time_names	Yes	Yes	Yes	Both	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
license			Yes	Global	No
local_infile	Yes	Yes	Yes	Global	Yes
lock_wait_timeout	Yes	Yes	Yes	Both	Yes
locked_in_memory			Yes	Global	No
log_bin			Yes	Global	No
log_bin_basename			Yes	Global	No
log_bin_index	Yes	Yes	Yes	Global	No
log_bin_trust_function_creators	Yes	Yes	Yes	Global	Yes
log_bin_use_v1_row_events	Yes	Yes	Yes	Global	No
log_error	Yes	Yes	Yes	Global	No
log_output	Yes	Yes	Yes	Global	Yes
log_queries_not_using_indexes	Yes	Yes	Yes	Global	Yes
log_slave_updates	Yes	Yes	Yes	Global	No
log_slow_admin_statements	Yes	Yes	Yes	Global	Yes
log_slow_slave_statements	Yes	Yes	Yes	Global	Yes
log_throttle_queries_not_using_indexes	Yes	Yes	Yes	Global	Yes
log_warnings	Yes	Yes	Yes	Varies	Yes
long_query_time	Yes	Yes	Yes	Both	Yes
low_priority_updates	Yes	Yes	Yes	Both	Yes
lower_case_file_system			Yes	Global	No
lower_case_table_names	Yes	Yes	Yes	Global	No
master_info_repository	Yes	Yes	Yes	Global	Yes
master_verify_checksum	Yes	Yes	Yes	Global	Yes
max_allowed_packet	Yes	Yes	Yes	Both	Yes
max_binlog_cache_size	Yes	Yes	Yes	Global	Yes
max_binlog_size	Yes	Yes	Yes	Global	Yes
max_binlog_stmt_cache_size	Yes	Yes	Yes	Global	Yes
max_connect_errors	Yes	Yes	Yes	Global	Yes
max_connections	Yes	Yes	Yes	Global	Yes
max_delayed_threads	Yes	Yes	Yes	Both	Yes
max_digest_length	Yes	Yes	Yes	Global	No
max_error_count	Yes	Yes	Yes	Both	Yes
max_heap_table_size	Yes	Yes	Yes	Both	Yes
max_insert_delayed_threads			Yes	Both	Yes
max_join_size	Yes	Yes	Yes	Both	Yes
max_length_for_sort_data	Yes	Yes	Yes	Both	Yes
max_prepared_stmt_count	Yes	Yes	Yes	Global	Yes
max_relay_log_size	Yes	Yes	Yes	Global	Yes
max_seeks_for_key	Yes	Yes	Yes	Both	Yes
max_sort_length	Yes	Yes	Yes	Both	Yes
max_sp_recursion_depth	Yes	Yes	Yes	Both	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
max_tmp_tables			Yes	Both	Yes
max_user_connections	Yes	Yes	Yes	Both	Yes
max_write_lock_count	Yes	Yes	Yes	Global	Yes
metadata_locks_cache_size	Yes	Yes	Yes	Global	No
metadata_locks_hash_instances	Yes	Yes	Yes	Global	No
min_examined_row_limit	Yes	Yes	Yes	Both	Yes
multi_range_count	Yes	Yes	Yes	Both	Yes
myisam_data_pointer_size	Yes	Yes	Yes	Global	Yes
myisam_max_sort_file_size	Yes	Yes	Yes	Global	Yes
myisam_mmap_size	Yes	Yes	Yes	Global	No
myisam_recover_options	Yes	Yes	Yes	Global	No
myisam_repair_threads	Yes	Yes	Yes	Both	Yes
myisam_sort_buffer_size	Yes	Yes	Yes	Both	Yes
myisam_stats_method	Yes	Yes	Yes	Both	Yes
myisam_use_mmap	Yes	Yes	Yes	Global	Yes
mysql_firewall_max_query_size	Yes	Yes	Yes	Global	No
mysql_firewall_mode	Yes	Yes	Yes	Global	Yes
mysql_firewall_trace	Yes	Yes	Yes	Global	Yes
named_pipe	Yes	Yes	Yes	Global	No
named_pipe_full_access_group	Yes	Yes	Yes	Global	No
ndb_autoincrement_prefetch_sz	Yes	Yes	Yes	Both	Yes
ndb_batch_size	Yes	Yes	Yes	Global	No
ndb_blob_read_batch_bytes	Yes	Yes	Yes	Both	Yes
ndb_blob_write_batch_bytes	Yes	Yes	Yes	Both	Yes
ndb_cache_check_time	Yes	Yes	Yes	Global	Yes
ndb_clear_apply_status	Yes		Yes	Global	Yes
ndb_cluster_connection_pool	Yes	Yes	Yes	Global	No
Ndb_conflict_last_conflict_epoch			Yes	Global	No
ndb_deferred_constraints	Yes	Yes	Yes	Both	Yes
ndb_deferred_constraints	Yes	Yes	Yes	Both	Yes
ndb_distribution	Yes	Yes	Yes	Global	Yes
ndb_distribution	Yes	Yes	Yes	Global	Yes
ndb_eventbuffer_free_percent	Yes	Yes	Yes	Global	Yes
ndb_eventbuffer_max_alloc	Yes	Yes	Yes	Global	Yes
ndb_extra_logging	Yes	Yes	Yes	Global	Yes
ndb_force_send	Yes	Yes	Yes	Both	Yes
ndb_index_stat_cache_entries	Yes	Yes	Yes	Both	Yes
ndb_index_stat_enable	Yes	Yes	Yes	Both	Yes
ndb_index_stat_option	Yes	Yes	Yes	Both	Yes
ndb_index_stat_update_freq	Yes	Yes	Yes	Both	Yes
ndb_join_pushdown			Yes	Both	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
ndb_log_apply_status	Yes	Yes	Yes	Global	No
ndb_log_apply_status	Yes	Yes	Yes	Global	No
ndb_log_bin	Yes		Yes	Both	Yes
ndb_log_binlog_index	Yes		Yes	Global	Yes
ndb_log_empty_epochs	Yes	Yes	Yes	Global	Yes
ndb_log_empty_epochs	Yes	Yes	Yes	Global	Yes
ndb_log_empty_update	Yes	Yes	Yes	Global	Yes
ndb_log_empty_update	Yes	Yes	Yes	Global	Yes
ndb_log_exclusive_reads	Yes	Yes	Yes	Both	Yes
ndb_log_exclusive_reads	Yes	Yes	Yes	Both	Yes
ndb_log_fail_terminate	Yes	Yes	Yes	Global	No
ndb_log_orig	Yes	Yes	Yes	Global	No
ndb_log_orig	Yes	Yes	Yes	Global	No
ndb_log_transaction_id	Yes	Yes	Yes	Global	No
ndb_log_transaction_id			Yes	Global	No
ndb_log_update_as_write	Yes	Yes	Yes	Global	Yes
ndb_log_update_minimal	Yes	Yes	Yes	Global	Yes
ndb_log_updated_only	Yes	Yes	Yes	Global	Yes
ndb_optimization_delay	Yes	Yes	Yes	Global	Yes
ndb_optimized_node_selection	Yes	Yes	Yes	Global	No
ndb_recv_thread_activation_threshold	Yes	Yes	Yes	Global	Yes
ndb_recv_thread_cpu_mask	Yes	Yes	Yes	Global	Yes
ndb_report_thresh_binlog_epoch_slip	Yes	Yes	Yes	Global	Yes
ndb_report_thresh_binlog_mem_usage	Yes	Yes	Yes	Global	Yes
ndb_show_foreign_key_mock_tables	Yes	Yes	Yes	Global	Yes
ndb_slave_conflict_role	Yes	Yes	Yes	Global	Yes
Ndb_slave_max_replicated_epoch			Yes	Global	No
ndb_table_no_logging			Yes	Session	Yes
ndb_table_temporary			Yes	Session	Yes
ndb_use_copying_alter_table			Yes	Both	No
ndb_use_exact_count			Yes	Both	Yes
ndb_use_transactions	Yes	Yes	Yes	Both	Yes
ndb_version			Yes	Global	No
ndb_version_string			Yes	Global	No
ndb_wait_connected	Yes	Yes	Yes	Global	No
ndb_wait_setup	Yes	Yes	Yes	Global	No
ndbinfo_database			Yes	Global	No
ndbinfo_max_bytes	Yes		Yes	Both	Yes
ndbinfo_max_rows	Yes		Yes	Both	Yes
ndbinfo_offline			Yes	Global	Yes
ndbinfo_show_hidden	Yes		Yes	Both	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
ndbinfo_table_prefix			Yes	Global	No
ndbinfo_version			Yes	Global	No
net_buffer_length	Yes	Yes	Yes	Both	Yes
net_read_timeout	Yes	Yes	Yes	Both	Yes
net_retry_count	Yes	Yes	Yes	Both	Yes
net_write_timeout	Yes	Yes	Yes	Both	Yes
new	Yes	Yes	Yes	Both	Yes
old	Yes	Yes	Yes	Global	No
old_alter_table	Yes	Yes	Yes	Both	Yes
old_passwords	Yes	Yes	Yes	Both	Yes
open_files_limit	Yes	Yes	Yes	Global	No
optimizer_prune_level	Yes	Yes	Yes	Both	Yes
optimizer_search_depth	Yes	Yes	Yes	Both	Yes
optimizer_switch	Yes	Yes	Yes	Both	Yes
optimizer_trace	Yes	Yes	Yes	Both	Yes
optimizer_trace_features	Yes	Yes	Yes	Both	Yes
optimizer_trace_limit	Yes	Yes	Yes	Both	Yes
optimizer_trace_max_mem_size	Yes	Yes	Yes	Both	Yes
optimizer_trace_offset	Yes	Yes	Yes	Both	Yes
performance_schema	Yes	Yes	Yes	Global	No
performance_schema_accounts_size	Yes	Yes	Yes	Global	No
performance_schema_digests_size	Yes	Yes	Yes	Global	No
performance_schema_events_stages_history_long_size	Yes	Yes	Yes	Global	No
performance_schema_events_stages_history_size	Yes	Yes	Yes	Global	No
performance_schema_events_statements_history_long_size	Yes	Yes	Yes	Global	No
performance_schema_events_statements_history_size	Yes	Yes	Yes	Global	No
performance_schema_events_waits_history_long_size	Yes	Yes	Yes	Global	No
performance_schema_events_waits_history_size	Yes	Yes	Yes	Global	No
performance_schema_hosts_size	Yes	Yes	Yes	Global	No
performance_schema_max_cond_classes	Yes	Yes	Yes	Global	No
performance_schema_max_cond_instances	Yes	Yes	Yes	Global	No
performance_schema_max_digest_length	Yes	Yes	Yes	Global	No
performance_schema_max_file_classes	Yes	Yes	Yes	Global	No
performance_schema_max_file_handles	Yes	Yes	Yes	Global	No
performance_schema_max_file_instances	Yes	Yes	Yes	Global	No
performance_schema_max_mutex_classes	Yes	Yes	Yes	Global	No
performance_schema_max_mutex_instances	Yes	Yes	Yes	Global	No
performance_schema_max_rwlock_classes	Yes	Yes	Yes	Global	No
performance_schema_max_rwlock_instances	Yes	Yes	Yes	Global	No
performance_schema_max_socket_classes	Yes	Yes	Yes	Global	No
performance_schema_max_socket_instances	Yes	Yes	Yes	Global	No

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
performance_schema_max_stage_classes	Yes	Yes	Yes	Global	No
performance_schema_max_statement_classes	Yes	Yes	Yes	Global	No
performance_schema_max_table_handles	Yes	Yes	Yes	Global	No
performance_schema_max_table_instances	Yes	Yes	Yes	Global	No
performance_schema_max_thread_classes	Yes	Yes	Yes	Global	No
performance_schema_max_thread_instances	Yes	Yes	Yes	Global	No
performance_schema_session_connect_attrs_size	Yes	Yes	Yes	Global	No
performance_schema_setup_actors_size	Yes	Yes	Yes	Global	No
performance_schema_setup_objects_size	Yes	Yes	Yes	Global	No
performance_schema_users_size	Yes	Yes	Yes	Global	No
pid_file	Yes	Yes	Yes	Global	No
plugin_dir	Yes	Yes	Yes	Global	No
plugin_load	Yes	Yes	Yes	Global	No
plugin_load_add	Yes	Yes	Yes	Global	No
port	Yes	Yes	Yes	Global	No
preload_buffer_size	Yes	Yes	Yes	Both	Yes
profiling			Yes	Both	Yes
profiling_history_size	Yes	Yes	Yes	Both	Yes
protocol_version			Yes	Global	No
proxy_user			Yes	Session	No
pseudo_slave_mode			Yes	Session	Yes
pseudo_thread_id			Yes	Session	Yes
query_alloc_block_size	Yes	Yes	Yes	Both	Yes
query_cache_limit	Yes	Yes	Yes	Global	Yes
query_cache_min_res_unit	Yes	Yes	Yes	Global	Yes
query_cache_size	Yes	Yes	Yes	Global	Yes
query_cache_type	Yes	Yes	Yes	Both	Yes
query_cache_wlock_invalidate	Yes	Yes	Yes	Both	Yes
query_prealloc_size	Yes	Yes	Yes	Both	Yes
rand_seed1			Yes	Session	Yes
rand_seed2			Yes	Session	Yes
range_alloc_block_size	Yes	Yes	Yes	Both	Yes
read_buffer_size	Yes	Yes	Yes	Both	Yes
read_only	Yes	Yes	Yes	Global	Yes
read_rnd_buffer_size	Yes	Yes	Yes	Both	Yes
relay_log	Yes	Yes	Yes	Global	No
relay_log_basename			Yes	Global	No
relay_log_index	Yes	Yes	Yes	Global	No
relay_log_info_file	Yes	Yes	Yes	Global	No
relay_log_info_repository	Yes	Yes	Yes	Global	Yes
relay_log_purge	Yes	Yes	Yes	Global	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
relay_log_recovery	Yes	Yes	Yes	Global	Varies
relay_log_space_limit	Yes	Yes	Yes	Global	No
report_host	Yes	Yes	Yes	Global	No
report_password	Yes	Yes	Yes	Global	No
report_port	Yes	Yes	Yes	Global	No
report_user	Yes	Yes	Yes	Global	No
rpl_semi_sync_master_enabled	Yes	Yes	Yes	Global	Yes
rpl_semi_sync_master_timeout	Yes	Yes	Yes	Global	Yes
rpl_semi_sync_master_trace_level	Yes	Yes	Yes	Global	Yes
rpl_semi_sync_master_wait_no_slave	Yes	Yes	Yes	Global	Yes
rpl_semi_sync_slave_enabled	Yes	Yes	Yes	Global	Yes
rpl_semi_sync_slave_trace_level	Yes	Yes	Yes	Global	Yes
rpl_stop_slave_timeout	Yes	Yes	Yes	Global	Yes
secure_auth	Yes	Yes	Yes	Global	Yes
secure_file_priv	Yes	Yes	Yes	Global	No
server_id	Yes	Yes	Yes	Global	Yes
server_id_bits	Yes	Yes	Yes	Global	No
server_uuid			Yes	Global	No
sha256_password_private_key_path	Yes	Yes	Yes	Global	No
sha256_password_public_key_path	Yes	Yes	Yes	Global	No
shared_memory	Yes	Yes	Yes	Global	No
shared_memory_base_name	Yes	Yes	Yes	Global	No
show_old_temporals	Yes	Yes	Yes	Both	Yes
simplified_binlog_gtid_recovery	Yes	Yes	Yes	Global	No
skip_external_locking	Yes	Yes	Yes	Global	No
skip_name_resolve	Yes	Yes	Yes	Global	No
skip_networking	Yes	Yes	Yes	Global	No
skip_show_database	Yes	Yes	Yes	Global	No
slave_allow_batching	Yes	Yes	Yes	Global	Yes
slave_checkpoint_group	Yes	Yes	Yes	Global	Yes
slave_checkpoint_period	Yes	Yes	Yes	Global	Yes
slave_compressed_protocol	Yes	Yes	Yes	Global	Yes
slave_exec_mode	Yes	Yes	Yes	Global	Yes
slave_load_tmpdir	Yes	Yes	Yes	Global	No
slave_max_allowed_packet	Yes	Yes	Yes	Global	Yes
slave_net_timeout	Yes	Yes	Yes	Global	Yes
slave_parallel_workers	Yes	Yes	Yes	Global	Yes
slave_pending_jobs_size_max	Yes	Yes	Yes	Global	Yes
slave_rows_search_algorithms	Yes	Yes	Yes	Global	Yes
slave_skip_errors	Yes	Yes	Yes	Global	No
slave_sql_verify_checksum	Yes	Yes	Yes	Global	Yes

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
slave_transaction_retries	Yes	Yes	Yes	Global	Yes
slave_type_conversions	Yes	Yes	Yes	Global	No
slow_launch_time	Yes	Yes	Yes	Global	Yes
slow_query_log	Yes	Yes	Yes	Global	Yes
slow_query_log_file	Yes	Yes	Yes	Global	Yes
socket	Yes	Yes	Yes	Global	No
sort_buffer_size	Yes	Yes	Yes	Both	Yes
sql_auto_is_null			Yes	Both	Yes
sql_big_selects			Yes	Both	Yes
sql_buffer_result			Yes	Both	Yes
sql_log_bin			Yes	Session	Yes
sql_log_off			Yes	Both	Yes
sql_mode	Yes	Yes	Yes	Both	Yes
sql_notes			Yes	Both	Yes
sql_quote_show_create			Yes	Both	Yes
sql_safe_updates			Yes	Both	Yes
sql_select_limit			Yes	Both	Yes
sql_slave_skip_counter			Yes	Global	Yes
sql_warnings			Yes	Both	Yes
ssl_ca	Yes	Yes	Yes	Global	No
ssl_capath	Yes	Yes	Yes	Global	No
ssl_cert	Yes	Yes	Yes	Global	No
ssl_cipher	Yes	Yes	Yes	Global	No
ssl_crl	Yes	Yes	Yes	Global	No
ssl_crlpath	Yes	Yes	Yes	Global	No
ssl_key	Yes	Yes	Yes	Global	No
storage_engine			Yes	Both	Yes
stored_program_cache	Yes	Yes	Yes	Global	Yes
sync_binlog	Yes	Yes	Yes	Global	Yes
sync_frm	Yes	Yes	Yes	Global	Yes
sync_master_info	Yes	Yes	Yes	Global	Yes
sync_relay_log	Yes	Yes	Yes	Global	Yes
sync_relay_log_info	Yes	Yes	Yes	Global	Yes
system_time_zone			Yes	Global	No
table_definition_cache	Yes	Yes	Yes	Global	Yes
table_open_cache	Yes	Yes	Yes	Global	Yes
table_open_cache_instances	Yes	Yes	Yes	Global	No
thread_cache_size	Yes	Yes	Yes	Global	Yes
thread_concurrency	Yes	Yes	Yes	Global	No
thread_handling	Yes	Yes	Yes	Global	No
thread_pool_algorithm	Yes	Yes	Yes	Global	No

Name	Cmd-Line	Option File	System Var	Var Scope	Dynamic
thread_pool_high_priority_connection	Yes	Yes	Yes	Both	Yes
thread_pool_max_unused_threads	Yes	Yes	Yes	Global	Yes
thread_pool_prio_kickup_timer	Yes	Yes	Yes	Both	Yes
thread_pool_size	Yes	Yes	Yes	Global	No
thread_pool_stall_limit	Yes	Yes	Yes	Global	Yes
thread_stack	Yes	Yes	Yes	Global	No
time_format			Yes	Global	No
time_zone			Yes	Both	Yes
timed_mutexes	Yes	Yes	Yes	Global	Yes
timestamp			Yes	Session	Yes
tmp_table_size	Yes	Yes	Yes	Both	Yes
tmpdir	Yes	Yes	Yes	Global	No
transaction_alloc_block_size	Yes	Yes	Yes	Both	Yes
transaction_allow_batching			Yes	Session	Yes
transaction_prealloc_size	Yes	Yes	Yes	Both	Yes
tx_isolation			Yes	Both	Yes
tx_read_only			Yes	Both	Yes
unique_checks			Yes	Both	Yes
updatable_views_with_limit	Yes	Yes	Yes	Both	Yes
validate_password_dictionary_file	Yes	Yes	Yes	Global	Varies
validate_password_length	Yes	Yes	Yes	Global	Yes
validate_password_mixed_case_count	Yes	Yes	Yes	Global	Yes
validate_password_number_count	Yes	Yes	Yes	Global	Yes
validate_password_policy	Yes	Yes	Yes	Global	Yes
validate_password_special_char_count	Yes	Yes	Yes	Global	Yes
validate_user_plugins	Yes	Yes	Yes	Global	No
version			Yes	Global	No
version_comment			Yes	Global	No
version_compile_machine			Yes	Global	No
version_compile_os			Yes	Global	No
wait_timeout	Yes	Yes	Yes	Both	Yes
warning_count			Yes	Session	No

Notes:

1. This option is dynamic, but should be set only by server. You should not set this variable manually.

5.1.5 Server Status Variable Reference

The following table lists all status variables applicable within `mysqld`.

The table lists each variable's data type and scope. The last column indicates whether the scope for each variable is Global, Session, or both. Please see the corresponding item descriptions for details on setting and using the variables. Where appropriate, direct links to further information about the items are provided.

Table 5.4 Status Variable Summary

Variable Name	Variable Type	Variable Scope
Aborted_clients	Integer	Global
Aborted_connects	Integer	Global
Audit_log_current_size	Integer	Global
Audit_log_event_max_drop_size	Integer	Global
Audit_log_events	Integer	Global
Audit_log_events_filtered	Integer	Global
Audit_log_events_lost	Integer	Global
Audit_log_events_written	Integer	Global
Audit_log_total_size	Integer	Global
Audit_log_write_waits	Integer	Global
Binlog_cache_disk_use	Integer	Global
Binlog_cache_use	Integer	Global
Binlog_stmt_cache_disk_use	Integer	Global
Binlog_stmt_cache_use	Integer	Global
Bytes_received	Integer	Both
Bytes_sent	Integer	Both
Com_admin_commands	Integer	Both
Com_alter_db	Integer	Both
Com_alter_db_upgrade	Integer	Both
Com_alter_event	Integer	Both
Com_alter_function	Integer	Both
Com_alter_procedure	Integer	Both
Com_alter_server	Integer	Both
Com_alter_table	Integer	Both
Com_alter_tablespace	Integer	Both
Com_alter_user	Integer	Both
Com_analyze	Integer	Both
Com_assign_to_keycache	Integer	Both
Com_begin	Integer	Both
Com_binlog	Integer	Both
Com_call_procedure	Integer	Both
Com_change_db	Integer	Both
Com_change_master	Integer	Both
Com_check	Integer	Both
Com_checksum	Integer	Both
Com_commit	Integer	Both
Com_create_db	Integer	Both
Com_create_event	Integer	Both
Com_create_function	Integer	Both
Com_create_index	Integer	Both

Variable Name	Variable Type	Variable Scope
Com_create_procedure	Integer	Both
Com_create_server	Integer	Both
Com_create_table	Integer	Both
Com_create_trigger	Integer	Both
Com_create_udf	Integer	Both
Com_create_user	Integer	Both
Com_create_view	Integer	Both
Com_dealloc_sql	Integer	Both
Com_delete	Integer	Both
Com_delete_multi	Integer	Both
Com_do	Integer	Both
Com_drop_db	Integer	Both
Com_drop_event	Integer	Both
Com_drop_function	Integer	Both
Com_drop_index	Integer	Both
Com_drop_procedure	Integer	Both
Com_drop_server	Integer	Both
Com_drop_table	Integer	Both
Com_drop_trigger	Integer	Both
Com_drop_user	Integer	Both
Com_drop_view	Integer	Both
Com_empty_query	Integer	Both
Com_execute_sql	Integer	Both
Com_flush	Integer	Both
Com_get_diagnostics	Integer	Both
Com_grant	Integer	Both
Com_ha_close	Integer	Both
Com_ha_open	Integer	Both
Com_ha_read	Integer	Both
Com_help	Integer	Both
Com_insert	Integer	Both
Com_insert_select	Integer	Both
Com_install_plugin	Integer	Both
Com_kill	Integer	Both
Com_load	Integer	Both
Com_lock_tables	Integer	Both
Com_optimize	Integer	Both
Com_preload_keys	Integer	Both
Com_prepare_sql	Integer	Both
Com_purge	Integer	Both
Com_purge_before_date	Integer	Both

Variable Name	Variable Type	Variable Scope
Com_release_savepoint	Integer	Both
Com_rename_table	Integer	Both
Com_rename_user	Integer	Both
Com_repair	Integer	Both
Com_replace	Integer	Both
Com_replace_select	Integer	Both
Com_reset	Integer	Both
Com_resignal	Integer	Both
Com_revoke	Integer	Both
Com_revoke_all	Integer	Both
Com_rollback	Integer	Both
Com_rollback_to_savepoint	Integer	Both
Com_savepoint	Integer	Both
Com_select	Integer	Both
Com_set_option	Integer	Both
Com_show_authors	Integer	Both
Com_show_binlog_events	Integer	Both
Com_show_binlogs	Integer	Both
Com_show_charsets	Integer	Both
Com_show_collations	Integer	Both
Com_show_contributors	Integer	Both
Com_show_create_db	Integer	Both
Com_show_create_event	Integer	Both
Com_show_create_func	Integer	Both
Com_show_create_proc	Integer	Both
Com_show_create_table	Integer	Both
Com_show_create_trigger	Integer	Both
Com_show_databases	Integer	Both
Com_show_engine_logs	Integer	Both
Com_show_engine_mutex	Integer	Both
Com_show_engine_status	Integer	Both
Com_show_errors	Integer	Both
Com_show_events	Integer	Both
Com_show_fields	Integer	Both
Com_show_function_code	Integer	Both
Com_show_function_status	Integer	Both
Com_show_grants	Integer	Both
Com_show_keys	Integer	Both
Com_show_master_status	Integer	Both
Com_show_ndb_status	Integer	Both
Com_show_open_tables	Integer	Both

Variable Name	Variable Type	Variable Scope
Com_show_plugins	Integer	Both
Com_show_privileges	Integer	Both
Com_show_procedure_code	Integer	Both
Com_show_procedure_status	Integer	Both
Com_show_processlist	Integer	Both
Com_show_profile	Integer	Both
Com_show_profiles	Integer	Both
Com_show_relaylog_events	Integer	Both
Com_show_slave_hosts	Integer	Both
Com_show_slave_status	Integer	Both
Com_show_status	Integer	Both
Com_show_storage_engines	Integer	Both
Com_show_table_status	Integer	Both
Com_show_tables	Integer	Both
Com_show_triggers	Integer	Both
Com_show_variables	Integer	Both
Com_show_warnings	Integer	Both
Com_signal	Integer	Both
Com_slave_start	Integer	Both
Com_slave_stop	Integer	Both
Com_stmt_close	Integer	Both
Com_stmt_execute	Integer	Both
Com_stmt_fetch	Integer	Both
Com_stmt_prepare	Integer	Both
Com_stmt_reprepare	Integer	Both
Com_stmt_reset	Integer	Both
Com_stmt_send_long_data	Integer	Both
Com_truncate	Integer	Both
Com_uninstall_plugin	Integer	Both
Com_unlock_tables	Integer	Both
Com_update	Integer	Both
Com_update_multi	Integer	Both
Com_xa_commit	Integer	Both
Com_xa_end	Integer	Both
Com_xa_prepare	Integer	Both
Com_xa_recover	Integer	Both
Com_xa_rollback	Integer	Both
Com_xa_start	Integer	Both
Compression	Integer	Session
Connection_control_delay_generated	Integer	Global
Connection_errors_accept	Integer	Global

Variable Name	Variable Type	Variable Scope
Connection_errors_internal	Integer	Global
Connection_errors_max_connections	Integer	Global
Connection_errors_peer_address	Integer	Global
Connection_errors_select	Integer	Global
Connection_errors_tcpwrap	Integer	Global
Connections	Integer	Global
Created_tmp_disk_tables	Integer	Both
Created_tmp_files	Integer	Global
Created_tmp_tables	Integer	Both
Delayed_errors	Integer	Global
Delayed_insert_threads	Integer	Global
Delayed_writes	Integer	Global
Firewall_access_denied	Integer	Global
Firewall_access_granted	Integer	Global
Firewall_cached_entries	Integer	Global
Flush_commands	Integer	Global
Handler_commit	Integer	Both
Handler_delete	Integer	Both
Handler_discover	Integer	Both
Handler_external_lock	Integer	Both
Handler_mrr_init	Integer	Both
Handler_prepare	Integer	Both
Handler_read_first	Integer	Both
Handler_read_key	Integer	Both
Handler_read_last	Integer	Both
Handler_read_next	Integer	Both
Handler_read_prev	Integer	Both
Handler_read_rnd	Integer	Both
Handler_read_rnd_next	Integer	Both
Handler_rollback	Integer	Both
Handler_savepoint	Integer	Both
Handler_savepoint_rollback	Integer	Both
Handler_update	Integer	Both
Handler_write	Integer	Both
Innodb_available_undo_logs	Integer	Global
Innodb_buffer_pool_bytes_data	Integer	Global
Innodb_buffer_pool_bytes_dirty	Integer	Global
Innodb_buffer_pool_dump_status	String	Global
Innodb_buffer_pool_load_status	String	Global
Innodb_buffer_pool_pages_data	Integer	Global
Innodb_buffer_pool_pages_dirty	Integer	Global

Variable Name	Variable Type	Variable Scope
Innodb_buffer_pool_pages_flushed	Integer	Global
Innodb_buffer_pool_pages_free	Integer	Global
Innodb_buffer_pool_pages_latched	Integer	Global
Innodb_buffer_pool_pages_misc	Integer	Global
Innodb_buffer_pool_pages_total	Integer	Global
Innodb_buffer_pool_read_ahead	Integer	Global
Innodb_buffer_pool_read_ahead_evicted	Integer	Global
Innodb_buffer_pool_read_ahead_rnd	Integer	Global
Innodb_buffer_pool_read_requests	Integer	Global
Innodb_buffer_pool_reads	Integer	Global
Innodb_buffer_pool_wait_free	Integer	Global
Innodb_buffer_pool_write_requests	Integer	Global
Innodb_data_fsyncs	Integer	Global
Innodb_data_pending_fsyncs	Integer	Global
Innodb_data_pending_reads	Integer	Global
Innodb_data_pending_writes	Integer	Global
Innodb_data_read	Integer	Global
Innodb_data_reads	Integer	Global
Innodb_data_writes	Integer	Global
Innodb_data_written	Integer	Global
Innodb_dblwr_pages_written	Integer	Global
Innodb_dblwr_writes	Integer	Global
Innodb_have_atomic_builtins	Integer	Global
Innodb_log_waits	Integer	Global
Innodb_log_write_requests	Integer	Global
Innodb_log_writes	Integer	Global
Innodb_num_open_files	Integer	Global
Innodb_os_log_fsyncs	Integer	Global
Innodb_os_log_pending_fsyncs	Integer	Global
Innodb_os_log_pending_writes	Integer	Global
Innodb_os_log_written	Integer	Global
Innodb_page_size	Integer	Global
Innodb_pages_created	Integer	Global
Innodb_pages_read	Integer	Global
Innodb_pages_written	Integer	Global
Innodb_row_lock_current_waits	Integer	Global
Innodb_row_lock_time	Integer	Global
Innodb_row_lock_time_avg	Integer	Global
Innodb_row_lock_time_max	Integer	Global
Innodb_row_lock_waits	Integer	Global
Innodb_rows_deleted	Integer	Global

Variable Name	Variable Type	Variable Scope
Innodb_rows_inserted	Integer	Global
Innodb_rows_read	Integer	Global
Innodb_rows_updated	Integer	Global
Innodb_truncated_status_writes	Integer	Global
Key_blocks_not_flushed	Integer	Global
Key_blocks_unused	Integer	Global
Key_blocks_used	Integer	Global
Key_read_requests	Integer	Global
Key_reads	Integer	Global
Key_write_requests	Integer	Global
Key_writes	Integer	Global
Last_query_cost	Numeric	Session
Last_query_partial_plans	Integer	Session
Max_used_connections	Integer	Global
Ndb_api_adaptive_send_deferred_count	Integer	Global
Ndb_api_adaptive_send_deferred_count_session	Integer	Global
Ndb_api_adaptive_send_deferred_count_slave	Integer	Global
Ndb_api_adaptive_send_forced_count	Integer	Global
Ndb_api_adaptive_send_forced_count_session	Integer	Global
Ndb_api_adaptive_send_forced_count_slave	Integer	Global
Ndb_api_adaptive_send_unforced_count	Integer	Global
Ndb_api_adaptive_send_unforced_count_session	Integer	Global
Ndb_api_adaptive_send_unforced_count_slave	Integer	Global
Ndb_api_bytes_received_count	Integer	Global
Ndb_api_bytes_received_count_session	Integer	Session
Ndb_api_bytes_received_count_slave	Integer	Global
Ndb_api_bytes_sent_count	Integer	Global
Ndb_api_bytes_sent_count_session	Integer	Session
Ndb_api_bytes_sent_count_slave	Integer	Global
Ndb_api_event_bytes_count	Integer	Global
Ndb_api_event_bytes_count_injector	Integer	Global
Ndb_api_event_data_count	Integer	Global
Ndb_api_event_data_count_injector	Integer	Global
Ndb_api_event_nondata_count	Integer	Global
Ndb_api_event_nondata_count_injector	Integer	Global
Ndb_api_pk_op_count	Integer	Global
Ndb_api_pk_op_count_session	Integer	Session
Ndb_api_pk_op_count_slave	Integer	Global
Ndb_api_pruned_scan_count	Integer	Global
Ndb_api_pruned_scan_count_session	Integer	Session
Ndb_api_pruned_scan_count_slave	Integer	Global

Variable Name	Variable Type	Variable Scope
Ndb_api_range_scan_count	Integer	Global
Ndb_api_range_scan_count_session	Integer	Session
Ndb_api_range_scan_count_slave	Integer	Global
Ndb_api_read_row_count	Integer	Global
Ndb_api_read_row_count_session	Integer	Session
Ndb_api_read_row_count_slave	Integer	Global
Ndb_api_scan_batch_count	Integer	Global
Ndb_api_scan_batch_count_session	Integer	Session
Ndb_api_scan_batch_count_slave	Integer	Global
Ndb_api_table_scan_count	Integer	Global
Ndb_api_table_scan_count_session	Integer	Session
Ndb_api_table_scan_count_slave	Integer	Global
Ndb_api_trans_abort_count	Integer	Global
Ndb_api_trans_abort_count_session	Integer	Session
Ndb_api_trans_abort_count_slave	Integer	Global
Ndb_api_trans_close_count	Integer	Global
Ndb_api_trans_close_count_session	Integer	Session
Ndb_api_trans_close_count_slave	Integer	Global
Ndb_api_trans_commit_count	Integer	Global
Ndb_api_trans_commit_count_session	Integer	Session
Ndb_api_trans_commit_count_slave	Integer	Global
Ndb_api_trans_local_read_row_count	Integer	Global
Ndb_api_trans_local_read_row_count_session	Integer	Session
Ndb_api_trans_local_read_row_count_slave	Integer	Global
Ndb_api_trans_start_count	Integer	Global
Ndb_api_trans_start_count_session	Integer	Session
Ndb_api_trans_start_count_slave	Integer	Global
Ndb_api_uk_op_count	Integer	Global
Ndb_api_uk_op_count_session	Integer	Session
Ndb_api_uk_op_count_slave	Integer	Global
Ndb_api_wait_exec_complete_count	Integer	Global
Ndb_api_wait_exec_complete_count_session	Integer	Session
Ndb_api_wait_exec_complete_count_slave	Integer	Global
Ndb_api_wait_meta_request_count	Integer	Global
Ndb_api_wait_meta_request_count_session	Integer	Session
Ndb_api_wait_meta_request_count_slave	Integer	Global
Ndb_api_wait_nanos_count	Integer	Global
Ndb_api_wait_nanos_count_session	Integer	Session
Ndb_api_wait_nanos_count_slave	Integer	Global
Ndb_api_wait_scan_result_count	Integer	Global
Ndb_api_wait_scan_result_count_session	Integer	Session

Variable Name	Variable Type	Variable Scope
Ndb_api_wait_scan_result_count_slave	Integer	Global
Ndb_cluster_node_id	Integer	Global
Ndb_config_from_host	Integer	Both
Ndb_config_from_port	Integer	Both
Ndb_conflict_fn_epoch	Integer	Global
Ndb_conflict_fn_epoch_trans	Integer	Global
Ndb_conflict_fn_epoch2	Integer	Global
Ndb_conflict_fn_epoch2_trans	Integer	Global
Ndb_conflict_fn_max	Integer	Global
Ndb_conflict_fn_max_del_win	Integer	Global
Ndb_conflict_fn_old	Integer	Global
Ndb_conflict_last_stable_epoch	Integer	Global
Ndb_conflict_reflected_op_discard_count	Integer	Global
Ndb_conflict_reflected_op_prepare_count	Integer	Global
Ndb_conflict_refresh_op_count	Integer	Global
Ndb_conflict_trans_conflict_commit_count	Integer	Global
Ndb_conflict_trans_detect_iter_count	Integer	Global
Ndb_conflict_trans_reject_count	Integer	Global
Ndb_conflict_trans_row_conflict_count	Integer	Global
Ndb_conflict_trans_row_reject_count	Integer	Global
Ndb_epoch_delete_delete_count	Integer	Global
Ndb_execute_count	Integer	Global
Ndb_last_commit_epoch_server	Integer	Global
Ndb_last_commit_epoch_session	Integer	Session
Ndb_cluster_node_id	Integer	Global
Ndb_number_of_data_nodes	Integer	Global
Ndb_pruned_scan_count	Integer	Global
Ndb_pushed_queries_defined	Integer	Global
Ndb_pushed_queries_dropped	Integer	Global
Ndb_pushed_queries_executed	Integer	Global
Ndb_pushed_reads	Integer	Global
Ndb_scan_count	Integer	Global
Not_flushed_delayed_rows	Integer	Global
Open_files	Integer	Global
Open_streams	Integer	Global
Open_table_definitions	Integer	Global
Open_tables	Integer	Both
Opened_files	Integer	Global
Opened_table_definitions	Integer	Both
Opened_tables	Integer	Both
Performance_schema_accounts_lost	Integer	Global

Variable Name	Variable Type	Variable Scope
Performance_schema_cond_classes_lost	Integer	Global
Performance_schema_cond_instances_lost	Integer	Global
Performance_schema_digest_lost	Integer	Global
Performance_schema_file_classes_lost	Integer	Global
Performance_schema_file_handles_lost	Integer	Global
Performance_schema_file_instances_lost	Integer	Global
Performance_schema_hosts_lost	Integer	Global
Performance_schema_locker_lost	Integer	Global
Performance_schema_mutex_classes_lost	Integer	Global
Performance_schema_mutex_instances_lost	Integer	Global
Performance_schema_rwlock_classes_lost	Integer	Global
Performance_schema_rwlock_instances_lost	Integer	Global
Performance_schema_session_connect_attrs_lost	Integer	Global
Performance_schema_socket_classes_lost	Integer	Global
Performance_schema_socket_instances_lost	Integer	Global
Performance_schema_stage_classes_lost	Integer	Global
Performance_schema_statement_classes_lost	Integer	Global
Performance_schema_table_handles_lost	Integer	Global
Performance_schema_table_instances_lost	Integer	Global
Performance_schema_thread_classes_lost	Integer	Global
Performance_schema_thread_instances_lost	Integer	Global
Performance_schema_users_lost	Integer	Global
Prepared_stmt_count	Integer	Global
Qcache_free_blocks	Integer	Global
Qcache_free_memory	Integer	Global
Qcache_hits	Integer	Global
Qcache_inserts	Integer	Global
Qcache_lowmem_prunes	Integer	Global
Qcache_not_cached	Integer	Global
Qcache_queries_in_cache	Integer	Global
Qcache_total_blocks	Integer	Global
Queries	Integer	Both
Questions	Integer	Both
Rpl_semi_sync_master_clients	Integer	Global
Rpl_semi_sync_master_net_avg_wait_time	Integer	Global
Rpl_semi_sync_master_net_wait_time	Integer	Global
Rpl_semi_sync_master_net_waits	Integer	Global
Rpl_semi_sync_master_no_times	Integer	Global
Rpl_semi_sync_master_no_tx	Integer	Global
Rpl_semi_sync_master_status	Boolean	Global
Rpl_semi_sync_master_timefunc_failures	Integer	Global

Variable Name	Variable Type	Variable Scope
Rpl_semi_sync_master_tx_avg_wait_time	Integer	Global
Rpl_semi_sync_master_tx_wait_time	Integer	Global
Rpl_semi_sync_master_tx_waits	Integer	Global
Rpl_semi_sync_master_wait_pos_backtraverse	Integer	Global
Rpl_semi_sync_master_wait_sessions	Integer	Global
Rpl_semi_sync_master_yes_tx	Integer	Global
Rpl_semi_sync_slave_status	Boolean	Global
Rsa_public_key	String	Global
Select_full_join	Integer	Both
Select_full_range_join	Integer	Both
Select_range	Integer	Both
Select_range_check	Integer	Both
Select_scan	Integer	Both
Slave_heartbeat_period	Numeric	Global
Slave_last_heartbeat	Datetime	Global
Slave_open_temp_tables	Integer	Global
Slave_received_heartbeats	Integer	Global
Slave_retried_transactions	Integer	Global
Slave_rows_last_search_algorithm_used	String	Global
Slave_running	Boolean	Global
Slow_launch_threads	Integer	Both
Slow_queries	Integer	Both
Sort_merge_passes	Integer	Both
Sort_range	Integer	Both
Sort_rows	Integer	Both
Sort_scan	Integer	Both
Ssl_accept_renegotiates	Integer	Global
Ssl_accepts	Integer	Global
Ssl_callback_cache_hits	Integer	Global
Ssl_cipher	String	Both
Ssl_cipher_list	String	Both
Ssl_client_connects	Integer	Global
Ssl_connect_renegotiates	Integer	Global
Ssl_ctx_verify_depth	Integer	Global
Ssl_ctx_verify_mode	Integer	Global
Ssl_default_timeout	Integer	Both
Ssl_finished_accepts	Integer	Global
Ssl_finished_connects	Integer	Global
Ssl_server_not_after	Integer	Both
Ssl_server_not_before	Integer	Both
Ssl_session_cache_hits	Integer	Global

Variable Name	Variable Type	Variable Scope
Ssl_session_cache_misses	Integer	Global
Ssl_session_cache_mode	String	Global
Ssl_session_cache_overflows	Integer	Global
Ssl_session_cache_size	Integer	Global
Ssl_session_cache_timeouts	Integer	Global
Ssl_sessions_reused	Integer	Both
Ssl_used_session_cache_entries	Integer	Global
Ssl_verify_depth	Integer	Both
Ssl_verify_mode	Integer	Both
Ssl_version	String	Both
Table_locks_immediate	Integer	Global
Table_locks_waited	Integer	Global
Table_open_cache_hits	Integer	Both
Table_open_cache_misses	Integer	Both
Table_open_cache_overflows	Integer	Both
Tc_log_max_pages_used	Integer	Global
Tc_log_page_size	Integer	Global
Tc_log_page_waits	Integer	Global
Threads_cached	Integer	Global
Threads_connected	Integer	Global
Threads_created	Integer	Global
Threads_running	Integer	Global
Uptime	Integer	Global
Uptime_since_flush_status	Integer	Global
validate_password_dictionary_file_last_parsed	Datetime	Global
validate_password_dictionary_file_words_count	Integer	Global

5.1.6 Server Command Options

When you start the `mysqld` server, you can specify program options using any of the methods described in Section 4.2.2, “Specifying Program Options”. The most common methods are to provide options in an option file or on the command line. However, in most cases it is desirable to make sure that the server uses the same options each time it runs. The best way to ensure this is to list them in an option file. See Section 4.2.2.2, “Using Option Files”. That section also describes option file format and syntax.

`mysqld` reads options from the `[mysqld]` and `[server]` groups. `mysqld_safe` reads options from the `[mysqld]`, `[server]`, `[mysqld_safe]`, and `[safe_mysqld]` groups. `mysql.server` reads options from the `[mysqld]` and `[mysql.server]` groups.

An embedded MySQL server usually reads options from the `[server]`, `[embedded]`, and `[`*`xxxxx`*`_SERVER]` groups, where *`xxxxx`* is the name of the application into which the server is embedded.

`mysqld` accepts many command options. For a brief summary, execute this command:

```
mysqld --help
```

To see the full list, use this command:

```
mysqld --verbose --help
```

Some of the items in the list are actually system variables that can be set at server startup. These can be displayed at runtime using the `SHOW VARIABLES` statement. Some items displayed by the preceding `mysqld` command do not appear in `SHOW VARIABLES` output; this is because they are options only and not system variables.

The following list shows some of the most common server options. Additional options are described in other sections:

- Options that affect security: See Section 6.1.4, “Security-Related mysqld Options and Variables”.
- SSL-related options: See Command Options for Encrypted Connections.
- Binary log control options: See Section 5.4.4, “The Binary Log”.
- Replication-related options: See Section 17.1.4, “Replication and Binary Logging Options and Variables”.
- Options for loading plugins such as pluggable storage engines: See Section 5.5.1, “Installing and Uninstalling Plugins”.
- Options specific to particular storage engines: See Section 14.14, “InnoDB Startup Options and System Variables” and Section 15.2.1, “MyISAM Startup Options”.

Some options control the size of buffers or caches. For a given buffer, the server might need to allocate internal data structures. These structures typically are allocated from the total memory allocated to the buffer, and the amount of space required might be platform dependent. This means that when you assign a value to an option that controls a buffer size, the amount of space actually available might differ from the value assigned. In some cases, the amount might be less than the value assigned. It is also possible that the server adjusts a value upward. For example, if you assign a value of 0 to an option for which the minimal value is 1024, the server sets the value to 1024.

Values for buffer sizes, lengths, and stack sizes are given in bytes unless otherwise specified.

Some options take file name values. Unless otherwise specified, the default file location is the data directory if the value is a relative path name. To specify the location explicitly, use an absolute path name. Suppose that the data directory is `/var/mysql/data`. If a file-valued option is given as a relative path name, it is located under `/var/mysql/data`. If the value is an absolute path name, its location is as given by the path name.

You can also set the values of server system variables at server startup by using variable names as options. To assign a value to a server system variable, use an option of the form `--var_name=value`. For example, `--key_buffer_size=32M` sets the `key_buffer_size` variable to a value of 32MB.

When you assign a value to a variable, MySQL might automatically correct the value to stay within a given range, or adjust the value to the closest permissible value if only certain values are permitted.

To restrict the maximum value to which a system variable can be set at runtime with the `SET` statement, specify this maximum by using an option of the form `--maximum-var_name=value` at server startup.

You can change the values of most system variables at runtime with the `SET` statement. See Section 13.7.4.1, “SET Syntax for Variable Assignment”.

Section 5.1.7, “Server System Variables”, provides a full description for all variables, and additional information for setting them at server startup and runtime. For information on changing system variables, see Section 5.1.1, “Configuring the Server”.

- `--help`, `-?`

Command-Line Format	`--help`

Display a short help message and exit. Use both the `--verbose` and `--help` options to see the full message.

- `--allow-suspicious-udfs`

Command-Line Format	`--allow-suspicious-udfs[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

This option controls whether user-defined functions that have only an `xxx` symbol for the main function can be loaded. By default, the option is off and only UDFs that have at least one auxiliary symbol can be loaded; this prevents attempts at loading functions from shared object files other than those containing legitimate UDFs. See UDF Security Precautions.

- `--ansi`

Command-Line Format	`--ansi`

Use standard (ANSI) SQL syntax instead of MySQL syntax. For more precise control over the server SQL mode, use the `--sql-mode` option instead. See Section 1.7, “MySQL Standards Compliance”, and Section 5.1.10, “Server SQL Modes”.

- `--basedir=`*`dir_name`*, `-b` *`dir_name`*

Command-Line Format	`--basedir=dir_name`
System Variable	`basedir`
Scope	Global
Dynamic	No
Type	Directory name
Default Value	`configuration-dependent default`

The path to the MySQL installation directory. This option sets the `basedir` system variable.

- `--bootstrap`

Command-Line Format	`--bootstrap`

This option is used by the `mysql_install_db` program to create the MySQL privilege tables without having to start a full MySQL server.

Replication and global transaction identifiers are automatically disabled whenever this option is used (Bug #13992602). See Section 17.1.3, “Replication with Global Transaction Identifiers”.

When the server operates in bootstap mode, some functionality is unavailable that limits the statements permitted in any file named by the `init_file` system variable. For more information, see the description of that variable.

- `--character-set-client-handshake`

Command-Line Format	`--character-set-client-handshake[={OFF|ON}]`
Type	Boolean
Default Value	`ON`

Do not ignore character set information sent by the client. To ignore client information and use the default server character set, use `--skip-character-set-client-handshake`; this makes MySQL behave like MySQL 4.0.

- `--chroot=dir_name`, `-r dir_name`

Command-Line Format	`--chroot=dir_name`
Type	Directory name

Put the `mysqld` server in a closed environment during startup by using the `chroot()` system call. This is a recommended security measure.

Note

Use of this option somewhat limits `LOAD DATA` and `SELECT ... INTO OUTFILE`.

- `--console`

Command-Line Format	`--console`
Platform Specific	Windows

(Windows only.) Write error log messages to `stderr` and `stdout` (the console). `mysqld` does not close the console window if this option is used.

`--log-error` takes precedence over `--console` if both are given.

- `--core-file`

Command-Line Format	`--core-file[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

Write a core file if `mysqld` dies. The name and location of the core file is system dependent. On Linux, a core file named `core.pid` is written to the current working directory of the process, which for `mysqld` is the data directory. `pid` represents the process ID of the server process. On macOS, a core file named `core.pid` is written to the `/cores` directory. On Solaris, use the `coreadm` command to specify where to write the core file and how to name it.

For some systems, to get a core file you must also specify the `--core-file-size` option to `mysqld_safe`. See Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”. On some systems, such as Solaris, you do not get a core file if you are also using the `--user` option. There might be additional restrictions or limitations. For example, it might be necessary to execute `ulimit -c unlimited` before starting the server. Consult your system documentation.

- `--datadir=dir_name`, `-h dir_name`

Command-Line Format	`--datadir=dir_name`
System Variable	`datadir`
Scope	Global
Dynamic	No
Type	Directory name

The path to the MySQL server data directory. This option sets the `datadir` system variable. See the description of that variable.

- `--debug[=debug_options]`, `-# [debug_options]`

Command-Line Format	`--debug[=debug_options]`
System Variable	`debug`
Scope	Global, Session
Dynamic	Yes
Type	String
Default Value (Windows)	`d:t:i:O,\mysqld.trace`
Default Value (Unix)	`d:t:i:o,/tmp/mysqld.trace`

If MySQL is configured with the `-DWITH_DEBUG=1 CMake` option, you can use this option to get a trace file of what `mysqld` is doing. A typical *`debug_options`* string is `d:t:o,`*`file_name`*. The default is `d:t:i:o,/tmp/mysqld.trace` on Unix and `d:t:i:O,\mysqld.trace` on Windows.

Using `-DWITH_DEBUG=1` to configure MySQL with debugging support enables you to use the `--debug="d,parser_debug"` option when you start the server. This causes the Bison parser that is used to process SQL statements to dump a parser trace to the server's standard error output. Typically, this output is written to the error log.

This option may be given multiple times. Values that begin with `+` or `-` are added to or subtracted from the previous value. For example, `--debug=T --debug=+P` sets the value to `P:T`.

For more information, see Section 5.8.3, “The DBUG Package”.

- `--debug-sync-timeout[=N]`

Command-Line Format	`--debug-sync-timeout[=#]`
Type	Integer

Controls whether the Debug Sync facility for testing and debugging is enabled. Use of Debug Sync requires that MySQL be configured with the `-DENABLE_DEBUG_SYNC=1 CMake` option (see Section 2.9.7, “MySQL Source-Configuration Options”). If Debug Sync is not compiled in, this option is not available. The option value is a timeout in seconds. The default value is 0, which disables Debug Sync. To enable it, specify a value greater than 0; this value also becomes the default timeout for individual synchronization points. If the option is given without a value, the timeout is set to 300 seconds.

For a description of the Debug Sync facility and how to use synchronization points, see MySQL Internals: Test Synchronization.

- `--default-authentication-plugin=plugin_name`

Command-Line Format	`--default-authentication-plugin=plugin_name`
Type	Enumeration
Default Value	`mysql_native_password`
Valid Values	`mysql_native_password` `sha256_password`

This option sets the default authentication plugin. These values are permitted:

- `mysql_native_password`: Use MySQL native passwords; see Section 6.4.1.1, “Native Pluggable Authentication”.
- `sha256_password`: Use SHA-256 passwords; see Section 6.4.1.4, “SHA-256 Pluggable Authentication”.

Note

Prior to MySQL 5.6.17, if you use this option to change the default authentication plugin to a value other than `mysql_native_password`, clients older than MySQL 5.5.7 cannot connect because they do not understand the resulting change to the authentication protocol.

In MySQL 5.6, there is no server system variable that corresponds to the `--default-authentication-plugin` option; to read the value set for this option, you must check the running MySQL Server process to see what options, if any, were employed when invoking `mysqld` (on Linux systems, you can do this with `ps ax | grep mysqld` or similar), and possibly the server options file as well. This is a known issue which is resolved in MySQL 5.7. (Bug #68858, Bug #16595944)

The `--default-authentication-plugin` value affects these aspects of server operation:

- It determines which authentication plugin the server assigns to new accounts created by `CREATE USER` and `GRANT` statements that do not explicitly specify an authentication plugin.
- The `old_passwords` system variable affects password hashing for accounts that use the `mysql_native_password` or `sha256_password` authentication plugin. If the default authentication plugin is one of those plugins, the server sets `old_passwords` at startup to the value required by the plugin password hashing method.
- For an account created with either of the following statements, the server associates the account with the default authentication plugin and assigns the account the given password, hashed as required by that plugin:

```
CREATE USER ... IDENTIFIED BY 'cleartext password';
GRANT ...  IDENTIFIED BY 'cleartext password';
```

- For an account created with either of the following statements, the server associates the account with the default authentication plugin and assigns the account the given password hash, if the password hash has the format required by the plugin:

```
CREATE USER ... IDENTIFIED BY PASSWORD 'encrypted password';
GRANT ...  IDENTIFIED BY PASSWORD 'encrypted password';
```

  If the password hash is not in the format required by the default authentication plugin, the statement fails.

- `--default-time-zone=timezone`

Command-Line Format	`--default-time-zone=name`
Type	String

Set the default server time zone. This option sets the global `time_zone` system variable. If this option is not given, the default time zone is the same as the system time zone (given by the value of the `system_time_zone` system variable.

- `--defaults-extra-file=file_name`

Read this option file after the global option file but (on Unix) before the user option file. If the file does not exist or is otherwise inaccessible, an error occurs. *`file_name`* is interpreted relative to the current directory if given as a relative path name rather than a full path name. This must be the first option on the command line if it is used.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=file_name`

  Read only the given option file. If the file does not exist or is otherwise inaccessible, an error occurs. `file_name` is interpreted relative to the current directory if given as a relative path name rather than a full path name.

  Note

  This must be the first option on the command line if it is used, except that if the server is started with the `--defaults-file` and `--install` (or `--install-manual`) options, `--install` (or `--install-manual`) must be first.

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- `--defaults-group-suffix=str`

  Read not only the usual option groups, but also groups with the usual names and a suffix of `str`. For example, `mysqld` normally reads the `[mysqld]` group. If the `--defaults-group-suffix=_other` option is given, `mysqld` also reads the `[mysqld_other]` group.

  For additional information about this and other option-file options, see Section 4.2.2.3, "Command-Line Options that Affect Option-File Handling".

- `--des-key-file=file_name`

Command-Line Format	`--des-key-file=file_name`

  Read the default DES keys from this file. These keys are used by the `DES_ENCRYPT()` and `DES_DECRYPT()` functions.

- `--exit-info[=flags]`, `-T [flags]`

Command-Line Format	`--exit-info[=flags]`
Type	Integer

  This is a bitmask of different flags that you can use for debugging the `mysqld` server. Do not use this option unless you know *exactly* what it does!

- `--external-locking`

Command-Line Format	`--external-locking[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

  Enable external locking (system locking), which is disabled by default. If you use this option on a system on which `lockd` does not fully work (such as Linux), it is easy for `mysqld` to deadlock.

  To disable external locking explicitly, use `--skip-external-locking`.

  External locking affects only `MyISAM` table access. For more information, including conditions under which it can and cannot be used, see Section 8.11.5, "External Locking".

- `--flush`

Command-Line Format	`--flush[={OFF|ON}]`
System Variable	`flush`

Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

Flush (synchronize) all changes to disk after each SQL statement. Normally, MySQL does a write of all changes to disk only after each SQL statement and lets the operating system handle the synchronizing to disk. See Section B.3.3.3, “What to Do If MySQL Keeps Crashing”.

Note

If `--flush` is specified, the value of `flush_time` does not matter and changes to `flush_time` have no effect on flush behavior.

- `--gdb`

Command-Line Format	`--gdb[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

Install an interrupt handler for `SIGINT` (needed to stop `mysqld` with `^C` to set breakpoints) and disable stack tracing and core file handling. See Section 5.8.1.4, “Debugging mysqld under gdb”.

- `--ignore-db-dir=`*`dir_name`*

Command-Line Format	`--ignore-db-dir=dir_name`
Type	Directory name

This option tells the server to ignore the given directory name for purposes of the `SHOW DATABASES` statement or `INFORMATION_SCHEMA` tables. For example, if a MySQL configuration locates the data directory at the root of a file system on Unix, the system might create a `lost+found` directory there that the server should ignore. Starting the server with `--ignore-db-dir=lost+found` causes that name not to be listed as a database.

To specify more than one name, use this option multiple times, once for each name. Specifying the option with an empty value (that is, as `--ignore-db-dir=`) resets the directory list to the empty list.

Instances of this option given at server startup are used to set the `ignore_db_dirs` system variable.

- `--innodb-`*`xxx`*

Set an option for the `InnoDB` storage engine. The `InnoDB` options are listed in Section 14.14, “InnoDB Startup Options and System Variables”.

- `--install [`*`service_name`*`]`

Command-Line Format	`--install [service_name]`

Platform Specific	Windows

(Windows only) Install the server as a Windows service that starts automatically during Windows startup. The default service name is `MySQL` if no *`service_name`* value is given. For more information, see Section 2.3.4.7, “Starting MySQL as a Windows Service”.

Note

If the server is started with the `--defaults-file` and `--install` options, `--install` must be first.

- `--install-manual [service_name]`

Command-Line Format	`--install-manual [service_name]`
Platform Specific	Windows

(Windows only) Install the server as a Windows service that must be started manually. It does not start automatically during Windows startup. The default service name is `MySQL` if no *`service_name`* value is given. For more information, see Section 2.3.4.7, “Starting MySQL as a Windows Service”.

Note

If the server is started with the `--defaults-file` and `--install-manual` options, `--install-manual` must be first.

- `--language=lang_name, -L lang_name`

Command-Line Format	`--language=name`
Deprecated	Yes; use `lc-messages-dir` instead
System Variable	`language`
Scope	Global
Dynamic	No
Type	Directory name
Default Value	`/usr/local/mysql/share/mysql/english/`

The language to use for error messages. *`lang_name`* can be given as the language name or as the full path name to the directory where the language files are installed. See Section 10.12, “Setting the Error Message Language”.

`--lc-messages-dir` and `--lc-messages` should be used rather than `--language`, which is deprecated and handled as a synonym for `--lc-messages-dir`. Expect the `--language` option to be removed in a future MySQL release.

- `--large-pages`

Command-Line Format	`--large-pages[={OFF|ON}]`
System Variable	`large_pages`
Scope	Global
Dynamic	No
Platform Specific	Linux
Type	Boolean

Default Value	OFF

Some hardware/operating system architectures support memory pages greater than the default (usually 4KB). The actual implementation of this support depends on the underlying hardware and operating system. Applications that perform a lot of memory accesses may obtain performance improvements by using large pages due to reduced Translation Lookaside Buffer (TLB) misses.

MySQL supports the Linux implementation of large page support (which is called HugeTLB in Linux). See Section 8.12.4.2, “Enabling Large Page Support”. For Solaris support of large pages, see the description of the `--super-large-pages` option.

`--large-pages` is disabled by default.

- `--lc-messages=locale_name`

Command-Line Format	--lc-messages=name
System Variable	lc_messages
Scope	Global, Session
Dynamic	Yes
Type	String
Default Value	en_US

The locale to use for error messages. The default is `en_US`. The server converts the argument to a language name and combines it with the value of `--lc-messages-dir` to produce the location for the error message file. See Section 10.12, “Setting the Error Message Language”.

- `--lc-messages-dir=dir_name`

Command-Line Format	--lc-messages-dir=dir_name
System Variable	lc_messages_dir
Scope	Global
Dynamic	No
Type	Directory name

The directory where error messages are located. The server uses the value together with the value of `--lc-messages` to produce the location for the error message file. See Section 10.12, “Setting the Error Message Language”.

- `--local-service`

Command-Line Format	--local-service

(Windows only) A `--local-service` option following the service name causes the server to run using the `LocalService` Windows account that has limited system privileges. If both `--defaults-file` and `--local-service` are given following the service name, they can be in any order. See Section 2.3.4.7, “Starting MySQL as a Windows Service”.

- `--log-error[=file_name]`

Command-Line Format	--log-error[=file_name]
System Variable	log_error
Scope	Global
Dynamic	No
Type	File name

Write the error log and startup messages to this file. See Section 5.4.2, “The Error Log”.

If the option names no file, the error log file name on Unix and Unix-like systems is *`host_name`*`.err` in the data directory. The file name on Windows is the same, unless the `--pid-file` option is specified. In that case, the file name is the PID file base name with a suffix of `.err` in the data directory.

If the option names a file, the error log file has that name (with an `.err` suffix added if the name has no suffix), located under the data directory unless an absolute path name is given to specify a different location.

On Windows, `--log-error` takes precedence over `--console` if both are given.

- `--log-isam[=`*`file_name`*`]`

Command-Line Format	`--log-isam[=file_name]`
Type	File name

Log all `MyISAM` changes to this file (used only when debugging `MyISAM`).

- `--log-raw`

Command-Line Format	`--log-raw[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

Passwords in certain statements written to the general query log, slow query log, and binary log are rewritten by the server not to occur literally in plain text. Password rewriting can be suppressed for the general query log by starting the server with the `--log-raw` option. This option may be useful for diagnostic purposes, to see the exact text of statements as received by the server, but for security reasons is not recommended for production use.

- `--log-short-format`

Command-Line Format	`--log-short-format[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

Log less information to the slow query log, if it has been activated.

- `--log-slow-admin-statements`

Command-Line Format	`--log-slow-admin-statements[={OFF|ON}]` (5.6.10)
Removed	5.6.11
Type	Boolean
Default Value	`OFF`

Include slow administrative statements in the statements written to the slow query log. Administrative statements include `ALTER TABLE`, `ANALYZE TABLE`, `CHECK TABLE`, `CREATE INDEX`, `DROP INDEX`, `OPTIMIZE TABLE`, and `REPAIR TABLE`.

This command-line option was removed in MySQL 5.6.11 and replaced by the `log_slow_admin_statements` system variable. The system variable can be set on the command line or in option files the same way as the option, so there is no need for any changes at server startup, but the system variable also makes it possible to examine or set the value at runtime.

- `--log-tc=file_name`

Command-Line Format	`--log-tc=file_name`
Type	File name
Default Value	`tc.log`

The name of the memory-mapped transaction coordinator log file (for XA transactions that affect multiple storage engines when the binary log is disabled). The default name is `tc.log`. The file is created under the data directory if not given as a full path name. This option is unused.

- `--log-tc-size=size`

Command-Line Format	`--log-tc-size=#`
Type	Integer
Default Value	`24576`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

The size in bytes of the memory-mapped transaction coordinator log. The default size is 24KB.

- `--log-warnings[=level]`, `-W [level]`

Command-Line Format	`--log-warnings[=#]`
System Variable	`log_warnings`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`1`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

Print out warnings such as `Aborted connection...` to the error log. This option is enabled (1) by default. To disable it, use `--log-warnings=0`. Specifying the option without a *`level`* value increments the current value by 1. Enabling this option by setting it greater than 0 is recommended, for example, if you use replication (you get more information about what is happening, such as messages about network failures and reconnections). If the value is greater than 1, aborted connections are written to the error log, and access-denied errors for new connection attempts are written. See Section B.3.2.10, “Communication Errors and Aborted Connections”.

If a replica server was started with `--log-warnings` enabled, the replica prints messages to the error log to provide information about its status, such as the binary log and relay log coordinates where it starts its job, when it is switching to another relay log, when it reconnects after a disconnect, and so forth. The server logs messages about statements that are unsafe for statement-based logging if `--log-warnings` is greater than 0.

- `--memlock`

Command-Line Format	`--memlock[={OFF|ON}]`

Type	Boolean
Default Value	OFF

Lock the mysqld process in memory. This option might help if you have a problem where the operating system is causing mysqld to swap to disk.

--memlock works on systems that support the mlockall() system call; this includes Solaris, most Linux distributions that use a 2.4 or higher kernel, and perhaps other Unix systems. On Linux systems, you can tell whether or not mlockall() (and thus this option) is supported by checking to see whether or not it is defined in the system mman.h file, like this:

```
shell> grep mlockall /usr/include/sys/mman.h
```

If mlockall() is supported, you should see in the output of the previous command something like the following:

```
extern int mlockall (int __flags) __THROW;
```

Important

Use of this option may require you to run the server as root, which, for reasons of security, is normally not a good idea. See Section 6.1.5, “How to Run MySQL as a Normal User”.

On Linux and perhaps other systems, you can avoid the need to run the server as root by changing the limits.conf file. See the notes regarding the memlock limit in Section 8.12.4.2, “Enabling Large Page Support”.

You must not use this option on a system that does not support the mlockall() system call; if you do so, mysqld is very likely to exit as soon as you try to start it.

- --myisam-block-size=*N*

Command-Line Format	--myisam-block-size=#
Type	Integer
Default Value	1024
Minimum Value	1024
Maximum Value	16384

The block size to be used for MyISAM index pages.

- --no-defaults

Do not read any option files. If program startup fails due to reading unknown options from an option file, --no-defaults can be used to prevent them from being read. This must be the first option on the command line if it is used.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- --old-style-user-limits

Command-Line Format	--old-style-user-limits[={OFF|ON}]
Type	Boolean

Default Value	OFF

Enable old-style user limits. (Before MySQL 5.0.3, account resource limits were counted separately for each host from which a user connected rather than per account row in the `user` table.) See Section 6.2.13, “Setting Account Resource Limits”.

- `--partition[=value]`

Command-Line Format	`--partition[={OFF|ON}]`
Disabled by	`skip-partition`
Type	Boolean
Default Value	`ON`

Enables or disables user-defined partitioning support in the MySQL Server.

- `--performance-schema-xxx`

Configure a Performance Schema option. For details, see Section 22.14, “Performance Schema Command Options”.

- `--plugin-load=plugin_list`

Command-Line Format	`--plugin-load=plugin_list`
System Variable	`plugin_load`
Scope	Global
Dynamic	No
Type	String

This option tells the server to load the named plugins at startup. If multiple `--plugin-load` options are given, only the last one is used. Additional plugins to load may be specified using `--plugin-load-add` options.

The option value is a semicolon-separated list of *`name=plugin_library`* and *`plugin_library`* values. Each *`name`* is the name of a plugin to load, and *`plugin_library`* is the name of the library file that contains the plugin code. If a plugin library is named without any preceding plugin name, the server loads all plugins in the library. The server looks for plugin library files in the directory named by the `plugin_dir` system variable.

For example, if plugins named `myplug1` and `myplug2` have library files `myplug1.so` and `myplug2.so`, use this option to perform an early plugin load:

```
shell> mysqld --plugin-load="myplug1=myplug1.so;myplug2=myplug2.so"
```

Quotes are used around the argument value here because otherwise semicolon (`;`) is interpreted as a special character by some command interpreters. (Unix shells treat it as a command terminator, for example.)

Each named plugin is loaded for a single invocation of `mysqld` only. After a restart, the plugin is not loaded unless `--plugin-load` is used again. This is in contrast to `INSTALL PLUGIN`, which adds an entry to the `mysql.plugins` table to cause the plugin to be loaded for every normal server startup.

During the normal startup sequence, the server determines which plugins to load by reading the `mysql.plugins` system table. If the server is started with the `--skip-grant-tables` option, plugins registered in the `mysql.plugins` table are not loaded and are unavailable. `--plugin-`

`load` enables plugins to be loaded even when `--skip-grant-tables` is given. `--plugin-load` also enables plugins to be loaded at startup that cannot be loaded at runtime.

For additional information about plugin loading, see Section 5.5.1, “Installing and Uninstalling Plugins”.

- `--plugin-load-add=plugin_list`

Command-Line Format	`--plugin-load-add=plugin_list`
System Variable	`plugin_load_add`
Scope	Global
Dynamic	No
Type	String

This option complements the `--plugin-load` option. `--plugin-load-add` adds a plugin or plugins to the set of plugins to be loaded at startup. The argument format is the same as for `--plugin-load`. `--plugin-load-add` can be used to avoid specifying a large set of plugins as a single long unwieldy `--plugin-load` argument.

`--plugin-load-add` can be given in the absence of `--plugin-load`, but any instance of `--plugin-load-add` that appears before `--plugin-load`. has no effect because `--plugin-load` resets the set of plugins to load. In other words, these options:

```
--plugin-load=x --plugin-load-add=y
```

are equivalent to this option:

```
--plugin-load="x;y"
```

But these options:

```
--plugin-load-add=y --plugin-load=x
```

are equivalent to this option:

```
--plugin-load=x
```

For additional information about plugin loading, see Section 5.5.1, “Installing and Uninstalling Plugins”.

- `--plugin-xxx`

Specifies an option that pertains to a server plugin. For example, many storage engines can be built as plugins, and for such engines, options for them can be specified with a `--plugin` prefix. Thus, the `--innodb-file-per-table` option for `InnoDB` can be specified as `--plugin-innodb-file-per-table`.

For boolean options that can be enabled or disabled, the `--skip` prefix and other alternative formats are supported as well (see Section 4.2.2.4, “Program Option Modifiers”). For example, `--skip-plugin-innodb-file-per-table` disables `innodb-file-per-table`.

The rationale for the `--plugin` prefix is that it enables plugin options to be specified unambiguously if there is a name conflict with a built-in server option. For example, were a plugin writer to name a plugin “sql” and implement a “mode” option, the option name might be `--sql-mode`, which would conflict with the built-in option of the same name. In such cases, references to the conflicting name are resolved in favor of the built-in option. To avoid the ambiguity, users can specify the plugin option as `--plugin-sql-mode`. Use of the `--plugin` prefix for plugin options is recommended to avoid any question of ambiguity.

- `--port=port_num`, `-P port_num`

Command-Line Format	`--port=port_num`
System Variable	`port`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`3306`
Minimum Value	`0`
Maximum Value	`65535`

The port number to use when listening for TCP/IP connections. On Unix and Unix-like systems, the port number must be 1024 or higher unless the server is started by the `root` operating system user. Setting this option to 0 causes the default value to be used.

- `--port-open-timeout=num`

Command-Line Format	`--port-open-timeout=#`
Type	Integer
Default Value	`0`

On some systems, when the server is stopped, the TCP/IP port might not become available immediately. If the server is restarted quickly afterward, its attempt to reopen the port can fail. This option indicates how many seconds the server should wait for the TCP/IP port to become free if it cannot be opened. The default is not to wait.

- `--print-defaults`

Print the program name and all options that it gets from option files. This must be the first option on the command line if it is used, except that it may be used immediately after `--defaults-file` or `--defaults-extra-file`.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--remove [service_name]`

Command-Line Format	`--remove [service_name]`
Platform Specific	Windows

(Windows only) Remove a MySQL Windows service. The default service name is `MySQL` if no `service_name` value is given. For more information, see Section 2.3.4.7, “Starting MySQL as a Windows Service”.

- `--safe-user-create`

Command-Line Format	`--safe-user-create[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

If this option is enabled, a user cannot create new MySQL users by using the `GRANT` statement unless the user has the `INSERT` privilege for the `mysql.user` system table or any column in the

table. If you want a user to have the ability to create new users that have those privileges that the user has the right to grant, you should grant the user the following privilege:

```
GRANT INSERT(user) ON mysql.user TO 'user_name'@'host_name';
```

This ensures that the user cannot change any privilege columns directly, but has to use the `GRANT` statement to give privileges to other users.

- `--skip-grant-tables`

Command-Line Format	`--skip-grant-tables[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

This option affects the server startup sequence:

- `--skip-grant-tables` causes the server not to read the grant tables in the `mysql` system database, and thus to start without using the privilege system at all. This gives anyone with access to the server *unrestricted access to all databases*.

  To cause a server started with `--skip-grant-tables` to load the grant tables at runtime, perform a privilege-flushing operation, which can be done in these ways:

  - Issue a MySQL `FLUSH PRIVILEGES` statement after connecting to the server.
  - Execute a `mysqladmin flush-privileges` or `mysqladmin reload` command from the command line.

  Privilege flushing might also occur implicitly as a result of other actions performed after startup, thus causing the server to start using the grant tables. For example, `mysql_upgrade` flushes the privileges during the upgrade procedure.

- `--skip-grant-tables` causes the server not to load certain other objects registered in the `mysql` system database:

  - Plugins installed using `INSTALL PLUGIN` and registered in the `mysql.plugin` system table.

    To cause plugins to be loaded even when using `--skip-grant-tables`, use the `--plugin-load` or `--plugin-load-add` option.

  - Scheduled events installed using `CREATE EVENT` and registered in the `mysql.event` system table.
  - User-defined functions (UDFs) installed using `CREATE FUNCTION` and registered in the `mysql.func` system table.

- `--skip-host-cache`

Command-Line Format	`--skip-host-cache`

Disable use of the internal host cache for faster name-to-IP resolution. With the cache disabled, the server performs a DNS lookup every time a client connects.

Use of `--skip-host-cache` is similar to setting the `host_cache_size` system variable to 0, but `host_cache_size` is more flexible because it can also be used to resize, enable, or disable the host cache at runtime, not just at server startup.

Starting the server with `--skip-host-cache` does not prevent runtime changes to the value of `host_cache_size`, but such changes have no effect and the cache is not re-enabled even if `host_cache_size` is set larger than 0.

For more information about how the host cache works, see Section 5.1.11.2, “DNS Lookups and the Host Cache”.

- `--skip-innodb`

Disable the `InnoDB` storage engine. In this case, because the default storage engine is `InnoDB`, the server cannot start unless you also use `--default-storage-engine` and `--default-tmp-storage-engine` to set the default to some other engine for both permanent and `TEMPORARY` tables.

As of MySQL 5.6.21, the `--skip-innodb` option is deprecated. Its use results in a warning. Expect this option to be removed in a future MySQL release.

- `--skip-new`

Command-Line Format	`--skip-new`

This option disables (what used to be considered) new, possibly unsafe behaviors. It results in these settings: `delay_key_write=OFF`, `concurrent_insert=NEVER`, `automatic_sp_privileges=OFF`. It also causes `OPTIMIZE TABLE` to be mapped to `ALTER TABLE` for storage engines for which `OPTIMIZE TABLE` is not supported.

- `--skip-partition`

Command-Line Format	`--skip-partition`
	
	`--disable-partition`

Disables user-defined partitioning. Partitioned tables can be seen using `SHOW TABLES` or by querying the `INFORMATION_SCHEMA.TABLES` table, but cannot be created or modified, nor can data in such tables be accessed. All partition-specific columns in the `INFORMATION_SCHEMA.PARTITIONS` table display `NULL`.

Since `DROP TABLE` removes table definition (`.frm`) files, this statement works on partitioned tables even when partitioning is disabled using the option. The statement, however, does not remove `.par` files associated with partitioned tables in such cases. For this reason, you should avoid dropping partitioned tables with partitioning disabled, or take action to remove the orphaned `.par` files manually.

- `--skip-show-database`

Command-Line Format	`--skip-show-database`
System Variable	`skip_show_database`
Scope	Global

Dynamic	No

This option sets the `skip_show_database` system variable that controls who is permitted to use the `SHOW DATABASES` statement. See Section 5.1.7, “Server System Variables”.

- `--skip-stack-trace`

Command-Line Format	`--skip-stack-trace`

Do not write stack traces. This option is useful when you are running `mysqld` under a debugger. On some systems, you also must use this option to get a core file. See Section 5.8, “Debugging MySQL”.

- `--slow-start-timeout=`*`timeout`*

Command-Line Format	`--slow-start-timeout=#`
Type	Integer
Default Value	`15000`

This option controls the Windows service control manager's service start timeout. The value is the maximum number of milliseconds that the service control manager waits before trying to kill the windows service during startup. The default value is 15000 (15 seconds). If the MySQL service takes too long to start, you may need to increase this value. A value of 0 means there is no timeout.

- `--socket=`*`path`*

Command-Line Format	`--socket={file_name|pipe_name}`
System Variable	`socket`
Scope	Global
Dynamic	No
Type	String
Default Value (Other)	`/tmp/mysql.sock`
Default Value (Windows)	`MySQL`

On Unix, this option specifies the Unix socket file to use when listening for local connections. The default value is `/tmp/mysql.sock`. If this option is given, the server creates the file in the data directory unless an absolute path name is given to specify a different directory. On Windows, the option specifies the pipe name to use when listening for local connections that use a named pipe. The default value is `MySQL` (not case-sensitive).

- `--sql-mode=`*`value`*`[,`*`value`*`[,`*`value`*`...]]`

Command-Line Format	`--sql-mode=name`
System Variable	`sql_mode`
Scope	Global, Session
Dynamic	Yes
Type	Set
Default Value	`NO_ENGINE_SUBSTITUTION`
Valid Values	`ALLOW_INVALID_DATES` `ANSI_QUOTES`
	`ERROR_FOR_DIVISION_BY_ZERO`

	HIGH_NOT_PRECEDENCE IGNORE_SPACE NO_AUTO_CREATE_USER NO_AUTO_VALUE_ON_ZERO NO_BACKSLASH_ESCAPES NO_DIR_IN_CREATE NO_ENGINE_SUBSTITUTION NO_FIELD_OPTIONS NO_KEY_OPTIONS NO_TABLE_OPTIONS NO_UNSIGNED_SUBTRACTION NO_ZERO_DATE NO_ZERO_IN_DATE ONLY_FULL_GROUP_BY PAD_CHAR_TO_FULL_LENGTH PIPES_AS_CONCAT REAL_AS_FLOAT STRICT_ALL_TABLES STRICT_TRANS_TABLES

Set the SQL mode. The default is `NO_ENGINE_SUBSTITUTION`. See Section 5.1.10, “Server SQL Modes”.

Note

MySQL installation programs may configure the SQL mode during the installation process. For example, `mysql_install_db` creates a default option file named `my.cnf` in the base installation directory. This file contains a line that sets the SQL mode; see Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”.

If the SQL mode differs from the default or from what you expect, check for a setting in an option file that the server reads at startup.

- `--ssl`, `--skip-ssl`

Command-Line Format	`--ssl[={OFF|ON}]`
Disabled by	`skip-ssl`
Type	Boolean

Default Value	OFF

This option specifies that the server permits but does not require encrypted connections. --ssl may be implied by setting ssl_xxx system variables, as indicated in the descriptions for those variables.

--ssl can be specified in negated form as --skip-ssl or a synonym (--ssl=OFF, --disable-ssl). In this case, the option specifies that the server does *not* permit encrypted connections, regardless of the settings of the ssl_xxx system variables.

For more information about configuring whether the server permits clients to connect using SSL and indicating where to find SSL keys and certificates, see Section 6.3.1, "Configuring MySQL to Use Encrypted Connections". Consider setting at least the ssl_cert and ssl_key system variables on the server side and the --ssl-ca (or --ssl-capath) option on the client side.

- --standalone

Command-Line Format	--standalone
Platform Specific	Windows

Available on Windows only; instructs the MySQL server not to run as a service.

- --super-large-pages

Command-Line Format	--super-large-pages[={OFF|ON}]
Platform Specific	Solaris
Type	Boolean
Default Value	OFF

Standard use of large pages in MySQL attempts to use the largest size supported, up to 4MB. Under Solaris, a "super large pages" feature enables uses of pages up to 256MB. This feature is available for recent SPARC platforms. It can be enabled or disabled by using the --super-large-pages or --skip-super-large-pages option.

- --symbolic-links, --skip-symbolic-links

Command-Line Format	--symbolic-links[={OFF|ON}]
Type	Boolean
Default Value	ON

Enable or disable symbolic link support. This option has different effects on Windows and Unix:

- On Windows, enabling symbolic links enables you to establish a symbolic link to a database directory by creating a *db_name*.sym file that contains the path to the real directory. See Section 8.12.3.3, "Using Symbolic Links for Databases on Windows".
- On Unix, enabling symbolic links means that you can link a MyISAM index file or data file to another directory with the INDEX DIRECTORY or DATA DIRECTORY option of the CREATE TABLE statement. If you delete or rename the table, the files that its symbolic links point to also are deleted or renamed. See Section 8.12.3.2, "Using Symbolic Links for MyISAM Tables on Unix".

- --sysdate-is-now

Command-Line Format	--sysdate-is-now[={OFF|ON}]
Type	Boolean

Default Value	OFF

SYSDATE() by default returns the time at which it executes, not the time at which the statement in which it occurs begins executing. This differs from the behavior of NOW(). This option causes SYSDATE() to be a synonym for NOW(). For information about the implications for binary logging and replication, see the description for SYSDATE() in Section 12.7, “Date and Time Functions” and for SET TIMESTAMP in Section 5.1.7, “Server System Variables”.

- --tc-heuristic-recover={COMMIT|ROLLBACK}

Command-Line Format	--tc-heuristic-recover=name
Type	Enumeration
Default Value	OFF
Valid Values	OFF COMMIT ROLLBACK

The decision to use in a manual heuristic recovery.

If a --tc-heuristic-recover option is specified, the server exits regardless of whether manual heuristic recovery is successful.

On systems with more than one storage engine capable of two-phase commit, the ROLLBACK option is not safe and causes recovery to halt with the following error:

```
[ERROR] --tc-heuristic-recover rollback
strategy is not safe on systems with more than one 2-phase-commit-capable
storage engine. Aborting crash recovery.
```

- --temp-pool

Command-Line Format	--temp-pool[={OFF|ON}]
Type	Boolean
Default Value (Other)	OFF
Default Value (Linux)	ON

This option is ignored except on Linux. On Linux, it causes most temporary files created by the server to use a small set of names, rather than a unique name for each new file. This works around a problem in the Linux kernel dealing with creating many new files with different names. With the old behavior, Linux seems to “leak” memory, because it is being allocated to the directory entry cache rather than to the disk cache.

- --transaction-isolation=*level*

Command-Line Format	--transaction-isolation=name
Type	Enumeration
Default Value	REPEATABLE-READ
Valid Values	READ-UNCOMMITTED READ-COMMITTED REPEATABLE-READ

	SERIALIZABLE

Sets the default transaction isolation level. The level value can be READ-UNCOMMITTED, READ-COMMITTED, REPEATABLE-READ, or SERIALIZABLE. See Section 13.3.6, "SET TRANSACTION Statement".

The default transaction isolation level can also be set at runtime using the SET TRANSACTION statement or by setting the tx_isolation system variable.

- --transaction-read-only

Command-Line Format	--transaction-read-only[={OFF|ON}]
Type	Boolean
Default Value	OFF

Sets the default transaction access mode. By default, read-only mode is disabled, so the mode is read/write.

To set the default transaction access mode at runtime, use the SET TRANSACTION statement or set the tx_read_only system variable. See Section 13.3.6, "SET TRANSACTION Statement".

- --tmpdir=*dir_name*, -t *dir_name*

Command-Line Format	--tmpdir=dir_name
System Variable	tmpdir
Scope	Global
Dynamic	No
Type	Directory name

The path of the directory to use for creating temporary files. It might be useful if your default /tmp directory resides on a partition that is too small to hold temporary tables. This option accepts several paths that are used in round-robin fashion. Paths should be separated by colon characters (:) on Unix and semicolon characters (;) on Windows.

--tmpdir can be a non-permanent location, such as a directory on a memory-based file system or a directory that is cleared when the server host restarts. If the MySQL server is acting as a replica, and you are using a non-permanent location for --tmpdir, consider setting a different temporary directory for the replica using the slave_load_tmpdir system variable. For a replica, the temporary files used to replicate LOAD DATA statements are stored in this directory, so with a permanent location they can survive machine restarts, although replication can now continue after a restart if the temporary files have been removed.

For more information about the storage location of temporary files, see Section B.3.3.5, "Where MySQL Stores Temporary Files".

- --user={*user_name*|*user_id*}, -u {*user_name*|*user_id*}

Command-Line Format	--user=name
Type	String

Run the mysqld server as the user having the name *user_name* or the numeric user ID *user_id*. ("User" in this context refers to a system login account, not a MySQL user listed in the grant tables.)

This option is *mandatory* when starting mysqld as root. The server changes its user ID during its startup sequence, causing it to run as that particular user rather than as root. See Section 6.1.1, "Security Guidelines".

To avoid a possible security hole where a user adds a `--user=root` option to a `my.cnf` file (thus causing the server to run as `root`), `mysqld` uses only the first `--user` option specified and produces a warning if there are multiple `--user` options. Options in `/etc/my.cnf` and `$MYSQL_HOME/my.cnf` are processed before command-line options, so it is recommended that you put a `--user` option in `/etc/my.cnf` and specify a value other than `root`. The option in `/etc/my.cnf` is found before any other `--user` options, which ensures that the server runs as a user other than `root`, and that a warning results if any other `--user` option is found.

- `--verbose`, `-v`

  Use this option with the `--help` option for detailed help.

- `--version`, `-V`

  Display version information and exit.

5.1.7 Server System Variables

The MySQL server maintains many system variables that configure its operation. Each system variable has a default value. System variables can be set at server startup using options on the command line or in an option file. Most of them can be changed dynamically at runtime using the `SET` statement, which enables you to modify operation of the server without having to stop and restart it. You can also use system variable values in expressions.

At runtime, setting a global system variable value requires the `SUPER` privilege. Setting a session system variable value normally requires no special privileges and can be done by any user, although there are exceptions. For more information, see Section 5.1.8.1, “System Variable Privileges”

There are several ways to see the names and values of system variables:

- To see the values that a server uses based on its compiled-in defaults and any option files that it reads, use this command:

```
mysqld --verbose --help
```

- To see the values that a server uses based only on its compiled-in defaults, ignoring the settings in any option files, use this command:

```
mysqld --no-defaults --verbose --help
```

- To see the current values used by a running server, use the `SHOW VARIABLES` statement.

This section provides a description of each system variable. For a system variable summary table, see Section 5.1.4, “Server System Variable Reference”. For more information about manipulation of system variables, see Section 5.1.8, “Using System Variables”.

For additional system variable information, see these sections:

- Section 5.1.8, “Using System Variables”, discusses the syntax for setting and displaying system variable values.
- Section 5.1.8.2, “Dynamic System Variables”, lists the variables that can be set at runtime.
- Information on tuning system variables can be found in Section 5.1.1, “Configuring the Server”.
- Section 14.14, “InnoDB Startup Options and System Variables”, lists `InnoDB` system variables.
- NDB Cluster System Variables, lists system variables which are specific to NDB Cluster.
- For information on server system variables specific to replication, see Section 17.1.4, “Replication and Binary Logging Options and Variables”.

Note

Some of the following variable descriptions refer to “enabling” or “disabling” a variable. These variables can be enabled with the `SET` statement by setting them to `ON` or `1`, or disabled by setting them to `OFF` or `0`. Boolean variables can be set at startup to the values `ON`, `TRUE`, `OFF`, and `FALSE` (not case-sensitive), as well as `1` and `0`. See Section 4.2.2.4, “Program Option Modifiers”.

Some system variables control the size of buffers or caches. For a given buffer, the server might need to allocate internal data structures. These structures typically are allocated from the total memory allocated to the buffer, and the amount of space required might be platform dependent. This means that when you assign a value to a system variable that controls a buffer size, the amount of space actually available might differ from the value assigned. In some cases, the amount might be less than the value assigned. It is also possible that the server adjusts a value upward. For example, if you assign a value of 0 to a variable for which the minimal value is 1024, the server sets the value to 1024.

Values for buffer sizes, lengths, and stack sizes are given in bytes unless otherwise specified.

Some system variables take file name values. Unless otherwise specified, the default file location is the data directory if the value is a relative path name. To specify the location explicitly, use an absolute path name. Suppose that the data directory is `/var/mysql/data`. If a file-valued variable is given as a relative path name, it is located under `/var/mysql/data`. If the value is an absolute path name, its location is as given by the path name.

- `authentication_windows_log_level`

Command-Line Format	`--authentication-windows-log-level=#`
System Variable	`authentication_windows_log_level`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`2`
Minimum Value	`0`
Maximum Value	`4`

This variable is available only if the `authentication_windows` Windows authentication plugin is enabled and debugging code is enabled. See Section 6.4.1.7, “Windows Pluggable Authentication”.

This variable sets the logging level for the Windows authentication plugin. The following table shows the permitted values.

Value	Description
0	No logging
1	Log only error messages
2	Log level 1 messages and warning messages
3	Log level 2 messages and information notes
4	Log level 3 messages and debug messages

- `authentication_windows_use_principal_name`

Command-Line Format	`--authentication-windows-use-principal-name[={OFF|ON}]`
System Variable	`authentication_windows_use_principal_name`
Scope	Global

Dynamic	No
Type	Boolean
Default Value	ON

This variable is available only if the authentication_windows Windows authentication plugin is enabled. See Section 6.4.1.7, “Windows Pluggable Authentication”.

A client that authenticates using the InitSecurityContext() function should provide a string identifying the service to which it connects (*targetName*). MySQL uses the principal name (UPN) of the account under which the server is running. The UPN has the form *user_id@computer_name* and need not be registered anywhere to be used. This UPN is sent by the server at the beginning of authentication handshake.

This variable controls whether the server sends the UPN in the initial challenge. By default, the variable is enabled. For security reasons, it can be disabled to avoid sending the server's account name to a client as cleartext. If the variable is disabled, the server always sends a 0x00 byte in the first challenge, the client does not specify *targetName*, and as a result, NTLM authentication is used.

If the server fails to obtain its UPN (which happens primarily in environments that do not support Kerberos authentication), the UPN is not sent by the server and NTLM authentication is used.

- autocommit

Command-Line Format	--autocommit[={OFF|ON}]
System Variable	autocommit
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	ON

The autocommit mode. If set to 1, all changes to a table take effect immediately. If set to 0, you must use COMMIT to accept a transaction or ROLLBACK to cancel it. If autocommit is 0 and you change it to 1, MySQL performs an automatic COMMIT of any open transaction. Another way to begin a transaction is to use a START TRANSACTION or BEGIN statement. See Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”.

By default, client connections begin with autocommit set to 1. To cause clients to begin with a default of 0, set the global autocommit value by starting the server with the --autocommit=0 option. To set the variable using an option file, include these lines:

```
[mysqld]
autocommit=0
```

- automatic_sp_privileges

Command-Line Format	--automatic-sp-privileges[={OFF|ON}]
System Variable	automatic_sp_privileges
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	ON

When this variable has a value of 1 (the default), the server automatically grants the EXECUTE and ALTER ROUTINE privileges to the creator of a stored routine, if the user cannot already execute

and alter or drop the routine. (The `ALTER ROUTINE` privilege is required to drop the routine.) The server also automatically drops those privileges from the creator when the routine is dropped. If `automatic_sp_privileges` is 0, the server does not automatically add or drop these privileges.

The creator of a routine is the account used to execute the `CREATE` statement for it. This might not be the same as the account named as the `DEFINER` in the routine definition.

If you start `mysqld` with `--skip-new`, `automatic_sp_privileges` is set to `OFF`.

See also Section 20.2.2, "Stored Routines and MySQL Privileges".

- `avoid_temporal_upgrade`

Command-Line Format	`--avoid-temporal-upgrade[={OFF|ON}]`
Introduced	5.6.24
Deprecated	5.6.24
System Variable	`avoid_temporal_upgrade`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

This variable controls whether `ALTER TABLE` implicitly upgrades temporal columns found to be in pre-5.6.4 format (`TIME`, `DATETIME`, and `TIMESTAMP` columns without support for fractional seconds precision). Upgrading such columns requires a table rebuild, which prevents any use of fast alterations that might otherwise apply to the operation to be performed.

This variable is disabled by default. Enabling it causes `ALTER TABLE` not to rebuild temporal columns and thereby be able to take advantage of possible fast alterations.

This variable was added in MySQL 5.6.24. It is deprecated; expect it to be removed in a future MySQL release.

- `back_log`

Command-Line Format	`--back-log=#`
System Variable	`back_log`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)
Minimum Value	`1`
Maximum Value	`65535`

The number of outstanding connection requests MySQL can have. This comes into play when the main MySQL thread gets very many connection requests in a very short time. It then takes some time (although very little) for the main thread to check the connection and start a new thread. The `back_log` value indicates how many requests can be stacked during this short time before MySQL momentarily stops answering new requests. You need to increase this only if you expect a large number of connections in a short period of time.

In other words, this value is the size of the listen queue for incoming TCP/IP connections. Your operating system has its own limit on the size of this queue. The manual page for the Unix

`listen()` system call should have more details. Check your OS documentation for the maximum value for this variable. `back_log` cannot be set higher than your operating system limit.

The default value is based on the following formula, capped to a limit of 900:

```
50 + (max_connections / 5)
```

- `basedir`

Command-Line Format	`--basedir=dir_name`
System Variable	`basedir`
Scope	Global
Dynamic	No
Type	Directory name
Default Value	`configuration-dependent default`

The path to the MySQL installation base directory.

- `big_tables`

Command-Line Format	`--big-tables[={OFF|ON}]`
System Variable	`big_tables`
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

If enabled, the server stores all temporary tables on disk rather than in memory. This prevents most `The table tbl_name is full` errors for `SELECT` operations that require a large temporary table, but also slows down queries for which in-memory tables would suffice.

The default value for new connections is `OFF` (use in-memory temporary tables). Normally, it should never be necessary to enable this variable because the server is able to handle large result sets automatically by using memory for small temporary tables and switching to disk-based tables as required.

- `bind_address`

Command-Line Format	`--bind-address=addr`
System Variable	`bind_address`
Scope	Global
Dynamic	No
Type	String
Default Value	`*`

The MySQL server listens on a single network socket for TCP/IP connections. This socket is bound to a single address, but it is possible for an address to map onto multiple network interfaces. To specify an address, set `bind_address=addr` at server startup, where *`addr`* is an IPv4 or IPv6 address or a host name. If *`addr`* is a host name, the server resolves the name to an IP address and binds to that address. If a host name resolves to multiple IP addresses, the server uses the first IPv4 address if there are any, or the first IPv6 address otherwise.

The server treats different types of addresses as follows:

- If the address is `*`, the server accepts TCP/IP connections on all server host IPv4 interfaces, and, if the server host supports IPv6, on all IPv6 interfaces. Use this address to permit both IPv4 and IPv6 connections on all server interfaces. This value is the default.
- If the address is `0.0.0.0`, the server accepts TCP/IP connections on all server host IPv4 interfaces.
- If the address is `::`, the server accepts TCP/IP connections on all server host IPv4 and IPv6 interfaces.
- If the address is an IPv4-mapped address, the server accepts TCP/IP connections for that address, in either IPv4 or IPv6 format. For example, if the server is bound to `::ffff:127.0.0.1`, clients can connect using `--host=127.0.0.1` or `--host=::ffff:127.0.0.1`.
- If the address is a "regular" IPv4 or IPv6 address (such as `127.0.0.1` or `::1`), the server accepts TCP/IP connections only for that IPv4 or IPv6 address.

If binding to the address fails, the server produces an error and does not start.

If you intend to bind the server to a specific address, be sure that the `mysql.user` system table contains an account with administrative privileges that you can use to connect to that address. Otherwise, you are not able to shut down the server. For example, if you bind the server to `*`, you can connect to it using all existing accounts. But if you bind the server to `::1`, it accepts connections only on that address. In that case, first make sure that the `'root'@'::1'` account is present in the `mysql.user` table so you can still connect to the server to shut it down.

- `block_encryption_mode`

Command-Line Format	`--block-encryption-mode=#`
Introduced	5.6.17
System Variable	`block_encryption_mode`
Scope	Global, Session
Dynamic	Yes
Type	String
Default Value	`aes-128-ecb`

This variable controls the block encryption mode for block-based algorithms such as AES. It affects encryption for `AES_ENCRYPT()` and `AES_DECRYPT()`.

`block_encryption_mode` takes a value in `aes-`*`keylen`*`-`*`mode`* format, where *`keylen`* is the key length in bits and *`mode`* is the encryption mode. The value is not case-sensitive. Permitted *`keylen`* values are 128, 192, and 256. Permitted encryption modes depend on whether MySQL was compiled using OpenSSL or yaSSL:

- For OpenSSL, permitted *`mode`* values are: `ECB`, `CBC`, `CFB1`, `CFB8`, `CFB128`, `OFB`
- For yaSSL, permitted *`mode`* values are: `ECB`, `CBC`

For example, this statement causes the AES encryption functions to use a key length of 256 bits and the CBC mode:

```
SET block_encryption_mode = 'aes-256-cbc';
```

An error occurs for attempts to set `block_encryption_mode` to a value containing an unsupported key length or a mode that the SSL library does not support.

This variable was added in MySQL 5.6.17.

- `bulk_insert_buffer_size`

Command-Line Format	`--bulk-insert-buffer-size=#`
System Variable	`bulk_insert_buffer_size`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`8388608`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

`MyISAM` uses a special tree-like cache to make bulk inserts faster for `INSERT ... SELECT`, `INSERT ... VALUES (...), (...), ...`, and `LOAD DATA` when adding data to nonempty tables. This variable limits the size of the cache tree in bytes per thread. Setting it to 0 disables this optimization. The default value is 8MB.

- `character_set_client`

System Variable	`character_set_client`
Scope	Global, Session
Dynamic	Yes
Type	String
Default Value	`utf8`

The character set for statements that arrive from the client. The session value of this variable is set using the character set requested by the client when the client connects to the server. (Many clients support a `--default-character-set` option to enable this character set to be specified explicitly. See also Section 10.4, “Connection Character Sets and Collations”.) The global value of the variable is used to set the session value in cases when the client-requested value is unknown or not available, or the server is configured to ignore client requests:

- The client requests a character set not known to the server. For example, a Japanese-enabled client requests `sjis` when connecting to a server not configured with `sjis` support.
- The client is from a version of MySQL older than MySQL 4.1, and thus does not request a character set.
- `mysqld` was started with the `--skip-character-set-client-handshake` option, which causes it to ignore client character set configuration. This reproduces MySQL 4.0 behavior and is useful should you wish to upgrade the server without upgrading all the clients.

Some character sets cannot be used as the client character set. Attempting to use them as the `character_set_client` value produces an error. See Impermissible Client Character Sets.

- `character_set_connection`

System Variable	`character_set_connection`
Scope	Global, Session
Dynamic	Yes
Type	String

Default Value	utf8

The character set used for literals specified without a character set introducer and for number-to-string conversion. For information about introducers, see Section 10.3.8, “Character Set Introducers”.

- character_set_database

System Variable	character_set_database
Scope	Global, Session
Dynamic	Yes
Type	String
Default Value	latin1
Footnote	This option is dynamic, but should be set only by server. You should not set this variable manually.

The character set used by the default database. The server sets this variable whenever the default database changes. If there is no default database, the variable has the same value as character_set_server.

- character_set_filesystem

Command-Line Format	--character-set-filesystem=name
System Variable	character_set_filesystem
Scope	Global, Session
Dynamic	Yes
Type	String
Default Value	binary

The file system character set. This variable is used to interpret string literals that refer to file names, such as in the LOAD DATA and SELECT ... INTO OUTFILE statements and the LOAD_FILE() function. Such file names are converted from character_set_client to character_set_filesystem before the file opening attempt occurs. The default value is binary, which means that no conversion occurs. For systems on which multibyte file names are permitted, a different value may be more appropriate. For example, if the system represents file names using UTF-8, set character_set_filesystem to 'utf8mb4'.

- character_set_results

System Variable	character_set_results
Scope	Global, Session
Dynamic	Yes
Type	String
Default Value	utf8

The character set used for returning query results to the client. This includes result data such as column values, result metadata such as column names, and error messages.

- character_set_server

Command-Line Format	--character-set-server=name
System Variable	character_set_server
Scope	Global, Session

Dynamic	Yes
Type	String
Default Value	`latin1`

The servers default character set. See Section 10.15, “Character Set Configuration”. If you set this variable, you should also set `collation_server` to specify the collation for the character set.

- `character_set_system`

System Variable	`character_set_system`
Scope	Global
Dynamic	No
Type	String
Default Value	`utf8`

The character set used by the server for storing identifiers. The value is always `utf8`.

- `character_sets_dir`

Command-Line Format	`--character-sets-dir=dir_name`
System Variable	`character_sets_dir`
Scope	Global
Dynamic	No
Type	Directory name

The directory where character sets are installed. See Section 10.15, “Character Set Configuration”.

- `collation_connection`

System Variable	`collation_connection`
Scope	Global, Session
Dynamic	Yes
Type	String

The collation of the connection character set. `collation_connection` is important for comparisons of literal strings. For comparisons of strings with column values, `collation_connection` does not matter because columns have their own collation, which has a higher collation precedence (see Section 10.8.4, “Collation Coercibility in Expressions”).

- `collation_database`

System Variable	`collation_database`
Scope	Global, Session
Dynamic	Yes
Type	String
Default Value	`latin1_swedish_ci`
Footnote	This option is dynamic, but should be set only by server. You should not set this variable manually.

The collation used by the default database. The server sets this variable whenever the default database changes. If there is no default database, the variable has the same value as `collation_server`.

- `collation_server`

Command-Line Format	`--collation-server=name`
System Variable	`collation_server`
Scope	Global, Session
Dynamic	Yes
Type	String
Default Value	`latin1_swedish_ci`

The server's default collation. See Section 10.15, “Character Set Configuration”.

- `completion_type`

Command-Line Format	`--completion-type=#`
System Variable	`completion_type`
Scope	Global, Session
Dynamic	Yes
Type	Enumeration
Default Value	`NO_CHAIN`

Valid Values	`NO_CHAIN`
	`CHAIN`
	`RELEASE`
	`0`
	`1`
	`2`

The transaction completion type. This variable can take the values shown in the following table. The variable can be assigned using either the name values or corresponding integer values.

Value	Description
`NO_CHAIN` (or 0)	`COMMIT` and `ROLLBACK` are unaffected. This is the default value.
`CHAIN` (or 1)	`COMMIT` and `ROLLBACK` are equivalent to `COMMIT AND CHAIN` and `ROLLBACK AND CHAIN`, respectively. (A new transaction starts immediately with the same isolation level as the just-terminated transaction.)
`RELEASE` (or 2)	`COMMIT` and `ROLLBACK` are equivalent to `COMMIT RELEASE` and `ROLLBACK RELEASE`, respectively. (The server disconnects after terminating the transaction.)

`completion_type` affects transactions that begin with `START TRANSACTION` or `BEGIN` and end with `COMMIT` or `ROLLBACK`. It does not apply to implicit commits resulting from execution of the statements listed in Section 13.3.3, “Statements That Cause an Implicit Commit”. It also does not apply for `XA COMMIT`, `XA ROLLBACK`, or when `autocommit=1`.

- `concurrent_insert`

Command-Line Format	`--concurrent-insert[=value]`

System Variable	concurrent_insert
Scope	Global
Dynamic	Yes
Type	Enumeration
Default Value	AUTO
Valid Values	NEVER AUTO ALWAYS 0 1 2

If AUTO (the default), MySQL permits INSERT and SELECT statements to run concurrently for MyISAM tables that have no free blocks in the middle of the data file.

This variable can take the values shown in the following table. The variable can be assigned using either the name values or corresponding integer values.

Value	Description
NEVER (or 0)	Disables concurrent inserts
AUTO (or 1)	(Default) Enables concurrent insert for MyISAM tables that do not have holes
ALWAYS (or 2)	Enables concurrent inserts for all MyISAM tables, even those that have holes. For a table with a hole, new rows are inserted at the end of the table if it is in use by another thread. Otherwise, MySQL acquires a normal write lock and inserts the row into the hole.

If you start mysqld with --skip-new, concurrent_insert is set to NEVER.

See also Section 8.11.3, “Concurrent Inserts”.

- connect_timeout

Command-Line Format	--connect-timeout=#
System Variable	connect_timeout
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	10
Minimum Value	2
Maximum Value	31536000

The number of seconds that the mysqld server waits for a connect packet before responding with Bad handshake. The default value is 10 seconds.

Increasing the connect_timeout value might help if clients frequently encounter errors of the form Lost connection to MySQL server at '*XXX*', system error: *errno*.

- `core_file`

System Variable	`core_file`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`OFF`

Whether to write a core file if the server unexpectedly exits. This variable is set by the `--core-file` option.

- `datadir`

Command-Line Format	`--datadir=dir_name`
System Variable	`datadir`
Scope	Global
Dynamic	No
Type	Directory name

The path to the MySQL server data directory. Relative paths are resolved with respect to the current directory. If the server is likely to be started automatically (that is, in contexts for which you cannot assume what the current directory is), it is best to specify the `datadir` value as an absolute path.

- `date_format`

This variable is unused. It is deprecated; expect it to be removed in a future MySQL release.

- `datetime_format`

This variable is unused. It is deprecated; expect it to be removed in a future MySQL release.

- `debug`

Command-Line Format	`--debug[=debug_options]`
System Variable	`debug`
Scope	Global, Session
Dynamic	Yes
Type	String
Default Value (Windows)	`d:t:i:O,\mysqld.trace`
Default Value (Unix)	`d:t:i:o,/tmp/mysqld.trace`

This variable indicates the current debugging settings. It is available only for servers built with debugging support. The initial value comes from the value of instances of the `--debug` option given at server startup. The global and session values may be set at runtime.

Setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.8.1, “System Variable Privileges”.

Assigning a value that begins with `+` or `-` cause the value to added to or subtracted from the current value:

```
mysql> SET debug = 'T';
mysql> SELECT @@debug;
+---------+
```

```
| @@debug |
+---------+
| T       |
+---------+

mysql> SET debug = '+P';
mysql> SELECT @@debug;
+---------+
| @@debug |
+---------+
| P:T     |
+---------+

mysql> SET debug = '-P';
mysql> SELECT @@debug;
+---------+
| @@debug |
+---------+
| T       |
+---------+
```

For more information, see Section 5.8.3, “The DBUG Package”.

- `debug_sync`

System Variable	`debug_sync`
Scope	Session
Dynamic	Yes
Type	String

This variable is the user interface to the Debug Sync facility. Use of Debug Sync requires that MySQL be configured with the `-DENABLE_DEBUG_SYNC=1 CMake` option (see Section 2.9.7, “MySQL Source-Configuration Options”). If Debug Sync is not compiled in, this system variable is not available.

The global variable value is read only and indicates whether the facility is enabled. By default, Debug Sync is disabled and the value of `debug_sync` is `OFF`. If the server is started with `--debug-sync-timeout=N`, where *`N`* is a timeout value greater than 0, Debug Sync is enabled and the value of `debug_sync` is `ON - current signal` followed by the signal name. Also, *`N`* becomes the default timeout for individual synchronization points.

The session value can be read by any user and has the same value as the global variable. The session value can be set to control synchronization points.

Setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.8.1, “System Variable Privileges”.

For a description of the Debug Sync facility and how to use synchronization points, see MySQL Internals: Test Synchronization.

- `default_storage_engine`

Command-Line Format	`--default-storage-engine=name`
System Variable	`default_storage_engine`
Scope	Global, Session
Dynamic	Yes
Type	Enumeration

Default Value	InnoDB

The default storage engine for tables. See Chapter 15, *Alternative Storage Engines*. This variable sets the storage engine for permanent tables only. To set the storage engine for TEMPORARY tables, set the default_tmp_storage_engine system variable.

To see which storage engines are available and enabled, use the SHOW ENGINES statement or query the INFORMATION_SCHEMA ENGINES table.

default_storage_engine should be used in preference to storage_engine, which is deprecated.

If you disable the default storage engine at server startup, you must set the default engine for both permanent and TEMPORARY tables to a different engine or the server cannot start.

- default_tmp_storage_engine

Command-Line Format	--default-tmp-storage-engine=name
System Variable	default_tmp_storage_engine
Scope	Global, Session
Dynamic	Yes
Type	Enumeration
Default Value	InnoDB

The default storage engine for TEMPORARY tables (created with CREATE TEMPORARY TABLE). To set the storage engine for permanent tables, set the default_storage_engine system variable.

If you disable the default storage engine at server startup, you must set the default engine for both permanent and TEMPORARY tables to a different engine or the server cannot start.

- default_week_format

Command-Line Format	--default-week-format=#
System Variable	default_week_format
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	0
Minimum Value	0
Maximum Value	7

The default mode value to use for the WEEK() function. See Section 12.7, “Date and Time Functions”.

- delay_key_write

Command-Line Format	--delay-key-write[={OFF|ON|ALL}]
System Variable	delay_key_write
Scope	Global
Dynamic	Yes
Type	Enumeration
Default Value	ON

Valid Values	`ON`
	`OFF`
	`ALL`

This variable specifies how to use delayed key writes. It applies only to `MyISAM` tables. Delayed key writing causes key buffers not to be flushed between writes. See also Section 15.2.1, “MyISAM Startup Options”.

This variable can have one of the following values to affect handling of the `DELAY_KEY_WRITE` table option that can be used in `CREATE TABLE` statements.

Option	Description
`OFF`	`DELAY_KEY_WRITE` is ignored.
`ON`	MySQL honors any `DELAY_KEY_WRITE` option specified in `CREATE TABLE` statements. This is the default value.
`ALL`	All new opened tables are treated as if they were created with the `DELAY_KEY_WRITE` option enabled.

Note

If you set this variable to `ALL`, you should not use `MyISAM` tables from within another program (such as another MySQL server or `myisamchk`) when the tables are in use. Doing so leads to index corruption.

If `DELAY_KEY_WRITE` is enabled for a table, the key buffer is not flushed for the table on every index update, but only when the table is closed. This speeds up writes on keys a lot, but if you use this feature, you should add automatic checking of all `MyISAM` tables by starting the server with the `myisam_recover_options` system variable set (for example, `myisam_recover_options='BACKUP,FORCE'`). See Section 5.1.7, “Server System Variables”, and Section 15.2.1, “MyISAM Startup Options”.

If you start `mysqld` with `--skip-new`, `delay_key_write` is set to `OFF`.

Warning

If you enable external locking with `--external-locking`, there is no protection against index corruption for tables that use delayed key writes.

- `delayed_insert_limit`

Command-Line Format	`--delayed-insert-limit=#`
Deprecated	Yes
System Variable	`delayed_insert_limit`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`100`
Minimum Value	`1`
Maximum Value (64-bit platforms)	`18446744073709551615`

Maximum Value (32-bit platforms)	4294967295

After inserting delayed_insert_limit delayed rows into a nontransactional table, the INSERT DELAYED handler thread checks whether there are any SELECT statements pending. If so, it permits them to execute before continuing to insert delayed rows.

This system variable is deprecated (because DELAYED inserts are deprecated); expect it to be removed in a future release.

- delayed_insert_timeout

Command-Line Format	--delayed-insert-timeout=#
Deprecated	Yes
System Variable	delayed_insert_timeout
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	300

How many seconds an INSERT DELAYED handler thread should wait for INSERT statements before terminating.

This system variable is deprecated (because DELAYED inserts are deprecated); expect it to be removed in a future release.

- delayed_queue_size

Command-Line Format	--delayed-queue-size=#
Deprecated	Yes
System Variable	delayed_queue_size
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	1000
Minimum Value	1
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295

This is a per-table limit on the number of rows to queue when handling INSERT DELAYED statements for nontransactional tables. If the queue becomes full, any client that issues an INSERT DELAYED statement waits until there is room in the queue again.

This system variable is deprecated (because DELAYED inserts are deprecated); expect it to be removed in a future release.

- disconnect_on_expired_password

Command-Line Format	--disconnect-on-expired-password[={OFF|ON}]
System Variable	disconnect_on_expired_password

Scope	Global
Dynamic	No
Type	Boolean
Default Value	`ON`

This variable controls how the server handles clients with expired passwords:

- If the client indicates that it can handle expired passwords, the value of `disconnect_on_expired_password` is irrelevant. The server permits the client to connect but puts it in sandbox mode.
- If the client does not indicate that it can handle expired passwords, the server handles the client according to the value of `disconnect_on_expired_password`:
  - If `disconnect_on_expired_password`: is enabled, the server disconnects the client.
  - If `disconnect_on_expired_password`: is disabled, the server permits the client to connect but puts it in sandbox mode.

For more information about the interaction of client and server settings relating to expired-password handling, see Section 6.2.10, “Server Handling of Expired Passwords”.

- `div_precision_increment`

Command-Line Format	`--div-precision-increment=#`
System Variable	`div_precision_increment`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`4`
Minimum Value	`0`
Maximum Value	`30`

This variable indicates the number of digits by which to increase the scale of the result of division operations performed with the `/` operator. The default value is 4. The minimum and maximum values are 0 and 30, respectively. The following example illustrates the effect of increasing the default value.

```
mysql> SELECT 1/7;
+--------+
| 1/7    |
+--------+
| 0.1429 |
+--------+
mysql> SET div_precision_increment = 12;
mysql> SELECT 1/7;
+----------------+
| 1/7            |
+----------------+
| 0.142857142857 |
+----------------+
```

- `end_markers_in_json`

Command-Line Format	`--end-markers-in-json[={OFF|ON}]`
System Variable	`end_markers_in_json`
Scope	Global, Session
Dynamic	Yes

Type	Boolean
Default Value	OFF

Whether optimizer JSON output should add end markers. See MySQL Internals: The end_markers_in_json System Variable.

- eq_range_index_dive_limit

Command-Line Format	--eq-range-index-dive-limit=#
System Variable	eq_range_index_dive_limit
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	10
Minimum Value	0
Maximum Value	4294967295

This variable indicates the number of equality ranges in an equality comparison condition when the optimizer should switch from using index dives to index statistics in estimating the number of qualifying rows. It applies to evaluation of expressions that have either of these equivalent forms, where the optimizer uses a nonunique index to look up *col_name* values:

```
col_name IN(val1, ..., valN)
col_name = val1 OR ... OR col_name = valN
```

In both cases, the expression contains *N* equality ranges. The optimizer can make row estimates using index dives or index statistics. If eq_range_index_dive_limit is greater than 0, the optimizer uses existing index statistics instead of index dives if there are eq_range_index_dive_limit or more equality ranges. Thus, to permit use of index dives for up to *N* equality ranges, set eq_range_index_dive_limit to *N* + 1. To disable use of index statistics and always use index dives regardless of *N*, set eq_range_index_dive_limit to 0.

For more information, see Equality Range Optimization of Many-Valued Comparisons.

To update table index statistics for best estimates, use ANALYZE TABLE.

- error_count

The number of errors that resulted from the last statement that generated messages. This variable is read only. See Section 13.7.5.18, “SHOW ERRORS Statement”.

- event_scheduler

Command-Line Format	--event-scheduler[=value]
System Variable	event_scheduler
Scope	Global
Dynamic	Yes
Type	Enumeration
Default Value	OFF
Valid Values	ON OFF

	DISABLED

This variable enables or disables, and starts or stops, the Event Scheduler. The possible status values are ON, OFF, and DISABLED, with the default being OFF. Turning the Event Scheduler OFF is not the same as disabling the Event Scheduler, which requires setting the status to DISABLED. This variable and its effects on the Event Scheduler's operation are discussed in greater detail in Section 20.4.2, “Event Scheduler Configuration”

- explicit_defaults_for_timestamp

Command-Line Format	--explicit-defaults-for-timestamp[={OFF|ON}]
Deprecated	Yes
System Variable	explicit_defaults_for_timestamp
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	OFF

This system variable determines whether the server enables certain nonstandard behaviors for default values and NULL-value handling in TIMESTAMP columns. By default, explicit_defaults_for_timestamp is disabled, which enables the nonstandard behaviors.

If explicit_defaults_for_timestamp is disabled, the server enables the nonstandard behaviors and handles TIMESTAMP columns as follows:

- TIMESTAMP columns not explicitly declared with the NULL attribute are automatically declared with the NOT NULL attribute. Assigning such a column a value of NULL is permitted and sets the column to the current timestamp.
- The first TIMESTAMP column in a table, if not explicitly declared with the NULL attribute or an explicit DEFAULT or ON UPDATE attribute, is automatically declared with the DEFAULT CURRENT_TIMESTAMP and ON UPDATE CURRENT_TIMESTAMP attributes.
- TIMESTAMP columns following the first one, if not explicitly declared with the NULL attribute or an explicit DEFAULT attribute, are automatically declared as DEFAULT '0000-00-00 00:00:00'

(the “zero” timestamp). For inserted rows that specify no explicit value for such a column, the column is assigned `'0000-00-00 00:00:00'` and no warning occurs.

Depending on whether strict SQL mode or the `NO_ZERO_DATE` SQL mode is enabled, a default value of `'0000-00-00 00:00:00'` may be invalid. Be aware that the `TRADITIONAL` SQL mode includes strict mode and `NO_ZERO_DATE`. See Section 5.1.10, “Server SQL Modes”.

In MySQL 5.6, the nonstandard behaviors just described are deprecated; expect them to be removed in a future MySQL release.

If `explicit_defaults_for_timestamp` is enabled, the server disables the nonstandard behaviors and handles `TIMESTAMP` columns as follows:

- It is not possible to assign a `TIMESTAMP` column a value of `NULL` to set it to the current timestamp. To assign the current timestamp, set the column to `CURRENT_TIMESTAMP` or a synonym such as `NOW()`.
- `TIMESTAMP` columns not explicitly declared with the `NOT NULL` attribute are automatically declared with the `NULL` attribute and permit `NULL` values. Assigning such a column a value of `NULL` sets it to `NULL`, not the current timestamp.
- `TIMESTAMP` columns declared with the `NOT NULL` attribute do not permit `NULL` values. For inserts that specify `NULL` for such a column, the result is either an error for a single-row insert or if strict SQL mode is enabled, or `'0000-00-00 00:00:00'` is inserted for multiple-row inserts with strict SQL mode disabled. In no case does assigning the column a value of `NULL` set it to the current timestamp.
- `TIMESTAMP` columns explicitly declared with the `NOT NULL` attribute and without an explicit `DEFAULT` attribute are treated as having no default value. For inserted rows that specify no explicit value for such a column, the result depends on the SQL mode. If strict SQL mode is enabled, an error occurs. If strict SQL mode is not enabled, the column is declared with the implicit default of `'0000-00-00 00:00:00'` and a warning occurs. This is similar to how MySQL treats other temporal types such as `DATETIME`.
- No `TIMESTAMP` column is automatically declared with the `DEFAULT CURRENT_TIMESTAMP` or `ON UPDATE CURRENT_TIMESTAMP` attributes. Those attributes must be explicitly specified.
- The first `TIMESTAMP` column in a table is not handled differently from `TIMESTAMP` columns following the first one.

If `explicit_defaults_for_timestamp` is disabled at server startup, this warning appears in the error log:

```
[Warning] TIMESTAMP with implicit DEFAULT value is deprecated.
Please use --explicit_defaults_for_timestamp server option (see
documentation for more details).
```

As indicated by the warning, to disable the deprecated nonstandard behaviors, enable the `explicit_defaults_for_timestamp` system variable at server startup.

Note

`explicit_defaults_for_timestamp` is itself deprecated because its only purpose is to permit control over deprecated `TIMESTAMP` behaviors that are to be removed in a future MySQL release. When removal of those behaviors occurs, `explicit_defaults_for_timestamp` has no purpose; expect it to be removed as well.

For additional information, see Section 11.2.6, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”.

- `external_user`

System Variable	`external_user`
Scope	Session
Dynamic	No
Type	String

The external user name used during the authentication process, as set by the plugin used to authenticate the client. With native (built-in) MySQL authentication, or if the plugin does not set the value, this variable is `NULL`. See Section 6.2.12, “Proxy Users”.

- `flush`

Command-Line Format	`--flush[={OFF|ON}]`
System Variable	`flush`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

If `ON`, the server flushes (synchronizes) all changes to disk after each SQL statement. Normally, MySQL does a write of all changes to disk only after each SQL statement and lets the operating system handle the synchronizing to disk. See Section B.3.3.3, “What to Do If MySQL Keeps Crashing”. This variable is set to `ON` if you start `mysqld` with the `--flush` option.

Note

If `flush` is enabled, the value of `flush_time` does not matter and changes to `flush_time` have no effect on flush behavior.

- `flush_time`

Command-Line Format	`--flush-time=#`
System Variable	`flush_time`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`0`
Minimum Value	`0`

If this is set to a nonzero value, all tables are closed every `flush_time` seconds to free up resources and synchronize unflushed data to disk. This option is best used only on systems with minimal resources. The default is 0.

Note

If `flush` is enabled, the value of `flush_time` does not matter and changes to `flush_time` have no effect on flush behavior.

- `foreign_key_checks`

System Variable	`foreign_key_checks`
Scope	Global, Session

Dynamic	Yes
Type	Boolean
Default Value	`ON`

If set to 1 (the default), foreign key constraints are checked. If set to 0, foreign key constraints are ignored, with a couple of exceptions. When re-creating a table that was dropped, an error is returned if the table definition does not conform to the foreign key constraints referencing the table. Likewise, an `ALTER TABLE` operation returns an error if a foreign key definition is incorrectly formed. For more information, see Section 13.1.17.5, “FOREIGN KEY Constraints”.

Beginning with MySQL NDB Cluster 7.3.2, setting this variable has the same effect on `NDB` tables as it does for `InnoDB` tables—previously, the setting was ignored and all such checks were enforced (Bug #14095855). Typically you leave this setting enabled during normal operation, to enforce referential integrity.

Disabling foreign key checking can be useful for reloading such tables in an order different from that required by their parent/child relationships. See Section 13.1.17.5, “FOREIGN KEY Constraints”.

Setting `foreign_key_checks` to 0 also affects data definition statements: `DROP SCHEMA` drops a schema even if it contains tables that have foreign keys that are referred to by tables outside the schema, and `DROP TABLE` drops tables that have foreign keys that are referred to by other tables.

Note

Setting `foreign_key_checks` to 1 does not trigger a scan of the existing table data. Therefore, rows added to the table while `foreign_key_checks=0` are not verified for consistency.

Warning

With `foreign_key_checks=0`, dropping an index required by a foreign key constraint places the table in an inconsistent state and causes the foreign key check that occurs at table load to fail. To avoid this problem, remove the foreign key constraint before dropping the index (Bug #70260).

- `ft_boolean_syntax`

Command-Line Format	`--ft-boolean-syntax=name`
System Variable	`ft_boolean_syntax`
Scope	Global
Dynamic	Yes
Type	String

Default Value	`+ -><()~*:""&|`

The list of operators supported by boolean full-text searches performed using `IN BOOLEAN MODE`. See Section 12.10.2, “Boolean Full-Text Searches”.

The default variable value is `'+ -><()~*:""&|'`. The rules for changing the value are as follows:

- Operator function is determined by position within the string.
- The replacement value must be 14 characters.
- Each character must be an ASCII nonalphanumeric character.
- Either the first or second character must be a space.
- No duplicates are permitted except the phrase quoting operators in positions 11 and 12. These two characters are not required to be the same, but they are the only two that may be.
- Positions 10, 13, and 14 (which by default are set to `:`, `&`, and `|`) are reserved for future extensions.

- `ft_max_word_len`

Command-Line Format	`--ft-max-word-len=#`
System Variable	`ft_max_word_len`
Scope	Global
Dynamic	No
Type	Integer
Minimum Value	`10`

The maximum length of the word to be included in a `MyISAM FULLTEXT` index.

Note

`FULLTEXT` indexes on `MyISAM` tables must be rebuilt after changing this variable. Use `REPAIR TABLE tbl_name QUICK`.

- `ft_min_word_len`

Command-Line Format	`--ft-min-word-len=#`
System Variable	`ft_min_word_len`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`4`
Minimum Value	`1`

The minimum length of the word to be included in a `MyISAM FULLTEXT` index.

Note

`FULLTEXT` indexes on `MyISAM` tables must be rebuilt after changing this variable. Use `REPAIR TABLE tbl_name QUICK`.

- `ft_query_expansion_limit`

Command-Line Format	`--ft-query-expansion-limit=#`
System Variable	`ft_query_expansion_limit`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`20`
Minimum Value	`0`
Maximum Value	`1000`

The number of top matches to use for full-text searches performed using `WITH QUERY EXPANSION`.

- `ft_stopword_file`

Command-Line Format	`--ft-stopword-file=file_name`
System Variable	`ft_stopword_file`
Scope	Global
Dynamic	No
Type	File name

The file from which to read the list of stopwords for full-text searches on `MyISAM` tables. The server looks for the file in the data directory unless an absolute path name is given to specify a different directory. All the words from the file are used; comments are *not* honored. By default, a built-in list of stopwords is used (as defined in the `storage/myisam/ft_static.c` file). Setting this variable to the empty string (`''`) disables stopword filtering. See also Section 12.10.4, “Full-Text Stopwords”.

Note

`FULLTEXT` indexes on `MyISAM` tables must be rebuilt after changing this variable or the contents of the stopword file. Use `REPAIR TABLE tbl_name QUICK`.

- `general_log`

Command-Line Format	`--general-log[={OFF|ON}]`
System Variable	`general_log`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

Whether the general query log is enabled. The value can be 0 (or `OFF`) to disable the log or 1 (or `ON`) to enable the log. The destination for log output is controlled by the `log_output` system variable; if that value is `NONE`, no log entries are written even if the log is enabled.

- `general_log_file`

Command-Line Format	`--general-log-file=file_name`
System Variable	`general_log_file`
Scope	Global
Dynamic	Yes

Type	File name
Default Value	`host_name.log`

The name of the general query log file. The default value is *`host_name`*`.log`, but the initial value can be changed with the `--general_log_file` option.

- `group_concat_max_len`

Command-Line Format	`--group-concat-max-len=#`
System Variable	`group_concat_max_len`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`1024`
Minimum Value	`4`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

The maximum permitted result length in bytes for the `GROUP_CONCAT()` function. The default is 1024.

- `have_compress`

  `YES` if the `zlib` compression library is available to the server, `NO` if not. If not, the `COMPRESS()` and `UNCOMPRESS()` functions cannot be used.

- `have_crypt`

  `YES` if the `crypt()` system call is available to the server, `NO` if not. If not, the `ENCRYPT()` function cannot be used.

- `have_dynamic_loading`

  `YES` if `mysqld` supports dynamic loading of plugins, `NO` if not. If the value is `NO`, you cannot use options such as `--plugin-load` to load plugins at server startup, or the `INSTALL PLUGIN` statement to load plugins at runtime.

- `have_geometry`

  `YES` if the server supports spatial data types, `NO` if not.

- `have_openssl`

  This variable is a synonym for `have_ssl`.

- `have_profiling`

  `YES` if statement profiling capability is present, `NO` if not. If present, the `profiling` system variable controls whether this capability is enabled or disabled. See Section 13.7.5.32, “SHOW PROFILES Statement”.

  This variable is deprecated; expect it to be removed in a future MySQL release.

- `have_query_cache`

  `YES` if `mysqld` supports the query cache, `NO` if not.

- have_rtree_keys

  YES if RTREE indexes are available, NO if not. (These are used for spatial indexes in MyISAM tables.)

- have_ssl

System Variable	have_ssl
Scope	Global
Dynamic	No
Type	String
Valid Values	YES (SSL support available) DISABLED (SSL support was compiled into server, but server was not started with necessary options to enable it)

  YES if mysqld supports SSL connections, DISABLED if the server was not started with the appropriate appropriate connection-encryption options to enable SSL connections. For more information, see Section 2.9.6, “Configuring SSL Library Support”.

- have_symlink

  YES if symbolic link support is enabled, NO if not. This is required on Unix for support of the DATA DIRECTORY and INDEX DIRECTORY table options, and on Windows for support of data directory symlinks. If the server is started with the --skip-symbolic-links option, the value is DISABLED.

- host_cache_size

Command-Line Format	--host-cache-size=#
System Variable	host_cache_size
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	-1 (signifies autosizing; do not assign this literal value)
Minimum Value	0
Maximum Value	65536

  The MySQL server maintains an in-memory host cache that contains client host name and IP address information and is used to avoid Domain Name System (DNS) lookups; see Section 5.1.11.2, “DNS Lookups and the Host Cache”.

  The host_cache_size variable controls the size of the host cache, as well as the size of the Performance Schema host_cache table that exposes the cache contents. Setting host_cache_size has these effects:

  - Setting the size to 0 disables the host cache. With the cache disabled, the server performs a DNS lookup every time a client connects.
  - Changing the size at runtime causes an implicit host cache flushing operation that clears the host cache, truncates the host_cache table, and unblocks any blocked hosts.

  The default value is autosized to 128, plus 1 for a value of max_connections up to 500, plus 1 for every increment of 20 over 500 in the max_connections value, capped to a limit of 2000.

  Using the --skip-host-cache option is similar to setting the host_cache_size system variable to 0, but host_cache_size is more flexible because it can also be used to resize, enable, and disable the host cache at runtime, not just at server startup. Starting the server with --skip-host-

cache does not prevent runtime changes to the value of host_cache_size, but such changes have no effect and the cache is not re-enabled even if host_cache_size is set larger than 0.

- hostname

System Variable	hostname
Scope	Global
Dynamic	No
Type	String

The server sets this variable to the server host name at startup.

- identity

This variable is a synonym for the last_insert_id variable. It exists for compatibility with other database systems. You can read its value with SELECT @@identity, and set it using SET identity.

- ignore_db_dirs

System Variable	ignore_db_dirs
Scope	Global
Dynamic	No
Type	String

A comma-separated list of names that are not considered as database directories in the data directory. The value is set from any instances of --ignore-db-dir given at server startup.

- init_connect

Command-Line Format	--init-connect=name
System Variable	init_connect
Scope	Global
Dynamic	Yes
Type	String

A string to be executed by the server for each client that connects. The string consists of one or more SQL statements, separated by semicolon characters.

For users that have the SUPER privilege, the content of init_connect is not executed. This is done so that an erroneous value for init_connect does not prevent all clients from connecting. For example, the value might contain a statement that has a syntax error, thus causing client connections to fail. Not executing init_connect for users that have the SUPER privilege enables them to open a connection and fix the init_connect value.

The server discards any result sets produced by statements in the value of init_connect.

- init_file

Command-Line Format	--init-file=file_name
System Variable	init_file
Scope	Global
Dynamic	No
Type	File name

If specified, this variable names a file containing SQL statements to be read and executed during the startup process. Each statement must be on a single line and should not include comments.

If the server is started with the `--bootstrap` option, it operates in bootstap mode and some functionality is unavailable that limits the statements permitted in the file. These include statements that relate to account management (such as `CREATE USER` or `GRANT`), replication, and global transaction identifiers. See Section 17.1.3, “Replication with Global Transaction Identifiers”.

- `innodb_xxx`

  `InnoDB` system variables are listed in Section 14.14, “InnoDB Startup Options and System Variables”. These variables control many aspects of storage, memory use, and I/O patterns for `InnoDB` tables, and are especially important now that `InnoDB` is the default storage engine.

- `insert_id`

  The value to be used by the following `INSERT` or `ALTER TABLE` statement when inserting an `AUTO_INCREMENT` value. This is mainly used with the binary log.

- `interactive_timeout`

Command-Line Format	`--interactive-timeout=#`
System Variable	`interactive_timeout`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`28800`
Minimum Value	`1`

  The number of seconds the server waits for activity on an interactive connection before closing it. An interactive client is defined as a client that uses the `CLIENT_INTERACTIVE` option to `mysql_real_connect()`. See also `wait_timeout`.

- `join_buffer_size`

Command-Line Format	`--join-buffer-size=#`
System Variable	`join_buffer_size`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`262144`
Minimum Value	`128`
Maximum Value (Other, 64-bit platforms)	`18446744073709547520`
Maximum Value (Other, 32-bit platforms)	`4294967295`
Maximum Value (Windows)	`4294967295`

  The minimum size of the buffer that is used for plain index scans, range index scans, and joins that do not use indexes and thus perform full table scans. Normally, the best way to get fast joins is to add indexes. Increase the value of `join_buffer_size` to get a faster full join when adding indexes is not possible. One join buffer is allocated for each full join between two tables. For a complex join between several tables for which indexes are not used, multiple join buffers might be necessary.

Unless a Block Nested-Loop or Batched Key Access algorithm is used, there is no gain from setting the buffer larger than required to hold each matching row, and all joins allocate at least the minimum size, so use caution in setting this variable to a large value globally. It is better to keep the global setting small and change the session setting to a larger value only in sessions that are doing large joins. Memory allocation time can cause substantial performance drops if the global size is larger than needed by most queries that use it.

When Block Nested-Loop is used, a larger join buffer can be beneficial up to the point where all required columns from all rows in the first table are stored in the join buffer. This depends on the query; the optimal size may be smaller than holding all rows from the first tables.

When Batched Key Access is used, the value of `join_buffer_size` defines how large the batch of keys is in each request to the storage engine. The larger the buffer, the more sequential access is made to the right hand table of a join operation, which can significantly improve performance.

The default is 256KB. The maximum permissible setting for `join_buffer_size` is 4GB−1. Larger values are permitted for 64-bit platforms (except 64-bit Windows, for which large values are truncated to 4GB−1 with a warning).

For additional information about join buffering, see Section 8.2.1.6, “Nested-Loop Join Algorithms”. For information about Batched Key Access, see Section 8.2.1.11, “Block Nested-Loop and Batched Key Access Joins”.

- `keep_files_on_create`

Command-Line Format	`--keep-files-on-create[={OFF|ON}]`
System Variable	`keep_files_on_create`
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

If a `MyISAM` table is created with no `DATA DIRECTORY` option, the `.MYD` file is created in the database directory. By default, if `MyISAM` finds an existing `.MYD` file in this case, it overwrites it. The same applies to `.MYI` files for tables created with no `INDEX DIRECTORY` option. To suppress this behavior, set the `keep_files_on_create` variable to `ON` (1), in which case `MyISAM` does not overwrite existing files and returns an error instead. The default value is `OFF` (0).

If a `MyISAM` table is created with a `DATA DIRECTORY` or `INDEX DIRECTORY` option and an existing `.MYD` or `.MYI` file is found, MyISAM always returns an error. It does not overwrite a file in the specified directory.

- `key_buffer_size`

Command-Line Format	`--key-buffer-size=#`
System Variable	`key_buffer_size`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`8388608`
Minimum Value	`8`
Maximum Value (64-bit platforms)	`OS_PER_PROCESS_LIMIT`

Maximum Value (32-bit platforms)	4294967295

Index blocks for MyISAM tables are buffered and are shared by all threads. key_buffer_size is the size of the buffer used for index blocks. The key buffer is also known as the key cache.

The maximum permissible setting for key_buffer_size is 4GB−1 on 32-bit platforms. Larger values are permitted for 64-bit platforms. The effective maximum size might be less, depending on your available physical RAM and per-process RAM limits imposed by your operating system or hardware platform. The value of this variable indicates the amount of memory requested. Internally, the server allocates as much memory as possible up to this amount, but the actual allocation might be less.

You can increase the value to get better index handling for all reads and multiple writes; on a system whose primary function is to run MySQL using the MyISAM storage engine, 25% of the machine's total memory is an acceptable value for this variable. However, you should be aware that, if you make the value too large (for example, more than 50% of the machine's total memory), your system might start to page and become extremely slow. This is because MySQL relies on the operating system to perform file system caching for data reads, so you must leave some room for the file system cache. You should also consider the memory requirements of any other storage engines that you may be using in addition to MyISAM.

For even more speed when writing many rows at the same time, use LOCK TABLES. See Section 8.2.4.1, “Optimizing INSERT Statements”.

You can check the performance of the key buffer by issuing a SHOW STATUS statement and examining the Key_read_requests, Key_reads, Key_write_requests, and Key_writes status variables. (See Section 13.7.5, “SHOW Statements”.) The Key_reads/Key_read_requests ratio should normally be less than 0.01. The Key_writes/Key_write_requests ratio is usually near 1 if you are using mostly updates and deletes, but might be much smaller if you tend to do updates that affect many rows at the same time or if you are using the DELAY_KEY_WRITE table option.

The fraction of the key buffer in use can be determined using key_buffer_size in conjunction with the Key_blocks_unused status variable and the buffer block size, which is available from the key_cache_block_size system variable:

```
1 - ((Key_blocks_unused * key_cache_block_size) / key_buffer_size)
```

This value is an approximation because some space in the key buffer is allocated internally for administrative structures. Factors that influence the amount of overhead for these structures include block size and pointer size. As block size increases, the percentage of the key buffer lost to overhead tends to decrease. Larger blocks results in a smaller number of read operations (because more keys are obtained per read), but conversely an increase in reads of keys that are not examined (if not all keys in a block are relevant to a query).

It is possible to create multiple MyISAM key caches. The size limit of 4GB applies to each cache individually, not as a group. See Section 8.10.2, “The MyISAM Key Cache”.

- key_cache_age_threshold

Command-Line Format	--key-cache-age-threshold=#
System Variable	key_cache_age_threshold
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	300

Minimum Value	100
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295

This value controls the demotion of buffers from the hot sublist of a key cache to the warm sublist. Lower values cause demotion to happen more quickly. The minimum value is 100. The default value is 300. See Section 8.10.2, “The MyISAM Key Cache”.

- key_cache_block_size

Command-Line Format	--key-cache-block-size=#
System Variable	key_cache_block_size
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	1024
Minimum Value	512
Maximum Value	16384

The size in bytes of blocks in the key cache. The default value is 1024. See Section 8.10.2, “The MyISAM Key Cache”.

- key_cache_division_limit

Command-Line Format	--key-cache-division-limit=#
System Variable	key_cache_division_limit
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	100
Minimum Value	1
Maximum Value	100

The division point between the hot and warm sublists of the key cache buffer list. The value is the percentage of the buffer list to use for the warm sublist. Permissible values range from 1 to 100. The default value is 100. See Section 8.10.2, “The MyISAM Key Cache”.

- large_files_support

System Variable	large_files_support
Scope	Global
Dynamic	No
Type	Boolean

Whether mysqld was compiled with options for large file support.

- large_pages

Command-Line Format	--large-pages[={OFF|ON}]

System Variable	large_pages
Scope	Global
Dynamic	No
Platform Specific	Linux
Type	Boolean
Default Value	OFF

Whether large page support is enabled (via the --large-pages option). See Section 8.12.4.2, "Enabling Large Page Support".

- large_page_size

System Variable	large_page_size
Scope	Global
Dynamic	No
Type	Integer
Default Value	0

If large page support is enabled, this shows the size of memory pages. Large memory pages are supported only on Linux; on other platforms, the value of this variable is always 0. See Section 8.12.4.2, "Enabling Large Page Support".

- last_insert_id

The value to be returned from LAST_INSERT_ID(). This is stored in the binary log when you use LAST_INSERT_ID() in a statement that updates a table. Setting this variable does not update the value returned by the mysql_insert_id() C API function.

- lc_messages

Command-Line Format	--lc-messages=name
System Variable	lc_messages
Scope	Global, Session
Dynamic	Yes
Type	String
Default Value	en_US

The locale to use for error messages. The default is en_US. The server converts the argument to a language name and combines it with the value of lc_messages_dir to produce the location for the error message file. See Section 10.12, "Setting the Error Message Language".

- lc_messages_dir

Command-Line Format	--lc-messages-dir=dir_name
System Variable	lc_messages_dir
Scope	Global
Dynamic	No
Type	Directory name

The directory where error messages are located. The server uses the value together with the value of lc_messages to produce the location for the error message file. See Section 10.12, "Setting the Error Message Language".

- `lc_time_names`

Command-Line Format	`--lc-time-names=value`
System Variable	`lc_time_names`
Scope	Global, Session
Dynamic	Yes
Type	String

This variable specifies the locale that controls the language used to display day and month names and abbreviations. This variable affects the output from the `DATE_FORMAT()`, `DAYNAME()` and `MONTHNAME()` functions. Locale names are POSIX-style values such as `'ja_JP'` or `'pt_BR'`. The default value is `'en_US'` regardless of your system's locale setting. For further information, see Section 10.16, “MySQL Server Locale Support”.

- `license`

System Variable	`license`
Scope	Global
Dynamic	No
Type	String
Default Value	`GPL`

The type of license the server has.

- `local_infile`

Command-Line Format	`--local-infile[={OFF|ON}]`
System Variable	`local_infile`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`ON`

This variable controls server-side `LOCAL` capability for `LOAD DATA` statements. Depending on the `local_infile` setting, the server refuses or permits local data loading by clients that have `LOCAL` enabled on the client side.

To explicitly cause the server to refuse or permit `LOAD DATA LOCAL` statements (regardless of how client programs and libraries are configured at build time or runtime), start `mysqld` with `local_infile` disabled or enabled, respectively. `local_infile` can also be set at runtime. For more information, see Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”.

- `lock_wait_timeout`

Command-Line Format	`--lock-wait-timeout=#`
System Variable	`lock_wait_timeout`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`31536000`
Minimum Value	`1`

Maximum Value	31536000

This variable specifies the timeout in seconds for attempts to acquire metadata locks. The permissible values range from 1 to 31536000 (1 year). The default is 31536000.

This timeout applies to all statements that use metadata locks. These include DML and DDL operations on tables, views, stored procedures, and stored functions, as well as `LOCK TABLES`, `FLUSH TABLES WITH READ LOCK`, and `HANDLER` statements.

This timeout does not apply to implicit accesses to system tables in the `mysql` database, such as grant tables modified by `GRANT` or `REVOKE` statements or table logging statements. The timeout does apply to system tables accessed directly, such as with `SELECT` or `UPDATE`.

The timeout value applies separately for each metadata lock attempt. A given statement can require more than one lock, so it is possible for the statement to block for longer than the `lock_wait_timeout` value before reporting a timeout error. When lock timeout occurs, `ER_LOCK_WAIT_TIMEOUT` is reported.

`lock_wait_timeout` does not apply to delayed inserts, which always execute with a timeout of 1 year. This is done to avoid unnecessary timeouts because a session that issues a delayed insert receives no notification of delayed insert timeouts.

- `locked_in_memory`

System Variable	locked_in_memory
Scope	Global
Dynamic	No

Whether `mysqld` was locked in memory with `--memlock`.

- `log_error`

Command-Line Format	--log-error[=file_name]
System Variable	log_error
Scope	Global
Dynamic	No
Type	File name

The name of the error log file, or empty if the server is writing error messages to the console rather than to a named file. See Section 5.4.2, “The Error Log”.

- `log_output`

Command-Line Format	--log-output=name
System Variable	log_output
Scope	Global
Dynamic	Yes
Type	Set
Default Value	FILE
Valid Values	TABLE FILE

	NONE

The destination or destinations for general query log and slow query log output. The value is a list one or more comma-separated words chosen from TABLE, FILE, and NONE. TABLE selects logging to the general_log and slow_log tables in the mysql system database. FILE selects logging to log files. NONE disables logging. If NONE is present in the value, it takes precedence over any other words that are present. TABLE and FILE can both be given to select both log output destinations.

This variable selects log output destinations, but does not enable log output. To do that, enable the general_log and slow_query_log system variables. For FILE logging, the general_log_file and slow_query_log_file system variables determine the log file locations. For more information, see Section 5.4.1, "Selecting General Query Log and Slow Query Log Output Destinations".

- log_queries_not_using_indexes

Command-Line Format	--log-queries-not-using-indexes[={OFF|ON}]
System Variable	log_queries_not_using_indexes
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	OFF

If you enable this variable with the slow query log enabled, queries that are expected to retrieve all rows are logged. See Section 5.4.5, "The Slow Query Log". This option does not necessarily mean that no index is used. For example, a query that uses a full index scan uses an index but would be logged because the index would not limit the number of rows.

- log_slow_admin_statements

Command-Line Format	--log-slow-admin-statements[={OFF|ON}]
Introduced	5.6.11
System Variable	log_slow_admin_statements
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	OFF

Include slow administrative statements in the statements written to the slow query log. Administrative statements include ALTER TABLE, ANALYZE TABLE, CHECK TABLE, CREATE INDEX, DROP INDEX, OPTIMIZE TABLE, and REPAIR TABLE.

This variable was added in MySQL 5.6.11 as a replacement for the --log-slow-admin-statements option. The system variable can be set on the command line or in option files the same way as the option, so there is no need for any changes at server startup, but the system variable also makes it possible to examine or set the value at runtime.

- log_throttle_queries_not_using_indexes

Command-Line Format	--log-throttle-queries-not-using-indexes=#
System Variable	log_throttle_queries_not_using_indexes
Scope	Global
Dynamic	Yes

Type	Integer
Default Value	0

If log_queries_not_using_indexes is enabled, the log_throttle_queries_not_using_indexes variable limits the number of such queries per minute that can be written to the slow query log. A value of 0 (the default) means “no limit”. For more information, see Section 5.4.5, “The Slow Query Log”.

- log_warnings

Command-Line Format	--log-warnings[=#]
System Variable	log_warnings
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	1
Minimum Value	0
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295

Whether to produce additional warning messages to the error log. This variable is enabled by default with a value of 1. To disable it, set it to 0. If the value is greater than 0, the server logs messages about statements that are unsafe for statement-based logging. If the value is greater than 1, the server logs aborted connections and access-denied errors for new connection attempts. See Section B.3.2.10, “Communication Errors and Aborted Connections”.

- long_query_time

Command-Line Format	--long-query-time=#
System Variable	long_query_time
Scope	Global, Session
Dynamic	Yes
Type	Numeric
Default Value	10
Minimum Value	0

If a query takes longer than this many seconds, the server increments the Slow_queries status variable. If the slow query log is enabled, the query is logged to the slow query log file. This value is measured in real time, not CPU time, so a query that is under the threshold on a lightly loaded system might be above the threshold on a heavily loaded one. The minimum and default values of long_query_time are 0 and 10, respectively. The value can be specified to a resolution of microseconds. For logging to a file, times are written including the microseconds part. For logging to tables, only integer times are written; the microseconds part is ignored. See Section 5.4.5, “The Slow Query Log”.

Smaller values of this variable result in more statements being considered long-running, with the result that more space is required for the slow query log. For very small values (less than one second), the log may grow quite large in a small time. Increasing the number of statements considered long-running may also result in false positives for the “excessive Number of Long Running Processes” alert in MySQL Enterprise Monitor. For these reasons, very small values should be used in test environments only, or, in production environments, only for a short period.

- `low_priority_updates`

Command-Line Format	`--low-priority-updates[={OFF|ON}]`
System Variable	`low_priority_updates`
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

If set to `1`, all `INSERT`, `UPDATE`, `DELETE`, and `LOCK TABLE WRITE` statements wait until there is no pending `SELECT` or `LOCK TABLE READ` on the affected table. The same effect can be obtained using `{INSERT | REPLACE | DELETE | UPDATE} LOW_PRIORITY ...` to lower the priority of only one query. This variable affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`). See Section 8.11.2, “Table Locking Issues”.

- `lower_case_file_system`

System Variable	`lower_case_file_system`
Scope	Global
Dynamic	No
Type	Boolean

This variable describes the case sensitivity of file names on the file system where the data directory is located. `OFF` means file names are case-sensitive, `ON` means they are not case-sensitive. This variable is read only because it reflects a file system attribute and setting it would have no effect on the file system.

- `lower_case_table_names`

Command-Line Format	`--lower-case-table-names[=#]`
System Variable	`lower_case_table_names`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`2`

If set to 0, table names are stored as specified and comparisons are case-sensitive. If set to 1, table names are stored in lowercase on disk and comparisons are not case-sensitive. If set to 2, table names are stored as given but compared in lowercase. This option also applies to database names and table aliases. For additional information, see Section 9.2.3, “Identifier Case Sensitivity”.

On Windows the default value is 1. On macOS, the default value is 2. On Linux, a value of 2 is not supported; the server forces the value to 0 instead.

You should *not* set `lower_case_table_names` to 0 if you are running MySQL on a system where the data directory resides on a case-insensitive file system (such as on Windows or macOS). It is an unsupported combination that could result in a hang condition when running an `INSERT INTO ...`

`SELECT ... FROM tbl_name` operation with the wrong `tbl_name` letter case. With `MyISAM`, accessing table names using different letter cases could cause index corruption.

As of MySQL 5.6.27, an error message is printed and the server exits if you attempt to start the server with `--lower_case_table_names=0` on a case-insensitive file system.

If you are using `InnoDB` tables, you should set this variable to 1 on all platforms to force names to be converted to lowercase.

The setting of this variable in MySQL 5.6 affects the behavior of replication filtering options with regard to case sensitivity. This is a change from previous versions of MySQL. (Bug #51639) See Section 17.2.3, "How Servers Evaluate Replication Filtering Rules", for more information.

- `max_allowed_packet`

Command-Line Format	`--max-allowed-packet=#`
System Variable	`max_allowed_packet`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`4194304`
Minimum Value	`1024`
Maximum Value	`1073741824`

The maximum size of one packet or any generated/intermediate string, or any parameter sent by the `mysql_stmt_send_long_data()` C API function. The default is 4MB.

The packet message buffer is initialized to `net_buffer_length` bytes, but can grow up to `max_allowed_packet` bytes when needed. This value by default is small, to catch large (possibly incorrect) packets.

You must increase this value if you are using large `BLOB` columns or long strings. It should be as big as the largest `BLOB` you want to use. The protocol limit for `max_allowed_packet` is 1GB. The value should be a multiple of 1024; nonmultiples are rounded down to the nearest multiple.

When you change the message buffer size by changing the value of the `max_allowed_packet` variable, you should also change the buffer size on the client side if your client program permits it. The default `max_allowed_packet` value built in to the client library is 1GB, but individual client programs might override this. For example, `mysql` and `mysqldump` have defaults of 16MB and 24MB, respectively. They also enable you to change the client-side value by setting `max_allowed_packet` on the command line or in an option file.

The session value of this variable is read only. The client can receive up to as many bytes as the session value. However, the server does not send to the client more bytes than the current global `max_allowed_packet` value. (The global value could be less than the session value if the global value is changed after the client connects.)

- `max_connect_errors`

Command-Line Format	`--max-connect-errors=#`
System Variable	`max_connect_errors`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`100`

Minimum Value	1
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295

After max_connect_errors successive connection requests from a host are interrupted without a successful connection, the server blocks that host from further connections. If a connection from a host is established successfully within fewer than max_connect_errors attempts after a previous connection was interrupted, the error count for the host is cleared to zero. To unblock blocked hosts, flush the host cache; see Flushing the Host Cache.

- max_connections

Command-Line Format	--max-connections=#
System Variable	max_connections
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	151
Minimum Value	1
Maximum Value	100000

The maximum permitted number of simultaneous client connections. For more information, see Section 5.1.11.1, “Connection Interfaces”.

- max_delayed_threads

Command-Line Format	--max-delayed-threads=#
Deprecated	Yes
System Variable	max_delayed_threads
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	20
Minimum Value	0
Maximum Value	16384

Do not start more than this number of threads to handle INSERT DELAYED statements for nontransactional tables. If you try to insert data into a new table after all INSERT DELAYED threads are in use, the row is inserted as if the DELAYED attribute was not specified. If you set this to 0, MySQL never creates a thread to handle DELAYED rows; in effect, this disables DELAYED entirely.

For the SESSION value of this variable, the only valid values are 0 or the GLOBAL value.

This system variable is deprecated (because DELAYED inserts are deprecated); expect it to be removed in a future release.

- max_digest_length

Command-Line Format	--max-digest-length=#
Introduced	5.6.24

System Variable	max_digest_length
Scope	Global
Dynamic	No
Type	Integer
Default Value	1024
Minimum Value	0
Maximum Value	1048576

The maximum number of bytes of memory reserved per session for computation of normalized statement digests. Once that amount of space is used during digest computation, truncation occurs: no further tokens from a parsed statement are collected or figure into its digest value. Statements that differ only after that many bytes of parsed tokens produce the same normalized statement digest and are considered identical if compared or if aggregated for digest statistics.

Decreasing the max_digest_length value reduces memory use but causes the digest value of more statements to become indistinguishable if they differ only at the end. Increasing the value permits longer statements to be distinguished but increases memory use, particularly for workloads that involve large numbers of simultaneous sessions (the server allocates max_digest_length bytes per session).

The parser uses this system variable as a limit on the maximum length of normalized statement digests that it computes. The Performance Schema, if it tracks statement digests, makes a copy of the digest value, using the performance_schema_max_digest_length. system variable as a limit on the maximum length of digests that it stores. Consequently, if performance_schema_max_digest_length is less than max_digest_length, digest values stored in the Performance Schema are truncated relative to the original digest values.

In MySQL 5.6.24 and 5.6.25, performance_schema_max_digest_length is not available and max_digest_length applies to all digest computation. Before MySQL 5.6.24, neither max_digest_length nor performance_schema_max_digest_length are available and a fixed maximum of 1024 bytes applies to all digest computation.

For more information about statement digesting, see Section 22.10, “Performance Schema Statement Digests”.

- max_error_count

Command-Line Format	--max-error-count=#
System Variable	max_error_count
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	64
Minimum Value	0
Maximum Value	65535

The maximum number of error, warning, and information messages to be stored for display by the SHOW ERRORS and SHOW WARNINGS statements. This is the same as the number of condition areas in the diagnostics area, and thus the number of conditions that can be inspected by GET DIAGNOSTICS.

- max_heap_table_size

Command-Line Format	--max-heap-table-size=#

System Variable	max_heap_table_size
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	16777216
Minimum Value	16384
Maximum Value (64-bit platforms)	1844674407370954752
Maximum Value (32-bit platforms)	4294967295

This variable sets the maximum size to which user-created MEMORY tables are permitted to grow. The value of the variable is used to calculate MEMORY table MAX_ROWS values. Setting this variable has no effect on any existing MEMORY table, unless the table is re-created with a statement such as CREATE TABLE or altered with ALTER TABLE or TRUNCATE TABLE. A server restart also sets the maximum size of existing MEMORY tables to the global max_heap_table_size value.

This variable is also used in conjunction with tmp_table_size to limit the size of internal in-memory tables. See Section 8.4.4, “Internal Temporary Table Use in MySQL”.

max_heap_table_size is not replicated. See Section 17.4.1.20, “Replication and MEMORY Tables”, and Section 17.4.1.35, “Replication and Variables”, for more information.

- max_insert_delayed_threads

Deprecated	Yes
System Variable	max_insert_delayed_threads
Scope	Global, Session
Dynamic	Yes
Type	Integer

This variable is a synonym for max_delayed_threads.

This system variable is deprecated (because DELAYED inserts are deprecated); expect it to be removed in a future release.

- max_join_size

Command-Line Format	--max-join-size=#
System Variable	max_join_size
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	18446744073709551615
Minimum Value	1
Maximum Value	18446744073709551615

Do not permit statements that probably need to examine more than max_join_size rows (for single-table statements) or row combinations (for multiple-table statements) or that are likely to do more than max_join_size disk seeks. By setting this value, you can catch statements where keys are not used properly and that would probably take a long time. Set it if your users tend to

perform joins that lack a `WHERE` clause, that take a long time, or that return millions of rows. For more information, see Using Safe-Updates Mode (--safe-updates).

Setting this variable to a value other than `DEFAULT` resets the value of `sql_big_selects` to `0`. If you set the `sql_big_selects` value again, the `max_join_size` variable is ignored.

If a query result is in the query cache, no result size check is performed, because the result has previously been computed and it does not burden the server to send it to the client.

- `max_length_for_sort_data`

Command-Line Format	`--max-length-for-sort-data=#`
System Variable	`max_length_for_sort_data`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`1024`
Minimum Value	`4`
Maximum Value	`8388608`

The cutoff on the size of index values that determines which `filesort` algorithm to use. See Section 8.2.1.13, “ORDER BY Optimization”.

- `max_prepared_stmt_count`

Command-Line Format	`--max-prepared-stmt-count=#`
System Variable	`max_prepared_stmt_count`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`16382`
Minimum Value	`0`
Maximum Value	`1048576`

This variable limits the total number of prepared statements in the server. (The sum of the number of prepared statements across all sessions.) It can be used in environments where there is the potential for denial-of-service attacks based on running the server out of memory by preparing huge numbers of statements. If the value is set lower than the current number of prepared statements, existing statements are not affected and can be used, but no new statements can be prepared until the current number drops below the limit. Setting the value to 0 disables prepared statements.

- `max_seeks_for_key`

Command-Line Format	`--max-seeks-for-key=#`
System Variable	`max_seeks_for_key`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value (Other, 64-bit platforms)	`18446744073709551615`

Default Value (Other, 32-bit platforms)	4294967295
Default Value (Windows)	4294967295
Minimum Value	1
Maximum Value (Other, 64-bit platforms)	18446744073709551615
Maximum Value (Other, 32-bit platforms)	4294967295
Maximum Value (Windows)	4294967295

Limit the assumed maximum number of seeks when looking up rows based on a key. The MySQL optimizer assumes that no more than this number of key seeks are required when searching for matching rows in a table by scanning an index, regardless of the actual cardinality of the index (see Section 13.7.5.23, “SHOW INDEX Statement”). By setting this to a low value (say, 100), you can force MySQL to prefer indexes instead of table scans.

- max_sort_length

Command-Line Format	--max-sort-length=#
System Variable	max_sort_length
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	1024
Minimum Value	4
Maximum Value	8388608

The number of bytes to use when sorting data values. The server uses only the first max_sort_length bytes of each value and ignores the rest. Consequently, values that differ only after the first max_sort_length bytes compare as equal for GROUP BY, ORDER BY, and DISTINCT operations.

Increasing the value of max_sort_length may require increasing the value of sort_buffer_size as well. For details, see Section 8.2.1.13, “ORDER BY Optimization”

- max_sp_recursion_depth

Command-Line Format	--max-sp-recursion-depth[=#]
System Variable	max_sp_recursion_depth
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	0
Maximum Value	255

The number of times that any given stored procedure may be called recursively. The default value for this option is 0, which completely disables recursion in stored procedures. The maximum value is 255.

Stored procedure recursion increases the demand on thread stack space. If you increase the value of max_sp_recursion_depth, it may be necessary to increase thread stack size by increasing the value of thread_stack at server startup.

- `max_tmp_tables`

  This variable is unused. It is deprecated; expect it to be removed in a future MySQL release.

- `max_user_connections`

Command-Line Format	`--max-user-connections=#`
System Variable	`max_user_connections`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`4294967295`

  The maximum number of simultaneous connections permitted to any given MySQL user account. A value of 0 (the default) means “no limit.”

  This variable has a global value that can be set at server startup or runtime. It also has a read-only session value that indicates the effective simultaneous-connection limit that applies to the account associated with the current session. The session value is initialized as follows:

  - If the user account has a nonzero `MAX_USER_CONNECTIONS` resource limit, the session `max_user_connections` value is set to that limit.
  - Otherwise, the session `max_user_connections` value is set to the global value.

  Account resource limits are specified using the `GRANT` statement. See Section 6.2.13, “Setting Account Resource Limits”, and Section 13.7.1.4, “GRANT Statement”.

- `max_write_lock_count`

Command-Line Format	`--max-write-lock-count=#`
System Variable	`max_write_lock_count`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value (Other, 64-bit platforms)	`18446744073709551615`
Default Value (Other, 32-bit platforms)	`4294967295`
Default Value (Windows)	`4294967295`
Minimum Value	`1`
Maximum Value (Other, 64-bit platforms)	`18446744073709551615`
Maximum Value (Other, 32-bit platforms)	`4294967295`
Maximum Value (Windows)	`4294967295`

  After this many write locks, permit some pending read lock requests to be processed in between. Write lock requests have higher priority than read lock requests. However, if `max_write_lock_count` is set to some low value (say, 10), read lock requests may be preferred

over pending write lock requests if the read lock requests have already been passed over in favor of 10 write lock requests. Normally this behavior does not occur because `max_write_lock_count` by default has a very large value.

- `metadata_locks_cache_size`

Command-Line Format	`--metadata-locks-cache-size=#`
System Variable	`metadata_locks_cache_size`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`1024`
Minimum Value	`1`
Maximum Value	`1048576`

The size of the metadata locks cache. The server uses this cache to avoid creation and destruction of synchronization objects. This is particularly helpful on systems where such operations are expensive, such as Windows XP.

- `metadata_locks_hash_instances`

Command-Line Format	`--metadata-locks-hash-instances=#`
System Variable	`metadata_locks_hash_instances`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`8`
Minimum Value	`1`
Maximum Value	`1024`

The set of metadata locks can be partitioned into separate hashes to permit connections accessing different objects to use different locking hashes and reduce contention. The `metadata_locks_hash_instances` system variable specifies the number of hashes (default 8).

- `min_examined_row_limit`

Command-Line Format	`--min-examined-row-limit=#`
System Variable	`min_examined_row_limit`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

Queries that examine fewer than this number of rows are not logged to the slow query log.

- `multi_range_count`

Command-Line Format	`--multi-range-count=#`
Deprecated	Yes
System Variable	`multi_range_count`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`256`
Minimum Value	`1`
Maximum Value	`4294967295`

This variable has no effect. It is deprecated; expect it to be removed in a future MySQL release.

- `myisam_data_pointer_size`

Command-Line Format	`--myisam-data-pointer-size=#`
System Variable	`myisam_data_pointer_size`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`6`
Minimum Value	`2`
Maximum Value	`7`

The default pointer size in bytes, to be used by `CREATE TABLE` for `MyISAM` tables when no `MAX_ROWS` option is specified. This variable cannot be less than 2 or larger than 7. The default value is 6. See Section B.3.2.11, “The table is full”.

- `myisam_max_sort_file_size`

Command-Line Format	`--myisam-max-sort-file-size=#`
System Variable	`myisam_max_sort_file_size`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value (Other, 64-bit platforms)	`9223372036853727232`
Default Value (Other, 32-bit platforms)	`2147483648`
Default Value (Windows)	`2146435072`
Maximum Value (Other, 64-bit platforms)	`9223372036853727232`
Maximum Value (Other, 32-bit platforms)	`2147483648`
Maximum Value (Windows)	`2146435072`

The maximum size of the temporary file that MySQL is permitted to use while re-creating a `MyISAM` index (during `REPAIR TABLE`, `ALTER TABLE`, or `LOAD DATA`). If the file size would be larger than

this value, the index is created using the key cache instead, which is slower. The value is given in bytes.

If MyISAM index files exceed this size and disk space is available, increasing the value may help performance. The space must be available in the file system containing the directory where the original index file is located.

- myisam_mmap_size

Command-Line Format	--myisam-mmap-size=#
System Variable	myisam_mmap_size
Scope	Global
Dynamic	No
Type	Integer
Default Value (64-bit platforms)	18446744073709551615
Default Value (32-bit platforms)	4294967295
Minimum Value	7
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295

The maximum amount of memory to use for memory mapping compressed MyISAM files. If many compressed MyISAM tables are used, the value can be decreased to reduce the likelihood of memory-swapping problems.

- myisam_recover_options

Command-Line Format	--myisam-recover-options[=list]
System Variable	myisam_recover_options
Scope	Global
Dynamic	No
Type	Enumeration
Default Value	OFF
Valid Values	OFF DEFAULT BACKUP FORCE QUICK

Set the MyISAM storage engine recovery mode. The variable value is any combination of the values of OFF, DEFAULT, BACKUP, FORCE, or QUICK. If you specify multiple values, separate them by commas. Specifying the variable with no value at server startup is the same as specifying DEFAULT, and specifying with an explicit value of "" disables recovery (same as a value of OFF). If recovery is enabled, each time mysqld opens a MyISAM table, it checks whether the table is marked as crashed or was not closed properly. (The last option works only if you are running with external

locking disabled.) If this is the case, `mysqld` runs a check on the table. If the table was corrupted, `mysqld` attempts to repair it.

The following options affect how the repair works.

Option	Description
`OFF`	No recovery.
`DEFAULT`	Recovery without backup, forcing, or quick checking.
`BACKUP`	If the data file was changed during recovery, save a backup of the *`tbl_name`*`.MYD` file as *`tbl_name-datetime`*`.BAK`.
`FORCE`	Run recovery even if we would lose more than one row from the `.MYD` file.
`QUICK`	Do not check the rows in the table if there are not any delete blocks.

Before the server automatically repairs a table, it writes a note about the repair to the error log. If you want to be able to recover from most problems without user intervention, you should use the options `BACKUP,FORCE`. This forces a repair of a table even if some rows would be deleted, but it keeps the old data file as a backup so that you can later examine what happened.

See Section 15.2.1, “MyISAM Startup Options”.

- `myisam_repair_threads`

Command-Line Format	`--myisam-repair-threads=#`
System Variable	`myisam_repair_threads`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`1`
Minimum Value	`1`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

If this value is greater than 1, `MyISAM` table indexes are created in parallel (each index in its own thread) during the `Repair by sorting` process. The default value is 1.

Note

Multithreaded repair is still *beta-quality* code.

- `myisam_sort_buffer_size`

Command-Line Format	`--myisam-sort-buffer-size=#`
System Variable	`myisam_sort_buffer_size`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`8388608`
Minimum Value	`4096`

Maximum Value (Other, 64-bit platforms)	18446744073709551615
Maximum Value (Other, 32-bit platforms)	4294967295
Maximum Value (Windows, 64-bit platforms)	18446744073709551615
Maximum Value (Windows, 32-bit platforms)	4294967295

The size of the buffer that is allocated when sorting MyISAM indexes during a REPAIR TABLE or when creating indexes with CREATE INDEX or ALTER TABLE.

- myisam_stats_method

Command-Line Format	--myisam-stats-method=name
System Variable	myisam_stats_method
Scope	Global, Session
Dynamic	Yes
Type	Enumeration
Default Value	nulls_unequal
Valid Values	nulls_equal nulls_unequal nulls_ignored

How the server treats NULL values when collecting statistics about the distribution of index values for MyISAM tables. This variable has three possible values, nulls_equal, nulls_unequal, and nulls_ignored. For nulls_equal, all NULL index values are considered equal and form a single value group that has a size equal to the number of NULL values. For nulls_unequal, NULL values are considered unequal, and each NULL forms a distinct value group of size 1. For nulls_ignored, NULL values are ignored.

The method that is used for generating table statistics influences how the optimizer chooses indexes for query execution, as described in Section 8.3.7, “InnoDB and MyISAM Index Statistics Collection”.

- myisam_use_mmap

Command-Line Format	--myisam-use-mmap[={OFF|ON}]
System Variable	myisam_use_mmap
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	OFF

Use memory mapping for reading and writing MyISAM tables.

- named_pipe

Command-Line Format	--named-pipe[={OFF|ON}]
System Variable	named_pipe
Scope	Global

Dynamic	No
Platform Specific	Windows
Type	Boolean
Default Value	`OFF`

(Windows only.) Indicates whether the server supports connections over named pipes.

- `named_pipe_full_access_group`

Command-Line Format	`--named-pipe-full-access-group=value`
Introduced	5.6.43
System Variable	`named_pipe_full_access_group`
Scope	Global
Dynamic	No
Platform Specific	Windows
Type	String
Default Value	`*everyone*`
Valid Values	`*everyone*` `empty string`

(Windows only.) The access control granted to clients on the named pipe created by the MySQL server is set to the minimum necessary for successful communication when the `named_pipe` system variable is enabled to support named-pipe connections. Newer MySQL client software can open named pipe connections without any additional configuration, however, older client software may still require full access to open a named pipe connection.

This variable sets the name of a Windows local group whose members are granted sufficient access by the MySQL server to use older named-pipe clients. Initially, the value is set to `'*everyone*'` by default, which permits users of the Everyone group on Windows to continue using older clients until the older clients are upgraded. In contrast, setting the value to an empty string means that no Windows user is granted full access to the named pipe. The default value `'*everyone*'` provides a language-independent way of referring to the Everyone group on Windows.

Ideally, a new Windows local group name (for example, `mysql_old_client_users`) should be created in Windows and then used to replace the default value for this variable when access to older client software is absolutely necessary. In this case, limit the membership of the group to as few users as possible, removing users from the group when their client software is upgraded. A non-member of the group who attempts to open a connection to MySQL with the older named-pipe client is denied access until the user is added to the group by a Windows administrator, and then the user logs out and logs in (required by Windows).

- `net_buffer_length`

Command-Line Format	`--net-buffer-length=#`
System Variable	`net_buffer_length`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`16384`
Minimum Value	`1024`
Maximum Value	`1048576`

Each client thread is associated with a connection buffer and result buffer. Both begin with a size given by `net_buffer_length` but are dynamically enlarged up to `max_allowed_packet` bytes as needed. The result buffer shrinks to `net_buffer_length` after each SQL statement.

This variable should not normally be changed, but if you have very little memory, you can set it to the expected length of statements sent by clients. If statements exceed this length, the connection buffer is automatically enlarged. The maximum value to which `net_buffer_length` can be set is 1MB.

The session value of this variable is read only.

- `net_read_timeout`

Command-Line Format	`--net-read-timeout=#`
System Variable	`net_read_timeout`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`30`
Minimum Value	`1`

The number of seconds to wait for more data from a connection before aborting the read. When the server is reading from the client, `net_read_timeout` is the timeout value controlling when to abort. When the server is writing to the client, `net_write_timeout` is the timeout value controlling when to abort. See also `slave_net_timeout`.

- `net_retry_count`

Command-Line Format	`--net-retry-count=#`
System Variable	`net_retry_count`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`10`
Minimum Value	`1`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

If a read or write on a communication port is interrupted, retry this many times before giving up. This value should be set quite high on FreeBSD because internal interrupts are sent to all threads.

- `net_write_timeout`

Command-Line Format	`--net-write-timeout=#`
System Variable	`net_write_timeout`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`60`

Minimum Value	1

The number of seconds to wait for a block to be written to a connection before aborting the write. See also net_read_timeout.

- new

Command-Line Format	--new[={OFF|ON}]
System Variable	new
Scope	Global, Session
Dynamic	Yes
Disabled by	skip-new
Type	Boolean
Default Value	OFF

This variable was used in MySQL 4.0 to turn on some 4.1 behaviors, and is retained for backward compatibility. Its value is always OFF.

In NDB Cluster, setting this variable to ON makes it possible to employ partitioning types other than KEY or LINEAR KEY with NDB tables. *This feature is experimental only, and not supported in production*. For additional information, see User-defined partitioning and the NDB storage engine (NDB Cluster).

- old

Command-Line Format	--old[={OFF|ON}]
System Variable	old
Scope	Global
Dynamic	No
Type	Boolean
Default Value	OFF

old is a compatibility variable. It is disabled by default, but can be enabled at startup to revert the server to behaviors present in older versions.

When old is enabled, it changes the default scope of index hints to that used prior to MySQL 5.1.17. That is, index hints with no FOR clause apply only to how indexes are used for row retrieval and not to resolution of ORDER BY or GROUP BY clauses. (See Section 8.9.3, “Index Hints”.) Take care about enabling this in a replication setup. With statement-based binary logging, having different modes for the source and replicas might lead to replication errors.

- old_alter_table

Command-Line Format	--old-alter-table[={OFF|ON}]
System Variable	old_alter_table
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	OFF

When this variable is enabled, the server does not use the optimized method of processing an ALTER TABLE operation. It reverts to using a temporary table, copying over the data, and

then renaming the temporary table to the original, as used by MySQL 5.0 and earlier. For more information on the operation of `ALTER TABLE`, see Section 13.1.7, “ALTER TABLE Statement”.

- `old_passwords`

Command-Line Format	`--old-passwords=value`
System Variable	`old_passwords`
Scope	Global, Session
Dynamic	Yes
Type	Enumeration
Default Value	`0`

Valid Values	`0`
	`1`
	`2`

This variable controls the password hashing method used by the `PASSWORD()` function. It also influences password hashing performed by `CREATE USER` and `GRANT` statements that specify a password using an `IDENTIFIED BY` clause.

The following table shows, for each password hashing method, the permitted value of `old_passwords` and which authentication plugins use the hashing method.

Password Hashing Method	old_passwords Value	Associated Authentication Plugin
MySQL 4.1 native hashing	0	`mysql_native_password`
Pre-4.1 (“old”) hashing	1	`mysql_old_password`
SHA-256 hashing	2	`sha256_password`

Note

Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated; expect support for them to be removed in a future MySQL release. Consequently, `old_passwords=1`, which causes `PASSWORD()` to generate pre-4.1 password hashes, is also deprecated. For account upgrade instructions, see Section 6.4.1.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”.

If `old_passwords=1`, `PASSWORD(str)` returns the same value as `OLD_PASSWORD(str)`. The latter function is not affected by the value of `old_passwords`.

If you set `old_passwords=2`, follow the instructions for using the `sha256_password` plugin at Section 6.4.1.4, “SHA-256 Pluggable Authentication”.

The server sets the global `old_passwords` value during startup to be consistent with the password hashing method required by the default authentication plugin. The default plugin is

`mysql_native_password` unless the `--default-authentication-plugin` option is set otherwise.

When a client successfully connects to the server, the server sets the session `old_passwords` value appropriately for the account authentication method. For example, if the account uses the `sha256_password` authentication plugin, the server sets `old_passwords=2`.

For additional information about authentication plugins and hashing formats, see Section 6.2.11, "Pluggable Authentication", and Section 6.1.2.4, "Password Hashing in MySQL".

- `open_files_limit`

Command-Line Format	`--open-files-limit=#`
System Variable	`open_files_limit`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`5000, with possible adjustment`
Minimum Value	`0`
Maximum Value	`platform dependent`

The number of file descriptors available to `mysqld` from the operating system:

- At startup, `mysqld` reserves descriptors with `setrlimit()`, using the value requested at by setting this variable directly or by using the `--open-files-limit` option to `mysqld_safe`. If `mysqld` produces the error `Too many open files`, try increasing the `open_files_limit` value. Internally, the maximum value for this variable is the maximum unsigned integer value, but the actual maximum is platform dependent.
- At runtime, the value of `open_files_limit` indicates the number of file descriptors actually permitted to `mysqld` by the operating system, which might differ from the value requested at startup. If the number of file descriptors requested during startup cannot be allocated, `mysqld` writes a warning to the error log.

The effective `open_files_limit` value is based on the value specified at system startup (if any) and the values of `max_connections` and `table_open_cache`, using these formulas:

- `10 + max_connections + (table_open_cache * 2)`
- `max_connections * 5`
- The operating system limit if that limit is positive but not Infinity.
- If the operating system limit is Infinity: `open_files_limit` value if specified at startup, 5000 if not.

The server attempts to obtain the number of file descriptors using the maximum of those values. If that many descriptors cannot be obtained, the server attempts to obtain as many as the system permits.

The effective value is 0 on systems where MySQL cannot change the number of open files.

On Unix, the value cannot be set greater than the value displayed by the `ulimit -n` command.

- `optimizer_prune_level`

Command-Line Format	`--optimizer-prune-level=#`

System Variable	optimizer_prune_level
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	1
Minimum Value	0
Maximum Value	1

Controls the heuristics applied during query optimization to prune less-promising partial plans from the optimizer search space. A value of 0 disables heuristics so that the optimizer performs an exhaustive search. A value of 1 causes the optimizer to prune plans based on the number of rows retrieved by intermediate plans.

- optimizer_search_depth

Command-Line Format	--optimizer-search-depth=#
System Variable	optimizer_search_depth
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	62
Minimum Value	0
Maximum Value	62

The maximum depth of search performed by the query optimizer. Values larger than the number of relations in a query result in better query plans, but take longer to generate an execution plan for a query. Values smaller than the number of relations in a query return an execution plan quicker, but the resulting plan may be far from being optimal. If set to 0, the system automatically picks a reasonable value.

- optimizer_switch

Command-Line Format	--optimizer-switch=value
System Variable	optimizer_switch
Scope	Global, Session
Dynamic	Yes
Type	Set
Valid Values	batched_key_access={on|off} block_nested_loop={on|off} engine_condition_pushdown={on|off} firstmatch={on|off} index_condition_pushdown={on|off} index_merge={on|off} index_merge_intersection={on|off} index_merge_sort_union={on|off}

	index_merge_union={on|off} loosescan={on|off} materialization={on|off} mrr={on|off} mrr_cost_based={on|off} semijoin={on|off} subquery_materialization_cost_based={on|off} use_index_extensions={on|off}

The optimizer_switch system variable enables control over optimizer behavior. The value of this variable is a set of flags, each of which has a value of on or off to indicate whether the corresponding optimizer behavior is enabled or disabled. This variable has global and session values and can be changed at runtime. The global default can be set at server startup.

To see the current set of optimizer flags, select the variable value:

```
mysql> SELECT @@optimizer_switch\G
*************************** 1. row ***************************
@@optimizer_switch: index_merge=on,index_merge_union=on,
                    index_merge_sort_union=on,
                    index_merge_intersection=on,
                    engine_condition_pushdown=on,
                    index_condition_pushdown=on,
                    mrr=on,mrr_cost_based=on,
                    block_nested_loop=on,batched_key_access=off,
                    materialization=on,semijoin=on,loosescan=on,
                    firstmatch=on,
                    subquery_materialization_cost_based=on,
                    use_index_extensions=on
```

For more information about the syntax of this variable and the optimizer behaviors that it controls, see Section 8.9.2, “Switchable Optimizations”.

- optimizer_trace

Command-Line Format	--optimizer-trace=value
System Variable	optimizer_trace
Scope	Global, Session
Dynamic	Yes
Type	String

This variable controls optimizer tracing. For details, see MySQL Internals: Tracing the Optimizer.

- optimizer_trace_features

Command-Line Format	--optimizer-trace-features=value
System Variable	optimizer_trace_features
Scope	Global, Session
Dynamic	Yes

Type	String

This variable enables or disables selected optimizer tracing features. For details, see MySQL Internals: Tracing the Optimizer.

- `optimizer_trace_limit`

Command-Line Format	`--optimizer-trace-limit=#`
System Variable	`optimizer_trace_limit`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`1`

The maximum number of optimizer traces to display. For details, see MySQL Internals: Tracing the Optimizer.

- `optimizer_trace_max_mem_size`

Command-Line Format	`--optimizer-trace-max-mem-size=#`
System Variable	`optimizer_trace_max_mem_size`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`16384`

The maximum cumulative size of stored optimizer traces. For details, see MySQL Internals: Tracing the Optimizer.

- `optimizer_trace_offset`

Command-Line Format	`--optimizer-trace-offset=#`
System Variable	`optimizer_trace_offset`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`-1`

The offset of optimizer traces to display. For details, see MySQL Internals: Tracing the Optimizer.

- `performance_schema_xxx`

Performance Schema system variables are listed in Section 22.15, “Performance Schema System Variables”. These variables may be used to configure Performance Schema operation.

- `pid_file`

Command-Line Format	`--pid-file=file_name`
System Variable	`pid_file`
Scope	Global
Dynamic	No
Type	File name

The path name of the file in which the server writes its process ID. The server creates the file in the data directory unless an absolute path name is given to specify a different directory. If you specify this variable, you must specify a value. If you do not specify this variable, MySQL uses a default value of *`host_name`*`.pid`, where *`host_name`* is the name of the host machine.

The process ID file is used by other programs such as `mysqld_safe` to determine the server's process ID. On Windows, this variable also affects the default error log file name. See Section 5.4.2, “The Error Log”.

- `plugin_dir`

Command-Line Format	`--plugin-dir=dir_name`
System Variable	`plugin_dir`
Scope	Global
Dynamic	No
Type	Directory name
Default Value	`BASEDIR/lib/plugin`

The path name of the plugin directory.

If the plugin directory is writable by the server, it may be possible for a user to write executable code to a file in the directory using `SELECT ... INTO DUMPFILE`. This can be prevented by making `plugin_dir` read only to the server or by setting `secure_file_priv` to a directory where `SELECT` writes can be made safely.

- `port`

Command-Line Format	`--port=port_num`
System Variable	`port`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`3306`
Minimum Value	`0`
Maximum Value	`65535`

The number of the port on which the server listens for TCP/IP connections. This variable can be set with the `--port` option.

- `preload_buffer_size`

Command-Line Format	`--preload-buffer-size=#`
System Variable	`preload_buffer_size`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`32768`
Minimum Value	`1024`
Maximum Value	`1073741824`

The size of the buffer that is allocated when preloading indexes.

- `profiling`

  If set to 0 or `OFF` (the default), statement profiling is disabled. If set to 1 or `ON`, statement profiling is enabled and the `SHOW PROFILE` and `SHOW PROFILES` statements provide access to profiling information. See Section 13.7.5.32, “SHOW PROFILES Statement”.

  This variable is deprecated; expect it to be removed in a future MySQL release.

- `profiling_history_size`

  The number of statements for which to maintain profiling information if `profiling` is enabled. The default value is 15. The maximum value is 100. Setting the value to 0 effectively disables profiling. See Section 13.7.5.32, “SHOW PROFILES Statement”.

  This variable is deprecated; expect it to be removed in a future MySQL release.

- `protocol_version`

System Variable	`protocol_version`
Scope	Global
Dynamic	No
Type	Integer

  The version of the client/server protocol used by the MySQL server.

- `proxy_user`

System Variable	`proxy_user`
Scope	Session
Dynamic	No
Type	String

  If the current client is a proxy for another user, this variable is the proxy user account name. Otherwise, this variable is `NULL`. See Section 6.2.12, “Proxy Users”.

- `pseudo_slave_mode`

System Variable	`pseudo_slave_mode`
Scope	Session
Dynamic	Yes
Type	Boolean

  This variable is for internal server use.

- `pseudo_thread_id`

System Variable	`pseudo_thread_id`
Scope	Session
Dynamic	Yes

Type	Integer

This variable is for internal server use.

Warning

Changing the session value of the `pseudo_thread_id` system variable changes the value returned by the `CONNECTION_ID()` function.

- `query_alloc_block_size`

Command-Line Format	`--query-alloc-block-size=#`
System Variable	`query_alloc_block_size`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`8192`
Minimum Value	`1024`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`
Unit	`bytes`
Block Size	`1024`

The allocation size in bytes of memory blocks that are allocated for objects created during statement parsing and execution. If you have problems with memory fragmentation, it might help to increase this parameter.

- `query_cache_limit`

Command-Line Format	`--query-cache-limit=#`
System Variable	`query_cache_limit`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`1048576`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

Do not cache results that are larger than this number of bytes. The default value is 1MB.

- `query_cache_min_res_unit`

Command-Line Format	`--query-cache-min-res-unit=#`
System Variable	`query_cache_min_res_unit`
Scope	Global
Dynamic	Yes

Type	Integer
Default Value	4096
Minimum Value	512
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295

The minimum size (in bytes) for blocks allocated by the query cache. The default value is 4096 (4KB). Tuning information for this variable is given in Section 8.10.3.3, “Query Cache Configuration”.

- query_cache_size

Command-Line Format	--query-cache-size=#
System Variable	query_cache_size
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	1048576
Minimum Value	0
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295

The amount of memory allocated for caching query results. By default, the query cache is disabled. This is achieved using a default value of 1M, with a default for query_cache_type of 0.

The permissible values are multiples of 1024; other values are rounded down to the nearest multiple. For nonzero values of query_cache_size, that many bytes of memory are allocated even if query_cache_type=0. See Section 8.10.3.3, “Query Cache Configuration”, for more information.

The query cache needs a minimum size of about 40KB to allocate its structures. (The exact size depends on system architecture.) If you set the value of query_cache_size too small, a warning occurs, as described in Section 8.10.3.3, “Query Cache Configuration”.

- query_cache_type

Command-Line Format	--query-cache-type=#
System Variable	query_cache_type
Scope	Global, Session
Dynamic	Yes
Type	Enumeration
Default Value	0
Valid Values	0 1

	2

Set the query cache type. Setting the GLOBAL value sets the type for all clients that connect thereafter. Individual clients can set the SESSION value to affect their own use of the query cache. Possible values are shown in the following table.

Option	Description
0 or OFF	Do not cache results in or retrieve results from the query cache. Note that this does not deallocate the query cache buffer. To do that, you should set query_cache_size to 0.
1 or ON	Cache all cacheable query results except for those that begin with SELECT SQL_NO_CACHE.
2 or DEMAND	Cache results only for cacheable queries that begin with SELECT SQL_CACHE.

This variable defaults to OFF.

If the server is started with query_cache_type set to 0, it does not acquire the query cache mutex at all, which means that the query cache cannot be enabled at runtime and there is reduced overhead in query execution.

- query_cache_wlock_invalidate

Command-Line Format	--query-cache-wlock-invalidate[={OFF|ON}]
System Variable	query_cache_wlock_invalidate
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	OFF

Normally, when one client acquires a WRITE lock on a MyISAM table, other clients are not blocked from issuing statements that read from the table if the query results are present in the query cache. Setting this variable to 1 causes acquisition of a WRITE lock for a table to invalidate any queries in the query cache that refer to the table. This forces other clients that attempt to access the table to wait while the lock is in effect.

- query_prealloc_size

Command-Line Format	--query-prealloc-size=#
System Variable	query_prealloc_size
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	8192
Minimum Value	8192
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295

Block Size	1024

The size in bytes of the persistent buffer used for statement parsing and execution. This buffer is not freed between statements. If you are running complex queries, a larger `query_prealloc_size` value might be helpful in improving performance, because it can reduce the need for the server to perform memory allocation during query execution operations.

- `rand_seed1`

System Variable	rand_seed1
Scope	Session
Dynamic	Yes
Type	Integer

The `rand_seed1` and `rand_seed2` variables exist as session variables only, and can be set but not read. The variables—but not their values—are shown in the output of `SHOW VARIABLES`.

The purpose of these variables is to support replication of the `RAND()` function. For statements that invoke `RAND()`, the source passes two values to the replica, where they are used to seed the random number generator. The replica uses these values to set the session variables `rand_seed1` and `rand_seed2` so that `RAND()` on the replica generates the same value as on the source.

- `rand_seed2`

  See the description for `rand_seed1`.

- `range_alloc_block_size`

Command-Line Format	--range-alloc-block-size=#
System Variable	range_alloc_block_size
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	4096
Minimum Value	4096
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295
Block Size	1024

The size in bytes of blocks that are allocated when doing range optimization.

- `read_buffer_size`

Command-Line Format	--read-buffer-size=#
System Variable	read_buffer_size
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	131072
Minimum Value	8192

Maximum Value	2147479552

Each thread that does a sequential scan for a MyISAM table allocates a buffer of this size (in bytes) for each table it scans. If you do many sequential scans, you might want to increase this value, which defaults to 131072. The value of this variable should be a multiple of 4KB. If it is set to a value that is not a multiple of 4KB, its value is rounded down to the nearest multiple of 4KB.

This option is also used in the following context for all storage engines:

- For caching the indexes in a temporary file (not a temporary table), when sorting rows for ORDER BY.
- For bulk insert into partitions.
- For caching results of nested queries.

read_buffer_size is also used in one other storage engine-specific way: to determine the memory block size for MEMORY tables.

For more information about memory use during different operations, see Section 8.12.4.1, "How MySQL Uses Memory".

- read_only

Command-Line Format	--read-only[={OFF|ON}]
System Variable	read_only
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	OFF

When the read_only system variable is enabled, the server permits no client updates except from users who have the SUPER privilege. This variable is disabled by default.

Even with read_only enabled, the server permits these operations:

- Updates performed by replica threads, if the server is a replica. In replication setups, it can be useful to enable read_only on replica servers to ensure that replicas accept updates only from the source server and not from clients.
- Use of ANALYZE TABLE or OPTIMIZE TABLE statements. The purpose of read-only mode is to prevent changes to table structure or contents. Analysis and optimization do not qualify as

such changes. This means, for example, that consistency checks on read-only replicas can be performed with `mysqlcheck --all-databases --analyze`.

- Operations on `TEMPORARY` tables.
- Inserts into the log tables (`mysql.general_log` and `mysql.slow_log`; see Section 5.4.1, "Selecting General Query Log and Slow Query Log Output Destinations").

`read_only` exists only as a `GLOBAL` variable, so changes to its value require the `SUPER` privilege. Changes to `read_only` on a source server are not replicated to replica servers. The value can be set on a replica server independent of the setting on the source.

The following conditions apply to attempts to enable `read_only`:

- The attempt fails and an error occurs if you have any explicit locks (acquired with `LOCK TABLES`) or have a pending transaction.
- The attempt blocks while other clients have any ongoing statement, active `LOCK TABLES WRITE`, or ongoing commit, until the locks are released and the statements and transactions end. While the attempt to enable `read_only` is pending, requests by other clients for table locks or to begin transactions also block until `read_only` has been set.
- The attempt blocks if there are active transactions that hold metadata locks, until those transactions end.
- `read_only` can be enabled while you hold a global read lock (acquired with `FLUSH TABLES WITH READ LOCK`) because that does not involve table locks.

- `read_rnd_buffer_size`

Command-Line Format	`--read-rnd-buffer-size=#`
System Variable	`read_rnd_buffer_size`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`262144`
Minimum Value	`1`
Maximum Value	`2147483647`

This variable is used for reads from `MyISAM` tables, and, for any storage engine, for Multi-Range Read optimization.

When reading rows from a `MyISAM` table in sorted order following a key-sorting operation, the rows are read through this buffer to avoid disk seeks. See Section 8.2.1.13, "ORDER BY Optimization". Setting the variable to a large value can improve `ORDER BY` performance by a lot. However, this is a buffer allocated for each client, so you should not set the global variable to a large value. Instead, change the session variable only from within those clients that need to run large queries.

For more information about memory use during different operations, see Section 8.12.4.1, "How MySQL Uses Memory". For information about Multi-Range Read optimization, see Section 8.2.1.10, "Multi-Range Read Optimization".

- `secure_auth`

Command-Line Format	`--secure-auth[={OFF|ON}]`
System Variable	`secure_auth`

Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`ON`

If this variable is enabled, the server blocks connections by clients that attempt to use accounts that have passwords stored in the old (pre-4.1) format. Enable this variable to prevent all use of passwords employing the old format (and hence insecure communication over the network). This variable is enabled by default.

Server startup fails with an error if this variable is enabled and the privilege tables are in pre-4.1 format. See Section B.3.2.4, “Client does not support authentication protocol”.

Note

Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated; expect support for them to be removed in a future MySQL release. Consequently, disabling secure authentication by disabling `secure_auth` is also deprecated.

- `secure_file_priv`

Command-Line Format	`--secure-file-priv=dir_name`
System Variable	`secure_file_priv`
Scope	Global
Dynamic	No
Type	String
Default Value (≥ 5.6.34)	`platform specific`
Default Value (≤ 5.6.33)	`empty string`

Valid Values (≥ 5.6.34)	`empty string`
	`dirname`
	`NULL`

Valid Values (≤ 5.6.33)	`empty string` `dirname`

This variable is used to limit the effect of data import and export operations, such as those performed by the `LOAD DATA` and `SELECT ... INTO OUTFILE` statements and the `LOAD_FILE()` function. These operations are permitted only to users who have the `FILE` privilege.

`secure_file_priv` may be set as follows:

- If empty, the variable has no effect. This is not a secure setting.
- If set to the name of a directory, the server limits import and export operations to work only with files in that directory. The directory must exist; the server does not create it.
- If set to `NULL`, the server disables import and export operations. This value is permitted as of MySQL 5.6.34.

Before MySQL 5.6.34, this variable is empty by default. As of 5.6.34, the default value is platform specific and depends on the value of the `INSTALL_LAYOUT` `CMake` option, as shown in the following

table. To specify the default `secure_file_priv` value explicitly if you are building from source, use the `INSTALL_SECURE_FILE_PRIVDIR` `CMake` option.

`INSTALL_LAYOUT` Value	**Default `secure_file_priv` Value**
`STANDALONE`, `WIN`	`NULL`
`DEB`, `RPM`, `SLES`, `SVR4`	`/var/lib/mysql-files`
Otherwise	`mysql-files` under the `CMAKE_INSTALL_PREFIX` value

To set the default `secure_file_priv` value for the `libmysqld` embedded server, use the `INSTALL_SECURE_FILE_PRIV_EMBEDDEDDIR` `CMake` option. The default value for this option is `NULL`.

As of MySQL 5.6.34, the server checks the value of `secure_file_priv` at startup and writes a warning to the error log if the value is insecure. A non-`NULL` value is considered insecure if it is empty, or the value is the data directory or a subdirectory of it, or a directory that is accessible by all users. If `secure_file_priv` is set to a nonexistent path, the server writes an error message to the error log and exits.

- `sha256_password_private_key_path`

Command-Line Format	`--sha256-password-private-key-path=file_name`
System Variable	`sha256_password_private_key_path`
Scope	Global
Dynamic	No
Type	File name
Default Value	`private_key.pem`

This variable is available if MySQL was compiled using OpenSSL (see Section 6.3.4, “SSL Library-Dependent Capabilities”). Its value is the path name of the RSA private key file for the `sha256_password` authentication plugin. If the file is named as a relative path, it is interpreted relative to the server data directory. The file must be in PEM format.

Important

Because this file stores a private key, its access mode should be restricted so that only the MySQL server can read it.

For information about `sha256_password`, see Section 6.4.1.4, “SHA-256 Pluggable Authentication”.

- `sha256_password_public_key_path`

Command-Line Format	`--sha256-password-public-key-path=file_name`
System Variable	`sha256_password_public_key_path`
Scope	Global
Dynamic	No
Type	File name
Default Value	`public_key.pem`

This variable is available if MySQL was compiled using OpenSSL (see Section 6.3.4, “SSL Library-Dependent Capabilities”). Its value is the path name of the RSA public key file for the `sha256_password` authentication plugin. If the file is named as a relative path, it is interpreted relative to the server data directory. The file must be in PEM format. Because this file stores a public key, copies can be freely distributed to client users. (Clients that explicitly specify a public key when

connecting to the server using RSA password encryption must use the same public key as that used by the server.)

For information about `sha256_password`, including information about how clients specify the RSA public key, see Section 6.4.1.4, “SHA-256 Pluggable Authentication”.

- `shared_memory`

Command-Line Format	`--shared-memory[={OFF|ON}]`
System Variable	`shared_memory`
Scope	Global
Dynamic	No
Platform Specific	Windows
Type	Boolean
Default Value	`OFF`

(Windows only.) Whether the server permits shared-memory connections.

- `shared_memory_base_name`

Command-Line Format	`--shared-memory-base-name=name`
System Variable	`shared_memory_base_name`
Scope	Global
Dynamic	No
Platform Specific	Windows
Type	String
Default Value	`MYSQL`

(Windows only.) The name of shared memory to use for shared-memory connections. This is useful when running multiple MySQL instances on a single physical machine. The default name is `MYSQL`. The name is case-sensitive.

This variable applies only if the server is started with the `shared_memory` system variable enabled to support shared-memory connections.

- `show_old_temporals`

Command-Line Format	`--show-old-temporals[={OFF|ON}]`
Introduced	5.6.24
Deprecated	5.6.24
System Variable	`show_old_temporals`
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

Whether `SHOW CREATE TABLE` output includes comments to flag temporal columns found to be in pre-5.6.4 format (`TIME`, `DATETIME`, and `TIMESTAMP` columns without support for fractional seconds precision). This variable is disabled by default. If enabled, `SHOW CREATE TABLE` output looks like this:


```
CREATE TABLE `mytbl` (
```

```
  `ts` timestamp /* 5.5 binary format */ NOT NULL DEFAULT CURRENT_TIMESTAMP,
  `dt` datetime /* 5.5 binary format */ DEFAULT NULL,
  `t` time /* 5.5 binary format */ DEFAULT NULL
) DEFAULT CHARSET=latin1
```

Output for the `COLUMN_TYPE` column of the `INFORMATION_SCHEMA.COLUMNS` table is affected similarly.

This variable was added in MySQL 5.6.24. It is deprecated; expect it to be removed in a future MySQL release.

- `skip_external_locking`

Command-Line Format	`--skip-external-locking[={OFF|ON}]`
System Variable	`skip_external_locking`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`ON`

This is `OFF` if `mysqld` uses external locking (system locking), `ON` if external locking is disabled. This affects only `MyISAM` table access.

This variable is set by the `--external-locking` or `--skip-external-locking` option. External locking is disabled by default.

External locking affects only `MyISAM` table access. For more information, including conditions under which it can and cannot be used, see Section 8.11.5, “External Locking”.

- `skip_name_resolve`

Command-Line Format	`--skip-name-resolve[={OFF|ON}]`
System Variable	`skip_name_resolve`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`OFF`

Whether to resolve host names when checking client connections. If this variable is `OFF`, `mysqld` resolves host names when checking client connections. If it is `ON`, `mysqld` uses only IP numbers; in this case, all `Host` column values in the grant tables must be IP addresses or `localhost`. See Section 5.1.11.2, “DNS Lookups and the Host Cache”.

Depending on the network configuration of your system and the `Host` values for your accounts, clients may need to connect using an explicit `--host` option, such as `--host=localhost`, `--host=127.0.0.1`, or `--host=::1`.

An attempt to connect to the host `127.0.0.1` normally resolves to the `localhost` account. However, this fails if the server is run with `skip_name_resolve` enabled. If you plan to do that, make sure an account exists that can accept a connection. For example, to be able to connect as `root` using `--host=127.0.0.1` or `--host=::1`, create these accounts:

```
CREATE USER 'root'@'127.0.0.1' IDENTIFIED BY 'root-password';
CREATE USER 'root'@'::1' IDENTIFIED BY 'root-password';
```

- `skip_networking`

Command-Line Format	`--skip-networking[={OFF|ON}]`
System Variable	`skip_networking`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`OFF`

This variable controls whether the server permits TCP/IP connections. By default, it is disabled (permit TCP connections). If enabled, the server permits only local (non-TCP/IP) connections and all interaction with `mysqld` must be made using named pipes or shared memory (on Windows) or Unix socket files (on Unix). This option is highly recommended for systems where only local clients are permitted. See Section 5.1.11.2, "DNS Lookups and the Host Cache".

- `skip_show_database`

Command-Line Format	`--skip-show-database`
System Variable	`skip_show_database`
Scope	Global
Dynamic	No

This prevents people from using the `SHOW DATABASES` statement if they do not have the `SHOW DATABASES` privilege. This can improve security if you have concerns about users being able to see databases belonging to other users. Its effect depends on the `SHOW DATABASES` privilege: If the variable value is `ON`, the `SHOW DATABASES` statement is permitted only to users who have the `SHOW DATABASES` privilege, and the statement displays all database names. If the value is `OFF`, `SHOW DATABASES` is permitted to all users, but displays the names of only those databases for which the user has the `SHOW DATABASES` or other privilege.

Caution

Because a global privilege is considered a privilege for all databases, *any* global privilege enables a user to see all database names with `SHOW DATABASES` or by examining the `INFORMATION_SCHEMA SCHEMATA` table.

- `slow_launch_time`

Command-Line Format	`--slow-launch-time=#`
System Variable	`slow_launch_time`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`2`

If creating a thread takes longer than this many seconds, the server increments the `Slow_launch_threads` status variable.

- `slow_query_log`

Command-Line Format	`--slow-query-log[={OFF|ON}]`
System Variable	`slow_query_log`
Scope	Global

Dynamic	Yes
Type	Boolean
Default Value	`OFF`

Whether the slow query log is enabled. The value can be 0 (or `OFF`) to disable the log or 1 (or `ON`) to enable the log. The destination for log output is controlled by the `log_output` system variable; if that value is `NONE`, no log entries are written even if the log is enabled.

"Slow" is determined by the value of the `long_query_time` variable. See Section 5.4.5, "The Slow Query Log".

- `slow_query_log_file`

Command-Line Format	`--slow-query-log-file=file_name`
System Variable	`slow_query_log_file`
Scope	Global
Dynamic	Yes
Type	File name
Default Value	`host_name-slow.log`

The name of the slow query log file. The default value is *`host_name`*`-slow.log`, but the initial value can be changed with the `--slow_query_log_file` option.

- `socket`

Command-Line Format	`--socket={file_name|pipe_name}`
System Variable	`socket`
Scope	Global
Dynamic	No
Type	String
Default Value (Other)	`/tmp/mysql.sock`
Default Value (Windows)	`MySQL`

On Unix platforms, this variable is the name of the socket file that is used for local client connections. The default is `/tmp/mysql.sock`. (For some distribution formats, the directory might be different, such as `/var/lib/mysql` for RPMs.)

On Windows, this variable is the name of the named pipe that is used for local client connections. The default value is `MySQL` (not case-sensitive).

- `sort_buffer_size`

Command-Line Format	`--sort-buffer-size=#`
System Variable	`sort_buffer_size`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`262144`
Minimum Value	`32768`
Maximum Value (Other, 64-bit platforms)	`18446744073709551615`

Maximum Value (Other, 32-bit platforms)	4294967295
Maximum Value (Windows)	4294967295

Each session that must perform a sort allocates a buffer of this size. sort_buffer_size is not specific to any storage engine and applies in a general manner for optimization. At minimum the sort_buffer_size value must be large enough to accommodate fifteen tuples in the sort buffer. Also, increasing the value of max_sort_length may require increasing the value of sort_buffer_size. For more information, see Section 8.2.1.13, "ORDER BY Optimization"

If you see many Sort_merge_passes per second in SHOW GLOBAL STATUS output, you can consider increasing the sort_buffer_size value to speed up ORDER BY or GROUP BY operations that cannot be improved with query optimization or improved indexing.

The optimizer tries to work out how much space is needed but can allocate more, up to the limit. Setting it larger than required globally slows down most queries that sort. It is best to increase it as a session setting, and only for the sessions that need a larger size. On Linux, there are thresholds of 256KB and 2MB where larger values may significantly slow down memory allocation, so you should consider staying below one of those values. Experiment to find the best value for your workload. See Section B.3.3.5, "Where MySQL Stores Temporary Files".

The maximum permissible setting for sort_buffer_size is 4GB−1. Larger values are permitted for 64-bit platforms (except 64-bit Windows, for which large values are truncated to 4GB−1 with a warning).

- sql_auto_is_null

System Variable	sql_auto_is_null
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	OFF

If this variable is enabled, then after a statement that successfully inserts an automatically generated AUTO_INCREMENT value, you can find that value by issuing a statement of the following form:

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

If the statement returns a row, the value returned is the same as if you invoked the LAST_INSERT_ID() function. For details, including the return value after a multiple-row insert, see Section 12.16, "Information Functions". If no AUTO_INCREMENT value was successfully inserted, the SELECT statement returns no row.

The behavior of retrieving an AUTO_INCREMENT value by using an IS NULL comparison is used by some ODBC programs, such as Access. See Obtaining Auto-Increment Values. This behavior can be disabled by setting sql_auto_is_null to OFF.

The default value of sql_auto_is_null is OFF.

- sql_big_selects

System Variable	sql_big_selects
Scope	Global, Session
Dynamic	Yes
Type	Boolean

Default Value	ON

If set to OFF, MySQL aborts SELECT statements that are likely to take a very long time to execute (that is, statements for which the optimizer estimates that the number of examined rows exceeds the value of max_join_size). This is useful when an inadvisable WHERE statement has been issued. The default value for a new connection is ON, which permits all SELECT statements.

If you set the max_join_size system variable to a value other than DEFAULT, sql_big_selects is set to OFF.

- sql_buffer_result

System Variable	sql_buffer_result
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	OFF

If enabled, sql_buffer_result forces results from SELECT statements to be put into temporary tables. This helps MySQL free the table locks early and can be beneficial in cases where it takes a long time to send results to the client. The default value is OFF.

- sql_log_off

System Variable	sql_log_off
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	OFF
Valid Values	OFF (enable logging) ON (disable logging)

This variable controls whether logging to the general query log is disabled for the current session (assuming that the general query log itself is enabled). The default value is OFF (that is, enable logging). To disable or enable general query logging for the current session, set the session sql_log_off variable to ON or OFF.

Setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.8.1, “System Variable Privileges”.

- sql_mode

Command-Line Format	--sql-mode=name
System Variable	sql_mode
Scope	Global, Session
Dynamic	Yes
Type	Set
Default Value	NO_ENGINE_SUBSTITUTION
Valid Values	ALLOW_INVALID_DATES ANSI_QUOTES

	ERROR_FOR_DIVISION_BY_ZERO HIGH_NOT_PRECEDENCE IGNORE_SPACE NO_AUTO_CREATE_USER NO_AUTO_VALUE_ON_ZERO NO_BACKSLASH_ESCAPES NO_DIR_IN_CREATE NO_ENGINE_SUBSTITUTION NO_FIELD_OPTIONS NO_KEY_OPTIONS NO_TABLE_OPTIONS NO_UNSIGNED_SUBTRACTION NO_ZERO_DATE NO_ZERO_IN_DATE ONLY_FULL_GROUP_BY PAD_CHAR_TO_FULL_LENGTH PIPES_AS_CONCAT REAL_AS_FLOAT STRICT_ALL_TABLES STRICT_TRANS_TABLES

The current server SQL mode, which can be set dynamically. The default is NO_ENGINE_SUBSTITUTION. For details, see Section 5.1.10, “Server SQL Modes”.

Note

MySQL installation programs may configure the SQL mode during the installation process. For example, mysql_install_db creates a default option file named my.cnf in the base installation directory. This file contains a line that sets the SQL mode; see Section 4.4.3, “mysql_install_db — Initialize MySQL Data Directory”.

If the SQL mode differs from the default or from what you expect, check for a setting in an option file that the server reads at startup.

- sql_notes

System Variable	sql_notes
Scope	Global, Session
Dynamic	Yes

Type	Boolean
Default Value	`ON`

If enabled (the default), diagnostics of `Note` level increment `warning_count` and the server records them. If disabled, `Note` diagnostics do not increment `warning_count` and the server does not record them. `mysqldump` includes output to disable this variable so that reloading the dump file does not produce warnings for events that do not affect the integrity of the reload operation.

- `sql_quote_show_create`

System Variable	`sql_quote_show_create`
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	`ON`

If enabled (the default), the server quotes identifiers for `SHOW CREATE TABLE` and `SHOW CREATE DATABASE` statements. If disabled, quoting is disabled. This option is enabled by default so that replication works for identifiers that require quoting. See Section 13.7.5.12, “SHOW CREATE TABLE Statement”, and Section 13.7.5.8, “SHOW CREATE DATABASE Statement”.

- `sql_safe_updates`

System Variable	`sql_safe_updates`
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

If this variable is enabled, `UPDATE` and `DELETE` statements that do not use a key in the `WHERE` clause or a `LIMIT` clause produce an error. This makes it possible to catch `UPDATE` and `DELETE` statements where keys are not used properly and that would probably change or delete a large number of rows. The default value is `OFF`.

For the `mysql` client, `sql_safe_updates` can be enabled by using the `--safe-updates` option. For more information, see Using Safe-Updates Mode (--safe-updates).

- `sql_select_limit`

System Variable	`sql_select_limit`
Scope	Global, Session
Dynamic	Yes
Type	Integer

The maximum number of rows to return from `SELECT` statements. For more information, see Using Safe-Updates Mode (--safe-updates).

The default value for a new connection is the maximum number of rows that the server permits per table. Typical default values are $(2^{32})-1$ or $(2^{64})-1$. If you have changed the limit, the default value can be restored by assigning a value of `DEFAULT`.

If a `SELECT` has a `LIMIT` clause, the `LIMIT` takes precedence over the value of `sql_select_limit`.

- `sql_warnings`

System Variable	sql_warnings
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	OFF

This variable controls whether single-row INSERT statements produce an information string if warnings occur. The default is OFF. Set the value to ON to produce an information string.

- ssl_ca

Command-Line Format	--ssl-ca=file_name
System Variable	ssl_ca
Scope	Global
Dynamic	No
Type	File name
Default Value	NULL

The path name of the Certificate Authority (CA) certificate file in PEM format. The file contains a list of trusted SSL Certificate Authorities. Setting this variable implies enabling --ssl.

- ssl_capath

Command-Line Format	--ssl-capath=dir_name
System Variable	ssl_capath
Scope	Global
Dynamic	No
Type	Directory name
Default Value	NULL

The path name of the directory that contains trusted SSL Certificate Authority (CA) certificate files in PEM format. Support for this capability depends on the SSL library used to compile MySQL; see Section 6.3.4, “SSL Library-Dependent Capabilities”. Setting this variable implies enabling --ssl.

- ssl_cert

Command-Line Format	--ssl-cert=file_name
System Variable	ssl_cert
Scope	Global
Dynamic	No
Type	File name
Default Value	NULL

The path name of the server SSL public key certificate file in PEM format. Setting this variable implies enabling --ssl.

- ssl_cipher

Command-Line Format	--ssl-cipher=name
System Variable	ssl_cipher

Scope	Global
Dynamic	No
Type	String
Default Value	NULL

The list of permissible ciphers for connection encryption. If no cipher in the list is supported, encrypted connections do not work. Setting this variable implies enabling `--ssl`.

For greatest portability, the cipher list should be a list of one or more cipher names, separated by colons. This format is understood both by OpenSSL and yaSSL. Examples:

```
[mysqld]
ssl_cipher="AES128-SHA"
ssl_cipher="DHE-RSA-AES256-SHA:AES128-SHA"
```

OpenSSL supports a more flexible syntax for specifying ciphers, as described in the OpenSSL documentation at https://www.openssl.org/docs/manmaster/man1/ciphers.html. yaSSL does not, so attempts to use that extended syntax fail for a MySQL distribution compiled using yaSSL.

For information about which encryption ciphers MySQL supports, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

- `ssl_crl`

Command-Line Format	`--ssl-crl=file_name`
System Variable	`ssl_crl`
Scope	Global
Dynamic	No
Type	File name
Default Value	NULL

The path name of the file containing certificate revocation lists in PEM format. Support for revocation-list capability depends on the SSL library used to compile MySQL. See Section 6.3.4, “SSL Library-Dependent Capabilities”. Setting this variable implies enabling `--ssl`.

- `ssl_crlpath`

Command-Line Format	`--ssl-crlpath=dir_name`
System Variable	`ssl_crlpath`
Scope	Global
Dynamic	No
Type	Directory name
Default Value	NULL

The path of the directory that contains certificate revocation-list files in PEM format. Support for revocation-list capability depends on the SSL library used to compile MySQL. See Section 6.3.4, “SSL Library-Dependent Capabilities”. Setting this variable implies enabling `--ssl`.

- `ssl_key`

Command-Line Format	`--ssl-key=file_name`
System Variable	`ssl_key`
Scope	Global

Dynamic	No
Type	File name
Default Value	`NULL`

The path name of the server SSL private key file in PEM format. For better security, use a certificate with an RSA key size of at least 2048 bits. Setting this variable implies enabling `--ssl`.

If the key file is protected by a passphrase, the server prompts the user for the passphrase. The password must be given interactively; it cannot be stored in a file. If the passphrase is incorrect, the program continues as if it could not read the key.

- `storage_engine`

System Variable	`storage_engine`
Scope	Global, Session
Dynamic	Yes
Type	Enumeration
Default Value	`InnoDB`

The default storage engine (table type). See Chapter 15, *Alternative Storage Engines*.

This variable is deprecated. Use `default_storage_engine` instead.

- `stored_program_cache`

Command-Line Format	`--stored-program-cache=#`
System Variable	`stored_program_cache`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`256`
Minimum Value	`256`
Maximum Value	`524288`

Sets a soft upper limit for the number of cached stored routines per connection. The value of this variable is specified in terms of the number of stored routines held in each of the two caches maintained by the MySQL Server for, respectively, stored procedures and stored functions.

Whenever a stored routine is executed this cache size is checked before the first or top-level statement in the routine is parsed; if the number of routines of the same type (stored procedures or stored functions according to which is being executed) exceeds the limit specified by this variable, the corresponding cache is flushed and memory previously allocated for cached objects is freed. This allows the cache to be flushed safely, even when there are dependencies between stored routines.

- `sync_frm`

Command-Line Format	`--sync-frm[={OFF|ON}]`
System Variable	`sync_frm`
Scope	Global
Dynamic	Yes
Type	Boolean

Default Value	ON

If this variable is set to 1, when any nontemporary table is created its .frm file is synchronized to disk (using fdatasync()). This is slower but safer in case of a crash. The default is 1.

- system_time_zone

System Variable	system_time_zone
Scope	Global
Dynamic	No
Type	String

The server system time zone. When the server begins executing, it inherits a time zone setting from the machine defaults, possibly modified by the environment of the account used for running the server or the startup script. The value is used to set system_time_zone. Typically the time zone is specified by the TZ environment variable. It also can be specified using the --timezone option of the mysqld_safe script.

The system_time_zone variable differs from time_zone. Although they might have the same value, the latter variable is used to initialize the time zone for each client that connects. See Section 5.1.13, “MySQL Server Time Zone Support”.

- table_definition_cache

Command-Line Format	--table-definition-cache=#
System Variable	table_definition_cache
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	-1 (signifies autosizing; do not assign this literal value)
Minimum Value	400
Maximum Value	524288

The number of table definitions (from .frm files) that can be stored in the definition cache. If you use a large number of tables, you can create a large table definition cache to speed up opening of tables. The table definition cache takes less space and does not use file descriptors, unlike the normal table cache. The minimum value is 400. The default value is based on the following formula, capped to a limit of 2000:

```
400 + (table_open_cache / 2)
```

For InnoDB, table_definition_cache acts as a soft limit for the number of open table instances in the InnoDB data dictionary cache. If the number of open table instances exceeds the table_definition_cache setting, the LRU mechanism begins to mark table instances for eviction and eventually removes them from the data dictionary cache. The limit helps address situations in which significant amounts of memory would be used to cache rarely used table instances until the next server restart. The number of table instances with cached metadata could be higher than the limit defined by table_definition_cache, because InnoDB system table instances and parent and child table instances with foreign key relationships are not placed on the LRU list and are not subject to eviction from memory.

Additionally, table_definition_cache defines a soft limit for the number of InnoDB file-per-table tablespaces that can be open at one time, which is also controlled by innodb_open_files. If both table_definition_cache and innodb_open_files are set, the highest setting is used. If neither variable is set, table_definition_cache, which has a higher default value, is used. If the

number of open tablespace file handles exceeds the limit defined by `table_definition_cache` or `innodb_open_files`, the LRU mechanism searches the tablespace file LRU list for files that are fully flushed and are not currently being extended. This process is performed each time a new tablespace is opened. If there are no “inactive” tablespaces, no tablespace files are closed.

- `table_open_cache`

Command-Line Format	`--table-open-cache=#`
System Variable	`table_open_cache`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`2000`
Minimum Value	`1`
Maximum Value	`524288`

The number of open tables for all threads. Increasing this value increases the number of file descriptors that `mysqld` requires. You can check whether you need to increase the table cache by checking the `Opened_tables` status variable. See Section 5.1.9, “Server Status Variables”. If the value of `Opened_tables` is large and you do not use `FLUSH TABLES` often (which just forces all tables to be closed and reopened), then you should increase the value of the `table_open_cache` variable. For more information about the table cache, see Section 8.4.3.1, “How MySQL Opens and Closes Tables”.

- `table_open_cache_instances`

Command-Line Format	`--table-open-cache-instances=#`
System Variable	`table_open_cache_instances`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`1`
Minimum Value	`1`
Maximum Value	`64`

The number of open tables cache instances (default 1). To improve scalability by reducing contention among sessions, the open tables cache can be partitioned into several smaller cache instances of size `table_open_cache` / `table_open_cache_instances` . A session needs to lock only one instance to access it for DML statements. This segments cache access among instances, permitting higher performance for operations that use the cache when there are many sessions accessing tables. (DDL statements still require a lock on the entire cache, but such statements are much less frequent than DML statements.)

A value of 8 or 16 is recommended on systems that routinely use 16 or more cores.

- `thread_cache_size`

Command-Line Format	`--thread-cache-size=#`
System Variable	`thread_cache_size`
Scope	Global
Dynamic	Yes
Type	Integer

Default Value	-1 (signifies autosizing; do not assign this literal value)
Minimum Value	0
Maximum Value	16384

How many threads the server should cache for reuse. When a client disconnects, the client's threads are put in the cache if there are fewer than thread_cache_size threads there. Requests for threads are satisfied by reusing threads taken from the cache if possible, and only when the cache is empty is a new thread created. This variable can be increased to improve performance if you have a lot of new connections. Normally, this does not provide a notable performance improvement if you have a good thread implementation. However, if your server sees hundreds of connections per second you should normally set thread_cache_size high enough so that most new connections use cached threads. By examining the difference between the Connections and Threads_created status variables, you can see how efficient the thread cache is. For details, see Section 5.1.9, “Server Status Variables”.

The default value is based on the following formula, capped to a limit of 100:

```
8 + (max_connections / 100)
```

- thread_concurrency

Command-Line Format	--thread-concurrency=#
Deprecated	Yes
System Variable	thread_concurrency
Scope	Global
Dynamic	No
Type	Integer
Default Value	10
Minimum Value	1
Maximum Value	512

This variable is specific to Solaris 8 and earlier systems, for which mysqld invokes the thr_setconcurrency() function with the variable value. This function enables applications to give the threads system a hint about the desired number of threads that should be run at the same time. Current Solaris versions document this as having no effect.

This variable is deprecated and is removed in MySQL 5.7. You should remove this from MySQL configuration files whenever you see it unless they are for Solaris 8 or earlier.

- thread_handling

Command-Line Format	--thread-handling=name
System Variable	thread_handling
Scope	Global
Dynamic	No
Type	Enumeration
Default Value	one-thread-per-connection
Valid Values	no-threads one-thread-per-connection

	`loaded-dynamically`

The thread-handling model used by the server for connection threads. The permissible values are `no-threads` (the server uses a single thread to handle one connection), `one-thread-per-connection` (the server uses one thread to handle each client connection), and `loaded-dynamically` (set by the thread pool plugin when it initializes). `no-threads` is useful for debugging under Linux; see Section 5.8, "Debugging MySQL".

If the thread pool plugin is enabled, the server sets the `thread_handling` value to `loaded-dynamically`. See Section 5.5.3.2, "Thread Pool Installation".

- `thread_pool_algorithm`

Command-Line Format	`--thread-pool-algorithm=#`
System Variable	`thread_pool_algorithm`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1`

This variable controls which algorithm the thread pool plugin uses:

- A value of 0 (the default) uses a conservative low-concurrency algorithm which is most well tested and is known to produce very good results.
- A value of 1 increases the concurrency and uses a more aggressive algorithm which at times has been known to perform 5–10% better on optimal thread counts, but has degrading performance as the number of connections increases. Its use should be considered as experimental and not supported.

This variable is available only if the thread pool plugin is enabled. See Section 5.5.3, "MySQL Enterprise Thread Pool".

- `thread_pool_high_priority_connection`

Command-Line Format	`--thread-pool-high-priority-connection=#`
System Variable	`thread_pool_high_priority_connection`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1`

This variable affects queuing of new statements prior to execution. If the value is 0 (false, the default), statement queuing uses both the low-priority and high-priority queues. If the value is 1 (true), queued statements always go to the high-priority queue.

This variable is available only if the thread pool plugin is enabled. See Section 5.5.3, "MySQL Enterprise Thread Pool".

- `thread_pool_max_unused_threads`

Command-Line Format	`--thread-pool-max-unused-threads=#`
System Variable	`thread_pool_max_unused_threads`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`4096`

The maximum permitted number of unused threads in the thread pool. This variable makes it possible to limit the amount of memory used by sleeping threads.

A value of 0 (the default) means no limit on the number of sleeping threads. A value of *`N`* where *`N`* is greater than 0 means 1 consumer thread and *`N`*−1 reserve threads. In this case, if a thread is ready to sleep but the number of sleeping threads is already at the maximum, the thread exits rather than going to sleep.

A sleeping thread is either sleeping as a consumer thread or a reserve thread. The thread pool permits one thread to be the consumer thread when sleeping. If a thread goes to sleep and there is no existing consumer thread, it sleeps as a consumer thread. When a thread must be woken up, a consumer thread is selected if there is one. A reserve thread is selected only when there is no consumer thread to wake up.

This variable is available only if the thread pool plugin is enabled. See Section 5.5.3, “MySQL Enterprise Thread Pool”.

- `thread_pool_prio_kickup_timer`

Command-Line Format	`--thread-pool-prio-kickup-timer=#`
System Variable	`thread_pool_prio_kickup_timer`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`1000`
Minimum Value	`0`
Maximum Value	`4294967294`

This variable affects statements waiting for execution in the low-priority queue. The value is the number of milliseconds before a waiting statement is moved to the high-priority queue. The default is 1000 (1 second).

This variable is available only if the thread pool plugin is enabled. See Section 5.5.3, “MySQL Enterprise Thread Pool”.

- `thread_pool_size`

Command-Line Format	`--thread-pool-size=#`
System Variable	`thread_pool_size`
Scope	Global
Dynamic	No

Type	Integer
Default Value	16
Minimum Value	1
Maximum Value	64

The number of thread groups in the thread pool. This is the most important parameter controlling thread pool performance. It affects how many statements can execute simultaneously. If a value outside the range of permissible values is specified, the thread pool plugin does not load and the server writes a message to the error log.

This variable is available only if the thread pool plugin is enabled. See Section 5.5.3, “MySQL Enterprise Thread Pool”.

- `thread_pool_stall_limit`

Command-Line Format	`--thread-pool-stall-limit=#`
System Variable	`thread_pool_stall_limit`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	6
Minimum Value	4
Maximum Value	600

This variable affects executing statements. The value is the amount of time a statement has to finish after starting to execute before it becomes defined as stalled, at which point the thread pool permits the thread group to begin executing another statement. The value is measured in 10 millisecond units, so the default of 6 means 60ms. Short wait values permit threads to start more quickly. Short values are also better for avoiding deadlock situations. Long wait values are useful for workloads that include long-running statements, to avoid starting too many new statements while the current ones execute.

This variable is available only if the thread pool plugin is enabled. See Section 5.5.3, “MySQL Enterprise Thread Pool”.

- `thread_stack`

Command-Line Format	`--thread-stack=#`
System Variable	`thread_stack`
Scope	Global
Dynamic	No
Type	Integer
Default Value (64-bit platforms)	262144
Default Value (32-bit platforms)	196608
Minimum Value	131072
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295

Block Size	1024

The stack size for each thread. Many of the limits detected by the `crash-me` test are dependent on this value. See Section 8.13.2, “The MySQL Benchmark Suite”. The default is large enough for normal operation. If the thread stack size is too small, it limits the complexity of the SQL statements that the server can handle, the recursion depth of stored procedures, and other memory-consuming actions.

- `time_format`

  This variable is unused. It is deprecated; expect it to be removed in a future MySQL release.

- `time_zone`

System Variable	time_zone
Scope	Global, Session
Dynamic	Yes
Type	String
Default Value	SYSTEM
Minimum Value	-12:59
Maximum Value	+13:00

  The current time zone. This variable is used to initialize the time zone for each client that connects. By default, the initial value of this is `'SYSTEM'` (which means, “use the value of `system_time_zone`”). The value can be specified explicitly at server startup with the `--default-time-zone` option. See Section 5.1.13, “MySQL Server Time Zone Support”.

  Note

  If set to `SYSTEM`, every MySQL function call that requires a time zone calculation makes a system library call to determine the current system time zone. This call may be protected by a global mutex, resulting in contention.

- `timed_mutexes`

Command-Line Format	--timed-mutexes[={OFF|ON}]
Deprecated	5.6.20
System Variable	timed_mutexes
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	OFF

  This variable is deprecated; it has no use. Expect it to be removed in a future MySQL release.

- `timestamp`

System Variable	timestamp
Scope	Session
Dynamic	Yes
Type	Numeric
Default Value	UNIX_TIMESTAMP()
Minimum Value	1

Maximum Value	2147483647

Set the time for this client. This is used to get the original timestamp if you use the binary log to restore rows. *timestamp_value* should be a Unix epoch timestamp (a value like that returned by UNIX_TIMESTAMP(), not a value in *'YYYY-MM-DD hh:mm:ss'* format) or DEFAULT.

Setting timestamp to a constant value causes it to retain that value until it is changed again. Setting timestamp to DEFAULT causes its value to be the current date and time as of the time it is accessed. The maximum value corresponds to '2038-01-19 03:14:07' UTC, the same as for the TIMESTAMP data type.

In MySQL 5.6, timestamp is a DOUBLE rather than BIGINT because its value includes a microseconds part.

SET timestamp affects the value returned by NOW() but not by SYSDATE(). This means that timestamp settings in the binary log have no effect on invocations of SYSDATE(). The server can be started with the --sysdate-is-now option to cause SYSDATE() to be a synonym for NOW(), in which case SET timestamp affects both functions.

- tmp_table_size

Command-Line Format	--tmp-table-size=#
System Variable	tmp_table_size
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	16777216
Minimum Value	1024
Maximum Value	18446744073709551615

The maximum size of internal in-memory temporary tables. This variable does not apply to user-created MEMORY tables.

The actual limit is the smaller of tmp_table_size and max_heap_table_size. When an in-memory temporary table exceeds the limit, MySQL automatically converts it to an on-disk MyISAM table. Increase the value of tmp_table_size (and max_heap_table_size if necessary) if you do many advanced GROUP BY queries and you have lots of memory.

You can compare the number of internal on-disk temporary tables created to the total number of internal temporary tables created by comparing Created_tmp_disk_tables and Created_tmp_tables values.

See also Section 8.4.4, “Internal Temporary Table Use in MySQL”.

- tmpdir

Command-Line Format	--tmpdir=dir_name
System Variable	tmpdir
Scope	Global
Dynamic	No
Type	Directory name

The path of the directory to use for creating temporary files. It might be useful if your default /tmp directory resides on a partition that is too small to hold temporary tables. This variable can be set

to a list of several paths that are used in round-robin fashion. Paths should be separated by colon characters (`:`) on Unix and semicolon characters (`;`) on Windows.

`tmpdir` can be a non-permanent location, such as a directory on a memory-based file system or a directory that is cleared when the server host restarts. If the MySQL server is acting as a replica, and you are using a non-permanent location for `tmpdir`, consider setting a different temporary directory for the replica using the `slave_load_tmpdir` variable. For a replication replica, the temporary files used to replicate `LOAD DATA` statements are stored in this directory, so with a permanent location they can survive machine restarts, although replication can now continue after a restart if the temporary files have been removed.

For more information about the storage location of temporary files, see Section B.3.3.5, “Where MySQL Stores Temporary Files”.

- `transaction_alloc_block_size`

Command-Line Format	`--transaction-alloc-block-size=#`
System Variable	`transaction_alloc_block_size`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`8192`
Minimum Value	`1024`
Maximum Value (64-bit platforms, ≤ 5.6.23)	`18446744073709551615`
Maximum Value (32-bit platforms, ≤ 5.6.23)	`4294967295`
Maximum Value (≥ 5.6.24)	`131072`
Block Size	`1024`

The amount in bytes by which to increase a per-transaction memory pool which needs memory. See the description of `transaction_prealloc_size`.

- `transaction_prealloc_size`

Command-Line Format	`--transaction-prealloc-size=#`
System Variable	`transaction_prealloc_size`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`4096`
Minimum Value	`1024`
Maximum Value (64-bit platforms, ≤ 5.6.23)	`18446744073709551615`
Maximum Value (32-bit platforms, ≤ 5.6.23)	`4294967295`
Maximum Value (≥ 5.6.24)	`131072`
Block Size	`1024`

There is a per-transaction memory pool from which various transaction-related allocations take memory. The initial size of the pool in bytes is `transaction_prealloc_size`. For every

allocation that cannot be satisfied from the pool because it has insufficient memory available, the pool is increased by `transaction_alloc_block_size` bytes. When the transaction ends, the pool is truncated to `transaction_prealloc_size` bytes.

By making `transaction_prealloc_size` sufficiently large to contain all statements within a single transaction, you can avoid many `malloc()` calls.

- `tx_isolation`

System Variable	`tx_isolation`
Scope	Global, Session
Dynamic	Yes
Type	Enumeration
Default Value	`REPEATABLE-READ`

Valid Values	`READ-UNCOMMITTED`
	`READ-COMMITTED`
	`REPEATABLE-READ`
	`SERIALIZABLE`

The transaction isolation level. The default is `REPEATABLE-READ`.

The transaction isolation level has three scopes: global, session, and next transaction. This three-scope implementation leads to some nonstandard isolation-level assignment semantics, as described later.

To set the global transaction isolation level at startup, use the `--transaction-isolation` server option.

At runtime, the isolation level can be set directly using the `SET` statement to assign a value to the `tx_isolation` system variable, or indirectly using the `SET TRANSACTION` statement. If you set `tx_isolation` directly to an isolation level name that contains a space, the name should be enclosed within quotation marks, with the space replaced by a dash. For example, use this `SET` statement to set the global value:

```
SET GLOBAL tx_isolation = 'READ-COMMITTED';
```

Setting the global `tx_isolation` value sets the isolation level for all subsequent sessions. Existing sessions are unaffected.

To set the session or next-level `tx_isolation` value, use the `SET` statement. For most session system variables, these statements are equivalent ways to set the value:

```
SET @@SESSION.var_name = value;
SET SESSION var_name = value;
SET var_name = value;
SET @@var_name = value;
```

As mentioned previously, the transaction isolation level has a next-transaction scope, in addition to the global and session scopes. To enable the next-transaction scope to be set, `SET` syntax for assigning session system variable values has nonstandard semantics for `tx_isolation`:

- To set the session isolation level, use any of these syntaxes:

```
SET @@SESSION.tx_isolation = value;
SET SESSION tx_isolation = value;
```

```
SET tx_isolation = value;
```

For each of those syntaxes, these semantics apply:

- Sets the isolation level for all subsequent transactions performed within the session.
- Permitted within transactions, but does not affect the current ongoing transaction.
- If executed between transactions, overrides any preceding statement that sets the next-transaction isolation level.
- Corresponds to `SET SESSION TRANSACTION ISOLATION LEVEL` (with the `SESSION` keyword).

- To set the next-transaction isolation level, use this syntax:

```
SET @@tx_isolation = value;
```

For that syntax, these semantics apply:

- Sets the isolation level only for the next single transaction performed within the session.
- Subsequent transactions revert to the session isolation level.
- Not permitted within transactions.
- Corresponds to `SET TRANSACTION ISOLATION LEVEL` (without the `SESSION` keyword).

For more information about `SET TRANSACTION` and its relationship to the `tx_isolation` system variable, see Section 13.3.6, “SET TRANSACTION Statement”.

- `tx_read_only`

System Variable	`tx_read_only`
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

The transaction access mode. The value can be `OFF` (read/write; the default) or `ON` (read only).

The transaction access mode has three scopes: global, session, and next transaction. This three-scope implementation leads to some nonstandard access-mode assignment semantics, as described later.

To set the global transaction access mode at startup, use the `--transaction-read-only` server option.

At runtime, the access mode can be set directly using the `SET` statement to assign a value to the `tx_read_only` system variable, or indirectly using the `SET TRANSACTION` statement. For example, use this `SET` statement to set the global value:

```
SET GLOBAL tx_read_only = ON;
```

Setting the global `tx_read_only` value sets the access mode for all subsequent sessions. Existing sessions are unaffected.

To set the session or next-level `tx_read_only` value, use the `SET` statement. For most session system variables, these statements are equivalent ways to set the value:

```
SET @@SESSION.var_name = value;
```

```
SET SESSION var_name = value;
SET var_name = value;
SET @@var_name = value;
```

As mentioned previously, the transaction access mode has a next-transaction scope, in addition to the global and session scopes. To enable the next-transaction scope to be set, `SET` syntax for assigning session system variable values has nonstandard semantics for `tx_read_only`,

- To set the session access mode, use any of these syntaxes:

  ```
  SET @@SESSION.tx_read_only = value;
  SET SESSION tx_read_only = value;
  SET tx_read_only = value;
  ```

  For each of those syntaxes, these semantics apply:

  - Sets the access mode for all subsequent transactions performed within the session.
  - Permitted within transactions, but does not affect the current ongoing transaction.
  - If executed between transactions, overrides any preceding statement that sets the next-transaction access mode.
  - Corresponds to `SET SESSION TRANSACTION {READ WRITE | READ ONLY}` (with the `SESSION` keyword).

- To set the next-transaction access mode, use this syntax:

  ```
  SET @@tx_read_only = value;
  ```

  For that syntax, these semantics apply:

  - Sets the access mode only for the next single transaction performed within the session.
  - Subsequent transactions revert to the session access mode.
  - Not permitted within transactions.
  - Corresponds to `SET TRANSACTION {READ WRITE | READ ONLY}` (without the `SESSION` keyword).

For more information about `SET TRANSACTION` and its relationship to the `tx_read_only` system variable, see Section 13.3.6, “SET TRANSACTION Statement”.

- `unique_checks`

System Variable	`unique_checks`
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	`ON`

If set to 1 (the default), uniqueness checks for secondary indexes in `InnoDB` tables are performed. If set to 0, storage engines are permitted to assume that duplicate keys are not present in input data. If you know for certain that your data does not contain uniqueness violations, you can set this to 0 to speed up large table imports to `InnoDB`.

Setting this variable to 0 does not *require* storage engines to ignore duplicate keys. An engine is still permitted to check for them and issue duplicate-key errors if it detects them.

- `updatable_views_with_limit`

Command-Line Format	`--updatable-views-with-limit[={OFF|ON}]`
System Variable	`updatable_views_with_limit`
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	`1`

This variable controls whether updates to a view can be made when the view does not contain all columns of the primary key defined in the underlying table, if the update statement contains a `LIMIT` clause. (Such updates often are generated by GUI tools.) An update is an `UPDATE` or `DELETE` statement. Primary key here means a `PRIMARY KEY`, or a `UNIQUE` index in which no column can contain `NULL`.

The variable can have two values:

- `1` or `YES`: Issue a warning only (not an error message). This is the default value.
- `0` or `NO`: Prohibit the update.

- `validate_password_xxx`

The `validate_password` plugin implements a set of system variables having names of the form `validate_password_xxx`. These variables affect password testing by that plugin; see Section 6.4.3.2, “Password Validation Plugin Options and Variables”.

- `validate_user_plugins`

Command-Line Format	`--validate-user-plugins[={OFF|ON}]`
Introduced	5.6.11
System Variable	`validate_user_plugins`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`ON`

If this variable is enabled (the default), the server checks each user account and produces a warning if conditions are found that would make the account unusable:

- The account requires an authentication plugin that is not loaded.
- The account requires the `sha256_password` authentication plugin but the server was started with neither SSL nor RSA enabled as required by this plugin.

Enabling `validate_user_plugins` slows down server initialization and `FLUSH PRIVILEGES`. If you do not require the additional checking, you can disable this variable at startup to avoid the performance decrement.

This variable was added in MySQL 5.6.11.

- `version`

The version number for the server. The value might also include a suffix indicating server build or configuration information. `-log` indicates that one or more of the general log, slow query log, or binary log are enabled. `-debug` indicates that the server was built with debugging support enabled.

- `version_comment`

System Variable	`version_comment`
Scope	Global
Dynamic	No
Type	String

The `CMake` configuration program has a `COMPILATION_COMMENT` option that permits a comment to be specified when building MySQL. This variable contains the value of that comment. See Section 2.9.7, “MySQL Source-Configuration Options”.

- `version_compile_machine`

System Variable	`version_compile_machine`
Scope	Global
Dynamic	No
Type	String

The type of the server binary.

- `version_compile_os`

System Variable	`version_compile_os`
Scope	Global
Dynamic	No
Type	String

The type of operating system on which MySQL was built.

- `wait_timeout`

Command-Line Format	`--wait-timeout=#`
System Variable	`wait_timeout`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`28800`
Minimum Value	`1`
Maximum Value (Other)	`31536000`
Maximum Value (Windows)	`2147483`

The number of seconds the server waits for activity on a noninteractive connection before closing it.

On thread startup, the session `wait_timeout` value is initialized from the global `wait_timeout` value or from the global `interactive_timeout` value, depending on the type of client (as defined by the `CLIENT_INTERACTIVE` connect option to `mysql_real_connect()`). See also `interactive_timeout`.

- `warning_count`

The number of errors, warnings, and notes that resulted from the last statement that generated messages. This variable is read only. See Section 13.7.5.41, “SHOW WARNINGS Statement”.

5.1.8 Using System Variables

The MySQL server maintains many system variables that configure its operation. Section 5.1.7, “Server System Variables”, describes the meaning of these variables. Each system variable has a default value. System variables can be set at server startup using options on the command line or in an option file. Most of them can be changed dynamically while the server is running by means of the `SET` statement, which enables you to modify operation of the server without having to stop and restart it. You can also use system variable values in expressions.

Many system variables are built in. System variables implemented by a server plugin are exposed when the plugin is installed and have names that begin with the plugin name. For example, the `audit_log` plugin implements a system variable named `audit_log_policy`.

There are two scopes in which system variables exist. Global variables affect the overall operation of the server. Session variables affect its operation for individual client connections. A given system variable can have both a global and a session value. Global and session system variables are related as follows:

- When the server starts, it initializes each global variable to its default value. These defaults can be changed by options specified on the command line or in an option file. (See Section 4.2.2, “Specifying Program Options”.)
- The server also maintains a set of session variables for each client that connects. The client's session variables are initialized at connect time using the current values of the corresponding global variables. For example, a client's SQL mode is controlled by the session `sql_mode` value, which is initialized when the client connects to the value of the global `sql_mode` value.

  For some system variables, the session value is not initialized from the corresponding global value; if so, that is indicated in the variable description.

System variable values can be set globally at server startup by using options on the command line or in an option file. At startup, the syntax for system variables is the same as for command options, so within variable names, dashes and underscores may be used interchangeably. For example, `--general_log=ON` and `--general-log=ON` are equivalent.

When you use a startup option to set a variable that takes a numeric value, the value can be given with a suffix of `K`, `M`, or `G` (either uppercase or lowercase) to indicate a multiplier of 1024, 1024^2 or 1024^3; that is, units of kilobytes, megabytes, or gigabytes, respectively. Thus, the following command starts the server with an `InnoDB` log file size of 16 megabytes and a maximum packet size of one gigabyte:

```
mysqld --innodb-log-file-size=16M --max-allowed-packet=1G
```

Within an option file, those variables are set like this:

```
[mysqld]
innodb_log_file_size=16M
max_allowed_packet=1G
```

The lettercase of suffix letters does not matter; `16M` and `16m` are equivalent, as are `1G` and `1g`.

To restrict the maximum value to which a system variable can be set at runtime with the `SET` statement, specify this maximum by using an option of the form `--maximum-var_name=value` at server startup. For example, to prevent the value of `innodb_log_file_size` from being increased to more than 32MB at runtime, use the option `--maximum-innodb-log-file-size=32M`.

Many system variables are dynamic and can be changed at runtime by using the `SET` statement. For a list, see Section 5.1.8.2, “Dynamic System Variables”. To change a system variable with `SET`, refer to it by name, optionally preceded by a modifier. At runtime, system variable names must be written using underscores, not dashes. The following examples briefly illustrate this syntax:

- Set a global system variable:

  ```
  SET GLOBAL max_connections = 1000;
  ```

```
SET @@GLOBAL.max_connections = 1000;
```

- Set a session system variable:

```
SET SESSION sql_mode = 'TRADITIONAL';
SET @@SESSION.sql_mode = 'TRADITIONAL';
SET @@sql_mode = 'TRADITIONAL';
```

For complete details about `SET` syntax, see Section 13.7.4.1, “SET Syntax for Variable Assignment”. For a description of the privilege requirements for setting system variables, see Section 5.1.8.1, “System Variable Privileges”

Suffixes for specifying a value multiplier can be used when setting a variable at server startup, but not to set the value with `SET` at runtime. On the other hand, with `SET` you can assign a variable's value using an expression, which is not true when you set a variable at server startup. For example, the first of the following lines is legal at server startup, but the second is not:

```
shell> mysql --max_allowed_packet=16M
shell> mysql --max_allowed_packet=16*1024*1024
```

Conversely, the second of the following lines is legal at runtime, but the first is not:

```
mysql> SET GLOBAL max_allowed_packet=16M;
mysql> SET GLOBAL max_allowed_packet=16*1024*1024;
```

Note

Some system variables can be enabled with the `SET` statement by setting them to `ON` or `1`, or disabled by setting them to `OFF` or `0`. However, to set such a variable on the command line or in an option file, you must set it to `1` or `0`; setting it to `ON` or `OFF` does not work. For example, on the command line, `--delay_key_write=1` works but `--delay_key_write=ON` does not.

To display system variable names and values, use the `SHOW VARIABLES` statement:

```
mysql> SHOW VARIABLES;
+---------------------------------+-----------------------------------+
| Variable_name                   | Value                             |
+---------------------------------+-----------------------------------+
| auto_increment_increment        | 1                                 |
| auto_increment_offset           | 1                                 |
| automatic_sp_privileges         | ON                                |
| back_log                        | 50                                |
| basedir                         | /home/mysql/                      |
| binlog_cache_size               | 32768                             |
| bulk_insert_buffer_size         | 8388608                           |
| character_set_client            | utf8                              |
| character_set_connection        | utf8                              |
| character_set_database          | latin1                            |
| character_set_filesystem        | binary                            |
| character_set_results           | utf8                              |
| character_set_server            | latin1                            |
| character_set_system            | utf8                              |
| character_sets_dir              | /home/mysql/share/mysql/charsets/ |
| collation_connection            | utf8_general_ci                   |
| collation_database              | latin1_swedish_ci                 |
| collation_server                | latin1_swedish_ci                 |
...
| innodb_additional_mem_pool_size | 1048576                           |
| innodb_autoextend_increment     | 8                                 |
| innodb_buffer_pool_size         | 8388608                           |
| innodb_checksums                | ON                                |
| innodb_commit_concurrency       | 0                                 |
| innodb_concurrency_tickets      | 500                               |
| innodb_data_file_path           | ibdata1:10M:autoextend            |
| innodb_data_home_dir            |                                   |
...
| version                         | 5.6.37-log                        |
| version_comment                 | Source distribution               |
```

```
| version_compile_machine         | i686                            |
| version_compile_os              | suse-linux                      |
| wait_timeout                    | 28800                           |
+---------------------------------+---------------------------------+
```

With a `LIKE` clause, the statement displays only those variables that match the pattern. To obtain a specific variable name, use a `LIKE` clause as shown:

```
SHOW VARIABLES LIKE 'max_join_size';
SHOW SESSION VARIABLES LIKE 'max_join_size';
```

To get a list of variables whose name match a pattern, use the `%` wildcard character in a `LIKE` clause:

```
SHOW VARIABLES LIKE '%size%';
SHOW GLOBAL VARIABLES LIKE '%size%';
```

Wildcard characters can be used in any position within the pattern to be matched. Strictly speaking, because `_` is a wildcard that matches any single character, you should escape it as `\_` to match it literally. In practice, this is rarely necessary.

For `SHOW VARIABLES`, if you specify neither `GLOBAL` nor `SESSION`, MySQL returns `SESSION` values.

The reason for requiring the `GLOBAL` keyword when setting `GLOBAL`-only variables but not when retrieving them is to prevent problems in the future:

- Were a `SESSION` variable to be removed that has the same name as a `GLOBAL` variable, a client with privileges sufficient to modify global variables might accidentally change the `GLOBAL` variable rather than just the `SESSION` variable for its own session.
- Were a `SESSION` variable to be added with the same name as a `GLOBAL` variable, a client that intends to change the `GLOBAL` variable might find only its own `SESSION` variable changed.

5.1.8.1 System Variable Privileges

A system variable can have a global value that affects server operation as a whole, a session value that affects only the current session, or both. To modify system variable runtime values, use the `SET` statement. See Section 13.7.4.1, “SET Syntax for Variable Assignment”. This section describes the privileges required to assign values to system variables at runtime.

Setting a global system variable runtime value requires the `SUPER` privilege.

To set a session system variable runtime value, use the `SET SESSION` statement. In contrast to setting global runtime values, setting session runtime values normally requires no special privileges and can be done by any user to affect the current session. For some system variables, setting the session value may have effects outside the current session and thus is a restricted operation that can be done only by users who have the `SUPER` privilege. If a session system variable is restricted in this way, the variable description indicates that restriction. Examples include `binlog_format` and `sql_log_bin`. Setting the session value of these variables affects binary logging for the current session, but may also have wider implications for the integrity of server replication and backups.

5.1.8.2 Dynamic System Variables

Many server system variables are dynamic and can be set at runtime. See Section 13.7.4.1, “SET Syntax for Variable Assignment”. For a description of the privilege requirements for setting system variables, see Section 5.1.8.1, “System Variable Privileges”

The following table lists all dynamic system variables applicable within `mysqld`.

The table lists each variable's data type and scope. The last column indicates whether the scope for each variable is Global, Session, or both. Please see the corresponding item descriptions for details on setting and using the variables. Where appropriate, direct links to further information about the items are provided.

Variables that have a type of “string” take a string value. Variables that have a type of “numeric” take a numeric value. Variables that have a type of “boolean” can be set to 0, 1, `ON` or `OFF`. Variables that are

marked as “enumeration” normally should be set to one of the available values for the variable, but can also be set to the number that corresponds to the desired enumeration value. For enumerated system variables, the first enumeration value corresponds to 0. This differs from the `ENUM` data type used for table columns, for which the first enumeration value corresponds to 1.

Table 5.5 Dynamic System Variable Summary

Variable Name	Variable Type	Variable Scope
audit_log_connection_policy	Enumeration	Global
audit_log_exclude_accounts	String	Global
audit_log_flush	Boolean	Global
audit_log_include_accounts	String	Global
audit_log_policy	Enumeration	Global
audit_log_rotate_on_size	Integer	Global
audit_log_statement_policy	Enumeration	Global
auto_increment_increment	Integer	Both
auto_increment_offset	Integer	Both
autocommit	Boolean	Both
automatic_sp_privileges	Boolean	Global
avoid_temporal_upgrade	Boolean	Global
big_tables	Boolean	Both
binlog_cache_size	Integer	Global
binlog_checksum	String	Global
binlog_direct_non_transactional_updates	Boolean	Both
binlog_error_action	Enumeration	Global
binlog_format	Enumeration	Both
binlog_max_flush_queue_time	Integer	Global
binlog_order_commits	Boolean	Global
binlog_row_image	Enumeration	Both
binlog_rows_query_log_events	Boolean	Both
binlog_stmt_cache_size	Integer	Global
binlogging_impossible_mode	Enumeration	Both
block_encryption_mode	String	Both
bulk_insert_buffer_size	Integer	Both
character_set_client	String	Both
character_set_connection	String	Both
character_set_database	String	Both
character_set_filesystem	String	Both
character_set_results	String	Both
character_set_server	String	Both
collation_connection	String	Both
collation_database	String	Both
collation_server	String	Both
completion_type	Enumeration	Both
concurrent_insert	Enumeration	Global

Variable Name	Variable Type	Variable Scope
connect_timeout	Integer	Global
connection_control_failed_connections_threshold	Integer	Global
connection_control_max_connection_delay	Integer	Global
connection_control_min_connection_delay	Integer	Global
debug	String	Both
debug_sync	String	Session
default_storage_engine	Enumeration	Both
default_tmp_storage_engine	Enumeration	Both
default_week_format	Integer	Both
delay_key_write	Enumeration	Global
delayed_insert_limit	Integer	Global
delayed_insert_timeout	Integer	Global
delayed_queue_size	Integer	Global
div_precision_increment	Integer	Both
end_markers_in_json	Boolean	Both
eq_range_index_dive_limit	Integer	Both
event_scheduler	Enumeration	Global
expire_logs_days	Integer	Global
explicit_defaults_for_timestamp	Boolean	Both
flush	Boolean	Global
flush_time	Integer	Global
foreign_key_checks	Boolean	Both
ft_boolean_syntax	String	Global
general_log	Boolean	Global
general_log_file	File name	Global
group_concat_max_len	Integer	Both
gtid_next	Enumeration	Session
gtid_purged	String	Global
host_cache_size	Integer	Global
identity	Integer	Session
init_connect	String	Global
init_slave	String	Global
innodb_adaptive_flushing	Boolean	Global
innodb_adaptive_flushing_lwm	Integer	Global
innodb_adaptive_hash_index	Boolean	Global
innodb_adaptive_max_sleep_delay	Integer	Global
innodb_api_bk_commit_interval	Integer	Global
innodb_api_trx_level	Integer	Global
innodb_autoextend_increment	Integer	Global
innodb_buffer_pool_dump_at_shutdown	Boolean	Global
innodb_buffer_pool_dump_now	Boolean	Global

Variable Name	Variable Type	Variable Scope
innodb_buffer_pool_filename	File name	Global
innodb_buffer_pool_load_abort	Boolean	Global
innodb_buffer_pool_load_now	Boolean	Global
innodb_change_buffer_max_size	Integer	Global
innodb_change_buffering	Enumeration	Global
innodb_change_buffering_debug	Integer	Global
innodb_checksum_algorithm	Enumeration	Global
innodb_cmp_per_index_enabled	Boolean	Global
innodb_commit_concurrency	Integer	Global
innodb_compression_failure_threshold_pct	Integer	Global
innodb_compression_level	Integer	Global
innodb_compression_pad_pct_max	Integer	Global
innodb_concurrency_tickets	Integer	Global
innodb_disable_sort_file_cache	Boolean	Global
innodb_fast_shutdown	Integer	Global
innodb_fil_make_page_dirty_debug	Integer	Global
innodb_file_format	String	Global
innodb_file_format_max	String	Global
innodb_file_per_table	Boolean	Global
innodb_flush_log_at_timeout	Integer	Global
innodb_flush_log_at_trx_commit	Enumeration	Global
innodb_flush_neighbors	Enumeration	Global
innodb_flushing_avg_loops	Integer	Global
innodb_ft_aux_table	String	Global
innodb_ft_enable_diag_print	Boolean	Global
innodb_ft_enable_stopword	Boolean	Both
innodb_ft_num_word_optimize	Integer	Global
innodb_ft_result_cache_limit	Integer	Global
innodb_ft_server_stopword_table	String	Global
innodb_ft_user_stopword_table	String	Both
innodb_io_capacity	Integer	Global
innodb_io_capacity_max	Integer	Global
innodb_large_prefix	Boolean	Global
innodb_limit_optimistic_insert_debug	Integer	Global
innodb_lock_wait_timeout	Integer	Both
innodb_log_checkpoint_now	Boolean	Global
innodb_log_compressed_pages	Boolean	Global
innodb_lru_scan_depth	Integer	Global
innodb_max_dirty_pages_pct	Numeric	Global
innodb_max_dirty_pages_pct_lwm	Numeric	Global
innodb_max_purge_lag	Integer	Global

Variable Name	Variable Type	Variable Scope
innodb_max_purge_lag_delay	Integer	Global
innodb_monitor_disable	String	Global
innodb_monitor_enable	String	Global
innodb_monitor_reset	Enumeration	Global
innodb_monitor_reset_all	Enumeration	Global
innodb_old_blocks_pct	Integer	Global
innodb_old_blocks_time	Integer	Global
innodb_online_alter_log_max_size	Integer	Global
innodb_optimize_fulltext_only	Boolean	Global
innodb_print_all_deadlocks	Boolean	Global
innodb_purge_batch_size	Integer	Global
innodb_random_read_ahead	Boolean	Global
innodb_read_ahead_threshold	Integer	Global
innodb_replication_delay	Integer	Global
innodb_rollback_segments	Integer	Global
innodb_saved_page_number_debug	Integer	Global
innodb_spin_wait_delay	Integer	Global
innodb_stats_auto_recalc	Boolean	Global
innodb_stats_include_delete_marked	Boolean	Global
innodb_stats_method	Enumeration	Global
innodb_stats_on_metadata	Boolean	Global
innodb_stats_persistent	Boolean	Global
innodb_stats_persistent_sample_pages	Integer	Global
innodb_stats_sample_pages	Integer	Global
innodb_stats_transient_sample_pages	Integer	Global
innodb_status_output	Boolean	Global
innodb_status_output_locks	Boolean	Global
innodb_strict_mode	Boolean	Both
innodb_support_xa	Boolean	Both
innodb_sync_spin_loops	Integer	Global
innodb_table_locks	Boolean	Both
innodb_thread_concurrency	Integer	Global
innodb_thread_sleep_delay	Integer	Global
innodb_tmpdir	Directory name	Both
innodb_trx_purge_view_update_only_debug	Boolean	Global
innodb_trx_rseg_n_slots_debug	Integer	Global
innodb_undo_logs	Integer	Global
insert_id	Integer	Session
interactive_timeout	Integer	Both
join_buffer_size	Integer	Both
keep_files_on_create	Boolean	Both

Variable Name	Variable Type	Variable Scope
key_buffer_size	Integer	Global
key_cache_age_threshold	Integer	Global
key_cache_block_size	Integer	Global
key_cache_division_limit	Integer	Global
last_insert_id	Integer	Session
lc_messages	String	Both
lc_time_names	String	Both
local_infile	Boolean	Global
lock_wait_timeout	Integer	Both
log_bin_trust_function_creators	Boolean	Global
log_output	Set	Global
log_queries_not_using_indexes	Boolean	Global
log_slow_admin_statements	Boolean	Global
log_slow_slave_statements	Boolean	Global
log_throttle_queries_not_using_indexes	Integer	Global
log_warnings	Integer	Varies
long_query_time	Numeric	Both
low_priority_updates	Boolean	Both
master_info_repository	String	Global
master_verify_checksum	Boolean	Global
max_allowed_packet	Integer	Both
max_binlog_cache_size	Integer	Global
max_binlog_size	Integer	Global
max_binlog_stmt_cache_size	Integer	Global
max_connect_errors	Integer	Global
max_connections	Integer	Global
max_delayed_threads	Integer	Both
max_error_count	Integer	Both
max_heap_table_size	Integer	Both
max_insert_delayed_threads	Integer	Both
max_join_size	Integer	Both
max_length_for_sort_data	Integer	Both
max_prepared_stmt_count	Integer	Global
max_relay_log_size	Integer	Global
max_seeks_for_key	Integer	Both
max_sort_length	Integer	Both
max_sp_recursion_depth	Integer	Both
max_tmp_tables	Integer	Both
max_user_connections	Integer	Both
max_write_lock_count	Integer	Global
min_examined_row_limit	Integer	Both

Variable Name	Variable Type	Variable Scope
multi_range_count	Integer	Both
myisam_data_pointer_size	Integer	Global
myisam_max_sort_file_size	Integer	Global
myisam_repair_threads	Integer	Both
myisam_sort_buffer_size	Integer	Both
myisam_stats_method	Enumeration	Both
myisam_use_mmap	Boolean	Global
mysql_firewall_mode	Boolean	Global
mysql_firewall_trace	Boolean	Global
ndb_autoincrement_prefetch_sz	Integer	Both
ndb_blob_read_batch_bytes	Integer	Both
ndb_blob_write_batch_bytes	Integer	Both
ndb_cache_check_time	Integer	Global
ndb_clear_apply_status	Boolean	Global
ndb_deferred_constraints	Integer	Both
ndb_deferred_constraints	Integer	Both
ndb_distribution	Enumeration	Global
ndb_distribution	Enumeration	Global
ndb_eventbuffer_free_percent	Integer	Global
ndb_eventbuffer_max_alloc	Integer	Global
ndb_extra_logging	Integer	Global
ndb_force_send	Boolean	Both
ndb_index_stat_cache_entries	Integer	Both
ndb_index_stat_enable	Boolean	Both
ndb_index_stat_option	String	Both
ndb_index_stat_update_freq	Integer	Both
ndb_join_pushdown	Boolean	Both
ndb_log_bin	Boolean	Both
ndb_log_binlog_index	Boolean	Global
ndb_log_empty_epochs	Boolean	Global
ndb_log_empty_epochs	Boolean	Global
ndb_log_empty_update	Boolean	Global
ndb_log_empty_update	Boolean	Global
ndb_log_exclusive_reads	Boolean	Both
ndb_log_exclusive_reads	Boolean	Both
ndb_log_update_as_write	Boolean	Global
ndb_log_update_minimal	Boolean	Global
ndb_log_updated_only	Boolean	Global
ndb_optimization_delay	Integer	Global
ndb_recv_thread_activation_threshold	Integer	Global
ndb_recv_thread_cpu_mask	Bitmap	Global

Variable Name	Variable Type	Variable Scope
ndb_report_thresh_binlog_epoch_slip	Integer	Global
ndb_report_thresh_binlog_mem_usage	Integer	Global
ndb_show_foreign_key_mock_tables	Boolean	Global
ndb_slave_conflict_role	Enumeration	Global
ndb_table_no_logging	Boolean	Session
ndb_table_temporary	Boolean	Session
ndb_use_exact_count	Boolean	Both
ndb_use_transactions	Boolean	Both
ndbinfo_max_bytes	Integer	Both
ndbinfo_max_rows	Integer	Both
ndbinfo_offline	Boolean	Global
ndbinfo_show_hidden	Boolean	Both
net_buffer_length	Integer	Both
net_read_timeout	Integer	Both
net_retry_count	Integer	Both
net_write_timeout	Integer	Both
new	Boolean	Both
old_alter_table	Boolean	Both
old_passwords	Enumeration	Both
optimizer_prune_level	Integer	Both
optimizer_search_depth	Integer	Both
optimizer_switch	Set	Both
optimizer_trace	String	Both
optimizer_trace_features	String	Both
optimizer_trace_limit	Integer	Both
optimizer_trace_max_mem_size	Integer	Both
optimizer_trace_offset	Integer	Both
preload_buffer_size	Integer	Both
profiling	Boolean	Both
profiling_history_size	Integer	Both
pseudo_slave_mode	Boolean	Session
pseudo_thread_id	Integer	Session
query_alloc_block_size	Integer	Both
query_cache_limit	Integer	Global
query_cache_min_res_unit	Integer	Global
query_cache_size	Integer	Global
query_cache_type	Enumeration	Both
query_cache_wlock_invalidate	Boolean	Both
query_prealloc_size	Integer	Both
rand_seed1	Integer	Session
rand_seed2	Integer	Session

Variable Name	Variable Type	Variable Scope
range_alloc_block_size	Integer	Both
read_buffer_size	Integer	Both
read_only	Boolean	Global
read_rnd_buffer_size	Integer	Both
relay_log_info_repository	String	Global
relay_log_purge	Boolean	Global
relay_log_recovery	Boolean	Global
rpl_semi_sync_master_enabled	Boolean	Global
rpl_semi_sync_master_timeout	Integer	Global
rpl_semi_sync_master_trace_level	Integer	Global
rpl_semi_sync_master_wait_no_slave	Boolean	Global
rpl_semi_sync_slave_enabled	Boolean	Global
rpl_semi_sync_slave_trace_level	Integer	Global
rpl_stop_slave_timeout	Integer	Global
secure_auth	Boolean	Global
server_id	Integer	Global
show_old_temporals	Boolean	Both
slave_allow_batching	Boolean	Global
slave_checkpoint_group	Integer	Global
slave_checkpoint_period	Integer	Global
slave_compressed_protocol	Boolean	Global
slave_exec_mode	Enumeration	Global
slave_max_allowed_packet	Integer	Global
slave_net_timeout	Integer	Global
slave_parallel_workers	Integer	Global
slave_pending_jobs_size_max	Integer	Global
slave_rows_search_algorithms	Set	Global
slave_sql_verify_checksum	Boolean	Global
slave_transaction_retries	Integer	Global
slow_launch_time	Integer	Global
slow_query_log	Boolean	Global
slow_query_log_file	File name	Global
sort_buffer_size	Integer	Both
sql_auto_is_null	Boolean	Both
sql_big_selects	Boolean	Both
sql_buffer_result	Boolean	Both
sql_log_bin	Boolean	Session
sql_log_off	Boolean	Both
sql_mode	Set	Both
sql_notes	Boolean	Both
sql_quote_show_create	Boolean	Both

Variable Name	Variable Type	Variable Scope
sql_safe_updates	Boolean	Both
sql_select_limit	Integer	Both
sql_slave_skip_counter	Integer	Global
sql_warnings	Boolean	Both
storage_engine	Enumeration	Both
stored_program_cache	Integer	Global
sync_binlog	Integer	Global
sync_frm	Boolean	Global
sync_master_info	Integer	Global
sync_relay_log	Integer	Global
sync_relay_log_info	Integer	Global
table_definition_cache	Integer	Global
table_open_cache	Integer	Global
thread_cache_size	Integer	Global
thread_pool_high_priority_connection	Integer	Both
thread_pool_max_unused_threads	Integer	Global
thread_pool_prio_kickup_timer	Integer	Both
thread_pool_stall_limit	Integer	Global
time_zone	String	Both
timed_mutexes	Boolean	Global
timestamp	Numeric	Session
tmp_table_size	Integer	Both
transaction_alloc_block_size	Integer	Both
transaction_allow_batching	Boolean	Session
transaction_prealloc_size	Integer	Both
tx_isolation	Enumeration	Both
tx_read_only	Boolean	Both
unique_checks	Boolean	Both
updatable_views_with_limit	Boolean	Both
validate_password_dictionary_file	File name	Global
validate_password_length	Integer	Global
validate_password_mixed_case_count	Integer	Global
validate_password_number_count	Integer	Global
validate_password_policy	Enumeration	Global
validate_password_special_char_count	Integer	Global
wait_timeout	Integer	Both

5.1.8.3 Structured System Variables

A structured variable differs from a regular system variable in two respects:

- Its value is a structure with components that specify server parameters considered to be closely related.

- There might be several instances of a given type of structured variable. Each one has a different name and refers to a different resource maintained by the server.

MySQL supports one structured variable type, which specifies parameters governing the operation of key caches. A key cache structured variable has these components:

- `key_buffer_size`
- `key_cache_block_size`
- `key_cache_division_limit`
- `key_cache_age_threshold`

This section describes the syntax for referring to structured variables. Key cache variables are used for syntax examples, but specific details about how key caches operate are found elsewhere, in Section 8.10.2, “The MyISAM Key Cache”.

To refer to a component of a structured variable instance, you can use a compound name in *`instance_name.component_name`* format. Examples:

```
hot_cache.key_buffer_size
hot_cache.key_cache_block_size
cold_cache.key_cache_block_size
```

For each structured system variable, an instance with the name of `default` is always predefined. If you refer to a component of a structured variable without any instance name, the `default` instance is used. Thus, `default.key_buffer_size` and `key_buffer_size` both refer to the same system variable.

Structured variable instances and components follow these naming rules:

- For a given type of structured variable, each instance must have a name that is unique *within* variables of that type. However, instance names need not be unique *across* structured variable types. For example, each structured variable has an instance named `default`, so `default` is not unique across variable types.
- The names of the components of each structured variable type must be unique across all system variable names. If this were not true (that is, if two different types of structured variables could share component member names), it would not be clear which default structured variable to use for references to member names that are not qualified by an instance name.
- If a structured variable instance name is not legal as an unquoted identifier, refer to it as a quoted identifier using backticks. For example, `hot-cache` is not legal, but `` `hot-cache` `` is.
- `global`, `session`, and `local` are not legal instance names. This avoids a conflict with notation such as `@@GLOBAL.`*`var_name`* for referring to nonstructured system variables.

Currently, the first two rules have no possibility of being violated because the only structured variable type is the one for key caches. These rules may assume greater significance if some other type of structured variable is created in the future.

With one exception, you can refer to structured variable components using compound names in any context where simple variable names can occur. For example, you can assign a value to a structured variable using a command-line option:

```
shell> mysqld --hot_cache.key_buffer_size=64K
```

In an option file, use this syntax:

```
[mysqld]
```

```
hot_cache.key_buffer_size=64K
```

If you start the server with this option, it creates a key cache named `hot_cache` with a size of 64KB in addition to the default key cache that has a default size of 8MB.

Suppose that you start the server as follows:

```
shell> mysqld --key_buffer_size=256K \
         --extra_cache.key_buffer_size=128K \
         --extra_cache.key_cache_block_size=2048
```

In this case, the server sets the size of the default key cache to 256KB. (You could also have written `--default.key_buffer_size=256K`.) In addition, the server creates a second key cache named `extra_cache` that has a size of 128KB, with the size of block buffers for caching table index blocks set to 2048 bytes.

The following example starts the server with three different key caches having sizes in a 3:1:1 ratio:

```
shell> mysqld --key_buffer_size=6M \
         --hot_cache.key_buffer_size=2M \
         --cold_cache.key_buffer_size=2M
```

Structured variable values may be set and retrieved at runtime as well. For example, to set a key cache named `hot_cache` to a size of 10MB, use either of these statements:

```
mysql> SET GLOBAL hot_cache.key_buffer_size = 10*1024*1024;
mysql> SET @@GLOBAL.hot_cache.key_buffer_size = 10*1024*1024;
```

To retrieve the cache size, do this:

```
mysql> SELECT @@GLOBAL.hot_cache.key_buffer_size;
```

However, the following statement does not work. The variable is not interpreted as a compound name, but as a simple string for a `LIKE` pattern-matching operation:

```
mysql> SHOW GLOBAL VARIABLES LIKE 'hot_cache.key_buffer_size';
```

This is the exception to being able to use structured variable names anywhere a simple variable name may occur.

5.1.9 Server Status Variables

The MySQL server maintains many status variables that provide information about its operation. You can view these variables and their values by using the `SHOW [GLOBAL | SESSION] STATUS` statement (see Section 13.7.5.36, “SHOW STATUS Statement”). The optional `GLOBAL` keyword aggregates the values over all connections, and `SESSION` shows the values for the current connection.

```
mysql> SHOW GLOBAL STATUS;
+-----------------------------------+------------+
| Variable_name                     | Value      |
+-----------------------------------+------------+
| Aborted_clients                   | 0          |
| Aborted_connects                  | 0          |
| Bytes_received                    | 155372598  |
| Bytes_sent                        | 1176560426 |
...
| Connections                       | 30023      |
| Created_tmp_disk_tables           | 0          |
| Created_tmp_files                 | 3          |
| Created_tmp_tables                | 2          |
...
| Threads_created                   | 217        |
| Threads_running                   | 88         |
| Uptime                            | 1389872    |
```

```
+---------------------------------+-----------+
```

Many status variables are reset to 0 by the `FLUSH STATUS` statement.

This section provides a description of each status variable. For a status variable summary, see Section 5.1.5, “Server Status Variable Reference”. For information about status variables specific to NDB Cluster, see NDB Cluster Status Variables.

The status variables have the following meanings.

- `Aborted_clients`

  The number of connections that were aborted because the client died without closing the connection properly. See Section B.3.2.10, “Communication Errors and Aborted Connections”.

- `Aborted_connects`

  The number of failed attempts to connect to the MySQL server. See Section B.3.2.10, “Communication Errors and Aborted Connections”.

  For additional connection-related information, check the `Connection_errors_xxx` status variables and the `host_cache` table.

- `Binlog_cache_disk_use`

  The number of transactions that used the temporary binary log cache but that exceeded the value of `binlog_cache_size` and used a temporary file to store statements from the transaction.

  The number of nontransactional statements that caused the binary log transaction cache to be written to disk is tracked separately in the `Binlog_stmt_cache_disk_use` status variable.

- `Binlog_cache_use`

  The number of transactions that used the binary log cache.

- `Binlog_stmt_cache_disk_use`

  The number of nontransaction statements that used the binary log statement cache but that exceeded the value of `binlog_stmt_cache_size` and used a temporary file to store those statements.

- `Binlog_stmt_cache_use`

  The number of nontransactional statements that used the binary log statement cache.

- `Bytes_received`

  The number of bytes received from all clients.

- `Bytes_sent`

  The number of bytes sent to all clients.

- `Com_xxx`

  The `Com_xxx` statement counter variables indicate the number of times each *`xxx`* statement has been executed. There is one status variable for each type of statement. For example, `Com_delete` and `Com_update` count `DELETE` and `UPDATE` statements, respectively. `Com_delete_multi` and `Com_update_multi` are similar but apply to `DELETE` and `UPDATE` statements that use multiple-table syntax.

  If a query result is returned from query cache, the server increments the `Qcache_hits` status variable, not `Com_select`. See Section 8.10.3.4, “Query Cache Status and Maintenance”.

All `Com_stmt_xxx` variables are increased even if a prepared statement argument is unknown or an error occurred during execution. In other words, their values correspond to the number of requests issued, not to the number of requests successfully completed.

The `Com_stmt_xxx` status variables are as follows:

- `Com_stmt_prepare`
- `Com_stmt_execute`
- `Com_stmt_fetch`
- `Com_stmt_send_long_data`
- `Com_stmt_reset`
- `Com_stmt_close`

Those variables stand for prepared statement commands. Their names refer to the `COM_xxx` command set used in the network layer. In other words, their values increase whenever prepared statement API calls such as `mysql_stmt_prepare()`, `mysql_stmt_execute()`, and so forth are executed. However, `Com_stmt_prepare`, `Com_stmt_execute` and `Com_stmt_close` also increase for `PREPARE`, `EXECUTE`, or `DEALLOCATE PREPARE`, respectively. Additionally, the values of the older statement counter variables `Com_prepare_sql`, `Com_execute_sql`, and `Com_dealloc_sql` increase for the `PREPARE`, `EXECUTE`, and `DEALLOCATE PREPARE` statements. `Com_stmt_fetch` stands for the total number of network round-trips issued when fetching from cursors.

`Com_stmt_reprepare` indicates the number of times statements were automatically reprepared by the server after metadata changes to tables or views referred to by the statement. A reprepare operation increments `Com_stmt_reprepare`, and also `Com_stmt_prepare`.

- `Compression`

  Whether the client connection uses compression in the client/server protocol.

- `Connection_errors_xxx`

  These variables provide information about errors that occur during the client connection process. They are global only and represent error counts aggregated across connections from all hosts. These variables track errors not accounted for by the host cache (see Section 5.1.11.2, “DNS Lookups and the Host Cache”), such as errors that are not associated with TCP connections, occur very early in the connection process (even before an IP address is known), or are not specific to any particular IP address (such as out-of-memory conditions).

  - `Connection_errors_accept`

    The number of errors that occurred during calls to `accept()` on the listening port.

  - `Connection_errors_internal`

    The number of connections refused due to internal errors in the server, such as failure to start a new thread or an out-of-memory condition.

  - `Connection_errors_max_connections`

    The number of connections refused because the server `max_connections` limit was reached.

  - `Connection_errors_peer_address`

    The number of errors that occurred while searching for connecting client IP addresses.

  - `Connection_errors_select`

    The number of errors that occurred during calls to `select()` or `poll()` on the listening port. (Failure of this operation does not necessarily means a client connection was rejected.)

  - `Connection_errors_tcpwrap`

    The number of connections refused by the `libwrap` library.

- `Connections`

  The number of connection attempts (successful or not) to the MySQL server.

- `Created_tmp_disk_tables`

  The number of internal on-disk temporary tables created by the server while executing statements.

  You can compare the number of internal on-disk temporary tables created to the total number of internal temporary tables created by comparing `Created_tmp_disk_tables` and `Created_tmp_tables` values.

  See also Section 8.4.4, “Internal Temporary Table Use in MySQL”.

- `Created_tmp_files`

  How many temporary files `mysqld` has created.

- `Created_tmp_tables`

  The number of internal temporary tables created by the server while executing statements.

  You can compare the number of internal on-disk temporary tables created to the total number of internal temporary tables created by comparing `Created_tmp_disk_tables` and `Created_tmp_tables` values.

  See also Section 8.4.4, “Internal Temporary Table Use in MySQL”.

  Each invocation of the `SHOW STATUS` statement uses an internal temporary table and increments the global `Created_tmp_tables` value.

- `Delayed_errors`

  The number of rows written with `INSERT DELAYED` for which some error occurred (probably `duplicate key`).

  This status variable is deprecated (because `DELAYED` inserts are deprecated); expect it to be removed in a future release.

- `Delayed_insert_threads`

  The number of `INSERT DELAYED` handler threads in use for nontransactional tables.

  This status variable is deprecated (because `DELAYED` inserts are deprecated); expect it to be removed in a future release.

- `Delayed_writes`

  The number of `INSERT DELAYED` rows written to nontransactional tables.

  This status variable is deprecated (because `DELAYED` inserts are deprecated); expect it to be removed in a future release.

- `Flush_commands`

  The number of times the server flushes tables, whether because a user executed a `FLUSH TABLES` statement or due to internal server operation. It is also incremented by receipt of a `COM_REFRESH` packet. This is in contrast to `Com_flush`, which indicates how many `FLUSH` statements have been executed, whether `FLUSH TABLES`, `FLUSH LOGS`, and so forth.

- `Handler_commit`

  The number of internal `COMMIT` statements.

- `Handler_delete`

  The number of times that rows have been deleted from tables.

- `Handler_external_lock`

  The server increments this variable for each call to its `external_lock()` function, which generally occurs at the beginning and end of access to a table instance. There might be differences among storage engines. This variable can be used, for example, to discover for a statement that accesses a partitioned table how many partitions were pruned before locking occurred: Check how much the counter increased for the statement, subtract 2 (2 calls for the table itself), then divide by 2 to get the number of partitions locked.

- `Handler_mrr_init`

  The number of times the server uses a storage engine's own Multi-Range Read implementation for table access.

- `Handler_prepare`

  A counter for the prepare phase of two-phase commit operations.

- `Handler_read_first`

  The number of times the first entry in an index was read. If this value is high, it suggests that the server is doing a lot of full index scans (for example, `SELECT col1 FROM foo`, assuming that `col1` is indexed).

- `Handler_read_key`

  The number of requests to read a row based on a key. If this value is high, it is a good indication that your tables are properly indexed for your queries.

- `Handler_read_last`

  The number of requests to read the last key in an index. With `ORDER BY`, the server issues a first-key request followed by several next-key requests, whereas with `ORDER BY DESC`, the server issues a last-key request followed by several previous-key requests.

- `Handler_read_next`

  The number of requests to read the next row in key order. This value is incremented if you are querying an index column with a range constraint or if you are doing an index scan.

- `Handler_read_prev`

  The number of requests to read the previous row in key order. This read method is mainly used to optimize `ORDER BY ... DESC`.

- `Handler_read_rnd`

  The number of requests to read a row based on a fixed position. This value is high if you are doing a lot of queries that require sorting of the result. You probably have a lot of queries that require MySQL to scan entire tables or you have joins that do not use keys properly.

- `Handler_read_rnd_next`

  The number of requests to read the next row in the data file. This value is high if you are doing a lot of table scans. Generally this suggests that your tables are not properly indexed or that your queries are not written to take advantage of the indexes you have.

- `Handler_rollback`

  The number of requests for a storage engine to perform a rollback operation.

- `Handler_savepoint`

  The number of requests for a storage engine to place a savepoint.

- `Handler_savepoint_rollback`

  The number of requests for a storage engine to roll back to a savepoint.

- `Handler_update`

  The number of requests to update a row in a table.

- `Handler_write`

  The number of requests to insert a row in a table.

- `Innodb_available_undo_logs`

  The total number of available `InnoDB` rollback segments. Supplements the `innodb_rollback_segments` system variable, which defines the number of active rollback segments.

- `Innodb_buffer_pool_dump_status`

  The progress of an operation to record the pages held in the `InnoDB` buffer pool, triggered by the setting of `innodb_buffer_pool_dump_at_shutdown` or `innodb_buffer_pool_dump_now`.

  For related information and examples, see Section 14.8.3.5, “Saving and Restoring the Buffer Pool State”.

- `Innodb_buffer_pool_load_status`

  The progress of an operation to warm up the `InnoDB` buffer pool by reading in a set of pages corresponding to an earlier point in time, triggered by the setting of `innodb_buffer_pool_load_at_startup` or `innodb_buffer_pool_load_now`. If the operation introduces too much overhead, you can cancel it by setting `innodb_buffer_pool_load_abort`.

  For related information and examples, see Section 14.8.3.5, “Saving and Restoring the Buffer Pool State”.

- `Innodb_buffer_pool_bytes_data`

  The total number of bytes in the `InnoDB` buffer pool containing data. The number includes both dirty and clean pages. For more accurate memory usage calculations than with `Innodb_buffer_pool_pages_data`, when compressed tables cause the buffer pool to hold pages of different sizes.

- `Innodb_buffer_pool_pages_data`

  The number of pages in the `InnoDB` buffer pool containing data. The number includes both dirty and clean pages. When using compressed tables, the reported `Innodb_buffer_pool_pages_data` value may be larger than `Innodb_buffer_pool_pages_total` (Bug #59550).

- `Innodb_buffer_pool_bytes_dirty`

  The total current number of bytes held in dirty pages in the `InnoDB` buffer pool. For more accurate memory usage calculations than with `Innodb_buffer_pool_pages_dirty`, when compressed tables cause the buffer pool to hold pages of different sizes.

- `Innodb_buffer_pool_pages_dirty`

  The current number of dirty pages in the `InnoDB` buffer pool.

- `Innodb_buffer_pool_pages_flushed`

  The number of requests to flush pages from the `InnoDB` buffer pool.

- `Innodb_buffer_pool_pages_free`

  The number of free pages in the `InnoDB` buffer pool.

- `Innodb_buffer_pool_pages_latched`

  The number of latched pages in the `InnoDB` buffer pool. These are pages currently being read or written, or that cannot be flushed or removed for some other reason. Calculation of this variable is expensive, so it is available only when the `UNIV_DEBUG` system is defined at server build time.

- `Innodb_buffer_pool_pages_misc`

  The number of pages in the `InnoDB` buffer pool that are busy because they have been allocated for administrative overhead, such as row locks or the adaptive hash index. This value can also be calculated as `Innodb_buffer_pool_pages_total` − `Innodb_buffer_pool_pages_free` − `Innodb_buffer_pool_pages_data`. When using compressed tables, `Innodb_buffer_pool_pages_misc` may report an out-of-bounds value (Bug #59550).

- `Innodb_buffer_pool_pages_total`

  The total size of the `InnoDB` buffer pool, in pages. When using compressed tables, the reported `Innodb_buffer_pool_pages_data` value may be larger than `Innodb_buffer_pool_pages_total` (Bug #59550)

- `Innodb_buffer_pool_read_ahead`

  The number of pages read into the `InnoDB` buffer pool by the read-ahead background thread.

- `Innodb_buffer_pool_read_ahead_evicted`

  The number of pages read into the `InnoDB` buffer pool by the read-ahead background thread that were subsequently evicted without having been accessed by queries.

- `Innodb_buffer_pool_read_ahead_rnd`

  The number of “random” read-aheads initiated by `InnoDB`. This happens when a query scans a large portion of a table but in random order.

- `Innodb_buffer_pool_read_requests`

  The number of logical read requests.

- `Innodb_buffer_pool_reads`

  The number of logical reads that `InnoDB` could not satisfy from the buffer pool, and had to read directly from disk.

- `Innodb_buffer_pool_wait_free`

  Normally, writes to the `InnoDB` buffer pool happen in the background. When `InnoDB` needs to read or create a page and no clean pages are available, `InnoDB` flushes some dirty pages first and waits for that operation to finish. This counter counts instances of these waits. If `innodb_buffer_pool_size` has been set properly, this value should be small.

- `Innodb_buffer_pool_write_requests`

  The number of writes done to the `InnoDB` buffer pool.

- `Innodb_data_fsyncs`

  The number of `fsync()` operations so far. The frequency of `fsync()` calls is influenced by the setting of the `innodb_flush_method` configuration option.

- `Innodb_data_pending_fsyncs`

  The current number of pending `fsync()` operations. The frequency of `fsync()` calls is influenced by the setting of the `innodb_flush_method` configuration option.

- `Innodb_data_pending_reads`

  The current number of pending reads.

- `Innodb_data_pending_writes`

  The current number of pending writes.

- `Innodb_data_read`

  The amount of data read since the server was started (in bytes).

- `Innodb_data_reads`

  The total number of data reads (OS file reads).

- `Innodb_data_writes`

  The total number of data writes.

- `Innodb_data_written`

  The amount of data written so far, in bytes.

- `Innodb_dblwr_pages_written`

  The number of pages that have been written to the doublewrite buffer. See Section 14.12.1, “InnoDB Disk I/O”.

- `Innodb_dblwr_writes`

  The number of doublewrite operations that have been performed. See Section 14.12.1, “InnoDB Disk I/O”.

- `Innodb_have_atomic_builtins`

  Indicates whether the server was built with atomic instructions.

- `Innodb_log_waits`

  The number of times that the log buffer was too small and a wait was required for it to be flushed before continuing.

- `Innodb_log_write_requests`

  The number of write requests for the `InnoDB` redo log.

- `Innodb_log_writes`

  The number of physical writes to the `InnoDB` redo log file.

- `Innodb_num_open_files`

  The number of files `InnoDB` currently holds open.

- `Innodb_os_log_fsyncs`

  The number of `fsync()` writes done to the `InnoDB` redo log files.

- `Innodb_os_log_pending_fsyncs`

  The number of pending `fsync()` operations for the `InnoDB` redo log files.

- `Innodb_os_log_pending_writes`

  The number of pending writes to the `InnoDB` redo log files.

- `Innodb_os_log_written`

  The number of bytes written to the `InnoDB` redo log files.

- `Innodb_page_size`

  `InnoDB` page size (default 16KB). Many values are counted in pages; the page size enables them to be easily converted to bytes.

- `Innodb_pages_created`

  The number of pages created by operations on `InnoDB` tables.

- `Innodb_pages_read`

  The number of pages read from the `InnoDB` buffer pool by operations on `InnoDB` tables.

- `Innodb_pages_written`

  The number of pages written by operations on `InnoDB` tables.

- `Innodb_row_lock_current_waits`

  The number of row locks currently being waited for by operations on `InnoDB` tables.

- `Innodb_row_lock_time`

  The total time spent in acquiring row locks for `InnoDB` tables, in milliseconds.

- `Innodb_row_lock_time_avg`

  The average time to acquire a row lock for `InnoDB` tables, in milliseconds.

- `Innodb_row_lock_time_max`

  The maximum time to acquire a row lock for `InnoDB` tables, in milliseconds.

- `Innodb_row_lock_waits`

  The number of times operations on `InnoDB` tables had to wait for a row lock.

- `Innodb_rows_deleted`

  The number of rows deleted from `InnoDB` tables.

- `Innodb_rows_inserted`

  The number of rows inserted into `InnoDB` tables.

- `Innodb_rows_read`

  The number of rows read from `InnoDB` tables.

- `Innodb_rows_updated`

  The number of rows updated in `InnoDB` tables.

- `Innodb_truncated_status_writes`

  The number of times output from the `SHOW ENGINE INNODB STATUS` statement has been truncated.

- `Key_blocks_not_flushed`

  The number of key blocks in the `MyISAM` key cache that have changed but have not yet been flushed to disk.

- `Key_blocks_unused`

  The number of unused blocks in the `MyISAM` key cache. You can use this value to determine how much of the key cache is in use; see the discussion of `key_buffer_size` in Section 5.1.7, “Server System Variables”.

- `Key_blocks_used`

  The number of used blocks in the `MyISAM` key cache. This value is a high-water mark that indicates the maximum number of blocks that have ever been in use at one time.

- `Key_read_requests`

  The number of requests to read a key block from the `MyISAM` key cache.

- `Key_reads`

  The number of physical reads of a key block from disk into the `MyISAM` key cache. If `Key_reads` is large, then your `key_buffer_size` value is probably too small. The cache miss rate can be calculated as `Key_reads`/`Key_read_requests`.

- `Key_write_requests`

  The number of requests to write a key block to the `MyISAM` key cache.

- `Key_writes`

  The number of physical writes of a key block from the `MyISAM` key cache to disk.

- `Last_query_cost`

  The total cost of the last compiled query as computed by the query optimizer. This is useful for comparing the cost of different query plans for the same query. The default value of 0 means that no query has been compiled yet. The default value is 0. `Last_query_cost` has session scope.

`Last_query_cost` can be computed accurately only for simple, “flat” queries, but not for complex queries such as those containing subqueries or `UNION`. For the latter, the value is set to 0.

- `Last_query_partial_plans`

  The number of iterations the query optimizer made in execution plan construction for the previous query. `Last_query_cost` has session scope.

- `Max_used_connections`

  The maximum number of connections that have been in use simultaneously since the server started.

- `Not_flushed_delayed_rows`

  The number of rows waiting to be written to nontransactional tables in `INSERT DELAYED` queues.

  This status variable is deprecated (because `DELAYED` inserts are deprecated); expect it to be removed in a future release.

- `Open_files`

  The number of files that are open. This count includes regular files opened by the server. It does not include other types of files such as sockets or pipes. Also, the count does not include files that storage engines open using their own internal functions rather than asking the server level to do so.

- `Open_streams`

  The number of streams that are open (used mainly for logging).

- `Open_table_definitions`

  The number of cached `.frm` files.

- `Open_tables`

  The number of tables that are open.

- `Opened_files`

  The number of files that have been opened with `my_open()` (a `mysys` library function). Parts of the server that open files without using this function do not increment the count.

- `Opened_table_definitions`

  The number of `.frm` files that have been cached.

- `Opened_tables`

  The number of tables that have been opened. If `Opened_tables` is big, your `table_open_cache` value is probably too small.

- `Performance_schema_`*`xxx`*

  Performance Schema status variables are listed in Section 22.16, “Performance Schema Status Variables”. These variables provide information about instrumentation that could not be loaded or created due to memory constraints.

- `Prepared_stmt_count`

  The current number of prepared statements. (The maximum number of statements is given by the `max_prepared_stmt_count` system variable.)

- `Qcache_free_blocks`

The number of free memory blocks in the query cache.

- `Qcache_free_memory`

  The amount of free memory for the query cache.

- `Qcache_hits`

  The number of query cache hits.

  The discussion at the beginning of this section indicates how to relate this statement-counting status variable to other such variables.

- `Qcache_inserts`

  The number of queries added to the query cache.

- `Qcache_lowmem_prunes`

  The number of queries that were deleted from the query cache because of low memory.

- `Qcache_not_cached`

  The number of noncached queries (not cacheable, or not cached due to the `query_cache_type` setting).

- `Qcache_queries_in_cache`

  The number of queries registered in the query cache.

- `Qcache_total_blocks`

  The total number of blocks in the query cache.

- `Queries`

  The number of statements executed by the server. This variable includes statements executed within stored programs, unlike the `Questions` variable. It does not count `COM_PING` or `COM_STATISTICS` commands.

  The discussion at the beginning of this section indicates how to relate this statement-counting status variable to other such variables.

- `Questions`

  The number of statements executed by the server. This includes only statements sent to the server by clients and not statements executed within stored programs, unlike the `Queries` variable. This variable does not count `COM_PING`, `COM_STATISTICS`, `COM_STMT_PREPARE`, `COM_STMT_CLOSE`, or `COM_STMT_RESET` commands.

  The discussion at the beginning of this section indicates how to relate this statement-counting status variable to other such variables.

- `Rpl_semi_sync_master_clients`

  The number of semisynchronous replicas.

  This variable is available only if the source-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_net_avg_wait_time`

  The average time in microseconds the source waited for a replica reply.

This variable is available only if the source-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_net_wait_time`

  The total time in microseconds the source waited for replica replies.

  This variable is available only if the source-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_net_waits`

  The total number of times the source waited for replica replies.

  This variable is available only if the source-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_no_times`

  The number of times the source turned off semisynchronous replication.

  This variable is available only if the source-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_no_tx`

  The number of commits that were not acknowledged successfully by a replica.

  This variable is available only if the source-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_status`

  Whether semisynchronous replication currently is operational on the source. The value is `ON` if the plugin has been enabled and a commit acknowledgment has occurred. It is `OFF` if the plugin is not enabled or the source has fallen back to asynchronous replication due to commit acknowledgment timeout.

  This variable is available only if the source-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_timefunc_failures`

  The number of times the source failed when calling time functions such as `gettimeofday()`.

  This variable is available only if the source-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_tx_avg_wait_time`

  The average time in microseconds the source waited for each transaction.

  This variable is available only if the source-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_tx_wait_time`

  The total time in microseconds the source waited for transactions.

  This variable is available only if the source-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_tx_waits`

  The total number of times the source waited for transactions.

  This variable is available only if the source-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_wait_pos_backtraverse`

  The total number of times the source waited for an event with binary coordinates lower than events waited for previously. This can occur when the order in which transactions start waiting for a reply is different from the order in which their binary log events are written.

  This variable is available only if the source-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_wait_sessions`

  The number of sessions currently waiting for replica replies.

  This variable is available only if the source-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_master_yes_tx`

  The number of commits that were acknowledged successfully by a replica.

  This variable is available only if the master-side semisynchronous replication plugin is installed.

- `Rpl_semi_sync_slave_status`

  Whether semisynchronous replication currently is operational on the replica. This is `ON` if the plugin has been enabled and the replica I/O thread is running, `OFF` otherwise.

  This variable is available only if the replica-side semisynchronous replication plugin is installed.

- `Rsa_public_key`

  This variable is available if MySQL was compiled using OpenSSL (see Section 6.3.4, “SSL Library-Dependent Capabilities”). Its value is the public key used by the `sha256_password` authentication plugin for RSA key pair-based password exchange. The value is nonempty only if the server successfully initializes the private and public keys in the files named by the `sha256_password_private_key_path` and `sha256_password_public_key_path` system variables. The value of `Rsa_public_key` comes from the latter file.

  For information about `sha256_password`, see Section 6.4.1.4, “SHA-256 Pluggable Authentication”.

- `Select_full_join`

  The number of joins that perform table scans because they do not use indexes. If this value is not 0, you should carefully check the indexes of your tables.

- `Select_full_range_join`

  The number of joins that used a range search on a reference table.

- `Select_range`

  The number of joins that used ranges on the first table. This is normally not a critical issue even if the value is quite large.

- `Select_range_check`

  The number of joins without keys that check for key usage after each row. If this is not 0, you should carefully check the indexes of your tables.

- `Select_scan`

  The number of joins that did a full scan of the first table.

- `Slave_heartbeat_period`

Shows the replication heartbeat interval (in seconds) on a replica.

- `Slave_last_heartbeat`

  Shows when the most recent heartbeat signal was received by a replica, as a `TIMESTAMP` value.

- `Slave_open_temp_tables`

  The number of temporary tables that the replica SQL thread currently has open. If the value is greater than zero, it is not safe to shut down the replica; see Section 17.4.1.29, “Replication and Temporary Tables”.

- `Slave_received_heartbeats`

  This counter increments with each replication heartbeat received by a replica since the last time that the replica was restarted or reset, or a `CHANGE MASTER TO` statement was issued.

- `Slave_retried_transactions`

  The total number of times since startup that the replica SQL thread has retried transactions.

- `Slave_rows_last_search_algorithm_used`

  The search algorithm that was most recently used by this replica to locate rows for row-based replication. The result shows whether the replica used indexes, a table scan, or hashing as the search algorithm for the last transaction executed on any channel.

  The method used depends on the setting for the `slave_rows_search_algorithms` system variable, and the keys that are available on the relevant table.

  This variable is available only for debug builds of MySQL.

- `Slave_running`

  This is `ON` if this server is a replica that is connected to a replication source, and both the I/O and SQL threads are running; otherwise, it is `OFF`.

- `Slow_launch_threads`

  The number of threads that have taken more than `slow_launch_time` seconds to create.

- `Slow_queries`

  The number of queries that have taken more than `long_query_time` seconds. This counter increments regardless of whether the slow query log is enabled. For information about that log, see Section 5.4.5, “The Slow Query Log”.

- `Sort_merge_passes`

  The number of merge passes that the sort algorithm has had to do. If this value is large, you should consider increasing the value of the `sort_buffer_size` system variable.

- `Sort_range`

  The number of sorts that were done using ranges.

- `Sort_rows`

  The number of sorted rows.

- `Sort_scan`

  The number of sorts that were done by scanning the table.

- `Ssl_accept_renegotiates`

  The number of negotiates needed to establish the connection.

- `Ssl_accepts`

  The number of accepted SSL connections.

- `Ssl_callback_cache_hits`

  The number of callback cache hits.

- `Ssl_cipher`

  The current encryption cipher (empty for unencrypted connections).

- `Ssl_cipher_list`

  The list of possible SSL ciphers (empty for non-SSL connections).

- `Ssl_client_connects`

  The number of SSL connection attempts to an SSL-enabled source.

- `Ssl_connect_renegotiates`

  The number of negotiates needed to establish the connection to an SSL-enabled source.

- `Ssl_ctx_verify_depth`

  The SSL context verification depth (how many certificates in the chain are tested).

- `Ssl_ctx_verify_mode`

  The SSL context verification mode.

- `Ssl_default_timeout`

  The default SSL timeout.

- `Ssl_finished_accepts`

  The number of successful SSL connections to the server.

- `Ssl_finished_connects`

  The number of successful replica connections to an SSL-enabled source.

- `Ssl_server_not_after`

  The last date for which the SSL certificate is valid. To check SSL certificate expiration information, use this statement:

```
mysql> SHOW STATUS LIKE 'Ssl_server_not%';
+-----------------------+--------------------------+
| Variable_name         | Value                    |
+-----------------------+--------------------------+
| Ssl_server_not_after  | Apr 28 14:16:39 2025 GMT |
| Ssl_server_not_before | May  1 14:16:39 2015 GMT |
+-----------------------+--------------------------+
```

  In MySQL 5.6, the value is empty unless the connection uses SSL.

- `Ssl_server_not_before`

  The first date for which the SSL certificate is valid.

In MySQL 5.6, the value is empty unless the connection uses SSL.

- `Ssl_session_cache_hits`

  The number of SSL session cache hits.

- `Ssl_session_cache_misses`

  The number of SSL session cache misses.

- `Ssl_session_cache_mode`

  The SSL session cache mode.

- `Ssl_session_cache_overflows`

  The number of SSL session cache overflows.

- `Ssl_session_cache_size`

  The SSL session cache size.

- `Ssl_session_cache_timeouts`

  The number of SSL session cache timeouts.

- `Ssl_sessions_reused`

  How many SSL connections were reused from the cache.

- `Ssl_used_session_cache_entries`

  How many SSL session cache entries were used.

- `Ssl_verify_depth`

  The verification depth for replication SSL connections.

- `Ssl_verify_mode`

  The verification mode used by the server for a connection that uses SSL. The value is a bitmask; bits are defined in the `openssl/ssl.h` header file:

```
# define SSL_VERIFY_NONE                 0x00
# define SSL_VERIFY_PEER                 0x01
# define SSL_VERIFY_FAIL_IF_NO_PEER_CERT 0x02
# define SSL_VERIFY_CLIENT_ONCE          0x04
```

  `SSL_VERIFY_PEER` indicates that the server asks for a client certificate. If the client supplies one, the server performs verification and proceeds only if verification is successful. `SSL_VERIFY_CLIENT_ONCE` indicates that a request for the client certificate is done only in the initial handshake.

- `Ssl_version`

  The SSL protocol version of the connection (for example, TLSv1). If the connection is not encrypted, the value is empty.

- `Table_locks_immediate`

  The number of times that a request for a table lock could be granted immediately.

- `Table_locks_waited`

The number of times that a request for a table lock could not be granted immediately and a wait was needed. If this is high and you have performance problems, you should first optimize your queries, and then either split your table or tables or use replication.

- `Table_open_cache_hits`

  The number of hits for open tables cache lookups.

- `Table_open_cache_misses`

  The number of misses for open tables cache lookups.

- `Table_open_cache_overflows`

  The number of overflows for the open tables cache. This is the number of times, after a table is opened or closed, a cache instance has an unused entry and the size of the instance is larger than `table_open_cache` / `table_open_cache_instances`.

- `Tc_log_max_pages_used`

  For the memory-mapped implementation of the log that is used by `mysqld` when it acts as the transaction coordinator for recovery of internal XA transactions, this variable indicates the largest number of pages used for the log since the server started. If the product of `Tc_log_max_pages_used` and `Tc_log_page_size` is always significantly less than the log size, the size is larger than necessary and can be reduced. (The size is set by the `--log-tc-size` option. This variable is unused: It is unneeded for binary log-based recovery, and the memory-mapped recovery log method is not used unless the number of storage engines that are capable of two-phase commit and that support XA transactions is greater than one. (`InnoDB` is the only applicable engine.)

- `Tc_log_page_size`

  The page size used for the memory-mapped implementation of the XA recovery log. The default value is determined using `getpagesize()`. This variable is unused for the same reasons as described for `Tc_log_max_pages_used`.

- `Tc_log_page_waits`

  For the memory-mapped implementation of the recovery log, this variable increments each time the server was not able to commit a transaction and had to wait for a free page in the log. If this value is large, you might want to increase the log size (with the `--log-tc-size` option). For binary log-based recovery, this variable increments each time the binary log cannot be closed because there are two-phase commits in progress. (The close operation waits until all such transactions are finished.)

- `Threads_cached`

  The number of threads in the thread cache.

- `Threads_connected`

  The number of currently open connections.

- `Threads_created`

  The number of threads created to handle connections. If `Threads_created` is big, you may want to increase the `thread_cache_size` value. The cache miss rate can be calculated as `Threads_created`/`Connections`.

- `Threads_running`

  The number of threads that are not sleeping.

- `Uptime`

  The number of seconds that the server has been up.

- `Uptime_since_flush_status`

  The number of seconds since the most recent `FLUSH STATUS` statement.

5.1.10 Server SQL Modes

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

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

- Setting the SQL Mode
- The Most Important SQL Modes
- Full List of SQL Modes
- Combination SQL Modes
- Strict SQL Mode

For answers to questions often asked about server SQL modes in MySQL, see Section A.3, “MySQL 5.6 FAQ: Server SQL Mode”.

When working with `InnoDB` tables, consider also the `innodb_strict_mode` system variable. It enables additional error checks for `InnoDB` tables.

Setting the SQL Mode

The default SQL mode is `NO_ENGINE_SUBSTITUTION`.

To set the SQL mode at server startup, use the `--sql-mode="modes"` option on the command line, or `sql-mode="modes"` in an option file such as `my.cnf` (Unix operating systems) or `my.ini` (Windows). *`modes`* is a list of different modes separated by commas. To clear the SQL mode explicitly, set it to an empty string using `--sql-mode=""` on the command line, or `sql-mode=""` in an option file.

Note

MySQL installation programs may configure the SQL mode during the installation process. For example, `mysql_install_db` creates a default option file named `my.cnf` in the base installation directory. This file contains a line that sets the SQL mode; see Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”.

If the SQL mode differs from the default or from what you expect, check for a setting in an option file that the server reads at startup.

To change the SQL mode at runtime, set the global or session `sql_mode` system variable using a `SET` statement:

```
SET GLOBAL sql_mode = 'modes';
SET SESSION sql_mode = 'modes';
```

Setting the GLOBAL variable requires the SUPER privilege and affects the operation of all clients that connect from that time on. Setting the SESSION variable affects only the current client. Each client can change its session sql_mode value at any time.

To determine the current global or session sql_mode setting, select its value:

```
SELECT @@GLOBAL.sql_mode;
SELECT @@SESSION.sql_mode;
```

Important

SQL mode and user-defined partitioning. Changing the server SQL mode after creating and inserting data into partitioned tables can cause major changes in the behavior of such tables, and could lead to loss or corruption of data. It is strongly recommended that you never change the SQL mode once you have created tables employing user-defined partitioning.

When replicating partitioned tables, differing SQL modes on the source and replica can also lead to problems. For best results, you should always use the same server SQL mode on the source and replica.

For more information, see Section 19.6, “Restrictions and Limitations on Partitioning”.

The Most Important SQL Modes

The most important sql_mode values are probably these:

- ANSI

  This mode changes syntax and behavior to conform more closely to standard SQL. It is one of the special combination modes listed at the end of this section.

- STRICT_TRANS_TABLES

  If a value could not be inserted as given into a transactional table, abort the statement. For a nontransactional table, abort the statement if the value occurs in a single-row statement or the first row of a multiple-row statement. More details are given later in this section.

- TRADITIONAL

  Make MySQL behave like a “traditional” SQL database system. A simple description of this mode is “give an error instead of a warning” when inserting an incorrect value into a column. It is one of the special combination modes listed at the end of this section.

  Note

  With TRADITIONAL mode enabled, an INSERT or UPDATE aborts as soon as an error occurs. If you are using a nontransactional storage engine, this may not be what you want because data changes made prior to the error may not be rolled back, resulting in a “partially done” update.

When this manual refers to “strict mode,” it means a mode with either or both STRICT_TRANS_TABLES or STRICT_ALL_TABLES enabled.

Full List of SQL Modes

The following list describes all supported SQL modes:

- ALLOW_INVALID_DATES

Do not perform full checking of dates. Check only that the month is in the range from 1 to 12 and the day is in the range from 1 to 31. This may be useful for Web applications that obtain year, month, and day in three different fields and store exactly what the user inserted, without date validation. This mode applies to `DATE` and `DATETIME` columns. It does not apply `TIMESTAMP` columns, which always require a valid date.

With `ALLOW_INVALID_DATES` disabled, the server requires that month and day values be legal, and not merely in the range 1 to 12 and 1 to 31, respectively. With strict mode disabled, invalid dates such as `'2004-04-31'` are converted to `'0000-00-00'` and a warning is generated. With strict mode enabled, invalid dates generate an error. To permit such dates, enable `ALLOW_INVALID_DATES`.

- `ANSI_QUOTES`

  Treat `"` as an identifier quote character (like the `` ` `` quote character) and not as a string quote character. You can still use `` ` `` to quote identifiers with this mode enabled. With `ANSI_QUOTES` enabled, you cannot use double quotation marks to quote literal strings because they are interpreted as identifiers.

- `ERROR_FOR_DIVISION_BY_ZERO`

  The `ERROR_FOR_DIVISION_BY_ZERO` mode affects handling of division by zero, which includes `MOD(N,0)`. For data-change operations (`INSERT`, `UPDATE`), its effect also depends on whether strict SQL mode is enabled.

  - If this mode is not enabled, division by zero inserts `NULL` and produces no warning.
  - If this mode is enabled, division by zero inserts `NULL` and produces a warning.
  - If this mode and strict mode are enabled, division by zero produces an error, unless `IGNORE` is given as well. For `INSERT IGNORE` and `UPDATE IGNORE`, division by zero inserts `NULL` and produces a warning.

  For `SELECT`, division by zero returns `NULL`. Enabling `ERROR_FOR_DIVISION_BY_ZERO` causes a warning to be produced as well, regardless of whether strict mode is enabled.

  As of MySQL 5.6.17, `ERROR_FOR_DIVISION_BY_ZERO` is deprecated and setting the `sql_mode` value to include it generates a warning.

- `HIGH_NOT_PRECEDENCE`

  The precedence of the `NOT` operator is such that expressions such as `NOT a BETWEEN b AND c` are parsed as `NOT (a BETWEEN b AND c)`. In some older versions of MySQL, the expression was parsed as `(NOT a) BETWEEN b AND c`. The old higher-precedence behavior can be obtained by enabling the `HIGH_NOT_PRECEDENCE` SQL mode.

```
mysql> SET sql_mode = '';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
        -> 0
mysql> SET sql_mode = 'HIGH_NOT_PRECEDENCE';
mysql> SELECT NOT 1 BETWEEN -5 AND 5;
        -> 1
```

- `IGNORE_SPACE`

  Permit spaces between a function name and the `(` character. This causes built-in function names to be treated as reserved words. As a result, identifiers that are the same as function names must be quoted as described in Section 9.2, “Schema Object Names”. For example, because there is a `COUNT()` function, the use of `count` as a table name in the following statement causes an error:

```
mysql> CREATE TABLE count (i INT);
ERROR 1064 (42000): You have an error in your SQL syntax
```

The table name should be quoted:

```
mysql> CREATE TABLE `count` (i INT);
Query OK, 0 rows affected (0.00 sec)
```

The IGNORE_SPACE SQL mode applies to built-in functions, not to user-defined functions or stored functions. It is always permissible to have spaces after a UDF or stored function name, regardless of whether IGNORE_SPACE is enabled.

For further discussion of IGNORE_SPACE, see Section 9.2.5, “Function Name Parsing and Resolution”.

- NO_AUTO_CREATE_USER

  Prevent the GRANT statement from automatically creating new users if it would otherwise do so, unless authentication information is specified. The statement must specify a nonempty password using IDENTIFIED BY or an authentication plugin using IDENTIFIED WITH.

- NO_AUTO_VALUE_ON_ZERO

  NO_AUTO_VALUE_ON_ZERO affects handling of AUTO_INCREMENT columns. Normally, you generate the next sequence number for the column by inserting either NULL or 0 into it. NO_AUTO_VALUE_ON_ZERO suppresses this behavior for 0 so that only NULL generates the next sequence number.

  This mode can be useful if 0 has been stored in a table's AUTO_INCREMENT column. (Storing 0 is not a recommended practice, by the way.) For example, if you dump the table with mysqldump and then reload it, MySQL normally generates new sequence numbers when it encounters the 0 values, resulting in a table with contents different from the one that was dumped. Enabling NO_AUTO_VALUE_ON_ZERO before reloading the dump file solves this problem. For this reason, mysqldump automatically includes in its output a statement that enables NO_AUTO_VALUE_ON_ZERO.

- NO_BACKSLASH_ESCAPES

  Disable the use of the backslash character (\) as an escape character within strings and identifiers. With this mode enabled, backslash becomes an ordinary character like any other.

- NO_DIR_IN_CREATE

  When creating a table, ignore all INDEX DIRECTORY and DATA DIRECTORY directives. This option is useful on replica servers.

- NO_ENGINE_SUBSTITUTION

  Control automatic substitution of the default storage engine when a statement such as CREATE TABLE or ALTER TABLE specifies a storage engine that is disabled or not compiled in.

  Because storage engines can be pluggable at runtime, unavailable engines are treated the same way:

  With NO_ENGINE_SUBSTITUTION disabled, for CREATE TABLE the default engine is used and a warning occurs if the desired engine is unavailable. For ALTER TABLE, a warning occurs and the table is not altered.

  With NO_ENGINE_SUBSTITUTION enabled, an error occurs and the table is not created or altered if the desired engine is unavailable.

- NO_FIELD_OPTIONS

  Do not print MySQL-specific column options in the output of SHOW CREATE TABLE. This mode is used by mysqldump in portability mode.

- NO_KEY_OPTIONS

  Do not print MySQL-specific index options in the output of SHOW CREATE TABLE. This mode is used by mysqldump in portability mode.

- NO_TABLE_OPTIONS

  Do not print MySQL-specific table options (such as ENGINE) in the output of SHOW CREATE TABLE. This mode is used by mysqldump in portability mode.

- NO_UNSIGNED_SUBTRACTION

  Subtraction between integer values, where one is of type UNSIGNED, produces an unsigned result by default. If the result would otherwise have been negative, an error results:

```
mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT CAST(0 AS UNSIGNED) - 1;
ERROR 1690 (22003): BIGINT UNSIGNED value is out of range in '(cast(0 as unsigned) - 1)'
```

  If the NO_UNSIGNED_SUBTRACTION SQL mode is enabled, the result is negative:

```
mysql> SET sql_mode = 'NO_UNSIGNED_SUBTRACTION';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+--------------------------+
| CAST(0 AS UNSIGNED) - 1 |
+--------------------------+
|                       -1 |
+--------------------------+
```

  If the result of such an operation is used to update an UNSIGNED integer column, the result is clipped to the maximum value for the column type, or clipped to 0 if NO_UNSIGNED_SUBTRACTION is enabled. With strict SQL mode enabled, an error occurs and the column remains unchanged.

  When NO_UNSIGNED_SUBTRACTION is enabled, the subtraction result is signed, *even if any operand is unsigned*. For example, compare the type of column c2 in table t1 with that of column c2 in table t2:

```
mysql> SET sql_mode='';
mysql> CREATE TABLE test (c1 BIGINT UNSIGNED NOT NULL);
mysql> CREATE TABLE t1 SELECT c1 - 1 AS c2 FROM test;
mysql> DESCRIBE t1;
+-------+---------------------+------+-----+---------+-------+
| Field | Type                | Null | Key | Default | Extra |
+-------+---------------------+------+-----+---------+-------+
| c2    | bigint(21) unsigned | NO   |     | 0       |       |
+-------+---------------------+------+-----+---------+-------+

mysql> SET sql_mode='NO_UNSIGNED_SUBTRACTION';
mysql> CREATE TABLE t2 SELECT c1 - 1 AS c2 FROM test;
mysql> DESCRIBE t2;
+-------+------------+------+-----+---------+-------+
| Field | Type       | Null | Key | Default | Extra |
+-------+------------+------+-----+---------+-------+
| c2    | bigint(21) | NO   |     | 0       |       |
+-------+------------+------+-----+---------+-------+
```

  This means that BIGINT UNSIGNED is not 100% usable in all contexts. See Section 12.11, “Cast Functions and Operators”.

- NO_ZERO_DATE

  The NO_ZERO_DATE mode affects whether the server permits '0000-00-00' as a valid date. Its effect also depends on whether strict SQL mode is enabled.

  - If this mode is not enabled, '0000-00-00' is permitted and inserts produce no warning.
  - If this mode is enabled, '0000-00-00' is permitted and inserts produce a warning.
  - If this mode and strict mode are enabled, '0000-00-00' is not permitted and inserts produce an error, unless IGNORE is given as well. For INSERT IGNORE and UPDATE IGNORE, '0000-00-00' is permitted and inserts produce a warning.

  As of MySQL 5.6.17, NO_ZERO_DATE is deprecated and setting the sql_mode value to include it generates a warning.

- NO_ZERO_IN_DATE

  The NO_ZERO_IN_DATE mode affects whether the server permits dates in which the year part is nonzero but the month or day part is 0. (This mode affects dates such as '2010-00-01' or '2010-01-00', but not '0000-00-00'. To control whether the server permits '0000-00-00', use the NO_ZERO_DATE mode.) The effect of NO_ZERO_IN_DATE also depends on whether strict SQL mode is enabled.

  - If this mode is not enabled, dates with zero parts are permitted and inserts produce no warning.
  - If this mode is enabled, dates with zero parts are inserted as '0000-00-00' and produce a warning.
  - If this mode and strict mode are enabled, dates with zero parts are not permitted and inserts produce an error, unless IGNORE is given as well. For INSERT IGNORE and UPDATE IGNORE, dates with zero parts are inserted as '0000-00-00' and produce a warning.

  As of MySQL 5.6.17, NO_ZERO_IN_DATE is deprecated and setting the sql_mode value to include it generates a warning.

- ONLY_FULL_GROUP_BY

  Reject queries for which the select list, HAVING condition, or ORDER BY list refer to nonaggregated columns that are not named in the GROUP BY clause.

  A MySQL extension to standard SQL permits references in the HAVING clause to aliased expressions in the select list. Enabling ONLY_FULL_GROUP_BY disables this extension, thus requiring the HAVING clause to be written using unaliased expressions.

  For additional discussion and examples, see Section 12.19.3, “MySQL Handling of GROUP BY”.

- PAD_CHAR_TO_FULL_LENGTH

  By default, trailing spaces are trimmed from CHAR column values on retrieval. If PAD_CHAR_TO_FULL_LENGTH is enabled, trimming does not occur and retrieved CHAR values are padded to their full length. This mode does not apply to VARCHAR columns, for which trailing spaces are retained on retrieval.

```
mysql> CREATE TABLE t1 (c1 CHAR(10));
Query OK, 0 rows affected (0.37 sec)

mysql> INSERT INTO t1 (c1) VALUES('xy');
Query OK, 1 row affected (0.01 sec)

mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)
```

```
mysql> SELECT c1, CHAR_LENGTH(c1) FROM t1;
+------+-----------------+
| c1   | CHAR_LENGTH(c1) |
+------+-----------------+
| xy   |               2 |
+------+-----------------+
1 row in set (0.00 sec)

mysql> SET sql_mode = 'PAD_CHAR_TO_FULL_LENGTH';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT c1, CHAR_LENGTH(c1) FROM t1;
+------------+-----------------+
| c1         | CHAR_LENGTH(c1) |
+------------+-----------------+
| xy         |              10 |
+------------+-----------------+
1 row in set (0.00 sec)
```

- `PIPES_AS_CONCAT`

  Treat `||` as a string concatenation operator (same as `CONCAT()`) rather than as a synonym for `OR`.

- `REAL_AS_FLOAT`

  Treat `REAL` as a synonym for `FLOAT`. By default, MySQL treats `REAL` as a synonym for `DOUBLE`.

- `STRICT_ALL_TABLES`

  Enable strict SQL mode for all storage engines. Invalid data values are rejected. For details, see Strict SQL Mode.

- `STRICT_TRANS_TABLES`

  Enable strict SQL mode for transactional storage engines, and when possible for nontransactional storage engines. For details, see Strict SQL Mode.

Combination SQL Modes

The following special modes are provided as shorthand for combinations of mode values from the preceding list.

- `ANSI`

  Equivalent to `REAL_AS_FLOAT`, `PIPES_AS_CONCAT`, `ANSI_QUOTES`, `IGNORE_SPACE`.

  `ANSI` mode also causes the server to return an error for queries where a set function *`S`* with an outer reference *`S(outer_ref)`* cannot be aggregated in the outer query against which the outer reference has been resolved. This is such a query:

  ```
  SELECT * FROM t1 WHERE t1.a IN (SELECT MAX(t1.b) FROM t2 WHERE ...);
  ```

  Here, `MAX(t1.b)` cannot aggregated in the outer query because it appears in the `WHERE` clause of that query. Standard SQL requires an error in this situation. If `ANSI` mode is not enabled, the server treats *`S(outer_ref)`* in such queries the same way that it would interpret *`S(const)`*.

  See Section 1.7, “MySQL Standards Compliance”.

- `DB2`

  Equivalent to `PIPES_AS_CONCAT`, `ANSI_QUOTES`, `IGNORE_SPACE`, `NO_KEY_OPTIONS`, `NO_TABLE_OPTIONS`, `NO_FIELD_OPTIONS`.

- `MAXDB`

Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS, NO_AUTO_CREATE_USER.

- MSSQL

  Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS.

- MYSQL323

  Equivalent to MYSQL323, HIGH_NOT_PRECEDENCE. This means HIGH_NOT_PRECEDENCE plus some SHOW CREATE TABLE behaviors specific to MYSQL323:

  - TIMESTAMP column display does not include DEFAULT or ON UPDATE attributes.
  - String column display does not include character set and collation attributes. For CHAR and VARCHAR columns, if the collation is binary, BINARY is appended to the column type.
  - The ENGINE=*engine_name* table option displays as TYPE=*engine_name*.
  - For MEMORY tables, the storage engine is displayed as HEAP.

- MYSQL40

  Equivalent to MYSQL40, HIGH_NOT_PRECEDENCE. This means HIGH_NOT_PRECEDENCE plus some behaviors specific to MYSQL40. These are the same as for MYSQL323, except that SHOW CREATE TABLE does not display HEAP as the storage engine for MEMORY tables.

- ORACLE

  Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS, NO_AUTO_CREATE_USER.

- POSTGRESQL

  Equivalent to PIPES_AS_CONCAT, ANSI_QUOTES, IGNORE_SPACE, NO_KEY_OPTIONS, NO_TABLE_OPTIONS, NO_FIELD_OPTIONS.

- TRADITIONAL

  Equivalent to STRICT_TRANS_TABLES, STRICT_ALL_TABLES, NO_ZERO_IN_DATE, NO_ZERO_DATE, ERROR_FOR_DIVISION_BY_ZERO, NO_AUTO_CREATE_USER, and NO_ENGINE_SUBSTITUTION.

Strict SQL Mode

Strict mode controls how MySQL handles invalid or missing values in data-change statements such as INSERT or UPDATE. A value can be invalid for several reasons. For example, it might have the wrong data type for the column, or it might be out of range. A value is missing when a new row to be inserted does not contain a value for a non-NULL column that has no explicit DEFAULT clause in its definition. (For a NULL column, NULL is inserted if the value is missing.) Strict mode also affects DDL statements such as CREATE TABLE.

If strict mode is not in effect, MySQL inserts adjusted values for invalid or missing values and produces warnings (see Section 13.7.5.41, “SHOW WARNINGS Statement”). In strict mode, you can produce this behavior by using INSERT IGNORE or UPDATE IGNORE.

For statements such as SELECT that do not change data, invalid values generate a warning in strict mode, not an error.

As of MySQL 5.6.11, strict mode produces an error for attempts to create a key that exceeds the maximum key length. Previously, this resulted in a warning and truncation of the key to the maximum key length (the same as when strict mode is not enabled).

Strict mode does not affect whether foreign key constraints are checked. `foreign_key_checks` can be used for that. (See Section 5.1.7, “Server System Variables”.)

Strict SQL mode is in effect if either `STRICT_ALL_TABLES` or `STRICT_TRANS_TABLES` is enabled, although the effects of these modes differ somewhat:

- For transactional tables, an error occurs for invalid or missing values in a data-change statement when either `STRICT_ALL_TABLES` or `STRICT_TRANS_TABLES` is enabled. The statement is aborted and rolled back.
- For nontransactional tables, the behavior is the same for either mode if the bad value occurs in the first row to be inserted or updated: The statement is aborted and the table remains unchanged. If the statement inserts or modifies multiple rows and the bad value occurs in the second or later row, the result depends on which strict mode is enabled:
  - For `STRICT_ALL_TABLES`, MySQL returns an error and ignores the rest of the rows. However, because the earlier rows have been inserted or updated, the result is a partial update. To avoid this, use single-row statements, which can be aborted without changing the table.
  - For `STRICT_TRANS_TABLES`, MySQL converts an invalid value to the closest valid value for the column and inserts the adjusted value. If a value is missing, MySQL inserts the implicit default value for the column data type. In either case, MySQL generates a warning rather than an error and continues processing the statement. Implicit defaults are described in Section 11.5, “Data Type Default Values”.

Strict mode also affects handling of division by zero, zero dates, and zeros in dates, in conjunction with the `ERROR_FOR_DIVISION_BY_ZERO`, `NO_ZERO_DATE`, and `NO_ZERO_IN_DATE` modes. For details, see the descriptions of those modes.

5.1.11 Connection Management

This section describes how MySQL Server manages connections. This includes a description of the available connection interfaces, how the server uses connection handler threads, details about the administrative connection interface, and management of DNS lookups.

5.1.11.1 Connection Interfaces

This section describes aspects of how the MySQL server manages client connections.

- Network Interfaces and Connection Manager Threads
- Client Connection Thread Management
- Connection Volume Management

Network Interfaces and Connection Manager Threads

The server is capable of listening for client connections on multiple network interfaces. Connection manager threads handle client connection requests on the network interfaces that the server listens to:

- On all platforms, one manager thread handles TCP/IP connection requests.
- On Unix, the same manager thread also handles Unix socket file connection requests.
- On Windows, one manager thread handles shared-memory connection requests, and another handles named-pipe connection requests.

The server does not create threads to handle interfaces that it does not listen to. For example, a Windows server that does not have support for named-pipe connections enabled does not create a thread to handle them.

Client Connection Thread Management

Connection manager threads associate each client connection with a thread dedicated to it that handles authentication and request processing for that connection. Manager threads create a new thread when necessary but try to avoid doing so by consulting the thread cache first to see whether it contains a thread that can be used for the connection. When a connection ends, its thread is returned to the thread cache if the cache is not full.

In this connection thread model, there are as many threads as there are clients currently connected, which has some disadvantages when server workload must scale to handle large numbers of connections. For example, thread creation and disposal becomes expensive. Also, each thread requires server and kernel resources, such as stack space. To accommodate a large number of simultaneous connections, the stack size per thread must be kept small, leading to a situation where it is either too small or the server consumes large amounts of memory. Exhaustion of other resources can occur as well, and scheduling overhead can become significant.

MySQL Enterprise Edition includes a thread pool plugin that provides an alternative thread-handling model designed to reduce overhead and improve performance. It implements a thread pool that increases server performance by efficiently managing statement execution threads for large numbers of client connections. See Section 5.5.3, “MySQL Enterprise Thread Pool”.

To control and monitor how the server manages threads that handle client connections, several system and status variables are relevant. (See Section 5.1.7, “Server System Variables”, and Section 5.1.9, “Server Status Variables”.)

- The `thread_cache_size` system variable determines the thread cache size. By default, the server autosizes the value at startup, but it can be set explicitly to override this default. A value of 0 disables caching, which causes a thread to be set up for each new connection and disposed of when the connection terminates. To enable *`N`* inactive connection threads to be cached, set `thread_cache_size` to *`N`* at server startup or at runtime. A connection thread becomes inactive when the client connection with which it was associated terminates.

- To monitor the number of threads in the cache and how many threads have been created because a thread could not be taken from the cache, check the `Threads_cached` and `Threads_created` status variables.

- When the thread stack is too small, this limits the complexity of the SQL statements the server can handle, the recursion depth of stored procedures, and other memory-consuming actions. To set a stack size of *`N`* bytes for each thread, start the server with `thread_stack` set to *`N`*.

Connection Volume Management

To control the maximum number of clients the server permits to connect simultaneously, set the `max_connections` system variable at server startup or at runtime. It may be necessary to increase `max_connections` if more clients attempt to connect simultaneously then the server is configured to handle (see Section B.3.2.6, “Too many connections”).

`mysqld` actually permits `max_connections` + 1 client connections. The extra connection is reserved for use by accounts that have the `SUPER` privilege. By granting the privilege to administrators and not to normal users (who should not need it), an administrator who also has the `PROCESS` privilege can connect to the server and use `SHOW PROCESSLIST` to diagnose problems even if the maximum number of unprivileged clients are connected. See Section 13.7.5.30, “SHOW PROCESSLIST Statement”.

If the server refuses a connection because the `max_connections` limit is reached, it increments the `Connection_errors_max_connections` status variable.

The maximum number of connections MySQL supports (that is, the maximum value to which `max_connections` can be set) depends on several factors:

- The quality of the thread library on a given platform.
- The amount of RAM available.
- The amount of RAM is used for each connection.
- The workload from each connection.
- The desired response time.
- The number of file descriptors available.

Linux or Solaris should be able to support at least 500 to 1000 simultaneous connections routinely and as many as 10,000 connections if you have many gigabytes of RAM available and the workload from each is low or the response time target undemanding.

Increasing the `max_connections` value increases the number of file descriptors that `mysqld` requires. If the required number of descriptors are not available, the server reduces the value of `max_connections`. For comments on file descriptor limits, see Section 8.4.3.1, “How MySQL Opens and Closes Tables”.

Increasing the `open_files_limit` system variable may be necessary, which may also require raising the operating system limit on how many file descriptors can be used by MySQL. Consult your operating system documentation to determine whether it is possible to increase the limit and how to do so. See also Section B.3.2.17, “File Not Found and Similar Errors”.

5.1.11.2 DNS Lookups and the Host Cache

The MySQL server maintains an in-memory host cache that contains information about clients: IP address, host name, and error information. The Performance Schema `host_cache` table exposes the contents of the host cache so that it can be examined using `SELECT` statements. This may help you diagnose the causes of connection problems. See Section 22.12.10.1, “The host_cache Table”.

The following sections discuss how the host cache works, as well as other topics such as how to configure and monitor the cache.

- Host Cache Operation
- Configuring the Host Cache
- Monitoring the Host Cache
- Flushing the Host Cache
- Dealing with Blocked Hosts

Host Cache Operation

The server uses the host cache only for non-localhost TCP connections. It does not use the cache for TCP connections established using a loopback interface address (for example, `127.0.0.1` or `::1`), or for connections established using a Unix socket file, named pipe, or shared memory.

The server uses the host cache for several purposes:

- By caching the results of IP-to-host name lookups, the server avoids doing a Domain Name System (DNS) lookup for each client connection. Instead, for a given host, it needs to perform a lookup only for the first connection from that host.
- The cache contains information about errors that occur during the client connection process. Some errors are considered “blocking.” If too many of these occur successively from a given

host without a successful connection, the server blocks further connections from that host. The `max_connect_errors` system variable determines the permitted number of successive errors before blocking occurs.

For each applicable new client connection, the server uses the client IP address to check whether the client host name is in the host cache. If so, the server refuses or continues to process the connection request depending on whether or not the host is blocked. If the host is not in the cache, the server attempts to resolve the host name. First, it resolves the IP address to a host name and resolves that host name back to an IP address. Then it compares the result to the original IP address to ensure that they are the same. The server stores information about the result of this operation in the host cache. If the cache is full, the least recently used entry is discarded.

The server performs host name resolution using the `gethostbyaddr()` and `gethostbyname()` system calls.

The server handles entries in the host cache like this:

1. When the first TCP client connection reaches the server from a given IP address, a new cache entry is created to record the client IP, host name, and client lookup validation flag. Initially, the host name is set to `NULL` and the flag is false. This entry is also used for subsequent client TCP connections from the same originating IP.

2. If the validation flag for the client IP entry is false, the server attempts an IP-to-host name-to-IP DNS resolution. If that is successful, the host name is updated with the resolved host name and the validation flag is set to true. If resolution is unsuccessful, the action taken depends on whether the error is permanent or transient. For permanent failures, the host name remains `NULL` and the validation flag is set to true. For transient failures, the host name and validation flag remain unchanged. (In this case, another DNS resolution attempt occurs the next time a client connects from this IP.)

3. If an error occurs while processing an incoming client connection from a given IP address, the server updates the corresponding error counters in the entry for that IP. For a description of the errors recorded, see Section 22.12.10.1, “The host_cache Table”.

To unblock blocked hosts, flush the host cache; see Dealing with Blocked Hosts.

It is possible for a blocked host to become unblocked even without flushing the host cache if activity from other hosts occurs:

- If the cache is full when a connection arrives from a client IP not in the cache, the server discards the least recently used cache entry to make room for the new entry.

- If the discarded entry is for a blocked host, that host becomes unblocked.

Some connection errors are not associated with TCP connections, occur very early in the connection process (even before an IP address is known), or are not specific to any particular IP address (such as out-of-memory conditions). For information about these errors, check the `Connection_errors_xxx` status variables (see Section 5.1.9, “Server Status Variables”).

Configuring the Host Cache

The host cache is enabled by default. The `host_cache_size` system variable controls its size, as well as the size of the Performance Schema `host_cache` table that exposes the cache contents. The cache size can be set at server startup and changed at runtime. For example, to set the size to 100 at startup, put these lines in the server `my.cnf` file:

```
[mysqld]
host_cache_size=200
```

To change the size to 300 at runtime, do this:

```
SET GLOBAL host_cache_size=300;
```

Setting `host_cache_size` to 0, either at server startup or at runtime, disables the host cache. With the cache disabled, the server performs a DNS lookup every time a client connects.

Changing the cache size at runtime causes an implicit host cache flushing operation that clears the host cache, truncates the `host_cache` table, and unblocks any blocked hosts; see Flushing the Host Cache.

Using the `--skip-host-cache` option is similar to setting the `host_cache_size` system variable to 0, but `host_cache_size` is more flexible because it can also be used to resize, enable, and disable the host cache at runtime, not just at server startup. Starting the server with `--skip-host-cache` does not prevent runtime changes to the value of `host_cache_size`, but such changes have no effect and the cache is not re-enabled even if `host_cache_size` is set larger than 0.

To disable DNS host name lookups, start the server with the `skip_name_resolve` system variable enabled. In this case, the server uses only IP addresses and not host names to match connecting hosts to rows in the MySQL grant tables. Only accounts specified in those tables using IP addresses can be used. (A client may not be able to connect if no account exists that specifies the client IP address.)

If you have a very slow DNS and many hosts, you might be able to improve performance either by enabling `skip_name_resolve` to disable DNS lookups, or by increasing the value of `host_cache_size` to make the host cache larger.

To disallow TCP/IP connections entirely, start the server with the `skip_networking` system variable enabled.

To adjust the permitted number of successive connection errors before host blocking occurs, set the `max_connect_errors` system variable. For example, to set the value at startup put these lines in the server `my.cnf` file:

```
[mysqld]
max_connect_errors=10000
```

To change the value at runtime, do this:

```
SET GLOBAL max_connect_errors=10000;
```

Monitoring the Host Cache

The Performance Schema `host_cache` table exposes the contents of the host cache. This table can be examined using `SELECT` statements, which may help you diagnose the causes of connection problems. The Performance Schema must be enabled or this table is empty. For information about this table, see Section 22.12.10.1, “The host_cache Table”.

Flushing the Host Cache

Flushing the host cache might be advisable or desirable under these conditions:

- Some of your client hosts change IP address.
- The error message `Host 'host_name' is blocked` occurs for connections from legitimate hosts. (See Dealing with Blocked Hosts.)

Flushing the host cache has these effects:

- It clears the in-memory host cache.
- It removes all rows from the Performance Schema `host_cache` table that exposes the cache contents.

- It unblocks any blocked hosts. This enables further connection attempts from those hosts.

To flush the host cache, use any of these methods:

- Change the value of the `host_cache_size` system variable. This requires the `SUPER` privilege.
- Execute a `TRUNCATE TABLE` statement that truncates the Performance Schema `host_cache` table. This requires the `DROP` privilege for the table.
- Execute a `FLUSH HOSTS` statement. This requires the `RELOAD` privilege.
- Execute a `mysqladmin flush-hosts` command. This requires the `RELOAD` privilege.

Dealing with Blocked Hosts

The server uses the host cache to track errors that occur during the client connection process. If the following error occurs, it means that `mysqld` has received many connection requests from the given host that were interrupted in the middle:

```
Host 'host_name' is blocked because of many connection errors.
Unblock with 'mysqladmin flush-hosts'
```

The value of the `max_connect_errors` system variable determines how many successive interrupted connection requests the server permits before blocking a host. After `max_connect_errors` failed requests without a successful connection, the server assumes that something is wrong (for example, that someone is trying to break in), and blocks the host from further connection requests.

To unblock blocked hosts, flush the host cache; see Flushing the Host Cache.

Alternatively, to avoid having the error message occur, set `max_connect_errors` as described in Configuring the Host Cache. The default value of `max_connect_errors` is 100. Increasing `max_connect_errors` to a large value makes it less likely that a host reaches the threshold and become blocked. However, if the `Host 'host_name' is blocked` error message occurs, first verify that there is nothing wrong with TCP/IP connections from the blocked hosts. It does no good to increase the value of `max_connect_errors` if there are network problems.

5.1.12 IPv6 Support

Support for IPv6 in MySQL includes these capabilities:

- MySQL Server can accept TCP/IP connections from clients connecting over IPv6. For example, this command connects over IPv6 to the MySQL server on the local host:

  ```
  shell> mysql -h ::1
  ```

  To use this capability, two things must be true:

  - Your system must be configured to support IPv6. See Section 5.1.12.1, “Verifying System Support for IPv6”.
  - The default MySQL server configuration permits IPv6 connections in addition to IPv4 connections. To change the default configuration, start the server with the `bind_address` system variable set to an appropriate value. See Section 5.1.7, “Server System Variables”.

- MySQL account names permit IPv6 addresses to enable DBAs to specify privileges for clients that connect to the server over IPv6. See Section 6.2.4, “Specifying Account Names”. IPv6 addresses can be specified in account names in statements such as `CREATE USER`, `GRANT`, and `REVOKE`. For example:

  ```
  mysql> CREATE USER 'bill'@'::1' IDENTIFIED BY 'secret';
  mysql> GRANT SELECT ON mydb.* TO 'bill'@'::1';
  ```

- IPv6 functions enable conversion between string and internal format IPv6 address formats, and checking whether values represent valid IPv6 addresses. For example, `INET6_ATON()` and `INET6_NTOA()` are similar to `INET_ATON()` and `INET_NTOA()`, but handle IPv6 addresses in addition to IPv4 addresses. See Section 12.20, “Miscellaneous Functions”.

The following sections describe how to set up MySQL so that clients can connect to the server over IPv6.

5.1.12.1 Verifying System Support for IPv6

Before MySQL Server can accept IPv6 connections, the operating system on your server host must support IPv6. As a simple test to determine whether that is true, try this command:

```
shell> ping6 ::1
16 bytes from ::1, icmp_seq=0 hlim=64 time=0.171 ms
16 bytes from ::1, icmp_seq=1 hlim=64 time=0.077 ms
...
```

To produce a description of your system's network interfaces, invoke `ifconfig -a` and look for IPv6 addresses in the output.

If your host does not support IPv6, consult your system documentation for instructions on enabling it. It might be that you need only reconfigure an existing network interface to add an IPv6 address. Or a more extensive change might be needed, such as rebuilding the kernel with IPv6 options enabled.

These links may be helpful in setting up IPv6 on various platforms:

- Windows
- Gentoo Linux
- Ubuntu Linux
- Linux (Generic)
- macOS

5.1.12.2 Configuring the MySQL Server to Permit IPv6 Connections

The MySQL server listens on a single network socket for TCP/IP connections. This socket is bound to a single address, but it is possible for an address to map onto multiple network interfaces. To specify an address, set `bind_address=`*`addr`* at server startup, where *`addr`* is an IPv4 or IPv6 address or a host name. For details, see the `bind_address` description in Section 5.1.7, “Server System Variables”.

5.1.12.3 Connecting Using the IPv6 Local Host Address

The following procedure shows how to configure MySQL to permit IPv6 connections by clients that connect to the local server using the `::1` local host address. The instructions given here assume that your system supports IPv6.

1. Start the MySQL server with an appropriate `bind_address` setting to permit it to accept IPv6 connections. For example, put the following lines in the server option file and restart the server:

   ```
   [mysqld]
   bind_address = *   # before 5.6.6, use :: rather than *
   ```

   Alternatively, you can bind the server to `::1`, but that makes the server more restrictive for TCP/IP connections. It accepts only IPv6 connections for that single address and rejects IPv4 connections. For more information, see the `bind_address` description in Section 5.1.7, “Server System Variables”.

2. As an administrator, connect to the server and create an account for a local user who connects from the `::1` local IPv6 host address:

```
mysql> CREATE USER 'ipv6user'@'::1' IDENTIFIED BY 'ipv6pass';
```

   For the permitted syntax of IPv6 addresses in account names, see Section 6.2.4, “Specifying Account Names”. In addition to the `CREATE USER` statement, you can issue `GRANT` statements that give specific privileges to the account, although that is not necessary for the remaining steps in this procedure.

3. Invoke the `mysql` client to connect to the server using the new account:

```
shell> mysql -h ::1 -u ipv6user -pipv6pass
```

4. Try some simple statements that show connection information:

```
mysql> STATUS
...
Connection:   ::1 via TCP/IP
...

mysql> SELECT CURRENT_USER(), @@bind_address;
+----------------+----------------+
| CURRENT_USER() | @@bind_address |
+----------------+----------------+
| ipv6user@::1   | ::             |
+----------------+----------------+
```

5.1.12.4 Connecting Using IPv6 Nonlocal Host Addresses

The following procedure shows how to configure MySQL to permit IPv6 connections by remote clients. It is similar to the preceding procedure for local clients, but the server and client hosts are distinct and each has its own nonlocal IPv6 address. The example uses these addresses:

```
Server host: 2001:db8:0:f101::1
Client host: 2001:db8:0:f101::2
```

These addresses are chosen from the nonroutable address range recommended by IANA for documentation purposes and suffice for testing on your local network. To accept IPv6 connections from clients outside the local network, the server host must have a public address. If your network provider assigns you an IPv6 address, you can use that. Otherwise, another way to obtain an address is to use an IPv6 broker; see Section 5.1.12.5, “Obtaining an IPv6 Address from a Broker”.

1. Start the MySQL server with an appropriate `bind_address` setting to permit it to accept IPv6 connections. For example, put the following lines in the server option file and restart the server:

```
[mysqld]
bind_address = *   # before 5.6.6, use :: rather than *
```

   Alternatively, you can bind the server to `2001:db8:0:f101::1`, but that makes the server more restrictive for TCP/IP connections. It accepts only IPv6 connections for that single address and rejects IPv4 connections. For more information, see the `bind_address` description in Section 5.1.7, “Server System Variables”.

2. On the server host (`2001:db8:0:f101::1`), create an account for a user who connects from the client host (`2001:db8:0:f101::2`):

```
mysql> CREATE USER 'remoteipv6user'@'2001:db8:0:f101::2' IDENTIFIED BY 'remoteipv6pass';
```

3. On the client host (`2001:db8:0:f101::2`), invoke the `mysql` client to connect to the server using the new account:

```
shell> mysql -h 2001:db8:0:f101::1 -u remoteipv6user -premoteipv6pass
```

4. Try some simple statements that show connection information:

```
mysql> STATUS
...
Connection:   2001:db8:0:f101::1 via TCP/IP
...

mysql> SELECT CURRENT_USER(), @@bind_address;
+----------------------------------+---------------+
| CURRENT_USER()                   | @@bind_address |
+----------------------------------+---------------+
| remoteipv6user@2001:db8:0:f101::2 | ::            |
+----------------------------------+---------------+
```

5.1.12.5 Obtaining an IPv6 Address from a Broker

If you do not have a public IPv6 address that enables your system to communicate over IPv6 outside your local network, you can obtain one from an IPv6 broker. The Wikipedia IPv6 Tunnel Broker page lists several brokers and their features, such as whether they provide static addresses and the supported routing protocols.

After configuring your server host to use a broker-supplied IPv6 address, start the MySQL server with an appropriate `bind_address` setting to permit the server to accept IPv6 connections. For example, put the following lines in the server option file and restart the server:

```
[mysqld]
bind_address = *   # before 5.6.6, use :: rather than *
```

Alternatively, you can bind the server to the specific IPv6 address provided by the broker, but that makes the server more restrictive for TCP/IP connections. It accepts only IPv6 connections for that single address and rejects IPv4 connections. For more information, see the `bind_address` description in Section 5.1.7, “Server System Variables”. In addition, if the broker allocates dynamic addresses, the address provided for your system might change the next time you connect to the broker. If so, any accounts you create that name the original address become invalid. To bind to a specific address but avoid this change-of-address problem, you may be able to arrange with the broker for a static IPv6 address.

The following example shows how to use Freenet6 as the broker and the `gogoc` IPv6 client package on Gentoo Linux.

1. Create an account at Freenet6 by visiting this URL and signing up:

   ```
   http://gogonet.gogo6.com
   ```

2. After creating the account, go to this URL, sign in, and create a user ID and password for the IPv6 broker:

   ```
   http://gogonet.gogo6.com/page/freenet6-registration
   ```

3. As `root`, install `gogoc`:

   ```
   shell> emerge gogoc
   ```

4. Edit `/etc/gogoc/gogoc.conf` to set the `userid` and `password` values. For example:

   ```
   userid=gogouser
   passwd=gogopass
   ```

5. Start `gogoc`:

   ```
   shell> /etc/init.d/gogoc start
   ```

   To start `gogoc` each time your system boots, execute this command:

   ```
   shell> rc-update add gogoc default
   ```

6. Use `ping6` to try to ping a host:

```
shell> ping6 ipv6.google.com
```

7. To see your IPv6 address:

```
shell> ifconfig tun
```

5.1.13 MySQL Server Time Zone Support

This section describes the time zone settings maintained by MySQL, how to load the system tables required for named time support, how to stay current with time zone changes, and how to enable leap-second support.

- Time Zone Variables
- Populating the Time Zone Tables
- Staying Current with Time Zone Changes
- Time Zone Leap Second Support

For information about time zone settings in replication setups, see Section 17.4.1.15, “Replication and System Functions” and Section 17.4.1.31, “Replication and Time Zones”.

Time Zone Variables

MySQL Server maintains several time zone settings:

- The system time zone. When the server starts, it attempts to determine the time zone of the host machine automatically and uses it to set the `system_time_zone` system variable. The value does not change thereafter.

  To explicitly specify the system time zone for MySQL Server at startup, set the `TZ` environment variable before you start `mysqld`. If you start the server using `mysqld_safe`, its `--timezone` option provides another way to set the system time zone. The permissible values for `TZ` and `--timezone` are system dependent. Consult your operating system documentation to see what values are acceptable.

- The server current time zone. The global `time_zone` system variable indicates the time zone the server currently is operating in. The initial `time_zone` value is `'SYSTEM'`, which indicates that the server time zone is the same as the system time zone.

  Note

  If set to `SYSTEM`, every MySQL function call that requires a time zone calculation makes a system library call to determine the current system time zone. This call may be protected by a global mutex, resulting in contention.

  The initial global server time zone value can be specified explicitly at startup with the `--default-time-zone` option on the command line, or you can use the following line in an option file:

```
default-time-zone='timezone'
```

  If you have the `SUPER` privilege, you can set the global server time zone value at runtime with this statement:

```
SET GLOBAL time_zone = timezone;
```

- Per-session time zones. Each client that connects has its own session time zone setting, given by the session `time_zone` variable. Initially, the session variable takes its value from the global `time_zone` variable, but the client can change its own time zone with this statement:

```
SET time_zone = timezone;
```

The session time zone setting affects display and storage of time values that are zone-sensitive. This includes the values displayed by functions such as `NOW()` or `CURTIME()`, and values stored in and retrieved from `TIMESTAMP` columns. Values for `TIMESTAMP` columns are converted from the session time zone to UTC for storage, and from UTC to the session time zone for retrieval.

The session time zone setting does not affect values displayed by functions such as `UTC_TIMESTAMP()` or values in `DATE`, `TIME`, or `DATETIME` columns. Nor are values in those data types stored in UTC; the time zone applies for them only when converting from `TIMESTAMP` values. If you want locale-specific arithmetic for `DATE`, `TIME`, or `DATETIME` values, convert them to UTC, perform the arithmetic, and then convert back.

The current global and session time zone values can be retrieved like this:

```
SELECT @@GLOBAL.time_zone, @@SESSION.time_zone;
```

`timezone` values can be given in several formats, none of which are case-sensitive:

- As the value `'SYSTEM'`, indicating that the server time zone is the same as the system time zone.
- As a string indicating an offset from UTC of the form *`[H]H:MM`*, prefixed with a `+` or `-`, such as `'+10:00'`, `'-6:00'`, or `'+05:30'`. A leading zero can optionally be used for hours values less than 10; MySQL prepends a leading zero when storing and retriving the value in such cases. MySQL converts `'-00:00'` or `'-0:00'` to `'+00:00'`.

  A time zone offset must be in the range `'-12:59'` to `'+13:00'`, inclusive.
- As a named time zone, such as `'Europe/Helsinki'`, `'US/Eastern'`, `'MET'`, or `'UTC'`.

Note

Named time zones can be used only if the time zone information tables in the `mysql` database have been created and populated. Otherwise, use of a named time zone results in an error:

```
mysql> SET time_zone = 'UTC';
ERROR 1298 (HY000): Unknown or incorrect time zone: 'UTC'
```

Populating the Time Zone Tables

Several tables in the `mysql` system database exist to store time zone information (see Section 5.3, “The mysql System Database”). The MySQL installation procedure creates the time zone tables, but does not load them. To do so manually, use the following instructions.

Note

Loading the time zone information is not necessarily a one-time operation because the information changes occasionally. When such changes occur, applications that use the old rules become out of date and you may find it necessary to reload the time zone tables to keep the information used by your MySQL server current. See Staying Current with Time Zone Changes.

If your system has its own *zoneinfo* database (the set of files describing time zones), use the `mysql_tzinfo_to_sql` program to load the time zone tables. Examples of such systems are Linux, macOS, FreeBSD, and Solaris. One likely location for these files is the `/usr/share/zoneinfo` directory. If your system has no zoneinfo database, you can use a downloadable package, as described later in this section.

To load the time zone tables from the command line, pass the zoneinfo directory path name to `mysql_tzinfo_to_sql` and send the output into the `mysql` program. For example:

```
mysql_tzinfo_to_sql /usr/share/zoneinfo | mysql -u root -p mysql
```

The `mysql` command shown here assumes that you connect to the server using an account such as `root` that has privileges for modifying tables in the `mysql` system database. Adjust the connection parameters as required.

`mysql_tzinfo_to_sql` reads your system's time zone files and generates SQL statements from them. `mysql` processes those statements to load the time zone tables.

`mysql_tzinfo_to_sql` also can be used to load a single time zone file or generate leap second information:

- To load a single time zone file *`tz_file`* that corresponds to a time zone name *`tz_name`*, invoke `mysql_tzinfo_to_sql` like this:

  ```
  mysql_tzinfo_to_sql tz_file tz_name | mysql -u root -p mysql
  ```

  With this approach, you must execute a separate command to load the time zone file for each named zone that the server needs to know about.

- If your time zone must account for leap seconds, initialize leap second information like this, where *`tz_file`* is the name of your time zone file:

  ```
  mysql_tzinfo_to_sql --leap tz_file | mysql -u root -p mysql
  ```

After running `mysql_tzinfo_to_sql`, restart the server so that it does not continue to use any previously cached time zone data.

If your system has no zoneinfo database (for example, Windows), you can use a package that is available for download at the MySQL Developer Zone:

```
https://dev.mysql.com/downloads/timezones.html
```

Warning

Do *not* use a downloadable time zone package if your system has a zoneinfo database. Use the `mysql_tzinfo_to_sql` utility instead. Otherwise, you may cause a difference in datetime handling between MySQL and other applications on your system.

You can use either a package that contains SQL statements to populate your existing time zone tables, or a package that contains pre-built `MyISAM` time zone tables to replace your existing tables:

- To use a time zone package that contains SQL statements, download and unpack it, then load the unpacked file contents into your existing time zone tables:

  ```
  mysql -u root -p mysql < file_name
  ```

  Then restart the server.

- To use a time zone package that contains `.frm`, `.MYD`, and `.MYI` files for the `MyISAM` time zone tables, download and unpack it. These table files are part of the `mysql` database, so you should place the files in the `mysql` subdirectory of your MySQL server's data directory. Stop the server before doing this and restart it afterward.

Staying Current with Time Zone Changes

When time zone rules change, applications that use the old rules become out of date. To stay current, it is necessary to make sure that your system uses current time zone information is used. For MySQL, there are multiple factors to consider in staying current:

- The operating system time affects the value that the MySQL server uses for times if its time zone is set to `SYSTEM`. Make sure that your operating system is using the latest time zone information. For most operating systems, the latest update or service pack prepares your system for the time

changes. Check the website for your operating system vendor for an update that addresses the time changes.

- If you replace the system's `/etc/localtime` time zone file with a version that uses rules differing from those in effect at `mysqld` startup, restart `mysqld` so that it uses the updated rules. Otherwise, `mysqld` might not notice when the system changes its time.
- If you use named time zones with MySQL, make sure that the time zone tables in the `mysql` database are up to date:
  - If your system has its own zoneinfo database, reload the MySQL time zone tables whenever the zoneinfo database is updated.
  - For systems that do not have their own zoneinfo database, check the MySQL Developer Zone for updates. When a new update is available, download it and use it to replace the content of your current time zone tables.

  For instructions for both methods, see Populating the Time Zone Tables. `mysqld` caches time zone information that it looks up, so after updating the time zone tables, restart `mysqld` to make sure that it does not continue to serve outdated time zone data.

If you are uncertain whether named time zones are available, for use either as the server's time zone setting or by clients that set their own time zone, check whether your time zone tables are empty. The following query determines whether the table that contains time zone names has any rows:

```
mysql> SELECT COUNT(*) FROM mysql.time_zone_name;
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+
```

A count of zero indicates that the table is empty. In this case, no applications currently are using named time zones, and you need not update the tables (unless you want to enable named time zone support). A count greater than zero indicates that the table is not empty and that its contents are available to be used for named time zone support. In this case, be sure to reload your time zone tables so that applications that use named time zones obtain correct query results.

To check whether your MySQL installation is updated properly for a change in Daylight Saving Time rules, use a test like the one following. The example uses values that are appropriate for the 2007 DST 1-hour change that occurs in the United States on March 11 at 2 a.m.

The test uses this query:

```
SELECT
  CONVERT_TZ('2007-03-11 2:00:00','US/Eastern','US/Central') AS time1,
  CONVERT_TZ('2007-03-11 3:00:00','US/Eastern','US/Central') AS time2;
```

The two time values indicate the times at which the DST change occurs, and the use of named time zones requires that the time zone tables be used. The desired result is that both queries return the same result (the input time, converted to the equivalent value in the 'US/Central' time zone).

Before updating the time zone tables, you see an incorrect result like this:

```
+---------------------+---------------------+
| time1               | time2               |
+---------------------+---------------------+
| 2007-03-11 01:00:00 | 2007-03-11 02:00:00 |
+---------------------+---------------------+
```

After updating the tables, you should see the correct result:

```
+---------------------+---------------------+
| time1               | time2               |
+---------------------+---------------------+
```

```
| 2007-03-11 01:00:00 | 2007-03-11 01:00:00 |
+---------------------+---------------------+
```

Time Zone Leap Second Support

Leap second values are returned with a time part that ends with `:59:59`. This means that a function such as `NOW()` can return the same value for two or three consecutive seconds during the leap second. It remains true that literal temporal values having a time part that ends with `:59:60` or `:59:61` are considered invalid.

If it is necessary to search for `TIMESTAMP` values one second before the leap second, anomalous results may be obtained if you use a comparison with *`'YYYY-MM-DD hh:mm:ss'`* values. The following example demonstrates this. It changes the session time zone to UTC so there is no difference between internal `TIMESTAMP` values (which are in UTC) and displayed values (which have time zone correction applied).

```
mysql> CREATE TABLE t1 (
         a INT,
         ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
         PRIMARY KEY (ts)
       );
Query OK, 0 rows affected (0.01 sec)

mysql> -- change to UTC
mysql> SET time_zone = '+00:00';
Query OK, 0 rows affected (0.00 sec)

mysql> -- Simulate NOW() = '2008-12-31 23:59:59'
mysql> SET timestamp = 1230767999;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 (a) VALUES (1);
Query OK, 1 row affected (0.00 sec)

mysql> -- Simulate NOW() = '2008-12-31 23:59:60'
mysql> SET timestamp = 1230768000;
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t1 (a) VALUES (2);
Query OK, 1 row affected (0.00 sec)

mysql> -- values differ internally but display the same
mysql> SELECT a, ts, UNIX_TIMESTAMP(ts) FROM t1;
+------+---------------------+--------------------+
| a    | ts                  | UNIX_TIMESTAMP(ts) |
+------+---------------------+--------------------+
|    1 | 2008-12-31 23:59:59 |         1230767999 |
|    2 | 2008-12-31 23:59:59 |         1230768000 |
+------+---------------------+--------------------+
2 rows in set (0.00 sec)

mysql> -- only the non-leap value matches
mysql> SELECT * FROM t1 WHERE ts = '2008-12-31 23:59:59';
+------+---------------------+
| a    | ts                  |
+------+---------------------+
|    1 | 2008-12-31 23:59:59 |
+------+---------------------+
1 row in set (0.00 sec)

mysql> -- the leap value with seconds=60 is invalid
mysql> SELECT * FROM t1 WHERE ts = '2008-12-31 23:59:60';
Empty set, 2 warnings (0.00 sec)
```

To work around this, you can use a comparison based on the UTC value actually stored in the column, which has the leap second correction applied:

```
mysql> -- selecting using UNIX_TIMESTAMP value return leap value
```

```
mysql> SELECT * FROM t1 WHERE UNIX_TIMESTAMP(ts) = 1230768000;
+------+---------------------+
| a    | ts                  |
+------+---------------------+
|    2 | 2008-12-31 23:59:59 |
+------+---------------------+
1 row in set (0.00 sec)
```

5.1.14 Server-Side Help Support

MySQL Server supports a `HELP` statement that returns information from the MySQL Reference Manual (see Section 13.8.3, “HELP Statement”). This information is stored in several tables in the `mysql` database (see Section 5.3, “The mysql System Database”). Proper operation of the `HELP` statement requires that these help tables be initialized.

For a new installation of MySQL using a binary or source distribution on Unix, help-table content initialization occurs when you initialize the data directory (see Section 2.10.1, “Initializing the Data Directory”). For an RPM distribution on Linux or binary distribution on Windows, content initialization occurs as part of the MySQL installation process.

For a MySQL upgrade using a binary distribution, help-table content is not upgraded automatically, but you can upgrade it manually. Locate the `fill_help_tables.sql` file in the `share` or `share/mysql` directory. Change location into that directory and process the file with the `mysql` client as follows (enter the command on one line):

```
mysql -u root -p
  --init-command="SET sql_log_bin=OFF"
  mysql < fill_help_tables.sql
```

The command shown here assumes that you connect to the server using an account such as `root` that has privileges for modifying tables in the `mysql` database. Adjust the connection parameters as required.

If you are working with Git and a MySQL development source tree, the source tree contains only a “stub” version of `fill_help_tables.sql`. To obtain a non-stub copy, use one from a source or binary distribution.

Note

Each MySQL series has its own series-specific reference manual, so help-table content is series specific as well. This has implications for replication because help-table content should match the MySQL series. If you load MySQL 5.6 help content into a MySQL 5.6 source server, it does not make sense to replicate that content to a replica server from a different MySQL series and for which that content is not appropriate. For this reason, as you upgrade individual servers in a replication scenario, you should upgrade each server's help tables, using the instructions given earlier.

5.1.15 The Server Shutdown Process

The server shutdown process takes place as follows:

1. The shutdown process is initiated.

   This can occur initiated several ways. For example, a user with the `SHUTDOWN` privilege can execute a `mysqladmin shutdown` command. `mysqladmin` can be used on any platform supported by MySQL. Other operating system-specific shutdown initiation methods are possible as well: The server shuts down on Unix when it receives a `SIGTERM` signal. A server running as a service on Windows shuts down when the services manager tells it to.

2. The server creates a shutdown thread if necessary.

Depending on how shutdown was initiated, the server might create a thread to handle the shutdown process. If shutdown was requested by a client, a shutdown thread is created. If shutdown is the result of receiving a `SIGTERM` signal, the signal thread might handle shutdown itself, or it might create a separate thread to do so. If the server tries to create a shutdown thread and cannot (for example, if memory is exhausted), it issues a diagnostic message that appears in the error log:

```
Error: Can't create thread to kill server
```

3. The server stops accepting new connections.

   To prevent new activity from being initiated during shutdown, the server stops accepting new client connections by closing the handlers for the network interfaces to which it normally listens for connections: the TCP/IP port, the Unix socket file, the Windows named pipe, and shared memory on Windows.

4. The server terminates current activity.

   For each thread associated with a client connection, the server breaks the connection to the client and marks the thread as killed. Threads die when they notice that they are so marked. Threads for idle connections die quickly. Threads that currently are processing statements check their state periodically and take longer to die. For additional information about thread termination, see Section 13.7.6.4, “KILL Statement”, in particular for the instructions about killed `REPAIR TABLE` or `OPTIMIZE TABLE` operations on `MyISAM` tables.

   For threads that have an open transaction, the transaction is rolled back.

   Note

   If a thread is updating a nontransactional table, an operation such as a multiple-row `UPDATE` or `INSERT` may leave the table partially updated because the operation can terminate before completion.

   If the server is a source replication server, it treats threads associated with currently connected replicas like other client threads. That is, each one is marked as killed and exits when it next checks its state.

   If the server is a replica, it stops the I/O and SQL threads, if they are active, before marking client threads as killed. The SQL thread is permitted to finish its current statement (to avoid causing replication problems), and then stops. If the SQL thread is in the middle of a transaction at this point, the server waits until the current replication event group (if any) has finished executing, or until the user issues a `KILL QUERY` or `KILL CONNECTION` statement. See also Section 13.4.2.6, “STOP SLAVE Statement”. Since nontransactional statements cannot be rolled back, in order to guarantee crash-safe replication, only transactional tables should be used.

   Note

   To guarantee crash safety on the replica, you must run the replica with `--relay-log-recovery` enabled.

   See also Section 17.2.2, “Relay Log and Replication Metadata Repositories”).

5. The server shuts down or closes storage engines.

   At this stage, the server flushes the table cache and closes all open tables.

   Each storage engine performs any actions necessary for tables that it manages. `InnoDB` flushes its buffer pool to disk (unless `innodb_fast_shutdown` is 2), writes the current LSN to the tablespace, and terminates its own internal threads. `MyISAM` flushes any pending index writes for a table.

6. The server exits.

5.2 The MySQL Data Directory

Information managed by the MySQL server is stored under a directory known as the data directory. The following list briefly describes the items typically found in the data directory, with cross references for additional information:

- Data directory subdirectories. Each subdirectory of the data directory is a database directory and corresponds to a database managed by the server. All MySQL installations have certain standard databases:
  - The `mysql` directory corresponds to the `mysql` system database, which contains information required by the MySQL server as it runs. See Section 5.3, “The mysql System Database”.
  - The `performance_schema` directory corresponds to the Performance Schema, which provides information used to inspect the internal execution of the server at runtime. See Chapter 22, *MySQL Performance Schema*.
  - The `ndbinfo` directory corresponds to the `ndbinfo` database that stores information specific to NDB Cluster (present only for installations built to include NDB Cluster). See Section 18.5.14, “ndbinfo: The NDB Cluster Information Database”.

  Other subdirectories correspond to databases created by users or applications.

  Note

  `INFORMATION_SCHEMA` is a standard database, but its implementation uses no corresponding database directory.
- Log files written by the server. See Section 5.4, “MySQL Server Logs”.
- `InnoDB` tablespace and log files. See Chapter 14, *The InnoDB Storage Engine*.
- The server process ID file (while the server is running).

Some items in the preceding list can be relocated elsewhere by reconfiguring the server. In addition, the `--datadir` option enables the location of the data directory itself to be changed. For a given MySQL installation, check the server configuration to determine whether items have been moved.

5.3 The mysql System Database

The `mysql` database is the system database. It contains tables that store information required by the MySQL server as it runs.

Tables in the `mysql` database fall into these categories:

- Grant System Tables
- Object Information System Tables
- Log System Tables
- Server-Side Help System Tables
- Time Zone System Tables
- Replication System Tables
- Optimizer System Tables
- Miscellaneous System Tables

The remainder of this section enumerates the tables in each category, with cross references for additional information. System tables use the `MyISAM` storage engine unless otherwise indicated.

Warning

Do *not* convert MySQL system tables in the `mysql` database from `MyISAM` to `InnoDB` tables. This is an unsupported operation. If you do this, MySQL does not restart until you restore the old system tables from a backup or regenerate them by reinitializing the data directory (see Section 2.10.1, “Initializing the Data Directory”).

Grant System Tables

These system tables contain grant information about user accounts and the privileges held by them:

- `user`: User accounts, global privileges, and other nonprivilege columns.
- `db`: Database-level privileges.
- `host`: Obsolete. MySQL install operations do not create this table as of MySQL 5.6.7.
- `tables_priv`: Table-level privileges.
- `columns_priv`: Column-level privileges.
- `procs_priv`: Stored procedure and function privileges.
- `proxies_priv`: Proxy-user privileges.

For more information about the structure, contents, and purpose of the grant tables, see Section 6.2.3, “Grant Tables”.

Object Information System Tables

These system tables contain information about stored programs, user-defined functions, and server-side plugins:

- `event`: The registry for Event Scheduler events installed using `CREATE EVENT`. If the server is started with the `--skip-grant-tables` option, the event scheduler is disabled and events registered in the table do not run. See Section 20.4.2, “Event Scheduler Configuration”.
- `func`: The registry for user-defined functions (UDFs) installed using `CREATE FUNCTION`. During the normal startup sequence, the server loads UDFs registered in this table. If the server is started with the `--skip-grant-tables` option, UDFs registered in the table are not loaded and are unavailable. See Section 5.6.1, “Installing and Uninstalling User-Defined Functions”.
- `plugin`: The registry for server-side plugins installed using `INSTALL PLUGIN`. During the normal startup sequence, the server loads plugins registered in this table. If the server is started with the `--skip-grant-tables` option, plugins registered in the table are not loaded and are unavailable. See Section 5.5.1, “Installing and Uninstalling Plugins”.
- `proc`: Information about stored procedures and functions. See Section 20.2, “Using Stored Routines”.

Log System Tables

The server uses these system tables for logging:

- `general_log`: The general query log table.
- `slow_log`: The slow query log table.

Log tables use the `CSV` storage engine.

For more information, see Section 5.4, “MySQL Server Logs”.

Server-Side Help System Tables

These system tables contain server-side help information:

- `help_category`: Information about help categories.
- `help_keyword`: Keywords associated with help topics.
- `help_relation`: Mappings between help keywords and topics.
- `help_topic`: Help topic contents.

For more information, see Section 5.1.14, “Server-Side Help Support”.

Time Zone System Tables

These system tables contain time zone information:

- `time_zone`: Time zone IDs and whether they use leap seconds.
- `time_zone_leap_second`: When leap seconds occur.
- `time_zone_name`: Mappings between time zone IDs and names.
- `time_zone_transition`, `time_zone_transition_type`: Time zone descriptions.

For more information, see Section 5.1.13, “MySQL Server Time Zone Support”.

Replication System Tables

The server uses these system tables to support replication:

- `ndb_binlog_index`: Binary log information for NDB Cluster replication. See Section 18.6.4, “NDB Cluster Replication Schema and Tables”.
- `slave_master_info`, `slave_relay_log_info`, `slave_worker_info`: Used to store replication information on replica servers. See Section 17.2.2, “Relay Log and Replication Metadata Repositories”.

  These tables use the `InnoDB` storage engine as of MySQL 5.6.6, `MyISAM` before that.

Optimizer System Tables

The `innodb_index_stats` and `innodb_table_stats` system tables are used for `InnoDB` persistent optimizer statistics. See Section 14.8.11.1, “Configuring Persistent Optimizer Statistics Parameters”.

These tables use the `InnoDB` storage engine.

Miscellaneous System Tables

Other system tables do not fall into the preceding categories:

- `firewall_users`, `firewall_whitelist`: If MySQL Enterprise Firewall is installed, these tables provide persistent storage for information used by the firewall. See Section 6.4.5, “MySQL Enterprise Firewall”.
- `servers`: Used by the `FEDERATED` storage engine. See Section 15.8.2.2, “Creating a FEDERATED Table Using CREATE SERVER”.

5.4 MySQL Server Logs

MySQL Server has several logs that can help you find out what activity is taking place.

Log Type	Information Written to Log
Error log	Problems encountered starting, running, or stopping `mysqld`
General query log	Established client connections and statements received from clients
Binary log	Statements that change data (also used for replication)
Relay log	Data changes received from a replication source server
Slow query log	Queries that took more than `long_query_time` seconds to execute
DDL log (metadata log)	Metadata operations performed by DDL statements

By default, no logs are enabled, except the error log on Windows. (The DDL log is always created when required, and has no user-configurable options; see Section 5.4.6, “The DDL Log”.) The following log-specific sections provide information about the server options that enable logging.

By default, the server writes files for all enabled logs in the data directory. You can force the server to close and reopen the log files (or in some cases switch to a new log file) by flushing the logs. Log flushing occurs when you issue a `FLUSH LOGS` statement; execute `mysqladmin` with a `flush-logs` or `refresh` argument; or execute `mysqldump` with a `--flush-logs` or `--master-data` option. See Section 13.7.6.3, “FLUSH Statement”, Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”, and Section 4.5.4, “`mysqldump` — A Database Backup Program”. In addition, the binary log is flushed when its size reaches the value of the `max_binlog_size` system variable.

You can control the general query and slow query logs during runtime. You can enable or disable logging, or change the log file name. You can tell the server to write general query and slow query entries to log tables, log files, or both. For details, see Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”, Section 5.4.3, “The General Query Log”, and Section 5.4.5, “The Slow Query Log”.

The relay log is used only on replica servers, to hold data changes from the source server that must also be made on the replica. For discussion of relay log contents and configuration, see Section 17.2.2.1, “The Relay Log”.

For information about log maintenance operations such as expiration of old log files, see Section 5.4.7, “Server Log Maintenance”.

For information about keeping logs secure, see Section 6.1.2.3, “Passwords and Logging”.

5.4.1 Selecting General Query Log and Slow Query Log Output Destinations

MySQL Server provides flexible control over the destination of output written to the general query log and the slow query log, if those logs are enabled. Possible destinations for log entries are log files or the `general_log` and `slow_log` tables in the `mysql` system database. File output, table output, or both can be selected.

- Log Control at Server Startup
- Log Control at Runtime
- Log Table Benefits and Characteristics

Log Control at Server Startup

The `log_output` system variable specifies the destination for log output. Setting this variable does not in itself enable the logs; they must be enabled separately.

- If `log_output` is not specified at startup, the default logging destination is `FILE`.
- If `log_output` is specified at startup, its value is a list one or more comma-separated words chosen from `TABLE` (log to tables), `FILE` (log to files), or `NONE` (do not log to tables or files). `NONE`, if present, takes precedence over any other specifiers.

The `general_log` system variable controls logging to the general query log for the selected log destinations. If specified at server startup, `general_log` takes an optional argument of 1 or 0 to enable or disable the log. To specify a file name other than the default for file logging, set the `general_log_file` variable. Similarly, the `slow_query_log` variable controls logging to the slow query log for the selected destinations and setting `slow_query_log_file` specifies a file name for file logging. If either log is enabled, the server opens the corresponding log file and writes startup messages to it. However, further logging of queries to the file does not occur unless the `FILE` log destination is selected.

Examples:

- To write general query log entries to the log table and the log file, use `--log_output=TABLE,FILE` to select both log destinations and `--general_log` to enable the general query log.
- To write general and slow query log entries only to the log tables, use `--log_output=TABLE` to select tables as the log destination and `--general_log` and `--slow_query_log` to enable both logs.
- To write slow query log entries only to the log file, use `--log_output=FILE` to select files as the log destination and `--slow_query_log` to enable the slow query log. In this case, because the default log destination is `FILE`, you could omit the `log_output` setting.

Log Control at Runtime

The system variables associated with log tables and files enable runtime control over logging:

- The `log_output` variable indicates the current logging destination. It can be modified at runtime to change the destination.
- The `general_log` and `slow_query_log` variables indicate whether the general query log and slow query log are enabled (`ON`) or disabled (`OFF`). You can set these variables at runtime to control whether the logs are enabled.
- The `general_log_file` and `slow_query_log_file` variables indicate the names of the general query log and slow query log files. You can set these variables at server startup or at runtime to change the names of the log files.
- To disable or enable general query logging for the current session, set the session `sql_log_off` variable to `ON` or `OFF`. (This assumes that the general query log itself is enabled.)

Log Table Benefits and Characteristics

The use of tables for log output offers the following benefits:

- Log entries have a standard format. To display the current structure of the log tables, use these statements:

```
SHOW CREATE TABLE mysql.general_log;
SHOW CREATE TABLE mysql.slow_log;
```

- Log contents are accessible through SQL statements. This enables the use of queries that select only those log entries that satisfy specific criteria. For example, to select log contents associated with a particular client (which can be useful for identifying problematic queries from that client), it is easier to do this using a log table than a log file.
- Logs are accessible remotely through any client that can connect to the server and issue queries (if the client has the appropriate log table privileges). It is not necessary to log in to the server host and directly access the file system.

The log table implementation has the following characteristics:

- In general, the primary purpose of log tables is to provide an interface for users to observe the runtime execution of the server, not to interfere with its runtime execution.

- `CREATE TABLE`, `ALTER TABLE`, and `DROP TABLE` are valid operations on a log table. For `ALTER TABLE` and `DROP TABLE`, the log table cannot be in use and must be disabled, as described later.
- By default, the log tables use the `CSV` storage engine that writes data in comma-separated values format. For users who have access to the `.CSV` files that contain log table data, the files are easy to import into other programs such as spreadsheets that can process CSV input.

  The log tables can be altered to use the `MyISAM` storage engine. You cannot use `ALTER TABLE` to alter a log table that is in use. The log must be disabled first. No engines other than `CSV` or `MyISAM` are legal for the log tables.
- To disable logging so that you can alter (or drop) a log table, you can use the following strategy. The example uses the general query log; the procedure for the slow query log is similar but uses the `slow_log` table and `slow_query_log` system variable.

```
SET @old_log_state = @@GLOBAL.general_log;
SET GLOBAL general_log = 'OFF';
ALTER TABLE mysql.general_log ENGINE = MyISAM;
SET GLOBAL general_log = @old_log_state;
```

- `TRUNCATE TABLE` is a valid operation on a log table. It can be used to expire log entries.
- `RENAME TABLE` is a valid operation on a log table. You can atomically rename a log table (to perform log rotation, for example) using the following strategy:

```
USE mysql;
DROP TABLE IF EXISTS general_log2;
CREATE TABLE general_log2 LIKE general_log;
RENAME TABLE general_log TO general_log_backup, general_log2 TO general_log;
```

- `CHECK TABLE` is a valid operation on a log table.
- `LOCK TABLES` cannot be used on a log table.
- `INSERT`, `DELETE`, and `UPDATE` cannot be used on a log table. These operations are permitted only internally to the server itself.
- `FLUSH TABLES WITH READ LOCK` and the state of the `read_only` system variable have no effect on log tables. The server can always write to the log tables.
- Entries written to the log tables are not written to the binary log and thus are not replicated to replica servers.
- To flush the log tables or log files, use `FLUSH TABLES` or `FLUSH LOGS`, respectively.
- Partitioning of log tables is not permitted.
- A `mysqldump` dump includes statements to recreate those tables so that they are not missing after reloading the dump file. Log table contents are not dumped.

5.4.2 The Error Log

This section discusses how to configure the MySQL server for logging of diagnostic messages to the error log. For information about selecting the error message character set and language, see Section 10.6, “Error Message Character Set”, and Section 10.12, “Setting the Error Message Language”.

The error log contains a record of `mysqld` startup and shutdown times. It also contains diagnostic messages such as errors, warnings, and notes that occur during server startup and shutdown, and while the server is running. For example, if `mysqld` notices that a table needs to be automatically checked or repaired, it writes a message to the error log.

On some operating systems, the error log contains a stack trace if `mysqld` exits abnormally. The trace can be used to determine where `mysqld` exited. See Section 5.8, “Debugging MySQL”.

If used to start `mysqld`, `mysqld_safe` may write messages to the error log. For example, when `mysqld_safe` notices abnormal `mysqld` exits, it restarts `mysqld` and writes a `mysqld restarted` message to the error log.

The following sections discuss aspects of configuring error logging. In the discussion, “console” means `stderr`, the standard error output. This is your terminal or console window unless the standard error output has been redirected to a different destination.

The server interprets options that determine where to write error messages somewhat differently for Windows and Unix systems. Be sure to configure error logging using the information appropriate to your platform.

5.4.2.1 Error Logging on Windows

On Windows, `mysqld` uses the `--log-error`, `--pid-file`, and `--console` options to determine whether `mysqld` writes the error log to the console or a file, and, if to a file, the file name:

- If `--console` is given, `mysqld` writes the error log to the console, unless `--log-error` is also given. `--log-error` takes precedence over `--console` if both are given.
- If `--log-error` is not given, or is given without naming a file, `mysqld` writes the error log to a file named *`host_name`*`.err` in the data directory, unless the `--pid-file` option is specified. In that case, the file name is the PID file base name with a suffix of `.err` in the data directory.
- If `--log-error` is given to name a file, `mysqld` writes the error log to that file (with an `.err` suffix added if the name has no suffix). The file location is under the data directory unless an absolute path name is given to specify a different location.

If the server writes the error log to a file, the `log_error` system variable indicates the error log file name.

In addition, the server by default writes events and error messages to the Windows Event Log within the Application log:

- Entries marked as `Error`, `Warning`, and `Note` are written to the Event Log, but not messages such as information statements from individual storage engines.
- Event Log entries have a source of `MySQL`.
- You cannot disable writing information to the Windows Event Log.

5.4.2.2 Error Logging on Unix and Unix-Like Systems

On Unix and Unix-like systems, `mysqld` uses the `--log-error` option to determine whether `mysqld` writes the error log to the console or a file, and, if to a file, the file name:

- If `--log-error` is not given, `mysqld` writes the error log to the console.
- If `--log-error` is given without naming a file, `mysqld` writes the error log to a file named *`host_name`*`.err` in the data directory.
- If `--log-error` is given to name a file, `mysqld` writes the error log to that file (with an `.err` suffix added if the name has no suffix). The file location is under the data directory unless an absolute path name is given to specify a different location.
- If `--log-error` is given in an option file in a `[mysqld]`, `[server]`, or `[mysqld_safe]` section, on systems that use `mysqld_safe` to start the server, `mysqld_safe` finds and uses the option, and passes it to `mysqld`.

Note

It is common for Yum or APT package installations to configure an error log file location under `/var/log` with an option like `log-error=/var/log/`

`mysqld.log` in a server configuration file. Removing the path name from the option causes the *`host_name`*`.err` file in the data directory to be used.

If the server writes the error log to a file, the `log_error` system variable indicates the error log file name.

5.4.2.3 Error Logging to the System Log

If you use `mysqld_safe` to start `mysqld`, `mysqld_safe` arranges for `mysqld` to write error messages to a log file or to `syslog`. `mysqld_safe` has three error-logging options, `--syslog`, `--skip-syslog`, and `--log-error`. The default with no logging options or with `--skip-syslog` is to use the default log file. To explicitly specify use of an error log file, specify `--log-error=`*`file_name`* to `mysqld_safe`, which then arranges for `mysqld` to write messages to a log file. To use `syslog` instead, specify the `--syslog` option.

Note

Error logging to the system log may require additional system configuration. Consult the system log documentation for your platform.

5.4.2.4 Error Log Filtering

The `--log-warnings` option or `log_warnings` system variable controls warning logging to the error log. The default value is enabled (1). To disable warning logging, set `--log-warnings` or `log_warnings` to 0. If the value is greater than 1, aborted connections are written to the error log, and access-denied errors for new connection attempts are written. See Section B.3.2.10, “Communication Errors and Aborted Connections”.

5.4.2.5 Error Log File Flushing and Renaming

If you flush the error log using a `FLUSH ERROR LOGS` or `FLUSH LOGS` statment, or a `mysqladmin flush-logs` command, the server closes and reopens any error log file to which it is writing. To rename an error log file, do so manually before flushing. Flushing the logs then opens a new file with the original file name. For example, assuming a log file name of *`host_name`*`.err`, use the following commands to rename the file and create a new one:

```
mv host_name.err host_name.err-old
mysqladmin flush-logs
mv host_name.err-old backup-directory
```

On Windows, use `rename` rather than `mv`.

If the location of the error log file is not writable by the server, the log-flushing operation fails to create a new log file. For example, on Linux, the server might write the error log to the `/var/log/mysqld.log` file, where the `/var/log` directory is owned by `root` and is not writable by `mysqld`. For information about handling this case, see Section 5.4.7, “Server Log Maintenance”.

If the server is not writing to a named error log file, no error log file renaming occurs when the error log is flushed.

5.4.3 The General Query Log

The general query log is a general record of what `mysqld` is doing. The server writes information to this log when clients connect or disconnect, and it logs each SQL statement received from clients. The general query log can be very useful when you suspect an error in a client and want to know exactly what the client sent to `mysqld`.

`mysqld` writes statements to the query log in the order that it receives them, which might differ from the order in which they are executed. This logging order is in contrast with that of the binary log, for which statements are written after they are executed but before any locks are released. In addition, the query log may contain statements that only select data while such statements are never written to the binary log.

When using statement-based binary logging on a replication source server, statements received by its replicas are written to the query log of each replica. Statements are written to the query log of the source server if a client reads events with the `mysqlbinlog` utility and passes them to the server.

However, when using row-based binary logging, updates are sent as row changes rather than SQL statements, and thus these statements are never written to the query log when `binlog_format` is `ROW`. A given update also might not be written to the query log when this variable is set to `MIXED`, depending on the statement used. See Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”, for more information.

By default, the general query log is disabled. To specify the initial general query log state explicitly, use `--general_log[={0|1}]`. With no argument or an argument of 1, `--general_log` enables the log. With an argument of 0, this option disables the log. To specify a log file name, use `--general_log_file=file_name`. To specify the log destination, use the `log_output` system variable (as described in Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”).

If you specify no name for the general query log file, the default name is *`host_name`*`.log`. The server creates the file in the data directory unless an absolute path name is given to specify a different directory.

To disable or enable the general query log or change the log file name at runtime, use the global `general_log` and `general_log_file` system variables. Set `general_log` to 0 (or `OFF`) to disable the log or to 1 (or `ON`) to enable it. Set `general_log_file` to specify the name of the log file. If a log file already is open, it is closed and the new file is opened.

When the general query log is enabled, the server writes output to any destinations specified by the `log_output` system variable. If you enable the log, the server opens the log file and writes startup messages to it. However, further logging of queries to the file does not occur unless the `FILE` log destination is selected. If the destination is `NONE`, the server writes no queries even if the general log is enabled. Setting the log file name has no effect on logging if the log destination value does not contain `FILE`.

Server restarts and log flushing do not cause a new general query log file to be generated (although flushing closes and reopens it). To rename the file and create a new one, use the following commands:

```
shell> mv host_name.log host_name-old.log
shell> mysqladmin flush-logs
shell> mv host_name-old.log backup-directory
```

On Windows, use `rename` rather than `mv`.

You can also rename the general query log file at runtime by disabling the log:

```
SET GLOBAL general_log = 'OFF';
```

With the log disabled, rename the log file externally (for example, from the command line). Then enable the log again:

```
SET GLOBAL general_log = 'ON';
```

This method works on any platform and does not require a server restart.

To disable or enable general query logging for the current session, set the session `sql_log_off` variable to `ON` or `OFF`. (This assumes that the general query log itself is enabled.)

Passwords in statements written to the general query log are rewritten by the server not to occur literally in plain text. Password rewriting can be suppressed for the general query log by starting the server with the `--log-raw` option. This option may be useful for diagnostic purposes, to see the exact text of statements as received by the server, but for security reasons is not recommended for production use.

One implication of the introduction of password rewriting is that statements that cannot be parsed (due, for example, to syntax errors) are no longer written to the general query log because they cannot be known to be password free. Use cases that require logging of all statements including those with errors should use the `--log-raw` option, bearing in mind that this also bypasses password rewriting.

Password rewriting occurs only when plain text passwords are expected. For statements with syntax that expect a password hash value, no rewriting occurs. If a plain text password is supplied erroneously for such syntax, the password is logged as given, without rewriting. For example, the following statement is logged as shown because a password hash value is expected:

```
CREATE USER 'user1'@'localhost' IDENTIFIED BY PASSWORD 'not-so-secret';
```

5.4.4 The Binary Log

The binary log contains “events” that describe database changes such as table creation operations or changes to table data. It also contains events for statements that potentially could have made changes (for example, a `DELETE` which matched no rows), unless row-based logging is used. The binary log also contains information about how long each statement took that updated data. The binary log has two important purposes:

- For replication, the binary log on a source replication server provides a record of the data changes to be sent to replica servers. The source server sends the events contained in its binary log to its replicas, which execute those events to make the same data changes that were made on the source. See Section 17.2, “Replication Implementation”.
- Certain data recovery operations require use of the binary log. After a backup has been restored, the events in the binary log that were recorded after the backup was made are re-executed. These events bring databases up to date from the point of the backup. See Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”.

The binary log is not used for statements such as `SELECT` or `SHOW` that do not modify data. To log all statements (for example, to identify a problem query), use the general query log. See Section 5.4.3, “The General Query Log”.

Running a server with binary logging enabled makes performance slightly slower. However, the benefits of the binary log in enabling you to set up replication and for restore operations generally outweigh this minor performance decrement.

The binary log is generally resilient to unexpected halts because only complete transactions are logged or read back. See Section 17.3.2, “Handling an Unexpected Halt of a Replica Server” for more information.

Passwords in statements written to the binary log are rewritten by the server not to occur literally in plain text. See also Section 6.1.2.3, “Passwords and Logging”.

The following discussion describes some of the server options and variables that affect the operation of binary logging. For a complete list, see Section 17.1.4.4, “Binary Log Options and Variables”.

To enable the binary log, start the server with the `--log-bin[=base_name]` option. If no *`base_name`* value is given, the default name is the value of the `--pid-file` option (which by default is the name of host machine) followed by `-bin`. If the base name is given, the server writes the file in the data directory unless the base name is given with a leading absolute path name to specify a different directory. It is recommended that you specify a base name explicitly rather than using the default of the host name; see Section B.3.7, “Known Issues in MySQL”, for the reason.

If you supply an extension in the log name (for example, `--log-bin=base_name.extension`), the extension is silently removed and ignored.

`mysqld` appends a numeric extension to the binary log base name to generate binary log file names. The number increases each time the server creates a new log file, thus creating an ordered series of files. The server creates a new file in the series each time any of the following events occurs:

- The server is started or restarted
- The server flushes the logs.
- The size of the current log file reaches `max_binlog_size`.

A binary log file may become larger than `max_binlog_size` if you are using large transactions because a transaction is written to the file in one piece, never split between files.

To keep track of which binary log files have been used, `mysqld` also creates a binary log index file that contains the names of the binary log files. By default, this has the same base name as the binary log file, with the extension `'.index'`. You can change the name of the binary log index file with the `--log-bin-index[=file_name]` option. You should not manually edit this file while `mysqld` is running; doing so would confuse `mysqld`.

The term “binary log file” generally denotes an individual numbered file containing database events. The term “binary log” collectively denotes the set of numbered binary log files plus the index file.

A client that has privileges sufficient to set restricted session system variables (see Section 5.1.8.1, “System Variable Privileges”) can disable binary logging of its own statements by using a `SET sql_log_bin=OFF` statement.

By default, the server logs the length of the event as well as the event itself and uses this to verify that the event was written correctly. You can also cause the server to write checksums for the events by setting the `binlog_checksum` system variable. When reading back from the binary log, the source uses the event length by default, but can be made to use checksums if available by enabling the `master_verify_checksum` system variable. The replica I/O thread also verifies events received from the source. You can cause the replica SQL thread to use checksums if available when reading from the relay log by enabling the `slave_sql_verify_checksum` system variable.

The format of the events recorded in the binary log is dependent on the binary logging format. Three format types are supported, row-based logging, statement-based logging and mixed-base logging. The binary logging format used depends on the MySQL version. For general descriptions of the logging formats, see Section 5.4.4.1, “Binary Logging Formats”. For detailed information about the format of the binary log, see MySQL Internals: The Binary Log.

The server evaluates the `--binlog-do-db` and `--binlog-ignore-db` options in the same way as it does the `--replicate-do-db` and `--replicate-ignore-db` options. For information about how this is done, see Section 17.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”.

A replication replica server by default does not write to its own binary log any data modifications that are received from the replication source. To log these modifications, start the replica with the `--log-slave-updates` option in addition to the `--log-bin` option (see Section 17.1.4.3, “Replica Server Options and Variables”). This is done when a replica is also to act as a source to other replicas in chained replication.

You can delete all binary log files with the `RESET MASTER` statement, or a subset of them with `PURGE BINARY LOGS`. See Section 13.7.6.6, “RESET Statement”, and Section 13.4.1.1, “PURGE BINARY LOGS Statement”.

If you are using replication, you should not delete old binary log files on the source until you are sure that no replica still needs to use them. For example, if your replicas never run more than three days behind, once a day you can execute `mysqladmin flush-logs` on the source and then remove any logs that are more than three days old. You can remove the files manually, but it is preferable to use `PURGE BINARY LOGS`, which also safely updates the binary log index file for you (and which can take a date argument). See Section 13.4.1.1, “PURGE BINARY LOGS Statement”.

You can display the contents of binary log files with the `mysqlbinlog` utility. This can be useful when you want to reprocess statements in the log for a recovery operation. For example, you can update a MySQL server from the binary log as follows:

```
shell> mysqlbinlog log_file | mysql -h server_name
```

`mysqlbinlog` also can be used to display replication replica relay log file contents because they are written using the same format as binary log files. For more information on the `mysqlbinlog` utility and how to use it, see Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”. For more information about the binary log and recovery operations, see Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”.

Binary logging is done immediately after a statement or transaction completes but before any locks are released or any commit is done. This ensures that the log is logged in commit order.

Updates to nontransactional tables are stored in the binary log immediately after execution.

Within an uncommitted transaction, all updates (`UPDATE`, `DELETE`, or `INSERT`) that change transactional tables such as `InnoDB` tables are cached until a `COMMIT` statement is received by the server. At that point, `mysqld` writes the entire transaction to the binary log before the `COMMIT` is executed.

Modifications to nontransactional tables cannot be rolled back. If a transaction that is rolled back includes modifications to nontransactional tables, the entire transaction is logged with a `ROLLBACK` statement at the end to ensure that the modifications to those tables are replicated.

When a thread that handles the transaction starts, it allocates a buffer of `binlog_cache_size` to buffer statements. If a statement is bigger than this, the thread opens a temporary file to store the transaction. The temporary file is deleted when the thread ends.

The `Binlog_cache_use` status variable shows the number of transactions that used this buffer (and possibly a temporary file) for storing statements. The `Binlog_cache_disk_use` status variable shows how many of those transactions actually had to use a temporary file. These two variables can be used for tuning `binlog_cache_size` to a large enough value that avoids the use of temporary files.

The `max_binlog_cache_size` system variable (default 4GB, which is also the maximum) can be used to restrict the total size used to cache a multiple-statement transaction. If a transaction is larger than this many bytes, it fails and rolls back. The minimum value is 4096.

If you are using the binary log and row based logging, concurrent inserts are converted to normal inserts for `CREATE ... SELECT` or `INSERT ... SELECT` statements. This is done to ensure that you can re-create an exact copy of your tables by applying the log during a backup operation. If you are using statement-based logging, the original statement is written to the log.

The binary log format has some known limitations that can affect recovery from backups. See Section 17.4.1, “Replication Features and Issues”.

Binary logging for stored programs is done as described in Section 20.7, “Stored Program Binary Logging”.

The binary log format differs in MySQL 5.6 from previous versions of MySQL, due to enhancements in replication. See Section 17.4.2, “Replication Compatibility Between MySQL Versions”.

If the server is unable to write to the binary log, flush binary log files, or synchronize the binary log to disk, the binary log on the replication source can become inconsistent and replication replicas can lose synchronization with the source. The `binlog_error_action` system variable controls the action taken if an error of this type is encountered with the binary log.

- The default setting in MySQL 5.6, `IGNORE_ERROR`, provides backward compatibility with older versions of MySQL. With this setting, the server continues the ongoing transaction and logs the error, then halts binary logging, but continues to perform updates. At this point, you can identify and correct the cause of the error. To resume binary logging, `log_bin` must be enabled again, which requires a server restart. Only use this option if you require backward compatibility, and the binary log is non-essential on this MySQL server instance. For example, you might use the binary log only for intermittent auditing or debugging of the server, and not use it for replication from the server or rely on it for point-in-time restore operations.

- The setting `ABORT_SERVER`, makes the server halt binary logging and shut down. At this point, you can identify and correct the cause of the error. On restart, recovery proceeds as in the case of an unexpected server halt (see Section 17.3.2, “Handling an Unexpected Halt of a Replica Server”). This is the recommended setting, particularly in complex replication environments.

By default, the binary log is not synchronized to disk at each write. So if the operating system or machine (not only the MySQL server) crashes, there is a chance that the last statements of the binary log are lost. To prevent this, use the `sync_binlog` system variable to synchronize the binary log to disk after every *`N`* commit groups. See Section 5.1.7, “Server System Variables”. The safest value for `sync_binlog` is 1, but this is also the slowest. Even with `sync_binlog` set to 1, there is still the chance of inconsistency between the table content and binary log content in case of a crash.

For example, if you are using `InnoDB` tables and the MySQL server processes a `COMMIT` statement, it writes many prepared transactions to the binary log in sequence, synchronizes the binary log, and then commits this transaction into `InnoDB`. If the server unexpectedly exits between those two operations, the transaction is rolled back by `InnoDB` at restart but still exists in the binary log. Such an issue is resolved assuming `--innodb_support_xa` is set to 1, the default. Although this option is related to the support of XA transactions in InnoDB, it also ensures that the binary log and InnoDB data files are synchronized. For this option to provide a greater degree of safety, the MySQL server should also be configured to synchronize the binary log and the `InnoDB` logs to disk before committing the transaction. The `InnoDB` logs are synchronized by default, and `sync_binlog=1` can be used to synchronize the binary log. The effect of this option is that at restart after a crash, after doing a rollback of transactions, the MySQL server scans the latest binary log file to collect transaction *`xid`* values and calculate the last valid position in the binary log file. The MySQL server then tells `InnoDB` to complete any prepared transactions that were successfully written to the to the binary log, and truncates the binary log to the last valid position. This ensures that the binary log reflects the exact data of `InnoDB` tables, and therefore the replica remains in synchrony with the source because it does not receive a statement which has been rolled back.

If the MySQL server discovers at crash recovery that the binary log is shorter than it should have been, it lacks at least one successfully committed `InnoDB` transaction. This should not happen if `sync_binlog=1` and the disk/file system do an actual sync when they are requested to (some do not), so the server prints an error message `The binary log file_name is shorter than its expected size`. In this case, this binary log is not correct and replication should be restarted from a fresh snapshot of the source's data.

The session values of the following system variables are written to the binary log and honored by the replication replica when parsing the binary log:

- `sql_mode` (except that the `NO_DIR_IN_CREATE` mode is not replicated; see Section 17.4.1.35, “Replication and Variables”)
- `foreign_key_checks`
- `unique_checks`
- `character_set_client`
- `collation_connection`
- `collation_database`
- `collation_server`
- `sql_auto_is_null`

5.4.4.1 Binary Logging Formats

The server uses several logging formats to record information in the binary log. The exact format employed depends on the version of MySQL being used. There are three logging formats:

- Replication capabilities in MySQL originally were based on propagation of SQL statements from source to replica. This is called *statement-based logging*. You can cause this format to be used by starting the server with `--binlog-format=STATEMENT`.
- In *row-based logging*, the source writes events to the binary log that indicate how individual table rows are affected. You can cause the server to use row-based logging by starting it with `--binlog-format=ROW`.
- A third option is also available: *mixed logging*. With mixed logging, statement-based logging is used by default, but the logging mode switches automatically to row-based in certain cases as described below. You can cause MySQL to use mixed logging explicitly by starting `mysqld` with the option `--binlog-format=MIXED`.

In MySQL 5.6, the default binary logging format is `STATEMENT`.

The logging format can also be set or limited by the storage engine being used. This helps to eliminate issues when replicating certain statements between a source and replica which are using different storage engines.

With statement-based replication, there may be issues with replicating nondeterministic statements. In deciding whether or not a given statement is safe for statement-based replication, MySQL determines whether it can guarantee that the statement can be replicated using statement-based logging. If MySQL cannot make this guarantee, it marks the statement as potentially unreliable and issues the warning, `Statement may not be safe to log in statement format`.

You can avoid these issues by using MySQL's row-based replication instead.

5.4.4.2 Setting The Binary Log Format

You can select the binary logging format explicitly by starting the MySQL server with `--binlog-format=type`. The supported values for *`type`* are:

- `STATEMENT` causes logging to be statement based.
- `ROW` causes logging to be row based.
- `MIXED` causes logging to use mixed format.

In MySQL 5.6, the default binary logging format is `STATEMENT`.

The logging format also can be switched at runtime. Set the global value of the `binlog_format` system variable to specify the format for clients that connect subsequent to the change:

```
mysql> SET GLOBAL binlog_format = 'STATEMENT';
mysql> SET GLOBAL binlog_format = 'ROW';
mysql> SET GLOBAL binlog_format = 'MIXED';
```

An individual client can control the logging format for its own statements by setting the session value of `binlog_format`:

```
mysql> SET SESSION binlog_format = 'STATEMENT';
mysql> SET SESSION binlog_format = 'ROW';
mysql> SET SESSION binlog_format = 'MIXED';
```

Changing the global `binlog_format` value requires privileges sufficient to set global system variables. Changing the session `binlog_format` value requires privileges sufficient to set restricted session system variables. See Section 5.1.8.1, “System Variable Privileges”.

There are several reasons why a client might want to set binary logging on a per-session basis:

- A session that makes many small changes to the database might want to use row-based logging.

- A session that performs updates that match many rows in the `WHERE` clause might want to use statement-based logging because it is more efficient to log a few statements than many rows.
- Some statements require a lot of execution time on the source, but result in just a few rows being modified. It might therefore be beneficial to replicate them using row-based logging.

There are exceptions when you cannot switch the replication format at runtime:

- From within a stored function or a trigger
- If the `NDB` storage engine is enabled
- If the session is currently in row-based replication mode and has open temporary tables

Trying to switch the format in any of these cases results in an error.

Switching the replication format at runtime is not recommended when any temporary tables exist, because temporary tables are logged only when using statement-based replication, whereas with row-based replication they are not logged. With mixed replication, temporary tables are usually logged; exceptions happen with user-defined functions (UDFs) and with the `UUID()` function.

Switching the replication format while replication is ongoing can also cause issues. Each MySQL Server can set its own and only its own binary logging format (true whether `binlog_format` is set with global or session scope). This means that changing the logging format on a replication source does not cause a replica to change its logging format to match. When using `STATEMENT` mode, the `binlog_format` system variable is not replicated. When using `MIXED` or `ROW` logging mode, it is replicated but is ignored by the replica.

A replica is not able to convert binary log entries received in `ROW` logging format to `STATEMENT` format for use in its own binary log. The replica must therefore use `ROW` or `MIXED` format if the source does. Changing the binary logging format on the source from `STATEMENT` to `ROW` or `MIXED` while replication is ongoing to a replica with `STATEMENT` format can cause replication to fail with errors such as `Error executing row event: 'Cannot execute statement: impossible to write to binary log since statement is in row format and BINLOG_FORMAT = STATEMENT.'` Changing the binary logging format on the replica to `STATEMENT` format when the source is still using `MIXED` or `ROW` format also causes the same type of replication failure. To change the format safely, you must stop replication and ensure that the same change is made on both the source and the replica.

If you are using `InnoDB` tables and the transaction isolation level is `READ COMMITTED` or `READ UNCOMMITTED`, only row-based logging can be used. It is *possible* to change the logging format to `STATEMENT`, but doing so at runtime leads very rapidly to errors because `InnoDB` can no longer perform inserts.

With the binary log format set to `ROW`, many changes are written to the binary log using the row-based format. Some changes, however, still use the statement-based format. Examples include all DDL (data definition language) statements such as `CREATE TABLE`, `ALTER TABLE`, or `DROP TABLE`.

The `--binlog-row-event-max-size` option is available for servers that are capable of row-based replication. Rows are stored into the binary log in chunks having a size in bytes not exceeding the value of this option. The value must be a multiple of 256. The default value is 8192.

Warning

When using *statement-based logging* for replication, it is possible for the data on the source and replica to become different if a statement is designed in such a way that the data modification is *nondeterministic*; that is, it is left to the will of the query optimizer. In general, this is not a good practice even outside of replication. For a detailed explanation of this issue, see Section B.3.7, “Known Issues in MySQL”.

For information about logs kept by replicas, see Section 17.2.2, “Relay Log and Replication Metadata Repositories”.

5.4.4.3 Mixed Binary Logging Format

When running in `MIXED` logging format, the server automatically switches from statement-based to row-based logging under the following conditions:

- When a function contains `UUID()`.
- When one or more tables with `AUTO_INCREMENT` columns are updated and a trigger or stored function is invoked. Like all other unsafe statements, this generates a warning if `binlog_format = STATEMENT`.

  For more information, see Section 17.4.1.1, “Replication and AUTO_INCREMENT”.
- When the body of a view requires row-based replication, the statement creating the view also uses it. For example, this occurs when the statement creating a view uses the `UUID()` function.
- When a call to a UDF is involved.
- When any `INSERT DELAYED` is executed for a nontransactional table.
- If a statement is logged by row and the session that executed the statement has any temporary tables, logging by row is used for all subsequent statements (except for those accessing temporary tables) until all temporary tables in use by that session are dropped.

  This is true whether or not any temporary tables are actually logged.

  Temporary tables cannot be logged using row-based format; thus, once row-based logging is used, all subsequent statements using that table are unsafe. The server approximates this condition by treating all statements executed during the session as unsafe until the session no longer holds any temporary tables.
- When `FOUND_ROWS()` or `ROW_COUNT()` is used. (Bug #12092, Bug #30244)
- When `USER()`, `CURRENT_USER()`, or `CURRENT_USER` is used. (Bug #28086)
- When a statement refers to one or more system variables. (Bug #31168)

  Exception. The following system variables, when used with session scope (only), do not cause the logging format to switch:

  - `auto_increment_increment`
  - `auto_increment_offset`
  - `character_set_client`
  - `character_set_connection`
  - `character_set_database`
  - `character_set_server`
  - `collation_connection`
  - `collation_database`
  - `collation_server`
  - `foreign_key_checks`
  - `identity`
  - `last_insert_id`

  - `lc_time_names`
  - `pseudo_thread_id`
  - `sql_auto_is_null`
  - `time_zone`
  - `timestamp`
  - `unique_checks`

  For information about determining system variable scope, see Section 5.1.8, “Using System Variables”.

  For information about how replication treats `sql_mode`, see Section 17.4.1.35, “Replication and Variables”.

- When one of the tables involved is a log table in the `mysql` database.
- When the `LOAD_FILE()` function is used. (Bug #39701)

Note

A warning is generated if you try to execute a statement using statement-based logging that should be written using row-based logging. The warning is shown both in the client (in the output of `SHOW WARNINGS`) and through the `mysqld` error log. A warning is added to the `SHOW WARNINGS` table each time such a statement is executed. However, only the first statement that generated the warning for each client session is written to the error log to prevent flooding the log.

In addition to the decisions above, individual engines can also determine the logging format used when information in a table is updated. The logging capabilities of an individual engine can be defined as follows:

- If an engine supports row-based logging, the engine is said to be *row-logging capable*.
- If an engine supports statement-based logging, the engine is said to be *statement-logging capable*.

A given storage engine can support either or both logging formats. The following table lists the formats supported by each engine.

Storage Engine	Row Logging Supported	Statement Logging Supported
`ARCHIVE`	Yes	Yes
`BLACKHOLE`	Yes	Yes
`CSV`	Yes	Yes
`EXAMPLE`	Yes	No
`FEDERATED`	Yes	Yes
`HEAP`	Yes	Yes
`InnoDB`	Yes	Yes when the transaction isolation level is `REPEATABLE READ` or `SERIALIZABLE`; No otherwise.
`MyISAM`	Yes	Yes

Storage Engine	Row Logging Supported	Statement Logging Supported
MERGE	Yes	Yes
NDB	Yes	No

Whether a statement is to be logged and the logging mode to be used is determined according to the type of statement (safe, unsafe, or binary injected), the binary logging format (STATEMENT, ROW, or MIXED), and the logging capabilities of the storage engine (statement capable, row capable, both, or neither). (Binary injection refers to logging a change that must be logged using ROW format.)

Statements may be logged with or without a warning; failed statements are not logged, but generate errors in the log. This is shown in the following decision table. **Type**, **binlog_format**, **SLC**, and **RLC** columns outline the conditions, and **Error / Warning** and **Logged as** columns represent the corresponding actions. **SLC** stands for “statement-logging capable”, and **RLC** stands for “row-logging capable”.

Type	binlog_format	SLC	RLC	Error / Warning	Logged as
*	*	No	No	Error: Cannot execute statement: Binary logging is impossible since at least one engine is involved that is both row-incapable and statement-incapable.	-
Safe	STATEMENT	Yes	No	-	STATEMENT
Safe	MIXED	Yes	No	-	STATEMENT
Safe	ROW	Yes	No	Error: Cannot execute statement: Binary logging is impossible since BINLOG_FORMAT = ROW and at least one table uses a storage engine that is not capable of row-based logging.	-
Unsafe	STATEMENT	Yes	No	Warning: Unsafe statement binlogged in statement format, since BINLOG_FORMAT = STATEMENT	STATEMENT
Unsafe	MIXED	Yes	No	Error: Cannot execute statement: Binary logging of an unsafe statement is impossible when	-

Type	binlog_format	SLC	RLC	Error / Warning	Logged as
				the storage engine is limited to statement-based logging, even if BINLOG_FORMAT = MIXED.	
Unsafe	ROW	Yes	No	Error: Cannot execute statement: Binary logging is impossible since BINLOG_FORMAT = ROW and at least one table uses a storage engine that is not capable of row-based logging.	-
Row Injection	STATEMENT	Yes	No	Error: Cannot execute row injection: Binary logging is not possible since at least one table uses a storage engine that is not capable of row-based logging.	-
Row Injection	MIXED	Yes	No	Error: Cannot execute row injection: Binary logging is not possible since at least one table uses a storage engine that is not capable of row-based logging.	-
Row Injection	ROW	Yes	No	Error: Cannot execute row injection: Binary logging is not possible since at least one table uses a storage engine that is not capable of row-based logging.	-
Safe	STATEMENT	No	Yes	Error: Cannot execute statement: Binary logging is impossible since BINLOG_FORMAT = STATEMENT and	-

Type	binlog_format	SLC	RLC	Error / Warning	Logged as
				at least one table uses a storage engine that is not capable of statement-based logging.	
Safe	MIXED	No	Yes	-	ROW
Safe	ROW	No	Yes	-	ROW
Unsafe	STATEMENT	No	Yes	Error: Cannot execute statement: Binary logging is impossible since BINLOG_FORMAT = STATEMENT and at least one table uses a storage engine that is not capable of statement-based logging.	-
Unsafe	MIXED	No	Yes	-	ROW
Unsafe	ROW	No	Yes	-	ROW
Row Injection	STATEMENT	No	Yes	Error: Cannot execute row injection: Binary logging is not possible since BINLOG_FORMAT = STATEMENT.	-
Row Injection	MIXED	No	Yes	-	ROW
Row Injection	ROW	No	Yes	-	ROW
Safe	STATEMENT	Yes	Yes	-	STATEMENT
Safe	MIXED	Yes	Yes	-	STATEMENT
Safe	ROW	Yes	Yes	-	ROW
Unsafe	STATEMENT	Yes	Yes	Warning: Unsafe statement binlogged in statement format since BINLOG_FORMAT = STATEMENT.	STATEMENT
Unsafe	MIXED	Yes	Yes	-	ROW
Unsafe	ROW	Yes	Yes	-	ROW
Row Injection	STATEMENT	Yes	Yes	Error: Cannot execute row injection: Binary logging	-

Type	binlog_format	SLC	RLC	Error / Warning	Logged as
				is not possible because BINLOG_FORMAT = STATEMENT.	
Row Injection	MIXED	Yes	Yes	-	ROW
Row Injection	ROW	Yes	Yes	-	ROW

When a warning is produced by the determination, a standard MySQL warning is produced (and is available using `SHOW WARNINGS`). The information is also written to the `mysqld` error log. Only one error for each error instance per client connection is logged to prevent flooding the log. The log message includes the SQL statement that was attempted.

If a replica server was started with `--log-warnings` enabled, the replica prints messages to the error log to provide information about its status, such as the binary log and relay log coordinates where it starts its job, when it is switching to another relay log, when it reconnects after a disconnect, and so forth.

5.4.4.4 Logging Format for Changes to mysql Database Tables

The contents of the grant tables in the `mysql` database can be modified directly (for example, with `INSERT` or `DELETE`) or indirectly (for example, with `GRANT` or `CREATE USER`). Statements that affect `mysql` database tables are written to the binary log using the following rules:

- Data manipulation statements that change data in `mysql` database tables directly are logged according to the setting of the `binlog_format` system variable. This pertains to statements such as `INSERT`, `UPDATE`, `DELETE`, `REPLACE`, `DO`, `LOAD DATA`, `SELECT`, and `TRUNCATE TABLE`.
- Statements that change the `mysql` database indirectly are logged as statements regardless of the value of `binlog_format`. This pertains to statements such as `GRANT`, `REVOKE`, `SET PASSWORD`, `RENAME USER`, `CREATE` (all forms except `CREATE TABLE ... SELECT`), `ALTER` (all forms), and `DROP` (all forms).

`CREATE TABLE ... SELECT` is a combination of data definition and data manipulation. The `CREATE TABLE` part is logged using statement format and the `SELECT` part is logged according to the value of `binlog_format`.

5.4.5 The Slow Query Log

The slow query log consists of SQL statements that take more than `long_query_time` seconds to execute and require at least `min_examined_row_limit` rows to be examined. The slow query log can be used to find queries that take a long time to execute and are therefore candidates for optimization. However, examining a long slow query log can be a time-consuming task. To make this easier, you can use the `mysqldumpslow` command to process a slow query log file and summarize its contents. See Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”.

The time to acquire the initial locks is not counted as execution time. `mysqld` writes a statement to the slow query log after it has been executed and after all locks have been released, so log order might differ from execution order.

- Slow Query Log Parameters
- Slow Query Log Contents

Slow Query Log Parameters

The minimum and default values of `long_query_time` are 0 and 10, respectively. The value can be specified to a resolution of microseconds. For logging to a file, times are written including the

microseconds part. For logging to tables, only integer times are written; the microseconds part is ignored.

By default, administrative statements are not logged, nor are queries that do not use indexes for lookups. This behavior can be changed using `log_slow_admin_statements` and `log_queries_not_using_indexes`, as described later.

By default, the slow query log is disabled. To specify the initial slow query log state explicitly, use `--slow_query_log[={0|1}]`. With no argument or an argument of 1, `--slow_query_log` enables the log. With an argument of 0, this option disables the log. To specify a log file name, use `--slow_query_log_file=file_name`. To specify the log destination, use the `log_output` system variable (as described in Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”).

If you specify no name for the slow query log file, the default name is *`host_name`*`-slow.log`. The server creates the file in the data directory unless an absolute path name is given to specify a different directory.

To disable or enable the slow query log or change the log file name at runtime, use the global `slow_query_log` and `slow_query_log_file` system variables. Set `slow_query_log` to 0 to disable the log or to 1 to enable it. Set `slow_query_log_file` to specify the name of the log file. If a log file already is open, it is closed and the new file is opened.

The server writes less information to the slow query log if you use the `--log-short-format` option.

To include slow administrative statements in the slow query log, enable the `log_slow_admin_statements` system variable. Administrative statements include `ALTER TABLE`, `ANALYZE TABLE`, `CHECK TABLE`, `CREATE INDEX`, `DROP INDEX`, `OPTIMIZE TABLE`, and `REPAIR TABLE`.

To include queries that do not use indexes for row lookups in the statements written to the slow query log, enable the `log_queries_not_using_indexes` system variable. (Even with that variable enabled, the server does not log queries that would not benefit from the presence of an index due to the table having fewer than two rows.)

When queries that do not use an index are logged, the slow query log may grow quickly. It is possible to put a rate limit on these queries by setting the `log_throttle_queries_not_using_indexes` system variable. By default, this variable is 0, which means there is no limit. Positive values impose a per-minute limit on logging of queries that do not use indexes. The first such query opens a 60-second window within which the server logs queries up to the given limit, then suppresses additional queries. If there are suppressed queries when the window ends, the server logs a summary that indicates how many there were and the aggregate time spent in them. The next 60-second window begins when the server logs the next query that does not use indexes.

The server uses the controlling parameters in the following order to determine whether to write a query to the slow query log:

1. The query must either not be an administrative statement, or `log_slow_admin_statements` must be enabled.
2. The query must have taken at least `long_query_time` seconds, or `log_queries_not_using_indexes` must be enabled and the query used no indexes for row lookups.
3. The query must have examined at least `min_examined_row_limit` rows.
4. The query must not be suppressed according to the `log_throttle_queries_not_using_indexes` setting.

The server does not log queries handled by the query cache.

By default, a replica does not write replicated queries to the slow query log. To change this, enable the `log_slow_slave_statements` system variable. Note that if row-based replication is in use (`binlog_format=ROW`), `log_slow_slave_statements` has no effect. Queries are only added to the replica's slow query log when they are logged in statement format in the binary log, that is, when `binlog_format=STATEMENT` is set, or when `binlog_format=MIXED` is set and the statement is logged in statement format. Slow queries that are logged in row format when `binlog_format=MIXED` is set, or that are logged when `binlog_format=ROW` is set, are not added to the replica's slow query log, even if `log_slow_slave_statements` is enabled.

Slow Query Log Contents

When the slow query log is enabled, the server writes output to any destinations specified by the `log_output` system variable. If you enable the log, the server opens the log file and writes startup messages to it. However, further logging of queries to the file does not occur unless the `FILE` log destination is selected. If the destination is `NONE`, the server writes no queries even if the slow query log is enabled. Setting the log file name has no effect on logging if `FILE` is not selected as an output destination.

If the slow query log is enabled and `FILE` is selected as an output destination, each statement written to the log is preceded by a line that begins with a `#` character and has these fields (with all fields on a single line):

- `Query_time: duration`

  The statement execution time in seconds.

- `Lock_time: duration`

  The time to acquire locks in seconds.

- `Rows_sent: N`

  The number of rows sent to the client.

- `Rows_examined:`

  The number of rows examined by the server layer (not counting any processing internal to storage engines).

Each statement written to the slow query log file is preceded by a `SET` statement that includes a timestamp indicating when the slow statement was logged (which occurs after the statement finishes executing).

Passwords in statements written to the slow query log are rewritten by the server not to occur literally in plain text. See Section 6.1.2.3, “Passwords and Logging”.

5.4.6 The DDL Log

The DDL log, or metadata log, records metadata operations generated by data definition statements such as `DROP TABLE` and `ALTER TABLE`. MySQL uses this log to recover from crashes occurring in the middle of a metadata operation. When executing the statement `DROP TABLE t1, t2`, we need to ensure that both `t1` and `t2` are dropped, and that each table drop is complete. Another example of this type of SQL statement is `ALTER TABLE t3 DROP PARTITION p2`, where we must make certain that the partition is completely dropped and that its definition is removed from the list of partitions for table `t3`.

A record of metadata operations such as those just described are written to the file `ddl_log.log`, in the MySQL data directory. This is a binary file; it is not intended to be human-readable, and you should not attempt to modify its contents in any way.

`ddl_log.log` is not created until it is actually needed for recording metadata statements, and is removed following a successful start of `mysqld`. Thus, it is possible for this file not to be present on a MySQL server that is functioning in a completely normal manner.

Currently, `ddl_log.log` can hold up to 1048573 entries, equivalent to 4 GB in size. Once this limit is exceeded, you must rename or remove the file before it is possible to execute any additional DDL statements. This is a known issue which we are working to resolve (Bug #83708).

There are no user-configurable server options or variables associated with this file.

5.4.7 Server Log Maintenance

As described in Section 5.4, “MySQL Server Logs”, MySQL Server can create several different log files to help you see what activity is taking place. However, you must clean up these files regularly to ensure that the logs do not take up too much disk space.

When using MySQL with logging enabled, you may want to back up and remove old log files from time to time and tell MySQL to start logging to new files. See Section 7.2, “Database Backup Methods”.

On a Linux (Red Hat) installation, you can use the `mysql-log-rotate` script for log maintenance. If you installed MySQL from an RPM distribution, this script should have been installed automatically. Be careful with this script if you are using the binary log for replication. You should not remove binary logs until you are certain that their contents have been processed by all replicas.

On other systems, you must install a short script yourself that you start from `cron` (or its equivalent) for handling log files.

For the binary log, you can set the `expire_logs_days` system variable to expire binary log files automatically after a given number of days (see Section 5.1.7, “Server System Variables”). If you are using replication, you should set the variable no lower than the maximum number of days your replicas might lag behind the source. To remove binary logs on demand, use the `PURGE BINARY LOGS` statement (see Section 13.4.1.1, “PURGE BINARY LOGS Statement”).

To force MySQL to start using new log files, flush the logs. Log flushing occurs when you execute a `FLUSH LOGS` statement or a `mysqladmin flush-logs`, `mysqladmin refresh`, `mysqldump --flush-logs`, or `mysqldump --master-data` command. See Section 13.7.6.3, “FLUSH Statement”, Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”, and Section 4.5.4, “`mysqldump` — A Database Backup Program”. In addition, the server flushes the binary log automatically when current binary log file size reaches the value of the `max_binlog_size` system variable.

`FLUSH LOGS` supports optional modifiers to enable selective flushing of individual logs (for example, `FLUSH BINARY LOGS`). See Section 13.7.6.3, “FLUSH Statement”.

A log-flushing operation has the following effects:

- If binary logging is enabled, the server closes the current binary log file and opens a new log file with the next sequence number.
- If general query logging or slow query logging to a log file is enabled, the server closes and reopens the log file.
- If the server was started with the `--log-error` option to cause the error log to be written to a file, the server closes and reopens the log file.

Execution of log-flushing statements or commands requires connecting to the server using an account that has the `RELOAD` privilege. On Unix and Unix-like systems, another way to flush the logs is to send a `SIGHUP` signal to the server, which can be done by `root` or the account that owns the server

process. Signals enable log flushing to be performed without having to connect to the server. However, `SIGHUP` has additional effects other than log flushing that might be undesirable. For details, see Section 4.10, “Unix Signal Handling in MySQL”.

As mentioned previously, flushing the binary log creates a new binary log file, whereas flushing the general query log, slow query log, or error log just closes and reopens the log file. For the latter logs, to cause a new log file to be created on Unix, rename the current log file first before flushing it. At flush time, the server opens the new log file with the original name. For example, if the general query log, slow query log, and error log files are named `mysql.log`, `mysql-slow.log`, and `err.log`, you can use a series of commands like this from the command line:

```
cd mysql-data-directory
mv mysql.log mysql.log.old
mv mysql-slow.log mysql-slow.log.old
mv err.log err.log.old
mysqladmin flush-logs
```

On Windows, use `rename` rather than `mv`.

At this point, you can make a backup of `mysql.log.old`, `mysql-slow.log.old`, and `err.log.old`, then remove them from disk.

To rename the general query log or slow query log at runtime, first connect to the server and disable the log:

```
SET GLOBAL general_log = 'OFF';
SET GLOBAL slow_query_log = 'OFF';
```

With the logs disabled, rename the log files externally (for example, from the command line). Then enable the logs again:

```
SET GLOBAL general_log = 'ON';
SET GLOBAL slow_query_log = 'ON';
```

This method works on any platform and does not require a server restart.

Note

For the server to recreate a given log file after you have renamed the file externally, the file location must be writable by the server. This may not always be the case. For example, on Linux, the server might write the error log as `/var/log/mysqld.log`, where `/var/log` is owned by `root` and not writable by `mysqld`. In this case, log-flushing operations fail to create a new log file.

To handle this situation, you must manually create the new log file with the proper ownership after renaming the original log file. For example, execute these commands as `root`:

```
mv /var/log/mysqld.log /var/log/mysqld.log.old
install -omysql -gmysql -m0644 /dev/null /var/log/mysqld.log
```

5.5 MySQL Server Plugins

MySQL supports an plugin API that enables creation of server plugins. Plugins can be loaded at server startup, or loaded and unloaded at runtime without restarting the server. The plugins supported by this interface include, but are not limited to, storage engines, `INFORMATION_SCHEMA` tables, full-text parser plugins, partitioning support, and server extensions.

MySQL distributions include several plugins that implement server extensions:

- Plugins for authenticating attempts by clients to connect to MySQL Server. Plugins are available for several authentication protocols. See Section 6.2.11, “Pluggable Authentication”.

- A connection-control plugin that enables administrators to introduce an increasing delay after a certain number of consecutive failed client connection attempts. See Section 6.4.2, “The Connection-Control Plugins”.
- A password-validation plugin implements password strength policies and assesses the strength of potential passwords. See Section 6.4.3, “The Password Validation Plugin”.
- Semisynchronous replication plugins implement an interface to replication capabilities that permit the source to proceed as long as at least one replica has responded to each transaction. See Section 17.3.9, “Semisynchronous Replication”.
- MySQL Enterprise Edition includes a thread pool plugin that manages connection threads to increase server performance by efficiently managing statement execution threads for large numbers of client connections. See Section 5.5.3, “MySQL Enterprise Thread Pool”.
- MySQL Enterprise Edition includes an audit plugin for monitoring and logging of connection and query activity. See Section 6.4.4, “MySQL Enterprise Audit”.
- MySQL Enterprise Edition includes a firewall plugin that implements an application-level firewall to enable database administrators to permit or deny SQL statement execution based on matching against allowlists of accepted statement patterns. See Section 6.4.5, “MySQL Enterprise Firewall”.

The following sections describe how to install and uninstall plugins, and how to determine at runtime which plugins are installed and obtain information about them. For information about writing plugins, see The MySQL Plugin API.

5.5.1 Installing and Uninstalling Plugins

Server plugins must be loaded into the server before they can be used. MySQL supports plugin loading at server startup and runtime. It is also possible to control the activation state of loaded plugins at startup, and to unload them at runtime.

While a plugin is loaded, information about it is available as described in Section 5.5.2, “Obtaining Server Plugin Information”.

- Installing Plugins
- Controlling Plugin Activation State
- Uninstalling Plugins

Installing Plugins

Before a server plugin can be used, it must be installed using one of the following methods. In the descriptions, *`plugin_name`* stands for a plugin name such as `innodb`, `csv`, or `validate_password`.

- Built-in Plugins
- Plugins Registered in the mysql.plugin System Table
- Plugins Named with Command-Line Options
- Plugins Installed with the INSTALL PLUGIN Statement

Built-in Plugins

A built-in plugin is known by the server automatically. By default, the server enables the plugin at startup. Some built-in plugins permit this to be changed with the *`--plugin_name`*`[=`*`activation_state`*`]` option.

Plugins Registered in the mysql.plugin System Table

The `mysql.plugin` system table serves as a registry of plugins (other than built-in plugins, which need not be registered). During the normal startup sequence, the server loads plugins registered in the table. By default, for a plugin loaded from the `mysql.plugin` table, the server also enables the plugin. This can be changed with the `--plugin_name[=activation_state]` option.

If the server is started with the `--skip-grant-tables` option, plugins registered in the `mysql.plugin` table are not loaded and are unavailable.

Plugins Named with Command-Line Options

A plugin located in a plugin library file can be loaded at server startup with the `--plugin-load` or `--plugin-load-add` option. Normally, for a plugin loaded at startup, the server also enables the plugin. This can be changed with the `--plugin_name[=activation_state]` option.

The value of each plugin-loading option is a semicolon-separated list of *`name=plugin_library`* and *`plugin_library`* values. Each *`name`* is the name of a plugin to load, and *`plugin_library`* is the name of the library file that contains the plugin code. If a plugin library is named without any preceding plugin name, the server loads all plugins in the library. The server looks for plugin library files in the directory named by the `plugin_dir` system variable.

Plugin-loading options do not register any plugin in the `mysql.plugin` table. For subsequent restarts, the server loads the plugin again only if `--plugin-load` or `--plugin-load-add` is given again. That is, the option produces a one-time plugin-installation operation that persists for a single server invocation.

`--plugin-load` and `--plugin-load-add` enable plugins to be loaded even when `--skip-grant-tables` is given (which causes the server to ignore the `mysql.plugin` table). `--plugin-load` and `--plugin-load-add` also enable plugins to be loaded at startup that cannot be loaded at runtime.

The `--plugin-load-add` option complements the `--plugin-load` option:

- Each instance of `--plugin-load` resets the set of plugins to load at startup, whereas `--plugin-load-add` adds a plugin or plugins to the set of plugins to be loaded without resetting the current set. Consequently, if multiple instances of `--plugin-load` are specified, only the last one takes effect. With multiple instances of `--plugin-load-add`, all of them take effect.
- The argument format is the same as for `--plugin-load`, but multiple instances of `--plugin-load-add` can be used to avoid specifying a large set of plugins as a single long unwieldy `--plugin-load` argument.
- `--plugin-load-add` can be given in the absence of `--plugin-load`, but any instance of `--plugin-load-add` that appears before `--plugin-load` has no effect because `--plugin-load` resets the set of plugins to load.

For example, these options:

```
--plugin-load=x --plugin-load-add=y
```

are equivalent to these options:

```
--plugin-load-add=x --plugin-load-add=y
```

and are also equivalent to this option:

```
--plugin-load="x;y"
```

But these options:

```
--plugin-load-add=y --plugin-load=x
```

are equivalent to this option:

```
--plugin-load=x
```

Plugins Installed with the INSTALL PLUGIN Statement

A plugin located in a plugin library file can be loaded at runtime with the `INSTALL PLUGIN` statement. The statement also registers the plugin in the `mysql.plugin` table to cause the server to load it on subsequent restarts. For this reason, `INSTALL PLUGIN` requires the `INSERT` privilege for the `mysql.plugin` table.

The plugin library file base name depends on your platform. Common suffixes are `.so` for Unix and Unix-like systems, `.dll` for Windows.

Example: The `--plugin-load-add` option installs a plugin at server startup. To install a plugin named `myplugin` from a plugin library file named `somepluglib.so`, use these lines in a `my.cnf` file:

```
[mysqld]
plugin-load-add=myplugin=somepluglib.so
```

In this case, the plugin is not registered in `mysql.plugin`. Restarting the server without the `--plugin-load-add` option causes the plugin not to be loaded at startup.

Alternatively, the `INSTALL PLUGIN` statement causes the server to load the plugin code from the library file at runtime:

```
INSTALL PLUGIN myplugin SONAME 'somepluglib.so';
```

`INSTALL PLUGIN` also causes “permanent” plugin registration: The plugin is listed in the `mysql.plugin` table to ensure that the server loads it on subsequent restarts.

Many plugins can be loaded either at server startup or at runtime. However, if a plugin is designed such that it must be loaded and initialized during server startup, attempts to load it at runtime using `INSTALL PLUGIN` produce an error:

```
mysql> INSTALL PLUGIN myplugin SONAME 'somepluglib.so';
ERROR 1721 (HY000): Plugin 'myplugin' is marked as not dynamically
installable. You have to stop the server to install it.
```

In this case, you must use `--plugin-load` or `--plugin-load-add`.

Many plugins can be loaded either at server startup or at runtime. However, if a plugin is designed such that it must be loaded and initialized during server startup, use `--plugin-load` or `--plugin-load-add` rather than `INSTALL PLUGIN`.

If a plugin is named both using a `--plugin-load` or `--plugin-load-add` option and (as a result of an earlier `INSTALL PLUGIN` statement) in the `mysql.plugin` table, the server starts but writes these messages to the error log:

```
[ERROR] Function 'plugin_name' already exists
[Warning] Couldn't load plugin named 'plugin_name'
with soname 'plugin_object_file'.
```

Controlling Plugin Activation State

If the server knows about a plugin when it starts (for example, because the plugin is named using a `--plugin-load-add` option or is registered in the `mysql.plugin` table), the server loads and enables the plugin by default. It is possible to control activation state for such a plugin using a `--plugin_name[=activation_state]` startup option, where *`plugin_name`* is the name of the plugin to affect, such as `innodb`, `csv`, or `validate_password`. As with other options, dashes and underscores are interchangeable in option names. Also, activation state values are not case-sensitive. For example, `--my_plugin=ON` and `--my-plugin=on` are equivalent.

- `--plugin_name=OFF`

  Tells the server to disable the plugin. This may not be possible for certain built-in plugins, such as `mysql_native_password`.

- `--plugin_name[=ON]`

  Tells the server to enable the plugin. (Specifying the option as `--plugin_name` without a value has the same effect.) If the plugin fails to initialize, the server runs with the plugin disabled.

- `--plugin_name=FORCE`

  Tells the server to enable the plugin, but if plugin initialization fails, the server does not start. In other words, this option forces the server to run with the plugin enabled or not at all.

- `--plugin_name=FORCE_PLUS_PERMANENT`

  Like `FORCE`, but in addition prevents the plugin from being unloaded at runtime. If a user attempts to do so with `UNINSTALL PLUGIN`, an error occurs.

Plugin activation states are visible in the `LOAD_OPTION` column of the `INFORMATION_SCHEMA.PLUGINS` table.

Suppose that `CSV`, `BLACKHOLE`, and `ARCHIVE` are built-in pluggable storage engines and that you want the server to load them at startup, subject to these conditions: The server is permitted to run if `CSV` initialization fails, must require that `BLACKHOLE` initialization succeeds, and should disable `ARCHIVE`. To accomplish that, use these lines in an option file:

```
[mysqld]
csv=ON
blackhole=FORCE
archive=OFF
```

The `--enable-plugin_name` option format is a synonym for `--plugin_name=ON`. The `--disable-plugin_name` and `--skip-plugin_name` option formats are synonyms for `--plugin_name=OFF`.

If a plugin is disabled, either explicitly with `OFF` or implicitly because it was enabled with `ON` but fails to initialize, aspects of server operation that require the plugin change. For example, if the plugin implements a storage engine, existing tables for the storage engine become inaccessible, and attempts to create new tables for the storage engine result in tables that use the default storage engine unless the `NO_ENGINE_SUBSTITUTION` SQL mode is enabled to cause an error to occur instead.

Disabling a plugin may require adjustment to other options. For example, if you start the server using `--skip-innodb` to disable `InnoDB`, other `innodb_xxx` options likely need to be omitted at startup. In addition, because `InnoDB` is the default storage engine, it cannot start unless you specify another available storage engine with `--default_storage_engine`. You must also set `--default_tmp_storage_engine`.

Uninstalling Plugins

At runtime, the `UNINSTALL PLUGIN` statement disables and uninstalls a plugin known to the server. The statement unloads the plugin and removes it from the `mysql.plugin` system table, if it is registered there. For this reason, `UNINSTALL PLUGIN` statement requires the `DELETE` privilege for the `mysql.plugin` table. With the plugin no longer registered in the table, the server does not load the plugin during subsequent restarts.

`UNINSTALL PLUGIN` can unload a plugin regardless of whether it was loaded at runtime with `INSTALL PLUGIN` or at startup with a plugin-loading option, subject to these conditions:

- It cannot unload plugins that are built in to the server. These can be identified as those that have a library name of `NULL` in the output from `INFORMATION_SCHEMA.PLUGINS` or `SHOW PLUGINS`.

- It cannot unload plugins for which the server was started with *--plugin_name*=FORCE_PLUS_PERMANENT, which prevents plugin unloading at runtime. These can be identified from the LOAD_OPTION column of the INFORMATION_SCHEMA.PLUGINS table.

To uninstall a plugin that currently is loaded at server startup with a plugin-loading option, use this procedure.

1. Remove from the my.cnf file any options related to the plugin.
2. Restart the server.
3. Plugins normally are installed using either a plugin-loading option at startup or with INSTALL PLUGIN at runtime, but not both. However, removing options for a plugin from the my.cnf file may not be sufficient to uninstall it if at some point INSTALL PLUGIN has also been used. If the plugin still appears in the output from INFORMATION_SCHEMA.PLUGINS or SHOW PLUGINS, use UNINSTALL PLUGIN to remove it from the mysql.plugin table. Then restart the server again.

5.5.2 Obtaining Server Plugin Information

There are several ways to determine which plugins are installed in the server:

- The INFORMATION_SCHEMA.PLUGINS table contains a row for each loaded plugin. Any that have a PLUGIN_LIBRARY value of NULL are built in and cannot be unloaded.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.PLUGINS\G
*************************** 1. row ***************************
           PLUGIN_NAME: binlog
        PLUGIN_VERSION: 1.0
         PLUGIN_STATUS: ACTIVE
           PLUGIN_TYPE: STORAGE ENGINE
   PLUGIN_TYPE_VERSION: 50158.0
        PLUGIN_LIBRARY: NULL
PLUGIN_LIBRARY_VERSION: NULL
         PLUGIN_AUTHOR: MySQL AB
    PLUGIN_DESCRIPTION: This is a pseudo storage engine to represent the binlog in a transaction
        PLUGIN_LICENSE: GPL
           LOAD_OPTION: FORCE
...
*************************** 10. row ***************************
           PLUGIN_NAME: InnoDB
        PLUGIN_VERSION: 1.0
         PLUGIN_STATUS: ACTIVE
           PLUGIN_TYPE: STORAGE ENGINE
   PLUGIN_TYPE_VERSION: 50158.0
        PLUGIN_LIBRARY: ha_innodb_plugin.so
PLUGIN_LIBRARY_VERSION: 1.0
         PLUGIN_AUTHOR: Innobase Oy
    PLUGIN_DESCRIPTION: Supports transactions, row-level locking,
                        and foreign keys
        PLUGIN_LICENSE: GPL
           LOAD_OPTION: ON
...
```

- The SHOW PLUGINS statement displays a row for each loaded plugin. Any that have a Library value of NULL are built in and cannot be unloaded.

```
mysql> SHOW PLUGINS\G
*************************** 1. row ***************************
   Name: binlog
 Status: ACTIVE
   Type: STORAGE ENGINE
Library: NULL
License: GPL
...
*************************** 10. row ***************************
   Name: InnoDB
 Status: ACTIVE
   Type: STORAGE ENGINE
```

```
Library: ha_innodb_plugin.so
License: GPL
...
```

- The `mysql.plugin` table shows which plugins have been registered with `INSTALL PLUGIN`. The table contains only plugin names and library file names, so it does not provide as much information as the `PLUGINS` table or the `SHOW PLUGINS` statement.

5.5.3 MySQL Enterprise Thread Pool

Note

MySQL Enterprise Thread Pool is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, https://www.mysql.com/products/.

MySQL Enterprise Edition includes MySQL Enterprise Thread Pool, implemented using a server plugin. The default thread-handling model in MySQL Server executes statements using one thread per client connection. As more clients connect to the server and execute statements, overall performance degrades. The thread pool plugin provides an alternative thread-handling model designed to reduce overhead and improve performance. The plugin implements a thread pool that increases server performance by efficiently managing statement execution threads for large numbers of client connections.

The thread pool addresses several problems of the model that uses one thread per connection:

- Too many thread stacks make CPU caches almost useless in highly parallel execution workloads. The thread pool promotes thread stack reuse to minimize the CPU cache footprint.
- With too many threads executing in parallel, context switching overhead is high. This also presents a challenge to the operating system scheduler. The thread pool controls the number of active threads to keep the parallelism within the MySQL server at a level that it can handle and that is appropriate for the server host on which MySQL is executing.
- Too many transactions executing in parallel increases resource contention. In `InnoDB`, this increases the time spent holding central mutexes. The thread pool controls when transactions start to ensure that not too many execute in parallel.

Additional Resources

Section A.15, “MySQL 5.6 FAQ: MySQL Enterprise Thread Pool”

5.5.3.1 Thread Pool Elements

MySQL Enterprise Thread Pool comprises these elements:

- A plugin library file implements a plugin for the thread pool code as well as several associated monitoring tables that provide information about thread pool operation.

  For a detailed description of how the thread pool works, see Section 5.5.3.3, “Thread Pool Operation”.

  The `INFORMATION_SCHEMA` tables are named `TP_THREAD_STATE`, `TP_THREAD_GROUP_STATE`, and `TP_THREAD_GROUP_STATS`. These tables provide information about thread pool operation. For more information, see Section 21.32, “INFORMATION_SCHEMA Thread Pool Tables”.

- Several system variables are related to the thread pool. The `thread_handling` system variable has a value of `loaded-dynamically` when the server successfully loads the thread pool plugin.

  The other related system variables are implemented by the thread pool plugin and are not available unless it is enabled. For information about using these variables, see Section 5.5.3.3, “Thread Pool Operation”, and Section 5.5.3.4, “Thread Pool Tuning”.

- The Performance Schema has instruments that expose information about the thread pool and may be used to investigate operational performance. To identify them, use this query:

```
SELECT * FROM performance_schema.setup_instruments
WHERE NAME LIKE '%thread_pool%';
```

  For more information, see Chapter 22, *MySQL Performance Schema*.

5.5.3.2 Thread Pool Installation

This section describes how to install MySQL Enterprise Thread Pool. For general information about installing plugins, see Section 5.5.1, “Installing and Uninstalling Plugins”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base name is `thread_pool`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To enable thread pool capability, load the plugins to be used by starting the server with the `--plugin-load-add` option. For example, if you name only the plugin library file, the server loads all plugins that it contains (that is, the thread pool plugin and all the `INFORMATION_SCHEMA` tables). To do this, put these lines in the server `my.cnf` file, adjusting the `.so` suffix for your platform as necessary:

```
[mysqld]
plugin-load-add=thread_pool.so
```

That is equivalent to loading all thread pool plugins by naming them individually:

```
[mysqld]
plugin-load-add=thread_pool=thread_pool.so
plugin-load-add=tp_thread_state=thread_pool.so
plugin-load-add=tp_thread_group_state=thread_pool.so
plugin-load-add=tp_thread_group_stats=thread_pool.so
```

If desired, you can load individual plugins from the library file. To load the thread pool plugin but not the `INFORMATION_SCHEMA` tables, use an option like this:

```
[mysqld]
plugin-load-add=thread_pool=thread_pool.so
```

To load the thread pool plugin and only the `TP_THREAD_STATE` `INFORMATION_SCHEMA` table, use options like this:

```
[mysqld]
plugin-load-add=thread_pool=thread_pool.so
plugin-load-add=tp_thread_state=thread_pool.so
```

Note

If you do not load all the `INFORMATION_SCHEMA` tables, some or all MySQL Enterprise Monitor thread pool graphs are empty.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.5.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE 'thread%' OR PLUGIN_NAME LIKE 'tp%';
+----------------------+---------------+
| PLUGIN_NAME          | PLUGIN_STATUS |
+----------------------+---------------+
| thread_pool          | ACTIVE        |
```

```
| TP_THREAD_STATE       | ACTIVE        |
| TP_THREAD_GROUP_STATE | ACTIVE        |
| TP_THREAD_GROUP_STATS | ACTIVE        |
+-----------------------+---------------+
```

If the server loads the thread pool plugin successfully, it sets the `thread_handling` system variable to `loaded-dynamically`.

If a plugin fails to initialize, check the server error log for diagnostic messages.

5.5.3.3 Thread Pool Operation

The thread pool consists of a number of thread groups, each of which manages a set of client connections. As connections are established, the thread pool assigns them to thread groups in round-robin fashion.

The thread pool exposes system variables that may be used to configure its operation:

- `thread_pool_algorithm`: The concurrency algorithm to use for scheduling.
- `thread_pool_high_priority_connection`: How to schedule statement execution for a session.
- `thread_pool_max_unused_threads`: How many sleeping threads to permit.
- `thread_pool_prio_kickup_timer`: How long before the thread pool moves a statement awaiting execution from the low-priority queue to the high-priority queue.
- `thread_pool_size`: The number of thread groups in the thread pool. This is the most important parameter controlling thread pool performance.
- `thread_pool_stall_limit`: The time before an executing statement is considered to be stalled.

To configure the number of thread groups, use the `thread_pool_size` system variable. The default number of groups is 16. For guidelines on setting this variable, see Section 5.5.3.4, “Thread Pool Tuning”.

The maximum number of threads per group is 4096 (or 4095 on some systems where one thread is used internally).

The thread pool separates connections and threads, so there is no fixed relationship between connections and the threads that execute statements received from those connections. This differs from the default thread-handling model that associates one thread with one connection such that a given thread executes all statements from its connection.

The thread pool tries to ensure a maximum of one thread executing in each group at any time, but sometimes permits more threads to execute temporarily for best performance:

- Each thread group has a listener thread that listens for incoming statements from the connections assigned to the group. When a statement arrives, the thread group either begins executing it immediately or queues it for later execution:
  - Immediate execution occurs if the statement is the only one received and no statements are queued or currently executing.
  - Queuing occurs if the statement cannot begin executing immediately.
- If immediate execution occurs, the listener thread performs it. (This means that temporarily no thread in the group is listening.) If the statement finishes quickly, the executing thread returns to listening for statements. Otherwise, the thread pool considers the statement stalled and starts another thread as a listener thread (creating it if necessary). To ensure that no thread group becomes blocked by

stalled statements, the thread pool has a background thread that regularly monitors thread group states.

By using the listening thread to execute a statement that can begin immediately, there is no need to create an additional thread if the statement finishes quickly. This ensures the most efficient execution possible in the case of a low number of concurrent threads.

When the thread pool plugin starts, it creates one thread per group (the listener thread), plus the background thread. Additional threads are created as necessary to execute statements.

- The value of the `thread_pool_stall_limit` system variable determines the meaning of “finishes quickly” in the previous item. The default time before threads are considered stalled is 60ms but can be set to a maximum of 6s. This parameter is configurable to enable you to strike a balance appropriate for the server work load. Short wait values permit threads to start more quickly. Short values are also better for avoiding deadlock situations. Long wait values are useful for workloads that include long-running statements, to avoid starting too many new statements while the current ones execute.
- The thread pool focuses on limiting the number of concurrent short-running statements. Before an executing statement reaches the stall time, it prevents other statements from beginning to execute. If the statement executes past the stall time, it is permitted to continue but no longer prevents other statements from starting. In this way, the thread pool tries to ensure that in each thread group there is never more than one short-running statement, although there might be multiple long-running statements. It is undesirable to let long-running statements prevent other statements from executing because there is no limit on the amount of waiting that might be necessary. For example, on a replication source, a thread that is sending binary log events to a replica effectively runs forever.
- A statement becomes blocked if it encounters a disk I/O operation or a user level lock (row lock or table lock). The block would cause the thread group to become unused, so there are callbacks to the thread pool to ensure that the thread pool can immediately start a new thread in this group to execute another statement. When a blocked thread returns, the thread pool permits it to restart immediately.
- There are two queues, a high-priority queue and a low-priority queue. The first statement in a transaction goes to the low-priority queue. Any following statements for the transaction go to the high-priority queue if the transaction is ongoing (statements for it have begun executing), or to the low-priority queue otherwise. Queue assignment can be affected by enabling the `thread_pool_high_priority_connection` system variable, which causes all queued statements for a session to go into the high-priority queue.

  Statements for a nontransactional storage engine, or a transactional engine if `autocommit` is enabled, are treated as low-priority statements because in this case each statement is a transaction. Thus, given a mix of statements for `InnoDB` and `MyISAM` tables, the thread pool prioritizes those for `InnoDB` over those for `MyISAM` unless `autocommit` is enabled. With `autocommit` enabled, all statements are low priority.
- When the thread group selects a queued statement for execution, it first looks in the high-priority queue, then in the low-priority queue. If a statement is found, it is removed from its queue and begins to execute.
- If a statement stays in the low-priority queue too long, the thread pool moves to the high-priority queue. The value of the `thread_pool_prio_kickup_timer` system variable controls the time before movement. For each thread group, a maximum of one statement per 10ms (100 per second) is moved from the low-priority queue to the high-priority queue.
- The thread pool reuses the most active threads to obtain a much better use of CPU caches. This is a small adjustment that has a great impact on performance.
- While a thread executes a statement from a user connection, Performance Schema instrumentation accounts thread activity to the user connection. Otherwise, Performance Schema accounts activity to the thread pool.

Here are examples of conditions under which a thread group might have multiple threads started to execute statements:

- One thread begins executing a statement, but runs long enough to be considered stalled. The thread group permits another thread to begin executing another statement even through the first thread is still executing.
- One thread begins executing a statement, then becomes blocked and reports this back to the thread pool. The thread group permits another thread to begin executing another statement.
- One thread begins executing a statement, becomes blocked, but does not report back that it is blocked because the block does not occur in code that has been instrumented with thread pool callbacks. In this case, the thread appears to the thread group to be still running. If the block lasts long enough for the statement to be considered stalled, the group permits another thread to begin executing another statement.

The thread pool is designed to be scalable across an increasing number of connections. It is also designed to avoid deadlocks that can arise from limiting the number of actively executing statements. It is important that threads that do not report back to the thread pool do not prevent other statements from executing and thus cause the thread pool to become deadlocked. Examples of such statements follow:

- Long-running statements. These would lead to all resources used by only a few statements and they could prevent all others from accessing the server.
- Binary log dump threads that read the binary log and send it to replicas. This is a kind of long-running “statement” that runs for a very long time, and that should not prevent other statements from executing.
- Statements blocked on a row lock, table lock, sleep, or any other blocking activity that has not been reported back to the thread pool by MySQL Server or a storage engine.

In each case, to prevent deadlock, the statement is moved to the stalled category when it does not complete quickly, so that the thread group can permit another statement to begin executing. With this design, when a thread executes or becomes blocked for an extended time, the thread pool moves the thread to the stalled category and for the rest of the statement's execution, it does not prevent other statements from executing.

The maximum number of threads that can occur is the sum of `max_connections` and `thread_pool_size`. This can happen in a situation where all connections are in execution mode and an extra thread is created per group to listen for more statements. This is not necessarily a state that happens often, but it is theoretically possible.

5.5.3.4 Thread Pool Tuning

This section provides guidelines on setting thread pool system variables for best performance, measured using a metric such as transactions per second.

`thread_pool_size` is the most important parameter controlling thread pool performance. It can be set only at server startup. Our experience in testing the thread pool indicates the following:

- If the primary storage engine is `InnoDB`, the optimal `thread_pool_size` setting is likely to be between 16 and 36, with the most common optimal values tending to be from 24 to 36. We have not seen any situation where the setting has been optimal beyond 36. There may be special cases where a value smaller than 16 is optimal.

  For workloads such as DBT2 and Sysbench, the optimum for `InnoDB` seems to be usually around 36. For very write-intensive workloads, the optimal setting can sometimes be lower.
- If the primary storage engine is `MyISAM`, the `thread_pool_size` setting should be fairly low. Optimal performance is often seen with values from 4 to 8. Higher values tend to have a slightly negative but not dramatic impact on performance.

Another system variable, `thread_pool_stall_limit`, is important for handling of blocked and long-running statements. If all calls that block the MySQL Server are reported to the thread pool, it would always know when execution threads are blocked. However, this may not always be true. For example, blocks could occur in code that has not been instrumented with thread pool callbacks. For such cases, the thread pool must be able to identify threads that appear to be blocked. This is done by means of a timeout that can be tuned using the `thread_pool_stall_limit` system variable, the value of which is measured in 10ms units. This parameter ensures that the server does not become completely blocked. The value of `thread_pool_stall_limit` has an upper limit of 6 seconds to prevent the risk of a deadlocked server.

`thread_pool_stall_limit` also enables the thread pool to handle long-running statements. If a long-running statement was permitted to block a thread group, all other connections assigned to the group would be blocked and unable to start execution until the long-running statement completed. In the worst case, this could take hours or even days.

The value of `thread_pool_stall_limit` should be chosen such that statements that execute longer than its value are considered stalled. Stalled statements generate a lot of extra overhead since they involve extra context switches and in some cases even extra thread creations. On the other hand, setting the `thread_pool_stall_limit` parameter too high means that long-running statements block a number of short-running statements for longer than necessary. Short wait values permit threads to start more quickly. Short values are also better for avoiding deadlock situations. Long wait values are useful for workloads that include long-running statements, to avoid starting too many new statements while the current ones execute.

Suppose a server executes a workload where 99.9% of the statements complete within 100ms even when the server is loaded, and the remaining statements take between 100ms and 2 hours fairly evenly spread. In this case, it would make sense to set `thread_pool_stall_limit` to 10 (10 × 10ms = 100ms). The default value of 6 (60ms) is suitable for servers that primarily execute very simple statements.

The `thread_pool_stall_limit` parameter can be changed at runtime to enable you to strike a balance appropriate for the server work load. Assuming that the `TP_THREAD_GROUP_STATS` table is enabled, you can use the following query to determine the fraction of executed statements that stalled:

```
SELECT SUM(STALLED_QUERIES_EXECUTED) / SUM(QUERIES_EXECUTED)
FROM INFORMATION_SCHEMA.TP_THREAD_GROUP_STATS;
```

This number should be as low as possible. To decrease the likelihood of statements stalling, increase the value of `thread_pool_stall_limit`.

When a statement arrives, what is the maximum time it can be delayed before it actually starts executing? Suppose that the following conditions apply:

- There are 200 statements queued in the low-priority queue.
- There are 10 statements queued in the high-priority queue.
- `thread_pool_prio_kickup_timer` is set to 10000 (10 seconds).
- `thread_pool_stall_limit` is set to 100 (1 second).

In the worst case, the 10 high-priority statements represent 10 transactions that continue executing for a long time. Thus, in the worst case, no statements are moved to the high-priority queue because it already contains statements awaiting execution. After 10 seconds, the new statement is eligible to be moved to the high-priority queue. However, before it can be moved, all the statements before it must be moved as well. This could take another 2 seconds because a maximum of 100 statements per second are moved to the high-priority queue. Now when the statement reaches the high-priority queue, there could potentially be many long-running statements ahead of it. In the worst case, every one of those becomes stalled and it takes 1 second for each statement before the next statement is retrieved from the high-priority queue. Thus, in this scenario, it takes 222 seconds before the new statement starts executing.

This example shows a worst case for an application. How to handle it depends on the application. If the application has high requirements for the response time, it should most likely throttle users at a higher level itself. Otherwise, it can use the thread pool configuration parameters to set some kind of a maximum waiting time.

5.5.4 MySQL Plugin Services

MySQL server plugins have access to server “plugin services.” The plugin services interface complements the plugin API by exposing server functionality that plugins can call. For developer information about writing plugin services, see MySQL Services for Plugins.

5.6 MySQL Server User-Defined Functions

MySQL Server enables user-defined functions (UDFs) to be created and loaded to extend server capabilities. Server capabilities can be implemented in whole or in part using UDFs. In addition, you can write your own UDFs.

MySQL distributions include UDFs that implement, or help to implement, several server capabilities:

- MySQL Enterprise Edition includes UDFs that perform encryption operations based on the OpenSSL library. See Section 6.5, “MySQL Enterprise Encryption”.
- MySQL Enterprise Edition includes a firewall capability that implements an application-level firewall to enable database administrators to permit or deny SQL statement execution based on matching against allowlists of accepted statement patterns. See Section 6.4.5, “MySQL Enterprise Firewall”.

The following sections describe how to install and uninstall UDFs, and how to determine at runtime which UDFs are installed and obtain information about them. For a table listing user-defined functions, see Section 12.2, “User-Defined Function Reference”. For information about writing UDFs, see Adding Functions to MySQL.

5.6.1 Installing and Uninstalling User-Defined Functions

User-defined functions (UDFs) must be loaded into the server before they can be used. MySQL supports manual UDF loading at runtime and automatic loading during server startup.

While a UDF is loaded, information about it is available as described in Section 5.6.2, “Obtaining User-Defined Function Information”.

- Installing User-Defined Functions
- Uninstalling User-Defined Functions
- Reinstalling or Upgrading User-Defined Functions

Installing User-Defined Functions

To load a UDF manually, use the `CREATE FUNCTION` statement. For example:

```
CREATE FUNCTION metaphon
  RETURNS STRING
  SONAME 'udf_example.so';
```

The UDF file base name depends on your platform. Common suffixes are `.so` for Unix and Unix-like systems, `.dll` for Windows.

`CREATE FUNCTION` has these effects:

- It loads the UDF into the server to make it available immediately.
- It registers the UDF in the `mysql.func` system table to make it persistent across server restarts. For this reason, `CREATE FUNCTION` requires the `INSERT` privilege for the `mysql` system database.

Automatic UDF loading occurs during the normal server startup sequence. The server loads UDFs registered in the `mysql.func` table. If the server is started with the `--skip-grant-tables` option, UDFs registered in the table are not loaded and are unavailable.

Uninstalling User-Defined Functions

To remove a UDF, use the `DROP FUNCTION` statement. For example:

```
DROP FUNCTION metaphon;
```

`DROP FUNCTION` has these effects:

- It unloads the UDF to make it unavailable.
- It removes the UDF from the `mysql.func` system table. For this reason, `DROP FUNCTION` requires the `DELETE` privilege for the `mysql` system database. With the UDF no longer registered in the `mysql.func` table, the server does not load the UDF during subsequent restarts.

While a UDF is loaded, information about it is available from the `mysql.func` system table. See Section 5.6.2, “Obtaining User-Defined Function Information”. `CREATE FUNCTION` adds the UDF to the table and `DROP FUNCTION` removes it.

Reinstalling or Upgrading User-Defined Functions

To reinstall or upgrade the shared library associated with a UDF, issue a `DROP FUNCTION` statement, upgrade the shared library, and then issue a `CREATE FUNCTION` statement. If you upgrade the shared library first and then use `DROP FUNCTION`, the server may unexpectedly shut down.

5.6.2 Obtaining User-Defined Function Information

The `mysql.func` system table shows which UDFs have been registered using `CREATE FUNCTION`:

```
SELECT * FROM mysql.func;
```

The `func` table has these columns:

- `name`

  The UDF name as referred to in SQL statements.

- `ret`

  The UDF return value type. Permitted values are 0 (`STRING`), 1 (`REAL`), 2 (`INTEGER`), 3 (`ROW`), or 4 (`DECIMAL`).

- `dl`

  The name of the UDF library file containing the executable UDF code. The file is located in the directory named by the `plugin_dir` system variable.

- `type`

  The UDF type, either `function` (scalar) or `aggregate`.

5.7 Running Multiple MySQL Instances on One Machine

In some cases, you might want to run multiple instances of MySQL on a single machine. You might want to test a new MySQL release while leaving an existing production setup undisturbed. Or you might want to give different users access to different `mysqld` servers that they manage themselves. (For example, you might be an Internet Service Provider that wants to provide independent MySQL installations for different customers.)

It is possible to use a different MySQL server binary per instance, or use the same binary for multiple instances, or any combination of the two approaches. For example, you might run a server from MySQL 5.5 and one from MySQL 5.6, to see how different versions handle a given workload. Or you might run multiple instances of the current production version, each managing a different set of databases.

Whether or not you use distinct server binaries, each instance that you run must be configured with unique values for several operating parameters. This eliminates the potential for conflict between instances. Parameters can be set on the command line, in option files, or by setting environment variables. See Section 4.2.2, “Specifying Program Options”. To see the values used by a given instance, connect to it and execute a `SHOW VARIABLES` statement.

The primary resource managed by a MySQL instance is the data directory. Each instance should use a different data directory, the location of which is specified using the `--datadir=dir_name` option. For methods of configuring each instance with its own data directory, and warnings about the dangers of failing to do so, see Section 5.7.1, “Setting Up Multiple Data Directories”.

In addition to using different data directories, several other options must have different values for each server instance:

- `--port=port_num`

  `--port` controls the port number for TCP/IP connections. Alternatively, if the host has multiple network addresses, you can set the `bind_address` system variable to cause each server to listen to a different address.

- `--socket={file_name|pipe_name}`

  `--socket` controls the Unix socket file path on Unix or the named-pipe name on Windows. On Windows, it is necessary to specify distinct pipe names only for those servers configured to permit named-pipe connections.

- `--shared-memory-base-name=name`

  This option is used only on Windows. It designates the shared-memory name used by a Windows server to permit clients to connect using shared memory. It is necessary to specify distinct shared-memory names only for those servers configured to permit shared-memory connections.

- `--pid-file=file_name`

  This option indicates the path name of the file in which the server writes its process ID.

If you use the following log file options, their values must differ for each server:

- `--general_log_file=file_name`
- `--log-bin[=file_name]`
- `--slow_query_log_file=file_name`
- `--log-error[=file_name]`

For further discussion of log file options, see Section 5.4, “MySQL Server Logs”.

To achieve better performance, you can specify the following option differently for each server, to spread the load between several physical disks:

- `--tmpdir=dir_name`

Having different temporary directories also makes it easier to determine which MySQL server created any given temporary file.

If you have multiple MySQL installations in different locations, you can specify the base directory for each installation with the `--basedir=dir_name` option. This causes each instance to automatically use a different data directory, log files, and PID file because the default for each of those parameters is relative to the base directory. In that case, the only other options you need to specify are the `--socket` and `--port` options. Suppose that you install different versions of MySQL using `tar` file binary distributions. These install in different locations, so you can start the server for each installation using the command `bin/mysqld_safe` under its corresponding base directory. `mysqld_safe` determines the proper `--basedir` option to pass to `mysqld`, and you need specify only the `--socket` and `--port` options to `mysqld_safe`.

As discussed in the following sections, it is possible to start additional servers by specifying appropriate command options or by setting environment variables. However, if you need to run multiple servers on a more permanent basis, it is more convenient to use option files to specify for each server those option values that must be unique to it. The `--defaults-file` option is useful for this purpose.

5.7.1 Setting Up Multiple Data Directories

Each MySQL Instance on a machine should have its own data directory. The location is specified using the `--datadir=dir_name` option.

There are different methods of setting up a data directory for a new instance:

- Create a new data directory.
- Copy an existing data directory.

The following discussion provides more detail about each method.

Warning

Normally, you should never have two servers that update data in the same databases. This may lead to unpleasant surprises if your operating system does not support fault-free system locking. If (despite this warning) you run multiple servers using the same data directory and they have logging enabled, you must use the appropriate options to specify log file names that are unique to each server. Otherwise, the servers try to log to the same files.

Even when the preceding precautions are observed, this kind of setup works only with `MyISAM` and `MERGE` tables, and not with any of the other storage engines. Also, this warning against sharing a data directory among servers always applies in an NFS environment. Permitting multiple MySQL servers to access a common data directory over NFS is a *very bad idea*. The primary problem is that NFS is the speed bottleneck. It is not meant for such use. Another risk with NFS is that you must devise a way to ensure that two or more servers do not interfere with each other. Usually NFS file locking is handled by the `lockd` daemon, but at the moment there is no platform that performs locking 100% reliably in every situation.

Create a New Data Directory

With this method, the data directory is in the same state as when you first install MySQL. It has the default set of MySQL accounts and no user data.

On Unix, initialize the data directory. See Section 2.10, “Postinstallation Setup and Testing”.

On Windows, the data directory is included in the MySQL distribution:

- MySQL Zip archive distributions for Windows contain an unmodified data directory. You can unpack such a distribution into a temporary location, then copy it `data` directory to where you are setting up the new instance.

- Windows MSI package installers create and set up the data directory that the installed server uses, but also create a pristine "template" data directory named `data` under the installation directory. After an installation has been performed using an MSI package, the template data directory can be copied to set up additional MySQL instances.

Copy an Existing Data Directory

With this method, any MySQL accounts or user data present in the data directory are carried over to the new data directory.

1. Stop the existing MySQL instance using the data directory. This must be a clean shutdown so that the instance flushes any pending changes to disk.
2. Copy the data directory to the location where the new data directory should be.
3. Copy the `my.cnf` or `my.ini` option file used by the existing instance. This serves as a basis for the new instance.
4. Modify the new option file so that any pathnames referring to the original data directory refer to the new data directory. Also, modify any other options that must be unique per instance, such as the TCP/IP port number and the log files. For a list of parameters that must be unique per instance, see Section 5.7, "Running Multiple MySQL Instances on One Machine".
5. Start the new instance, telling it to use the new option file.

5.7.2 Running Multiple MySQL Instances on Windows

You can run multiple servers on Windows by starting them manually from the command line, each with appropriate operating parameters, or by installing several servers as Windows services and running them that way. General instructions for running MySQL from the command line or as a service are given in Section 2.3, "Installing MySQL on Microsoft Windows". The following sections describe how to start each server with different values for those options that must be unique per server, such as the data directory. These options are listed in Section 5.7, "Running Multiple MySQL Instances on One Machine".

5.7.2.1 Starting Multiple MySQL Instances at the Windows Command Line

The procedure for starting a single MySQL server manually from the command line is described in Section 2.3.4.5, "Starting MySQL from the Windows Command Line". To start multiple servers this way, you can specify the appropriate options on the command line or in an option file. It is more convenient to place the options in an option file, but it is necessary to make sure that each server gets its own set of options. To do this, create an option file for each server and tell the server the file name with a `--defaults-file` option when you run it.

Suppose that you want to run one instance of `mysqld` on port 3307 with a data directory of `C:\mydata1`, and another instance on port 3308 with a data directory of `C:\mydata2`. Use this procedure:

1. Make sure that each data directory exists, including its own copy of the `mysql` database that contains the grant tables.
2. Create two option files. For example, create one file named `C:\my-opts1.cnf` that looks like this:

```
[mysqld]
datadir = C:/mydata1
port = 3307
```

   Create a second file named `C:\my-opts2.cnf` that looks like this:

```
[mysqld]
datadir = C:/mydata2
```

```
port = 3308
```

3. Use the `--defaults-file` option to start each server with its own option file:

```
C:\> C:\mysql\bin\mysqld --defaults-file=C:\my-opts1.cnf
C:\> C:\mysql\bin\mysqld --defaults-file=C:\my-opts2.cnf
```

   Each server starts in the foreground (no new prompt appears until the server exits later), so you must issue those two commands in separate console windows.

To shut down the servers, connect to each using the appropriate port number:

```
C:\> C:\mysql\bin\mysqladmin --port=3307 --host=127.0.0.1 --user=root --password shutdown
C:\> C:\mysql\bin\mysqladmin --port=3308 --host=127.0.0.1 --user=root --password shutdown
```

Servers configured as just described permit clients to connect over TCP/IP. If your version of Windows supports named pipes and you also want to permit named-pipe connections, specify options that enable the named pipe and specify its name. Each server that supports named-pipe connections must use a unique pipe name. For example, the `C:\my-opts1.cnf` file might be written like this:

```
[mysqld]
datadir = C:/mydata1
port = 3307
enable-named-pipe
socket = mypipe1
```

Modify `C:\my-opts2.cnf` similarly for use by the second server. Then start the servers as described previously.

A similar procedure applies for servers that you want to permit shared-memory connections. Enable such connections by starting the server with the `shared_memory` system variable enabled and specify a unique shared-memory name for each server by setting the `shared_memory_base_name` system variable.

5.7.2.2 Starting Multiple MySQL Instances as Windows Services

On Windows, a MySQL server can run as a Windows service. The procedures for installing, controlling, and removing a single MySQL service are described in Section 2.3.4.7, “Starting MySQL as a Windows Service”.

To set up multiple MySQL services, you must make sure that each instance uses a different service name in addition to the other parameters that must be unique per instance.

For the following instructions, suppose that you want to run the `mysqld` server from two different versions of MySQL that are installed at `C:\mysql-5.5.9` and `C:\mysql-5.6.51`, respectively. (This might be the case if you are running 5.5.9 as your production server, but also want to conduct tests using 5.6.51.)

To install MySQL as a Windows service, use the `--install` or `--install-manual` option. For information about these options, see Section 2.3.4.7, “Starting MySQL as a Windows Service”.

Based on the preceding information, you have several ways to set up multiple services. The following instructions describe some examples. Before trying any of them, shut down and remove any existing MySQL services.

- **Approach 1:** Specify the options for all services in one of the standard option files. To do this, use a different service name for each server. Suppose that you want to run the 5.5.9 `mysqld` using the service name of `mysqld1` and the 5.6.51 `mysqld` using the service name `mysqld2`. In this case, you can use the `[mysqld1]` group for 5.5.9 and the `[mysqld2]` group for 5.6.51. For example, you can set up `C:\my.cnf` like this:

```
# options for mysqld1 service
```

```
[mysqld1]
basedir = C:/mysql-5.5.9
port = 3307
enable-named-pipe
socket = mypipe1

# options for mysqld2 service
[mysqld2]
basedir = C:/mysql-5.6.51
port = 3308
enable-named-pipe
socket = mypipe2
```

Install the services as follows, using the full server path names to ensure that Windows registers the correct executable program for each service:

```
C:\> C:\mysql-5.5.9\bin\mysqld --install mysqld1
C:\> C:\mysql-5.6.51\bin\mysqld --install mysqld2
```

To start the services, use the services manager, or `NET START` or `SC START` with the appropriate service names:

```
C:\> SC START mysqld1
C:\> SC START mysqld2
```

To stop the services, use the services manager, or use `NET STOP` or `SC STOP` with the appropriate service names:

```
C:\> SC STOP mysqld1
C:\> SC STOP mysqld2
```

- **Approach 2:** Specify options for each server in separate files and use `--defaults-file` when you install the services to tell each server what file to use. In this case, each file should list options using a `[mysqld]` group.

  With this approach, to specify options for the 5.5.9 `mysqld`, create a file `C:\my-opts1.cnf` that looks like this:

  ```
  [mysqld]
  basedir = C:/mysql-5.5.9
  port = 3307
  enable-named-pipe
  socket = mypipe1
  ```

  For the 5.6.51 `mysqld`, create a file `C:\my-opts2.cnf` that looks like this:

  ```
  [mysqld]
  basedir = C:/mysql-5.6.51
  port = 3308
  enable-named-pipe
  socket = mypipe2
  ```

  Install the services as follows (enter each command on a single line):

  ```
  C:\> C:\mysql-5.5.9\bin\mysqld --install mysqld1
            --defaults-file=C:\my-opts1.cnf
  C:\> C:\mysql-5.6.51\bin\mysqld --install mysqld2
            --defaults-file=C:\my-opts2.cnf
  ```

  When you install a MySQL server as a service and use a `--defaults-file` option, the service name must precede the option.

  After installing the services, start and stop them the same way as in the preceding example.

To remove multiple services, use `SC DELETE` *`mysqld_service_name`* for each one. Alternatively, use `mysqld --remove` for each one, specifying a service name following the `--remove` option. If the service name is the default (`MySQL`), you can omit it when using `mysqld --remove`.

5.7.3 Running Multiple MySQL Instances on Unix

One way is to run multiple MySQL instances on Unix is to compile different servers with different default TCP/IP ports and Unix socket files so that each one listens on different network interfaces. Compiling in different base directories for each installation also results automatically in a separate, compiled-in data directory, log file, and PID file location for each server.

Assume that an existing 5.5 server is configured for the default TCP/IP port number (3306) and Unix socket file (`/tmp/mysql.sock`). To configure a new 5.6.51 server to have different operating parameters, use a `CMake` command something like this:

```
shell> cmake . -DMYSQL_TCP_PORT=port_number \
             -DMYSQL_UNIX_ADDR=file_name \
             -DCMAKE_INSTALL_PREFIX=/usr/local/mysql-5.6.51
```

Here, *`port_number`* and *`file_name`* must be different from the default TCP/IP port number and Unix socket file path name, and the `CMAKE_INSTALL_PREFIX` value should specify an installation directory different from the one under which the existing MySQL installation is located.

If you have a MySQL server listening on a given port number, you can use the following command to find out what operating parameters it is using for several important configurable variables, including the base directory and Unix socket file name:

```
shell> mysqladmin --host=host_name --port=port_number variables
```

With the information displayed by that command, you can tell what option values *not* to use when configuring an additional server.

If you specify `localhost` as the host name, `mysqladmin` defaults to using a Unix socket file rather than TCP/IP. To explicitly specify the transport protocol, use the `--protocol={TCP|SOCKET|PIPE|MEMORY}` option.

You need not compile a new MySQL server just to start with a different Unix socket file and TCP/IP port number. It is also possible to use the same server binary and start each invocation of it with different parameter values at runtime. One way to do so is by using command-line options:

```
shell> mysqld_safe --socket=file_name --port=port_number
```

To start a second server, provide different `--socket` and `--port` option values, and pass a `--datadir=dir_name` option to `mysqld_safe` so that the server uses a different data directory.

Alternatively, put the options for each server in a different option file, then start each server using a `--defaults-file` option that specifies the path to the appropriate option file. For example, if the option files for two server instances are named `/usr/local/mysql/my.cnf` and `/usr/local/mysql/my.cnf2`, start the servers like this: command:

```
shell> mysqld_safe --defaults-file=/usr/local/mysql/my.cnf
shell> mysqld_safe --defaults-file=/usr/local/mysql/my.cnf2
```

Another way to achieve a similar effect is to use environment variables to set the Unix socket file name and TCP/IP port number:

```
shell> MYSQL_UNIX_PORT=/tmp/mysqld-new.sock
shell> MYSQL_TCP_PORT=3307
shell> export MYSQL_UNIX_PORT MYSQL_TCP_PORT
shell> mysql_install_db --user=mysql
shell> mysqld_safe --datadir=/path/to/datadir &
```

This is a quick way of starting a second server to use for testing. The nice thing about this method is that the environment variable settings apply to any client programs that you invoke from the same shell. Thus, connections for those clients are automatically directed to the second server.

Section 4.9, “Environment Variables”, includes a list of other environment variables you can use to affect MySQL programs.

On Unix, the `mysqld_multi` script provides another way to start multiple servers. See Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”.

5.7.4 Using Client Programs in a Multiple-Server Environment

To connect with a client program to a MySQL server that is listening to different network interfaces from those compiled into your client, you can use one of the following methods:

- Start the client with `--host=host_name --port=port_number` to connect using TCP/IP to a remote server, with `--host=127.0.0.1 --port=port_number` to connect using TCP/IP to a local server, or with `--host=localhost --socket=file_name` to connect to a local server using a Unix socket file or a Windows named pipe.
- Start the client with `--protocol=TCP` to connect using TCP/IP, `--protocol=SOCKET` to connect using a Unix socket file, `--protocol=PIPE` to connect using a named pipe, or `--protocol=MEMORY` to connect using shared memory. For TCP/IP connections, you may also need to specify `--host` and `--port` options. For the other types of connections, you may need to specify a `--socket` option to specify a Unix socket file or Windows named-pipe name, or a `--shared-memory-base-name` option to specify the shared-memory name. Shared-memory connections are supported only on Windows.
- On Unix, set the `MYSQL_UNIX_PORT` and `MYSQL_TCP_PORT` environment variables to point to the Unix socket file and TCP/IP port number before you start your clients. If you normally use a specific socket file or port number, you can place commands to set these environment variables in your `.login` file so that they apply each time you log in. See Section 4.9, “Environment Variables”.
- Specify the default Unix socket file and TCP/IP port number in the `[client]` group of an option file. For example, you can use `C:\my.cnf` on Windows, or the `.my.cnf` file in your home directory on Unix. See Section 4.2.2.2, “Using Option Files”.
- In a C program, you can specify the socket file or port number arguments in the `mysql_real_connect()` call. You can also have the program read option files by calling `mysql_options()`. See C API Function Descriptions.
- If you are using the Perl `DBD::mysql` module, you can read options from MySQL option files. For example:

```
$dsn = "DBI:mysql:test;mysql_read_default_group=client;"
       . "mysql_read_default_file=/usr/local/mysql/data/my.cnf";
$dbh = DBI->connect($dsn, $user, $password);
```

See Section 23.9, “MySQL Perl API”.

Other programming interfaces may provide similar capabilities for reading option files.

5.8 Debugging MySQL

This section describes debugging techniques that assist efforts to track down problems in MySQL.

5.8.1 Debugging a MySQL Server

If you are using some functionality that is very new in MySQL, you can try to run `mysqld` with the `--skip-new` option (which disables all new, potentially unsafe functionality). See Section B.3.3.3, “What to Do If MySQL Keeps Crashing”.

If `mysqld` does not want to start, verify that you have no `my.cnf` files that interfere with your setup! You can check your `my.cnf` arguments with `mysqld --print-defaults` and avoid using them by starting with `mysqld --no-defaults ...`.

If `mysqld` starts to eat up CPU or memory or if it “hangs,” you can use `mysqladmin processlist status` to find out if someone is executing a query that takes a long time. It may be a good idea to run

`mysqladmin -i10 processlist status` in some window if you are experiencing performance problems or problems when new clients cannot connect.

The command `mysqladmin debug` dumps some information about locks in use, used memory and query usage to the MySQL log file. This may help solve some problems. This command also provides some useful information even if you have not compiled MySQL for debugging!

If the problem is that some tables are getting slower and slower you should try to optimize the table with `OPTIMIZE TABLE` or `myisamchk`. See Chapter 5, *MySQL Server Administration*. You should also check the slow queries with `EXPLAIN`.

You should also read the OS-specific section in this manual for problems that may be unique to your environment. See Section 2.1, “General Installation Guidance”.

5.8.1.1 Compiling MySQL for Debugging

If you have some very specific problem, you can always try to debug MySQL. To do this you must configure MySQL with the `-DWITH_DEBUG=1` option. You can check whether MySQL was compiled with debugging by doing: `mysqld --help`. If the `--debug` flag is listed with the options then you have debugging enabled. `mysqladmin ver` also lists the `mysqld` version as `mysql ... --debug` in this case.

If `mysqld` stops crashing when you configure it with the `-DWITH_DEBUG=1` CMake option, you probably have found a compiler bug or a timing bug within MySQL. In this case, you can try to add `-g` using the `CMAKE_C_FLAGS` and `CMAKE_CXX_FLAGS` CMake options and not use `-DWITH_DEBUG=1`. If `mysqld` dies, you can at least attach to it with `gdb` or use `gdb` on the core file to find out what happened.

When you configure MySQL for debugging you automatically enable a lot of extra safety check functions that monitor the health of `mysqld`. If they find something “unexpected,” an entry is written to `stderr`, which `mysqld_safe` directs to the error log! This also means that if you are having some unexpected problems with MySQL and are using a source distribution, the first thing you should do is to configure MySQL for debugging. If you believe that you have found a bug, please use the instructions at Section 1.6, “How to Report Bugs or Problems”.

In the Windows MySQL distribution, `mysqld.exe` is by default compiled with support for trace files.

5.8.1.2 Creating Trace Files

If the `mysqld` server does not start or it crashes easily, you can try to create a trace file to find the problem.

To do this, you must have a `mysqld` that has been compiled with debugging support. You can check this by executing `mysqld -V`. If the version number ends with `-debug`, it is compiled with support for trace files. (On Windows, the debugging server is named `mysqld-debug` rather than `mysqld`.)

Start the `mysqld` server with a trace log in `/tmp/mysqld.trace` on Unix or `\mysqld.trace` on Windows:

```
shell> mysqld --debug
```

On Windows, you should also use the `--standalone` flag to not start `mysqld` as a service. In a console window, use this command:

```
C:\> mysqld-debug --debug --standalone
```

After this, you can use the `mysql.exe` command-line tool in a second console window to reproduce the problem. You can stop the `mysqld` server with `mysqladmin shutdown`.

The trace file can become **very large**! To generate a smaller trace file, you can use debugging options something like this:

```
mysqld --debug=d,info,error,query,general,where:O,/tmp/mysqld.trace
```

This only prints information with the most interesting tags to the trace file.

If you file a bug, please add only those lines from the trace file to the bug report that indicate where something seems to go wrong. If you cannot locate the wrong place, open a bug report and upload the whole trace file to the report, so that a MySQL developer can take a look at it. For instructions, see Section 1.6, “How to Report Bugs or Problems”.

The trace file is made with the **DBUG** package by Fred Fish. See Section 5.8.3, “The DBUG Package”.

5.8.1.3 Using WER with PDB to create a Windows crashdump

Program Database (PDB) files (with file name suffix `pdb`) are included in ZIP Archive distributions (but not MSI distributions) of MySQL. These files provide information for debugging your MySQL installation in the event of a problem.

The PDB file contains more detailed information about `mysqld` and other tools that enables more detailed trace and dump files to be created. You can use these with `WinDbg` or Visual Studio to debug `mysqld`.

For more information on PDB files, see Microsoft Knowledge Base Article 121366. For more information on the debugging options available, see Debugging Tools for Windows.

To use WinDbg, either install the full Windows Driver Kit (WDK) or install the standalone version.

Important

The `.exe` and `.pdb` files must be an exact match (both version number and MySQL server edition) or WinDBG complains while attempting to load the symbols.

1. To generate a minidump `mysqld.dmp`, enable the `core-file` option under the [mysqld] section in `my.ini`. Restart the MySQL server after making these changes.
2. Create a directory to store the generated files, such as `c:\symbols`
3. Determine the path to your `windbg.exe` executable using the Find GUI or from the command line, for example: `dir /s /b windbg.exe` -- a common default is *C:\Program Files\Debugging Tools for Windows (x64)\windbg.exe*
4. Launch `windbg.exe` giving it the paths to `mysqld-debug.exe`, `mysqld.pdb`, `mysqld.dmp`, and the source code. Alternatively, pass in each path from the WinDbg GUI. For example:

```
windbg.exe -i "C:\mysql-5.6.51-winx64\bin\"^
 -z "C:\mysql-5.6.51-winx64\data\mysqld.dmp"^
 -srcpath "E:\ade\mysql_archives\5.6\5.6.51\mysql-5.6.51"^
 -y "C:\mysql-5.6.51-winx64\bin;SRV*c:\symbols*http://msdl.microsoft.com/download/symbols"^
 -v -n -c "!analyze -vvvvv"
```

Note

The `^` character and newline are removed by the Windows command line processor, so be sure the spaces remain intact.

5.8.1.4 Debugging mysqld under gdb

On most systems you can also start `mysqld` from `gdb` to get more information if `mysqld` crashes.

With some older `gdb` versions on Linux you must use `run --one-thread` if you want to be able to debug `mysqld` threads. In this case, you can only have one thread active at a time.

NPTL threads (the new thread library on Linux) may cause problems while running `mysqld` under `gdb`. Some symptoms are:

- `mysqld` hangs during startup (before it writes `ready for connections`).
- `mysqld` crashes during a `pthread_mutex_lock()` or `pthread_mutex_unlock()` call.

In this case, you should set the following environment variable in the shell before starting `gdb`:

```
LD_ASSUME_KERNEL=2.4.1
export LD_ASSUME_KERNEL
```

When running `mysqld` under `gdb`, you should disable the stack trace with `--skip-stack-trace` to be able to catch segfaults within `gdb`.

Use the `--gdb` option to `mysqld` to install an interrupt handler for `SIGINT` (needed to stop `mysqld` with `^C` to set breakpoints) and disable stack tracing and core file handling.

It is very hard to debug MySQL under `gdb` if you do a lot of new connections the whole time as `gdb` does not free the memory for old threads. You can avoid this problem by starting `mysqld` with `thread_cache_size` set to a value equal to `max_connections` + 1. In most cases just using `--thread_cache_size=5'` helps a lot!

If you want to get a core dump on Linux if `mysqld` dies with a SIGSEGV signal, you can start `mysqld` with the `--core-file` option. This core file can be used to make a backtrace that may help you find out why `mysqld` died:

```
shell> gdb mysqld core
gdb>   backtrace full
gdb>   quit
```

See Section B.3.3.3, “What to Do If MySQL Keeps Crashing”.

If you are using `gdb` on Linux, you should install a `.gdb` file, with the following information, in your current directory:

```
set print sevenbit off
handle SIGUSR1 nostop noprint
handle SIGUSR2 nostop noprint
handle SIGWAITING nostop noprint
handle SIGLWP nostop noprint
handle SIGPIPE nostop
handle SIGALRM nostop
handle SIGHUP nostop
handle SIGTERM nostop noprint
```

Here is an example how to debug `mysqld`:

```
shell> gdb /usr/local/libexec/mysqld
gdb> run
...
backtrace full # Do this when mysqld crashes
```

Include the preceding output in a bug report, which you can file using the instructions in Section 1.6, “How to Report Bugs or Problems”.

If `mysqld` hangs, you can try to use some system tools like `strace` or `/usr/proc/bin/pstack` to examine where `mysqld` has hung.

```
strace /tmp/log libexec/mysqld
```

If you are using the Perl `DBI` interface, you can turn on debugging information by using the `trace` method or by setting the `DBI_TRACE` environment variable.

5.8.1.5 Using a Stack Trace

On some operating systems, the error log contains a stack trace if `mysqld` dies unexpectedly. You can use this to find out where (and maybe why) `mysqld` died. See Section 5.4.2, “The Error Log”. To get a stack trace, you must not compile `mysqld` with the `-fomit-frame-pointer` option to gcc. See Section 5.8.1.1, “Compiling MySQL for Debugging”.

A stack trace in the error log looks something like this:

```
mysqld got signal 11;
Attempting backtrace. You can use the following information
to find out where mysqld died. If you see no messages after
this, something went terribly wrong...

stack_bottom = 0x41fd0110 thread_stack 0x40000
mysqld(my_print_stacktrace+0x32)[0x9da402]
mysqld(handle_segfault+0x28a)[0x6648e9]
/lib/libpthread.so.0[0x7f1a5af000f0]
/lib/libc.so.6(strcmp+0x2)[0x7f1a5a10f0f2]
mysqld(_Z21check_change_passwordP3THDPKcS2_Pcj+0x7c)[0x7412cb]
mysqld(_ZN16set_var_password5checkEP3THD+0xd0)[0x688354]
mysqld(_Z17sql_set_variablesP3THDP4ListI12set_var_baseE+0x68)[0x688494]
mysqld(_Z21mysql_execute_commandP3THD+0x41a0)[0x67a170]
mysqld(_Z11mysql_parseP3THDPKcjPS2_+0x282)[0x67f0ad]
mysqld(_Z16dispatch_command19enum_server_commandP3THDPcj+0xbb7[0x67fdf8]
mysqld(_Z10do_commandP3THD+0x24d)[0x6811b6]
mysqld(handle_one_connection+0x11c)[0x66e05e]
```

If resolution of function names for the trace fails, the trace contains less information:

```
mysqld got signal 11;
Attempting backtrace. You can use the following information
to find out where mysqld died. If you see no messages after
this, something went terribly wrong...

stack_bottom = 0x41fd0110 thread_stack 0x40000
[0x9da402]
[0x6648e9]
[0x7f1a5af000f0]
[0x7f1a5a10f0f2]
[0x7412cb]
[0x688354]
[0x688494]
[0x67a170]
[0x67f0ad]
[0x67fdf8]
[0x6811b6]
[0x66e05e]
```

In the latter case, you can use the `resolve_stack_dump` utility to determine where `mysqld` died by using the following procedure:

1. Copy the numbers from the stack trace to a file, for example `mysqld.stack`. The numbers should not include the surrounding square brackets:

   ```
   0x9da402
   0x6648e9
   0x7f1a5af000f0
   0x7f1a5a10f0f2
   0x7412cb
   0x688354
   0x688494
   0x67a170
   0x67f0ad
   0x67fdf8
   0x6811b6
   0x66e05e
   ```

2. Make a symbol file for the `mysqld` server:

```
shell> nm -n libexec/mysqld > /tmp/mysqld.sym
```

If `mysqld` is not linked statically, use the following command instead:

```
shell> nm -D -n libexec/mysqld > /tmp/mysqld.sym
```

If you want to decode C++ symbols, use the `--demangle`, if available, to `nm`. If your version of `nm` does not have this option, you must use the `c++filt` command after the stack dump has been produced to demangle the C++ names.

3. Execute the following command:

```
shell> resolve_stack_dump -s /tmp/mysqld.sym -n mysqld.stack
```

If you were not able to include demangled C++ names in your symbol file, process the `resolve_stack_dump` output using `c++filt`:

```
shell> resolve_stack_dump -s /tmp/mysqld.sym -n mysqld.stack | c++filt
```

This prints out where `mysqld` died. If that does not help you find out why `mysqld` died, you should create a bug report and include the output from the preceding command with the bug report.

However, in most cases it does not help us to have just a stack trace to find the reason for the problem. To be able to locate the bug or provide a workaround, in most cases we need to know the statement that killed `mysqld` and preferably a test case so that we can repeat the problem! See Section 1.6, “How to Report Bugs or Problems”.

Newer versions of `glibc` stack trace functions also print the address as relative to the object. On `glibc`-based systems (Linux), the trace for an unexpected exit within a plugin looks something like:

```
plugin/auth/auth_test_plugin.so(+0x9a6)[0x7ff4d11c29a6]
```

To translate the relative address (`+0x9a6`) into a file name and line number, use this command:

```
shell> addr2line -fie auth_test_plugin.so 0x9a6
auth_test_plugin
mysql-trunk/plugin/auth/test_plugin.c:65
```

The `addr2line` utility is part of the `binutils` package on Linux.

On Solaris, the procedure is similar. The Solaris `printstack()` already prints relative addresses:

```
plugin/auth/auth_test_plugin.so:0x1510
```

To translate, use this command:

```
shell> gaddr2line -fie auth_test_plugin.so 0x1510
mysql-trunk/plugin/auth/test_plugin.c:88
```

Windows already prints the address, function name and line:

```
000007FEF07E10A4 auth_test_plugin.dll!auth_test_plugin()[test_plugin.c:72]
```

5.8.1.6 Using Server Logs to Find Causes of Errors in mysqld

Before starting `mysqld` with the general query log enabled, you should check all your tables with `myisamchk`. See Chapter 5, *MySQL Server Administration*.

If `mysqld` dies or hangs, you should start `mysqld` with the general query log enabled. See Section 5.4.3, “The General Query Log”. When `mysqld` dies again, you can examine the end of the log file for the query that killed `mysqld`.

If you use the default general query log file, the log is stored in the database directory as *`host_name`*`.log` In most cases it is the last query in the log file that killed `mysqld`, but if possible you

should verify this by restarting `mysqld` and executing the found query from the `mysql` command-line tools. If this works, you should also test all complicated queries that did not complete.

You can also try the command `EXPLAIN` on all `SELECT` statements that takes a long time to ensure that `mysqld` is using indexes properly. See Section 13.8.2, “EXPLAIN Statement”.

You can find the queries that take a long time to execute by starting `mysqld` with the slow query log enabled. See Section 5.4.5, “The Slow Query Log”.

If you find the text `mysqld restarted` in the error log (normally a file named *`host_name`*`.err`) you probably have found a query that causes `mysqld` to fail. If this happens, you should check all your tables with `myisamchk` (see Chapter 5, *MySQL Server Administration*), and test the queries in the MySQL log files to see whether one fails. If you find such a query, try first upgrading to the newest MySQL version. If this does not help, report a bug, see Section 1.6, “How to Report Bugs or Problems”.

If you have started `mysqld` with the `myisam_recover_options` system variable set, MySQL automatically checks and tries to repair `MyISAM` tables if they are marked as 'not closed properly' or 'crashed'. If this happens, MySQL writes an entry in the `hostname.err` file `'Warning: Checking table ...'` which is followed by `Warning: Repairing table` if the table needs to be repaired. If you get a lot of these errors, without `mysqld` having died unexpectedly just before, then something is wrong and needs to be investigated further. See Section 5.1.6, “Server Command Options”.

When the server detects `MyISAM` table corruption, it writes additional information to the error log, such as the name and line number of the source file, and the list of threads accessing the table. Example: `Got an error from thread_id=1, mi_dynrec.c:368`. This is useful information to include in bug reports.

It is not a good sign if `mysqld` did die unexpectedly, but in this case, you should not investigate the `Checking table...` messages, but instead try to find out why `mysqld` died.

5.8.1.7 Making a Test Case If You Experience Table Corruption

The following procedure applies to `MyISAM` tables. For information about steps to take when encountering `InnoDB` table corruption, see Section 1.6, “How to Report Bugs or Problems”.

If you encounter corrupted `MyISAM` tables or if `mysqld` always fails after some update statements, you can test whether the issue is reproducible by doing the following:

1. Stop the MySQL daemon with `mysqladmin shutdown`.
2. Make a backup of the tables to guard against the very unlikely case that the repair does something bad.
3. Check all tables with `myisamchk -s database/*.MYI`. Repair any corrupted tables with `myisamchk -r database/`*`table`*`.MYI`.
4. Make a second backup of the tables.
5. Remove (or move away) any old log files from the MySQL data directory if you need more space.
6. Start `mysqld` with the binary log enabled. If you want to find a statement that crashes `mysqld`, you should start the server with the general query log enabled as well. See Section 5.4.3, “The General Query Log”, and Section 5.4.4, “The Binary Log”.
7. When you have gotten a crashed table, stop the `mysqld` server.
8. Restore the backup.
9. Restart the `mysqld` server *without* the binary log enabled.
10. Re-execute the statements with `mysqlbinlog binary-log-file | mysql`. The binary log is saved in the MySQL database directory with the name `hostname-bin.`*`NNNNNN`*.

11. If the tables are corrupted again or you can get `mysqld` to die with the above command, you have found a reproducible bug. FTP the tables and the binary log to our bugs database using the instructions given in Section 1.6, “How to Report Bugs or Problems”. If you are a support customer, you can use the MySQL Customer Support Center (https://www.mysql.com/support/) to alert the MySQL team about the problem and have it fixed as soon as possible.

5.8.2 Debugging a MySQL Client

To be able to debug a MySQL client with the integrated debug package, you should configure MySQL with `-DWITH_DEBUG=1`. See Section 2.9.7, “MySQL Source-Configuration Options”.

Before running a client, you should set the `MYSQL_DEBUG` environment variable:

```
shell> MYSQL_DEBUG=d:t:O,/tmp/client.trace
shell> export MYSQL_DEBUG
```

This causes clients to generate a trace file in `/tmp/client.trace`.

If you have problems with your own client code, you should attempt to connect to the server and run your query using a client that is known to work. Do this by running `mysql` in debugging mode (assuming that you have compiled MySQL with debugging on):

```
shell> mysql --debug=d:t:O,/tmp/client.trace
```

This provides useful information in case you mail a bug report. See Section 1.6, “How to Report Bugs or Problems”.

If your client crashes at some 'legal' looking code, you should check that your `mysql.h` include file matches your MySQL library file. A very common mistake is to use an old `mysql.h` file from an old MySQL installation with new MySQL library.

5.8.3 The DBUG Package

The MySQL server and most MySQL clients are compiled with the DBUG package originally created by Fred Fish. When you have configured MySQL for debugging, this package makes it possible to get a trace file of what the program is doing. See Section 5.8.1.2, “Creating Trace Files”.

This section summarizes the argument values that you can specify in debug options on the command line for MySQL programs that have been built with debugging support.

The DBUG package can be used by invoking a program with the `--debug[=debug_options]` or `-#[debug_options]` option. If you specify the `--debug` or `-#` option without a *`debug_options`* value, most MySQL programs use a default value. The server default is `d:t:i:o,/tmp/mysqld.trace` on Unix and `d:t:i:O,\mysqld.trace` on Windows. The effect of this default is:

- `d`: Enable output for all debug macros
- `t`: Trace function calls and exits
- `i`: Add PID to output lines
- `o,/tmp/mysqld.trace`, `O,\mysqld.trace`: Set the debug output file.

Most client programs use a default *`debug_options`* value of `d:t:o,/tmp/`*`program_name`*`.trace`, regardless of platform.

Here are some example debug control strings as they might be specified on a shell command line:

```
--debug=d:t
--debug=d:f,main,subr1:F:L:t,20
--debug=d,input,output,files:n
```

```
--debug=d:t:i:O,\\mysqld.trace
```

For `mysqld`, it is also possible to change DBUG settings at runtime by setting the `debug` system variable. This variable has global and session values:

```
mysql> SET GLOBAL debug = 'debug_options';
mysql> SET SESSION debug = 'debug_options';
```

Changing the global `debug` value requires privileges sufficient to set global system variables. Changing the session `debug` value requires privileges sufficient to set restricted session system variables. See Section 5.1.8.1, “System Variable Privileges”.

The *`debug_options`* value is a sequence of colon-separated fields:

```
field_1:field_2:...:field_N
```

Each field within the value consists of a mandatory flag character, optionally preceded by a `+` or `-` character, and optionally followed by a comma-delimited list of modifiers:

```
[+|-]flag[,modifier,modifier,...,modifier]
```

The following table describes the permitted flag characters. Unrecognized flag characters are silently ignored.

Flag	Description
`d`	Enable output from DBUG_*XXX* macros for the current state. May be followed by a list of keywords, which enables output only for the DBUG macros with that keyword. An empty list of keywords enables output for all macros. In MySQL, common debug macro keywords to enable are `enter`, `exit`, `error`, `warning`, `info`, and `loop`.
`D`	Delay after each debugger output line. The argument is the delay, in tenths of seconds, subject to machine capabilities. For example, `D,20` specifies a delay of two seconds.
`f`	Limit debugging, tracing, and profiling to the list of named functions. An empty list enables all functions. The appropriate `d` or `t` flags must still be given; this flag only limits their actions if they are enabled.
`F`	Identify the source file name for each line of debug or trace output.
`i`	Identify the process with the PID or thread ID for each line of debug or trace output.
`L`	Identify the source file line number for each line of debug or trace output.
`n`	Print the current function nesting depth for each line of debug or trace output.
`N`	Number each line of debug output.
`o`	Redirect the debugger output stream to the specified file. The default output is `stderr`.
`O`	Like `o`, but the file is really flushed between each write. When needed, the file is closed and reopened between each write.
`p`	Limit debugger actions to specified processes. A process must be identified with the `DBUG_PROCESS` macro and match one in the list for debugger actions to occur.
`P`	Print the current process name for each line of debug or trace output.
`r`	When pushing a new state, do not inherit the previous state's function nesting level. Useful when the output is to start at the left margin.
`S`	Do function `_sanity(_file_,_line_)` at each debugged function until `_sanity()` returns something that differs from 0.
`t`	Enable function call/exit trace lines. May be followed by a list (containing only one modifier) giving a numeric maximum trace level, beyond which no output occurs for either debugging or tracing macros. The default is a compile time option.

The leading + or - character and trailing list of modifiers are used for flag characters such as d or f that can enable a debug operation for all applicable modifiers or just some of them:

- With no leading + or -, the flag value is set to exactly the modifier list as given.
- With a leading + or -, the modifiers in the list are added to or subtracted from the current modifier list.

The following examples show how this works for the d flag. An empty d list enabled output for all debug macros. A nonempty list enables output only for the macro keywords in the list.

These statements set the d value to the modifier list as given:

```
mysql> SET debug = 'd';
mysql> SELECT @@debug;
+---------+
| @@debug |
+---------+
| d       |
+---------+
mysql> SET debug = 'd,error,warning';
mysql> SELECT @@debug;
+-----------------+
| @@debug         |
+-----------------+
| d,error,warning |
+-----------------+
```

A leading + or - adds to or subtracts from the current d value:

```
mysql> SET debug = '+d,loop';
mysql> SELECT @@debug;
+----------------------+
| @@debug              |
+----------------------+
| d,error,warning,loop |
+----------------------+
mysql> SET debug = '-d,error,loop';
mysql> SELECT @@debug;
+-----------+
| @@debug   |
+-----------+
| d,warning |
+-----------+
```

Adding to “all macros enabled” results in no change:

```
mysql> SET debug = 'd';
mysql> SELECT @@debug;
+---------+
| @@debug |
+---------+
| d       |
+---------+
mysql> SET debug = '+d,loop';
mysql> SELECT @@debug;
+---------+
| @@debug |
+---------+
| d       |
+---------+
```

Disabling all enabled macros disables the d flag entirely:

```
mysql> SET debug = 'd,error,loop';
mysql> SELECT @@debug;
+--------------+
| @@debug      |
+--------------+
| d,error,loop |
```

```
+--------------+
mysql> SET debug = '-d,error,loop';
mysql> SELECT @@debug;
+---------+
| @@debug |
+---------+
|         |
+---------+
```

Note

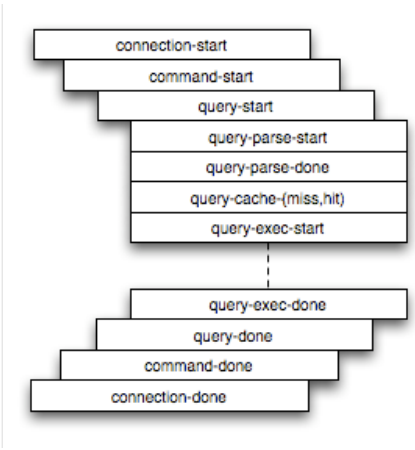
Prior to MySQL 5.6.12, the `+` and `-` modifiers were not always handled correctly and could leave a flag value in an incorrect state. Verify your `debug`-setting sequence in advance or set it without using `+` or `-`.

5.8.4 Tracing mysqld Using DTrace

The DTrace probes in the MySQL server are designed to provide information about the execution of queries within MySQL and the different areas of the system being utilized during that process. The organization and triggering of the probes means that the execution of an entire query can be monitored with one level of probes (`query-start` and `query-done`) but by monitoring other probes you can get successively more detailed information about the execution of the query in terms of the locks used, sort methods and even row-by-row and storage-engine level execution information.

The DTrace probes are organized so that you can follow the entire query process, from the point of connection from a client, through the query execution, row-level operations, and back out again. You can think of the probes as being fired within a specific sequence during a typical client connect/execute/disconnect sequence, as shown in the following figure.

Figure 5.1 DTrace Probe Sequence



Global information is provided in the arguments to the DTrace probes at various levels. Global information, that is, the connection ID and user/host and where relevant the query string, is provided at key levels (`connection-start`, `command-start`, `query-start`, and `query-exec-start`). As you go deeper into the probes, it is assumed either you are only interested in the individual executions (row-level probes provide information on the database and table name only), or that you combine the row-level probes with the notional parent probes to provide the information about a specific query. Examples of this are given as the format and arguments of each probe are provided.

MySQL includes support for DTrace probes on these platforms:

- Solaris 10 Update 5 (Solaris 5/08) on SPARC, x86 and x86_64 platforms
- OS X 10.4 and higher
- Oracle Linux 6 and higher with UEK kernel (as of MySQL 5.6.20)

Enabling the probes should be automatic on these platforms. To explicitly enable or disable the probes during building, use the `-DENABLE_DTRACE=1` or `-DENABLE_DTRACE=0` option to `CMake`.

If a non-Solaris platform includes DTrace support, building `mysqld` on that platform includes DTrace support.

Additional Resources

- For more information on DTrace and writing DTrace scripts, read the DTrace User Guide.
- For an introduction to DTrace, see the MySQL Dev Zone article Getting started with DTracing MySQL.

5.8.4.1 mysqld DTrace Probe Reference

MySQL supports the following static probes, organized into groups of functionality.

Table 5.6 MySQL DTrace Probes

Group	Probes
Connection	`connection-start`, `connection-done`
Command	`command-start`, `command-done`
Query	`query-start`, `query-done`
Query Parsing	`query-parse-start`, `query-parse-done`
Query Cache	`query-cache-hit`, `query-cache-miss`
Query Execution	`query-exec-start`, `query-exec-done`
Row Level	`insert-row-start`, `insert-row-done`
	`update-row-start`, `update-row-done`
	`delete-row-start`, `delete-row-done`
Row Reads	`read-row-start`, `read-row-done`
Index Reads	`index-read-row-start`, `index-read-row-done`
Lock	`handler-rdlock-start`, `handler-rdlock-done`
	`handler-wrlock-start`, `handler-wrlock-done`
	`handler-unlock-start`, `handler-unlock-done`
Filesort	`filesort-start`, `filesort-done`
Statement	`select-start`, `select-done`
	`insert-start`, `insert-done`
	`insert-select-start`, `insert-select-done`
	`update-start`, `update-done`
	`multi-update-start`, `multi-update-done`
	`delete-start`, `delete-done`
	`multi-delete-start`, `multi-delete-done`
Network	`net-read-start`, `net-read-done`, `net-write-start`, `net-write-done`
Keycache	`keycache-read-start`, `keycache-read-block`, `keycache-read-done`, `keycache-read-hit`, `keycache-read-miss`, `keycache-write-start`, `keycache-write-block`, `keycache-write-done`

Note

When extracting the argument data from the probes, each argument is available as `arg`*N*, starting with `arg0`. To identify each argument within the definitions

> they are provided with a descriptive name, but you must access the information using the corresponding `argN` parameter.

Connection Probes

The `connection-start` and `connection-done` probes enclose a connection from a client, regardless of whether the connection is through a socket or network connection.

```
connection-start(connectionid, user, host)
connection-done(status, connectionid)
```

- `connection-start`: Triggered after a connection and successful login/authentication have been completed by a client. The arguments contain the connection information:
  - `connectionid`: An `unsigned long` containing the connection ID. This is the same as the process ID shown as the `Id` value in the output from `SHOW PROCESSLIST`.
  - `user`: The username used when authenticating. The value is blank for the anonymous user.
  - `host`: The host of the client connection. For a connection made using Unix sockets, the value is blank.
- `connection-done`: Triggered just as the connection to the client has been closed. The arguments are:
  - `status`: The status of the connection when it was closed. A logout operation has a value of 0; any other termination of the connection has a nonzero value.
  - `connectionid`: The connection ID of the connection that was closed.

The following D script quantifies and summarizes the average duration of individual connections, and provides a count, dumping the information every 60 seconds:

```
#!/usr/sbin/dtrace -s


mysql*:::connection-start
{
  self->start = timestamp;
}

mysql*:::connection-done
/self->start/
{
  @ = quantize(((timestamp - self->start)/1000000));
  self->start = 0;
}

tick-60s
{
  printa(@);
}
```

When executed on a server with a large number of clients you might see output similar to this:

```
  1  57413                         :tick-60s

           value  ------------- Distribution ------------- count
              -1 |                                         0
               0 |@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@ 30011
               1 |                                         59
               2 |                                         5
               4 |                                         20
               8 |                                         29
              16 |                                         18
              32 |                                         27
              64 |                                         30
```

```
           128 |                                      11
           256 |                                      10
           512 |                                      1
          1024 |                                      6
          2048 |                                      8
          4096 |                                      9
          8192 |                                      8
         16384 |                                      2
         32768 |                                      1
         65536 |                                      1
        131072 |                                      0
        262144 |                                      1
        524288 |                                      0
```

Command Probes

The command probes are executed before and after a client command is executed, including any SQL statement that might be executed during that period. Commands include operations such as the initialization of the DB, use of the `COM_CHANGE_USER` operation (supported by the MySQL protocol), and manipulation of prepared statements. Many of these commands are used only by the MySQL client API from various connectors such as PHP and Java.

```
command-start(connectionid, command, user, host)
command-done(status)
```

- `command-start`: Triggered when a command is submitted to the server.
  - `connectionid`: The connection ID of the client executing the command.
  - `command`: An integer representing the command that was executed. Possible values are shown in the following table.

Value	Name	Description
00	COM_SLEEP	Internal thread state
01	COM_QUIT	Close connection
02	COM_INIT_DB	Select database (`USE ...`)
03	COM_QUERY	Execute a query
04	COM_FIELD_LIST	Get a list of fields
05	COM_CREATE_DB	Create a database (deprecated)
06	COM_DROP_DB	Drop a database (deprecated)
07	COM_REFRESH	Refresh connection
08	COM_SHUTDOWN	Shutdown server
09	COM_STATISTICS	Get statistics
10	COM_PROCESS_INFO	Get processes (`SHOW PROCESSLIST`)
11	COM_CONNECT	Initialize connection
12	COM_PROCESS_KILL	Kill process
13	COM_DEBUG	Get debug information
14	COM_PING	Ping
15	COM_TIME	Internal thread state
16	COM_DELAYED_INSERT	Internal thread state
17	COM_CHANGE_USER	Change user
18	COM_BINLOG_DUMP	Used by a replica or `mysqlbinlog` to initiate a binary log read
19	COM_TABLE_DUMP	Used by a replica to get the source table information
20	COM_CONNECT_OUT	Used by a replica to log a connection to the server

Value	Name	Description
21	COM_REGISTER_SLAVE	Used by a replica during registration
22	COM_STMT_PREPARE	Prepare a statement
23	COM_STMT_EXECUTE	Execute a statement
24	COM_STMT_SEND_LONG_DATA	Used by a client when requesting extended data
25	COM_STMT_CLOSE	Close a prepared statement
26	COM_STMT_RESET	Reset a prepared statement
27	COM_SET_OPTION	Set a server option
28	COM_STMT_FETCH	Fetch a prepared statement

  - `user`: The user executing the command.
  - `host`: The client host.
- `command-done`: Triggered when the command execution completes. The `status` argument contains 0 if the command executed successfully, or 1 if the statement was terminated before normal completion.

The `command-start` and `command-done` probes are best used when combined with the statement probes to get an idea of overall execution time.

Query Probes

The `query-start` and `query-done` probes are triggered when a specific query is received by the server and when the query has been completed and the information has been successfully sent to the client.

```
query-start(query, connectionid, database, user, host)
query-done(status)
```

- `query-start`: Triggered after the query string has been received from the client. The arguments are:
  - `query`: The full text of the submitted query.
  - `connectionid`: The connection ID of the client that submitted the query. The connection ID equals the connection ID returned when the client first connects and the `Id` value in the output from `SHOW PROCESSLIST`.
  - `database`: The database name on which the query is being executed.
  - `user`: The username used to connect to the server.
  - `host`: The hostname of the client.
- `query-done`: Triggered once the query has been executed and the information has been returned to the client. The probe includes a single argument, `status`, which returns 0 when the query is successfully executed and 1 if there was an error.

You can get a simple report of the execution time for each query using the following D script:

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

dtrace:::BEGIN
{
   printf("%-20s %-20s %-40s %-9s\n", "Who", "Database", "Query", "Time(ms)");
}
```

```
mysql*:::query-start
{
   self->query = copyinstr(arg0);
   self->connid = arg1;
   self->db    = copyinstr(arg2);
   self->who   = strjoin(copyinstr(arg3),strjoin("@",copyinstr(arg4)));
   self->querystart = timestamp;
}

mysql*:::query-done
{
   printf("%-20s %-20s %-40s %-9d\n",self->who,self->db,self->query,
          (timestamp - self->querystart) / 1000000);
}
```

When executing the above script you should get a basic idea of the execution time of your queries:

```
shell> ./query.d
Who                 Database            Query                                    Time(ms)
root@localhost      test                select * from t1 order by i limit 10     0
root@localhost      test                set global query_cache_size=0            0
root@localhost      test                select * from t1 order by i limit 10     776
root@localhost      test                select * from t1 order by i limit 10     773
root@localhost      test                select * from t1 order by i desc limit 10 795
```

Query Parsing Probes

The query parsing probes are triggered before the original SQL statement is parsed and when the parsing of the statement and determination of the execution model required to process the statement has been completed:

```
query-parse-start(query)
query-parse-done(status)
```

- `query-parse-start`: Triggered just before the statement is parsed by the MySQL query parser. The single argument, `query`, is a string containing the full text of the original query.
- `query-parse-done`: Triggered when the parsing of the original statement has been completed. The `status` is an integer describing the status of the operation. A `0` indicates that the query was successfully parsed. A `1` indicates that the parsing of the query failed.

For example, you could monitor the execution time for parsing a given query using the following D script:

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

mysql*:::query-parse-start
{
   self->parsestart = timestamp;
   self->parsequery = copyinstr(arg0);
}

mysql*:::query-parse-done
/arg0 == 0/
{
   printf("Parsing %s: %d microseconds\n", self->parsequery,((timestamp - self->parsestart)/1000));
}

mysql*:::query-parse-done
/arg0 != 0/
{
   printf("Error parsing %s: %d microseconds\n", self->parsequery,((timestamp - self->parsestart)/1000)
}
```

In the above script a predicate is used on `query-parse-done` so that different output is generated based on the status value of the probe.

When running the script and monitoring the execution:

```
shell> ./query-parsing.d
Error parsing select from t1 join (t2) on (t1.i = t2.i) order by t1.s,t1.i limit 10: 36 ms
Parsing select * from t1 join (t2) on (t1.i = t2.i) order by t1.s,t1.i limit 10: 176 ms
```

Query Cache Probes

The query cache probes are fired when executing any query. The `query-cache-hit` query is triggered when a query exists in the query cache and can be used to return the query cache information. The arguments contain the original query text and the number of rows returned from the query cache for the query. If the query is not within the query cache, or the query cache is not enabled, then the `query-cache-miss` probe is triggered instead.

```
query-cache-hit(query, rows)
query-cache-miss(query)
```

- `query-cache-hit`: Triggered when the query has been found within the query cache. The first argument, `query`, contains the original text of the query. The second argument, `rows`, is an integer containing the number of rows in the cached query.
- `query-cache-miss`: Triggered when the query is not found within the query cache. The first argument, `query`, contains the original text of the query.

The query cache probes are best combined with a probe on the main query so that you can determine the differences in times between using or not using the query cache for specified queries. For example, in the following D script, the query and query cache information are combined into the information output during monitoring:

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

dtrace:::BEGIN
{
   printf("%-20s %-20s %-40s %2s %-9s\n", "Who", "Database", "Query", "QC", "Time(ms)");
}

mysql*:::query-start
{
   self->query = copyinstr(arg0);
   self->connid = arg1;
   self->db    = copyinstr(arg2);
   self->who   = strjoin(copyinstr(arg3),strjoin("@",copyinstr(arg4)));
   self->querystart = timestamp;
   self->qc = 0;
}

mysql*:::query-cache-hit
{
   self->qc = 1;
}

mysql*:::query-cache-miss
{
   self->qc = 0;
}

mysql*:::query-done
{
   printf("%-20s %-20s %-40s %-2s %-9d\n",self->who,self->db,self->query,(self->qc ? "Y" : "N"),
          (timestamp - self->querystart) / 1000000);
}
```

When executing the script you can see the effects of the query cache. Initially the query cache is disabled. If you set the query cache size and then execute the query multiple times you should see that the query cache is being used to return the query data:

```
shell> ./query-cache.d
root@localhost       test                select * from t1 order by i limit 10     N  1072
root@localhost                           set global query_cache_size=262144       N  0
root@localhost       test                select * from t1 order by i limit 10     N  781
root@localhost       test                select * from t1 order by i limit 10     Y  0
```

Query Execution Probes

The query execution probe is triggered when the actual execution of the query starts, after the parsing and checking the query cache but before any privilege checks or optimization. By comparing the difference between the start and done probes you can monitor the time actually spent servicing the query (instead of just handling the parsing and other elements of the query).

```
query-exec-start(query, connectionid, database, user, host, exec_type)
query-exec-done(status)
```

Note

The information provided in the arguments for `query-start` and `query-exec-start` are almost identical and designed so that you can choose to monitor either the entire query process (using `query-start`) or only the execution (using `query-exec-start`) while exposing the core information about the user, client, and query being executed.

- `query-exec-start`: Triggered when the execution of a individual query is started. The arguments are:
  - `query`: The full text of the submitted query.
  - `connectionid`: The connection ID of the client that submitted the query. The connection ID equals the connection ID returned when the client first connects and the `Id` value in the output from `SHOW PROCESSLIST`.
  - `database`: The database name on which the query is being executed.
  - `user`: The username used to connect to the server.
  - `host`: The hostname of the client.
  - `exec_type`: The type of execution. Execution types are determined based on the contents of the query and where it was submitted. The values for each type are shown in the following table.

Value	Description
0	Executed query from sql_parse, top-level query.
1	Executed prepared statement
2	Executed cursor statement
3	Executed query in stored procedure

- `query-exec-done`: Triggered when the execution of the query has completed. The probe includes a single argument, `status`, which returns 0 when the query is successfully executed and 1 if there was an error.

Row-Level Probes

The `*row-{start,done}` probes are triggered each time a row operation is pushed down to a storage engine. For example, if you execute an `INSERT` statement with 100 rows of data, then the `insert-row-start` and `insert-row-done` probes are triggered 100 times each, for each row insert.

```
insert-row-start(database, table)
insert-row-done(status)
```

```
update-row-start(database, table)
update-row-done(status)

delete-row-start(database, table)
delete-row-done(status)
```

- `insert-row-start`: Triggered before a row is inserted into a table.
- `insert-row-done`: Triggered after a row is inserted into a table.
- `update-row-start`: Triggered before a row is updated in a table.
- `update-row-done`: Triggered before a row is updated in a table.
- `delete-row-start`: Triggered before a row is deleted from a table.
- `delete-row-done`: Triggered before a row is deleted from a table.

The arguments supported by the probes are consistent for the corresponding `start` and `done` probes in each case:

- `database`: The database name.
- `table`: The table name.
- `status`: The status; 0 for success or 1 for failure.

Because the row-level probes are triggered for each individual row access, these probes can be triggered many thousands of times each second, which may have a detrimental effect on both the monitoring script and MySQL. The DTrace environment should limit the triggering on these probes to prevent the performance being adversely affected. Either use the probes sparingly, or use counter or aggregation functions to report on these probes and then provide a summary when the script terminates or as part of a `query-done` or `query-exec-done` probes.

The following example script summarizes the duration of each row operation within a larger query:

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

dtrace:::BEGIN
{
   printf("%-2s %-10s %-10s %9s %9s %-s \n",
          "St", "Who", "DB", "ConnID", "Dur ms", "Query");
}

mysql*:::query-start
{
   self->query = copyinstr(arg0);
   self->who   = strjoin(copyinstr(arg3),strjoin("@",copyinstr(arg4)));
   self->db    = copyinstr(arg2);
   self->connid = arg1;
   self->querystart = timestamp;
   self->rowdur = 0;
}

mysql*:::query-done
{
   this->elapsed = (timestamp - self->querystart) /1000000;
   printf("%2d %-10s %-10s %9d %9d %s\n",
          arg0, self->who, self->db,
          self->connid, this->elapsed, self->query);
}

mysql*:::query-done
/ self->rowdur /
{
```

```
   printf("%34s %9d %s\n", "", (self->rowdur/1000000), "-> Row ops");
}

mysql*:::insert-row-start
{
   self->rowstart = timestamp;
}

mysql*:::delete-row-start
{
   self->rowstart = timestamp;
}

mysql*:::update-row-start
{
   self->rowstart = timestamp;
}

mysql*:::insert-row-done
{
   self->rowdur += (timestamp-self->rowstart);
}

mysql*:::delete-row-done
{
   self->rowdur += (timestamp-self->rowstart);
}

mysql*:::update-row-done
{
   self->rowdur += (timestamp-self->rowstart);
}
```

Running the above script with a query that inserts data into a table, you can monitor the exact time spent performing the raw row insertion:

```
St Who        DB            ConnID    Dur ms Query
 0 @localhost test              13     20767 insert into t1(select * from t2)
                                        4827 -> Row ops
```

Read Row Probes

The read row probes are triggered at a storage engine level each time a row read operation occurs. These probes are specified within each storage engine (as opposed to the `*row-start` probes which are in the storage engine interface). These probes can therefore be used to monitor individual storage engine row-level operations and performance. Because these probes are triggered around the storage engine row read interface, they may be hit a significant number of times during a basic query.

```
read-row-start(database, table, scan_flag)
read-row-done(status)
```

- `read-row-start`: Triggered when a row is read by the storage engine from the specified `database` and `table`. The `scan_flag` is set to 1 (true) when the read is part of a table scan (that is, a sequential read), or 0 (false) when the read is of a specific record.
- `read-row-done`: Triggered when a row read operation within a storage engine completes. The `status` returns 0 on success, or a positive value on failure.

Index Probes

The index probes are triggered each time a row is read using one of the indexes for the specified table. The probe is triggered within the corresponding storage engine for the table.

```
index-read-row-start(database, table)
index-read-row-done(status)
```

- `index-read-row-start`: Triggered when a row is read by the storage engine from the specified `database` and `table`.

- `index-read-row-done`: Triggered when an indexed row read operation within a storage engine completes. The `status` returns 0 on success, or a positive value on failure.

Lock Probes

The lock probes are called whenever an external lock is requested by MySQL for a table using the corresponding lock mechanism on the table as defined by the table's engine type. There are three different types of lock, the read lock, write lock, and unlock operations. Using the probes you can determine the duration of the external locking routine (that is, the time taken by the storage engine to implement the lock, including any time waiting for another lock to become free) and the total duration of the lock/unlock process.

```
handler-rdlock-start(database, table)
handler-rdlock-done(status)

handler-wrlock-start(database, table)
handler-wrlock-done(status)

handler-unlock-start(database, table)
handler-unlock-done(status)
```

- `handler-rdlock-start`: Triggered when a read lock is requested on the specified `database` and `table`.
- `handler-wrlock-start`: Triggered when a write lock is requested on the specified `database` and `table`.
- `handler-unlock-start`: Triggered when an unlock request is made on the specified `database` and `table`.
- `handler-rdlock-done`: Triggered when a read lock request completes. The `status` is 0 if the lock operation succeeded, or `>0` on failure.
- `handler-wrlock-done`: Triggered when a write lock request completes. The `status` is 0 if the lock operation succeeded, or `>0` on failure.
- `handler-unlock-done`: Triggered when an unlock request completes. The `status` is 0 if the unlock operation succeeded, or `>0` on failure.

You can use arrays to monitor the locking and unlocking of individual tables and then calculate the duration of the entire table lock using the following script:

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

mysql*:::handler-rdlock-start
{
   self->rdlockstart = timestamp;
   this->lockref = strjoin(copyinstr(arg0),strjoin("@",copyinstr(arg1)));
   self->lockmap[this->lockref] = self->rdlockstart;
   printf("Start: Lock->Read   %s.%s\n",copyinstr(arg0),copyinstr(arg1));
}

mysql*:::handler-wrlock-start
{
   self->wrlockstart = timestamp;
   this->lockref = strjoin(copyinstr(arg0),strjoin("@",copyinstr(arg1)));
   self->lockmap[this->lockref] = self->rdlockstart;
   printf("Start: Lock->Write  %s.%s\n",copyinstr(arg0),copyinstr(arg1));
}

mysql*:::handler-unlock-start
{
   self->unlockstart = timestamp;
   this->lockref = strjoin(copyinstr(arg0),strjoin("@",copyinstr(arg1)));
   printf("Start: Lock->Unlock %s.%s (%d ms lock duration)\n",
```

```
          copyinstr(arg0),copyinstr(arg1),
          (timestamp - self->lockmap[this->lockref])/1000000);
}

mysql*:::handler-rdlock-done
{
   printf("End:   Lock->Read   %d ms\n",
          (timestamp - self->rdlockstart)/1000000);
}

mysql*:::handler-wrlock-done
{
   printf("End:   Lock->Write  %d ms\n",
          (timestamp - self->wrlockstart)/1000000);
}

mysql*:::handler-unlock-done
{
   printf("End:   Lock->Unlock %d ms\n",
          (timestamp - self->unlockstart)/1000000);
}
```

When executed, you should get information both about the duration of the locking process itself, and of the locks on a specific table:

```
Start: Lock->Read   test.t2
End:   Lock->Read   0 ms
Start: Lock->Unlock test.t2 (25743 ms lock duration)
End:   Lock->Unlock 0 ms
Start: Lock->Read   test.t2
End:   Lock->Read   0 ms
Start: Lock->Unlock test.t2 (1 ms lock duration)
End:   Lock->Unlock 0 ms
Start: Lock->Read   test.t2
End:   Lock->Read   0 ms
Start: Lock->Unlock test.t2 (1 ms lock duration)
End:   Lock->Unlock 0 ms
Start: Lock->Read   test.t2
End:   Lock->Read   0 ms
```

Filesort Probes

The filesort probes are triggered whenever a filesort operation is applied to a table. For more information on filesort and the conditions under which it occurs, see Section 8.2.1.13, “ORDER BY Optimization”.

```
filesort-start(database, table)
filesort-done(status, rows)
```

- `filesort-start`: Triggered when the filesort operation starts on a table. The two arguments to the probe, `database` and `table`, identify the table being sorted.
- `filesort-done`: Triggered when the filesort operation completes. Two arguments are supplied, the `status` (0 for success, 1 for failure), and the number of rows sorted during the filesort process.

An example of this is in the following script, which tracks the duration of the filesort process in addition to the duration of the main query:

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

dtrace:::BEGIN
{
   printf("%-2s %-10s %-10s %9s %18s %-s \n",
          "St", "Who", "DB", "ConnID", "Dur microsec", "Query");
}

mysql*:::query-start
```

```
{
   self->query = copyinstr(arg0);
   self->who   = strjoin(copyinstr(arg3),strjoin("@",copyinstr(arg4)));
   self->db    = copyinstr(arg2);
   self->connid = arg1;
   self->querystart = timestamp;
   self->filesort = 0;
   self->fsdb = "";
   self->fstable = "";
}

mysql*:::filesort-start
{
  self->filesort = timestamp;
  self->fsdb = copyinstr(arg0);
  self->fstable = copyinstr(arg1);
}

mysql*:::filesort-done
{
   this->elapsed = (timestamp - self->filesort) /1000;
   printf("%2d %-10s %-10s %9d %18d Filesort on %s\n",
          arg0, self->who, self->fsdb,
          self->connid, this->elapsed, self->fstable);
}

mysql*:::query-done
{
   this->elapsed = (timestamp - self->querystart) /1000;
   printf("%2d %-10s %-10s %9d %18d %s\n",
          arg0, self->who, self->db,
          self->connid, this->elapsed, self->query);
}
```

Executing a query on a large table with an `ORDER BY` clause that triggers a filesort, and then creating an index on the table and then repeating the same query, you can see the difference in execution speed:

```
St Who        DB            ConnID       Dur microsec Query
 0 @localhost test             14          11335469 Filesort on t1
 0 @localhost test             14          11335787 select * from t1 order by i limit 100
 0 @localhost test             14         466734378 create index t1a on t1 (i)
 0 @localhost test             14             26472 select * from t1 order by i limit 100
```

Statement Probes

The individual statement probes are provided to give specific information about different statement types. For the start probes the string of the query is provided as a the only argument. Depending on the statement type, the information provided by the corresponding done probe differs. For all done probes the status of the operation (`0` for success, `>0` for failure) is provided. For `SELECT`, `INSERT`, `INSERT ... (SELECT FROM ...)`, `DELETE`, and `DELETE FROM t1,t2` operations the number of rows affected is returned.

For `UPDATE` and `UPDATE t1,t2 ...` statements the number of rows matched and the number of rows actually changed is provided. This is because the number of rows actually matched by the corresponding `WHERE` clause, and the number of rows changed can differ. MySQL does not update the value of a row if the value already matches the new setting.

```
select-start(query)
select-done(status,rows)

insert-start(query)
insert-done(status,rows)

insert-select-start(query)
insert-select-done(status,rows)

update-start(query)
update-done(status,rowsmatched,rowschanged)
```

```
multi-update-start(query)
multi-update-done(status,rowsmatched,rowschanged)

delete-start(query)
delete-done(status,rows)

multi-delete-start(query)
multi-delete-done(status,rows)
```

- `select-start`: Triggered before a `SELECT` statement.
- `select-done`: Triggered at the end of a `SELECT` statement.
- `insert-start`: Triggered before a `INSERT` statement.
- `insert-done`: Triggered at the end of an `INSERT` statement.
- `insert-select-start`: Triggered before an `INSERT ... SELECT` statement.
- `insert-select-done`: Triggered at the end of an `INSERT ... SELECT` statement.
- `update-start`: Triggered before an `UPDATE` statement.
- `update-done`: Triggered at the end of an `UPDATE` statement.
- `multi-update-start`: Triggered before an `UPDATE` statement involving multiple tables.
- `multi-update-done`: Triggered at the end of an `UPDATE` statement involving multiple tables.
- `delete-start`: Triggered before a `DELETE` statement.
- `delete-done`: Triggered at the end of a `DELETE` statement.
- `multi-delete-start`: Triggered before a `DELETE` statement involving multiple tables.
- `multi-delete-done`: Triggered at the end of a `DELETE` statement involving multiple tables.

The arguments for the statement probes are:

- `query`: The query string.
- `status`: The status of the query. `0` for success, and `>0` for failure.
- `rows`: The number of rows affected by the statement. This returns the number rows found for `SELECT`, the number of rows deleted for `DELETE`, and the number of rows successfully inserted for `INSERT`.
- `rowsmatched`: The number of rows matched by the `WHERE` clause of an `UPDATE` operation.
- `rowschanged`: The number of rows actually changed during an `UPDATE` operation.

You use these probes to monitor the execution of these statement types without having to monitor the user or client executing the statements. A simple example of this is to track the execution times:

```
#!/usr/sbin/dtrace -s

#pragma D option quiet

dtrace:::BEGIN
{
   printf("%-60s %-8s %-8s %-8s\n", "Query", "RowsU", "RowsM", "Dur (ms)");
}

mysql*:::update-start, mysql*:::insert-start,
mysql*:::delete-start, mysql*:::multi-delete-start,
```

```
mysql*:::multi-delete-done, mysql*:::select-start,
mysql*:::insert-select-start, mysql*:::multi-update-start
{
    self->query = copyinstr(arg0);
    self->querystart = timestamp;
}

mysql*:::insert-done, mysql*:::select-done,
mysql*:::delete-done, mysql*:::multi-delete-done, mysql*:::insert-select-done
/ self->querystart /
{
    this->elapsed = ((timestamp - self->querystart)/1000000);
    printf("%-60s %-8d %-8d %d\n",
           self->query,
           0,
           arg1,
           this->elapsed);
    self->querystart = 0;
}

mysql*:::update-done, mysql*:::multi-update-done
/ self->querystart /
{
    this->elapsed = ((timestamp - self->querystart)/1000000);
    printf("%-60s %-8d %-8d %d\n",
           self->query,
           arg1,
           arg2,
           this->elapsed);
    self->querystart = 0;
}
```

When executed you can see the basic execution times and rows matches:

```
Query                                                       RowsU    RowsM    Dur (ms)
select * from t2                                            0        275      0
insert into t2 (select * from t2)                           0        275      9
update t2 set i=5 where i > 75                              110      110      8
update t2 set i=5 where i < 25                              254      134      12
delete from t2 where i < 5                                  0        0        0
```

Another alternative is to use the aggregation functions in DTrace to aggregate the execution time of individual statements together:

```
#!/usr/sbin/dtrace -s

#pragma D option quiet


mysql*:::update-start, mysql*:::insert-start,
mysql*:::delete-start, mysql*:::multi-delete-start,
mysql*:::multi-delete-done, mysql*:::select-start,
mysql*:::insert-select-start, mysql*:::multi-update-start
{
    self->querystart = timestamp;
}

mysql*:::select-done
{
        @statements["select"] = sum(((timestamp - self->querystart)/1000000));
}

mysql*:::insert-done, mysql*:::insert-select-done
{
        @statements["insert"] = sum(((timestamp - self->querystart)/1000000));
}

mysql*:::update-done, mysql*:::multi-update-done
{
        @statements["update"] = sum(((timestamp - self->querystart)/1000000));
}
```

```
mysql*:::delete-done, mysql*:::multi-delete-done
{
        @statements["delete"] = sum(((timestamp - self->querystart)/1000000));
}

tick-30s
{
        printa(@statements);
}
```

The script just shown aggregates the times spent doing each operation, which could be used to help benchmark a standard suite of tests.

```
 delete                                                    0
  update                                                   0
  insert                                                  23
  select                                                2484

  delete                                                   0
  update                                                   0
  insert                                                  39
  select                                               10744

  delete                                                   0
  update                                                  26
  insert                                                  56
  select                                               10944

  delete                                                   0
  update                                                  26
  insert                                                2287
  select                                               15985
```

Network Probes

The network probes monitor the transfer of information from the MySQL server and clients of all types over the network. The probes are defined as follows:

```
net-read-start()
net-read-done(status, bytes)
net-write-start(bytes)
net-write-done(status)
```

- `net-read-start`: Triggered when a network read operation is started.
- `net-read-done`: Triggered when the network read operation completes. The `status` is an `integer` representing the return status for the operation, `0` for success and `1` for failure. The `bytes` argument is an integer specifying the number of bytes read during the process.
- `net-start-bytes`: Triggered when data is written to a network socket. The single argument, `bytes`, specifies the number of bytes written to the network socket.
- `net-write-done`: Triggered when the network write operation has completed. The single argument, `status`, is an integer representing the return status for the operation, `0` for success and `1` for failure.

You can use the network probes to monitor the time spent reading from and writing to network clients during execution. The following D script provides an example of this. Both the cumulative time for the read or write is calculated, and the number of bytes. Note that the dynamic variable size has been increased (using the `dynvarsize` option) to cope with the rapid firing of the individual probes for the network reads/writes.

```
#!/usr/sbin/dtrace -s

#pragma D option quiet
#pragma D option dynvarsize=4m

dtrace:::BEGIN
```

```
{
   printf("%-2s %-30s %-10s %9s %18s %-s \n",
          "St", "Who", "DB", "ConnID", "Dur microsec", "Query");
}

mysql*:::query-start
{
   self->query = copyinstr(arg0);
   self->who   = strjoin(copyinstr(arg3),strjoin("@",copyinstr(arg4)));
   self->db    = copyinstr(arg2);
   self->connid = arg1;
   self->querystart = timestamp;
   self->netwrite = 0;
   self->netwritecum = 0;
   self->netwritebase = 0;
   self->netread = 0;
   self->netreadcum = 0;
   self->netreadbase = 0;
}

mysql*:::net-write-start
{
   self->netwrite += arg0;
   self->netwritebase = timestamp;
}

mysql*:::net-write-done
{
   self->netwritecum += (timestamp - self->netwritebase);
   self->netwritebase = 0;
}

mysql*:::net-read-start
{
   self->netreadbase = timestamp;
}

mysql*:::net-read-done
{
   self->netread += arg1;
   self->netreadcum += (timestamp - self->netreadbase);
   self->netreadbase = 0;
}

mysql*:::query-done
{
   this->elapsed = (timestamp - self->querystart) /1000000;
   printf("%2d %-30s %-10s %9d %18d %s\n",
          arg0, self->who, self->db,
          self->connid, this->elapsed, self->query);
   printf("Net read: %d bytes (%d ms) write: %d bytes (%d ms)\n",
               self->netread, (self->netreadcum/1000000),
               self->netwrite, (self->netwritecum/1000000));
}
```

When executing the above script on a machine with a remote client, you can see that approximately a third of the time spent executing the query is related to writing the query results back to the client.

```
St Who                            DB            ConnID       Dur microsec Query
 0 root@::ffff:198.51.100.108      test              31               3495 select * from t1 limit 1000000
Net read: 0 bytes (0 ms) write: 10000075 bytes (1220 ms)
```

Keycache Probes

The keycache probes are triggered when using the index key cache used with the MyISAM storage engine. Probes exist to monitor when data is read into the keycache, cached key data is written from the cache into a cached file, or when accessing the keycache.

Keycache usage indicates when data is read or written from the index files into the cache, and can be used to monitor how efficient the memory allocated to the keycache is being used. A high number of

keycache reads across a range of queries may indicate that the keycache is too small for size of data being accessed.

```
keycache-read-start(filepath, bytes, mem_used, mem_free)
keycache-read-block(bytes)
keycache-read-hit()
keycache-read-miss()
keycache-read-done(mem_used, mem_free)
keycache-write-start(filepath, bytes, mem_used, mem_free)
keycache-write-block(bytes)
keycache-write-done(mem_used, mem_free)
```

When reading data from the index files into the keycache, the process first initializes the read operation (indicated by `keycache-read-start`), then loads blocks of data (`keycache-read-block`), and then the read block is either matches the data being identified (`keycache-read-hit`) or more data needs to be read (`keycache-read-miss`). Once the read operation has completed, reading stops with the `keycache-read-done`.

Data is read from the index file into the keycache only when the specified key is not already within the keycache.

- `keycache-read-start`: Triggered when the keycache read operation is started. Data is read from the specified `filepath`, reading the specified number of `bytes`. The `mem_used` and `mem_avail` indicate memory currently used by the keycache and the amount of memory available within the keycache.
- `keycache-read-block`: Triggered when the keycache reads a block of data, of the specified number of `bytes`, from the index file into the keycache.
- `keycache-read-hit`: Triggered when the block of data read from the index file matches the key data requested.
- `keycache-read-miss`: Triggered when the block of data read from the index file does not match the key data needed.
- `keycache-read-done`: Triggered when the keycache read operation has completed. The `mem_used` and `mem_avail` indicate memory currently used by the keycache and the amount of memory available within the keycache.

Keycache writes occur when the index information is updated during an `INSERT`, `UPDATE`, or `DELETE` operation, and the cached key information is flushed back to the index file.

- `keycache-write-start`: Triggered when the keycache write operation is started. Data is written to the specified `filepath`, reading the specified number of `bytes`. The `mem_used` and `mem_avail` indicate memory currently used by the keycache and the amount of memory available within the keycache.
- `keycache-write-block`: Triggered when the keycache writes a block of data, of the specified number of `bytes`, to the index file from the keycache.
- `keycache-write-done`: Triggered when the keycache write operation has completed. The `mem_used` and `mem_avail` indicate memory currently used by the keycache and the amount of memory available within the keycache.

Chapter 6 Security

Table of Contents

6.1 General Security Issues 766
6.1.1 Security Guidelines 766
6.1.2 Keeping Passwords Secure 767
6.1.3 Making MySQL Secure Against Attackers 775
6.1.4 Security-Related mysqld Options and Variables 777
6.1.5 How to Run MySQL as a Normal User 778
6.1.6 Security Considerations for LOAD DATA LOCAL 778
6.1.7 Client Programming Security Guidelines 780
6.2 Access Control and Account Management 781
6.2.1 Account User Names and Passwords 782
6.2.2 Privileges Provided by MySQL 784
6.2.3 Grant Tables 790
6.2.4 Specifying Account Names 796
6.2.5 Access Control, Stage 1: Connection Verification 798
6.2.6 Access Control, Stage 2: Request Verification 801
6.2.7 Adding Accounts, Assigning Privileges, and Dropping Accounts 803
6.2.8 When Privilege Changes Take Effect 806
6.2.9 Assigning Account Passwords 806
6.2.10 Server Handling of Expired Passwords 808
6.2.11 Pluggable Authentication 810
6.2.12 Proxy Users 814
6.2.13 Setting Account Resource Limits 820
6.2.14 Troubleshooting Problems Connecting to MySQL 822
6.2.15 SQL-Based Account Activity Auditing 826
6.3 Using Encrypted Connections 828
6.3.1 Configuring MySQL to Use Encrypted Connections 829
6.3.2 Encrypted Connection TLS Protocols and Ciphers 831
6.3.3 Creating SSL and RSA Certificates and Keys 835
6.3.4 SSL Library-Dependent Capabilities 840
6.3.5 Connecting to MySQL Remotely from Windows with SSH 841
6.4 Security Plugins 842
6.4.1 Authentication Plugins 842
6.4.2 The Connection-Control Plugins 871
6.4.3 The Password Validation Plugin 877
6.4.4 MySQL Enterprise Audit 883
6.4.5 MySQL Enterprise Firewall 908
6.5 MySQL Enterprise Encryption 923
6.5.1 MySQL Enterprise Encryption Installation 923
6.5.2 MySQL Enterprise Encryption Usage and Examples 924
6.5.3 MySQL Enterprise Encryption User-Defined Function Reference 926
6.5.4 MySQL Enterprise Encryption User-Defined Function Descriptions 926
6.6 SELinux 930
6.6.1 Check if SELinux is Enabled 931
6.6.2 Changing the SELinux Mode 931
6.6.3 MySQL Server SELinux Policies 932
6.6.4 SELinux File Context 932
6.6.5 SELinux TCP Port Context 933
6.6.6 Troubleshooting SELinux 934

When thinking about security within a MySQL installation, you should consider a wide range of possible topics and how they affect the security of your MySQL server and related applications:

- General factors that affect security. These include choosing good passwords, not granting unnecessary privileges to users, ensuring application security by preventing SQL injections and data corruption, and others. See Section 6.1, “General Security Issues”.
- Security of the installation itself. The data files, log files, and the all the application files of your installation should be protected to ensure that they are not readable or writable by unauthorized parties. For more information, see Section 2.10, “Postinstallation Setup and Testing”.
- Access control and security within the database system itself, including the users and databases granted with access to the databases, views and stored programs in use within the database. For more information, see Section 6.2, “Access Control and Account Management”
- The features offered by security-related plugins. See Section 6.4, “Security Plugins”.
- Network security of MySQL and your system. The security is related to the grants for individual users, but you may also wish to restrict MySQL so that it is available only locally on the MySQL server host, or to a limited set of other hosts.
- Ensure that you have adequate and appropriate backups of your database files, configuration and log files. Also be sure that you have a recovery solution in place and test that you are able to successfully recover the information from your backups. See Chapter 7, *Backup and Recovery*.

6.1 General Security Issues

This section describes general security issues to be aware of and what you can do to make your MySQL installation more secure against attack or misuse. For information specifically about the access control system that MySQL uses for setting up user accounts and checking database access, see Section 2.10, “Postinstallation Setup and Testing”.

For answers to some questions that are often asked about MySQL Server security issues, see Section A.9, “MySQL 5.6 FAQ: Security”.

6.1.1 Security Guidelines

Anyone using MySQL on a computer connected to the Internet should read this section to avoid the most common security mistakes.

In discussing security, it is necessary to consider fully protecting the entire server host (not just the MySQL server) against all types of applicable attacks: eavesdropping, altering, playback, and denial of service. We do not cover all aspects of availability and fault tolerance here.

MySQL uses security based on Access Control Lists (ACLs) for all connections, queries, and other operations that users can attempt to perform. There is also support for SSL-encrypted connections between MySQL clients and servers. Many of the concepts discussed here are not specific to MySQL at all; the same general ideas apply to almost all applications.

When running MySQL, follow these guidelines:

- **Do not ever give anyone (except MySQL `root` accounts) access to the `user` table in the `mysql` system database!** This is critical.
- Learn how the MySQL access privilege system works (see Section 6.2, “Access Control and Account Management”). Use the `GRANT` and `REVOKE` statements to control access to MySQL. Do not grant more privileges than necessary. Never grant privileges to all hosts.

  Checklist:

  - Try `mysql -u root`. If you are able to connect successfully to the server without being asked for a password, anyone can connect to your MySQL server as the MySQL `root` user with full

privileges! Review the MySQL installation instructions, paying particular attention to the information about setting a `root` password. See Section 2.10.4, “Securing the Initial MySQL Accounts”.

- Use the `SHOW GRANTS` statement to check which accounts have access to what. Then use the `REVOKE` statement to remove those privileges that are not necessary.

- Do not store cleartext passwords in your database. If your computer becomes compromised, the intruder can take the full list of passwords and use them. Instead, use `SHA2()` or some other one-way hashing function and store the hash value.

  To prevent password recovery using rainbow tables, do not use these functions on a plain password; instead, choose some string to be used as a salt, and use hash(hash(password)+salt) values.

- Do not choose passwords from dictionaries. Special programs exist to break passwords. Even passwords like “xfish98” are very bad. Much better is “duag98” which contains the same word “fish” but typed one key to the left on a standard QWERTY keyboard. Another method is to use a password that is taken from the first characters of each word in a sentence (for example, “Four score and seven years ago” results in a password of “Fsasya”). The password is easy to remember and type, but difficult to guess for someone who does not know the sentence. In this case, you can additionally substitute digits for the number words to obtain the phrase “4 score and 7 years ago”, yielding the password “4sa7ya” which is even more difficult to guess.

- Invest in a firewall. This protects you from at least 50% of all types of exploits in any software. Put MySQL behind the firewall or in a demilitarized zone (DMZ).

  Checklist:

  - Try to scan your ports from the Internet using a tool such as `nmap`. MySQL uses port 3306 by default. This port should not be accessible from untrusted hosts. As a simple way to check whether your MySQL port is open, try the following command from some remote machine, where *`server_host`* is the host name or IP address of the host on which your MySQL server runs:

    ```
    shell> telnet server_host 3306
    ```

    If `telnet` hangs or the connection is refused, the port is blocked, which is how you want it to be. If you get a connection and some garbage characters, the port is open, and should be closed on your firewall or router, unless you really have a good reason to keep it open.

- Applications that access MySQL should not trust any data entered by users, and should be written using proper defensive programming techniques. See Section 6.1.7, “Client Programming Security Guidelines”.

- Do not transmit plain (unencrypted) data over the Internet. This information is accessible to everyone who has the time and ability to intercept it and use it for their own purposes. Instead, use an encrypted protocol such as SSL or SSH. MySQL supports internal SSL connections. Another technique is to use SSH port-forwarding to create an encrypted (and compressed) tunnel for the communication.

- Learn to use the `tcpdump` and `strings` utilities. In most cases, you can check whether MySQL data streams are unencrypted by issuing a command like the following:

  ```
  shell> tcpdump -l -i eth0 -w - src or dst port 3306 | strings
  ```

  This works under Linux and should work with small modifications under other systems.

Warning

If you do not see cleartext data, this does not always mean that the information actually is encrypted. If you need high security, consult with a security expert.

6.1.2 Keeping Passwords Secure

Passwords occur in several contexts within MySQL. The following sections provide guidelines that enable end users and administrators to keep these passwords secure and avoid exposing them. There is also a discussion of how MySQL uses password hashing internally and of a plugin that you can use to enforce stricter passwords.

6.1.2.1 End-User Guidelines for Password Security

MySQL users should use the following guidelines to keep passwords secure.

When you run a client program to connect to the MySQL server, it is inadvisable to specify your password in a way that exposes it to discovery by other users. The methods you can use to specify your password when you run client programs are listed here, along with an assessment of the risks of each method. In short, the safest methods are to have the client program prompt for the password or to specify the password in a properly protected option file.

- Use the `mysql_config_editor` utility, which enables you to store authentication credentials in an encrypted login path file named `.mylogin.cnf`. The file can be read later by MySQL client programs to obtain authentication credentials for connecting to MySQL Server. See Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”.

- Use a `--password=`*`password`* or `-p`*`password`* option on the command line. For example:

```
shell> mysql -u francis -pfrank db_name
```

Warning

This is convenient *but insecure*. On some systems, your password becomes visible to system status programs such as `ps` that may be invoked by other users to display command lines. MySQL clients typically overwrite the command-line password argument with zeros during their initialization sequence. However, there is still a brief interval during which the value is visible. Also, on some systems this overwriting strategy is ineffective and the password remains visible to `ps`. (SystemV Unix systems and perhaps others are subject to this problem.)

If your operating environment is set up to display your current command in the title bar of your terminal window, the password remains visible as long as the command is running, even if the command has scrolled out of view in the window content area.

- Use the `--password` or `-p` option on the command line with no password value specified. In this case, the client program solicits the password interactively:

```
shell> mysql -u francis -p db_name
Enter password: ********
```

The `*` characters indicate where you enter your password. The password is not displayed as you enter it.

It is more secure to enter your password this way than to specify it on the command line because it is not visible to other users. However, this method of entering a password is suitable only for programs that you run interactively. If you want to invoke a client from a script that runs noninteractively, there is no opportunity to enter the password from the keyboard. On some systems, you may even find that the first line of your script is read and interpreted (incorrectly) as your password.

- Store your password in an option file. For example, on Unix, you can list your password in the `[client]` section of the `.my.cnf` file in your home directory:

```
[client]
password=password
```

To keep the password safe, the file should not be accessible to anyone but yourself. To ensure this, set the file access mode to `400` or `600`. For example:

```
shell> chmod 600 .my.cnf
```

To name from the command line a specific option file containing the password, use the `--defaults-file=file_name` option, where `file_name` is the full path name to the file. For example:

```
shell> mysql --defaults-file=/home/francis/mysql-opts
```

Section 4.2.2.2, “Using Option Files”, discusses option files in more detail.

- Store your password in the `MYSQL_PWD` environment variable. See Section 4.9, “Environment Variables”.

  This method of specifying your MySQL password must be considered *extremely insecure* and should not be used. Some versions of `ps` include an option to display the environment of running processes. On some systems, if you set `MYSQL_PWD`, your password is exposed to any other user who runs `ps`. Even on systems without such a version of `ps`, it is unwise to assume that there are no other methods by which users can examine process environments.

On Unix, the `mysql` client writes a record of executed statements to a history file (see Section 4.5.1.3, “mysql Client Logging”). By default, this file is named `.mysql_history` and is created in your home directory. Passwords can be written as plain text in SQL statements such as `CREATE USER`, `GRANT`, and `SET PASSWORD`, so if you use these statements, they are logged in the history file. To keep this file safe, use a restrictive access mode, the same way as described earlier for the `.my.cnf` file.

If your command interpreter is configured to maintain a history, any file in which the commands are saved contains MySQL passwords entered on the command line. For example, `bash` uses `~/.bash_history`. Any such file should have a restrictive access mode.

6.1.2.2 Administrator Guidelines for Password Security

Database administrators should use the following guidelines to keep passwords secure.

MySQL stores passwords for user accounts in the `mysql.user` system table. Access to this table should never be granted to any nonadministrative accounts.

Account passwords can be expired so that users must reset them. See Section 6.2.10, “Server Handling of Expired Passwords”.

The `validate_password` plugin can be used to enforce a policy on acceptable password. See Section 6.4.3, “The Password Validation Plugin”.

A user who has access to modify the plugin directory (the value of the `plugin_dir` system variable) or the `my.cnf` file that specifies the plugin directory location can replace plugins and modify the capabilities provided by plugins, including authentication plugins.

Files such as log files to which passwords might be written should be protected. See Section 6.1.2.3, “Passwords and Logging”.

6.1.2.3 Passwords and Logging

Passwords can be written as plain text in SQL statements such as `CREATE USER`, `GRANT`, `SET PASSWORD`, and statements that invoke the `PASSWORD()` function. If such statements are logged by the MySQL server as written, passwords in them become visible to anyone with access to the logs.

Statement logging avoids writing passwords as cleartext for the following statements:

```
CREATE USER ... IDENTIFIED BY ...
GRANT ... IDENTIFIED BY ...
SET PASSWORD ...
SLAVE START ... PASSWORD = ...
CREATE SERVER ... OPTIONS(... PASSWORD ...)
```

```
ALTER SERVER ... OPTIONS(... PASSWORD ...)
```

Passwords in those statements are rewritten to not appear literally in statement text written to the general query log, slow query log, and binary log. Rewriting does not apply to other statements. In particular, `INSERT` or `UPDATE` statements for the `mysql.user` system table that refer to literal passwords are logged as is, so you should avoid such statements. (Direct modification of grant tables is discouraged, anyway.)

For the general query log, password rewriting can be suppressed by starting the server with the `--log-raw` option. For security reasons, this option is not recommended for production use. For diagnostic purposes, it may be useful to see the exact text of statements as received by the server.

Contents of the audit log file produced by the audit log plugin are not encrypted. For security reasons, this file should be written to a directory accessible only to the MySQL server and users with a legitimate reason to view the log. See Section 6.4.4.2, “MySQL Enterprise Audit Security Considerations”.

To guard log files against unwarranted exposure, locate them in a directory that restricts access to the server and the database administrator. If the server logs to tables in the `mysql` database, grant access to those tables only to the database administrator.

Replicas store the password for the replication source in the source info repository, which can be either a file or a table (see Section 17.2.2, “Relay Log and Replication Metadata Repositories”). Ensure that the repository can be accessed only by the database administrator. An alternative to storing the password in a file is to use the `START SLAVE` statement to specify credentials for connecting to the source.

Use a restricted access mode to protect database backups that include log tables or log files containing passwords.

6.1.2.4 Password Hashing in MySQL

Note

The information in this section applies only for accounts that use the `mysql_native_password` or `mysql_old_password` authentication plugins.

MySQL lists user accounts in the `user` table of the `mysql` database. Each MySQL account can be assigned a password, although the `user` table does not store the cleartext version of the password, but a hash value computed from it.

MySQL uses passwords in two phases of client/server communication:

- When a client attempts to connect to the server, there is an initial authentication step in which the client must present a password that has a hash value matching the hash value stored in the `user` table for the account the client wants to use.
- After the client connects, it can (if it has sufficient privileges) set or change the password hash for accounts listed in the `user` table. The client can do this by using the `PASSWORD()` function to generate a password hash, or by using a password-generating statement (`CREATE USER`, `GRANT`, or `SET PASSWORD`).

In other words, the server *checks* hash values during authentication when a client first attempts to connect. The server *generates* hash values if a connected client invokes the `PASSWORD()` function or uses a password-generating statement to set or change a password.

Password hashing methods in MySQL have the history described following. These changes are illustrated by changes in the result from the `PASSWORD()` function that computes password hash values and in the structure of the `user` table where passwords are stored.

The Original (Pre-4.1) Hashing Method

The original hashing method produced a 16-byte string. Such hashes look like this:

```
mysql> SELECT PASSWORD('mypass');
+--------------------+
| PASSWORD('mypass') |
+--------------------+
| 6f8c114b58f2ce9e   |
+--------------------+
```

To store account passwords, the `Password` column of the `user` table was at this point 16 bytes long.

The 4.1 Hashing Method

MySQL 4.1 introduced password hashing that provided better security and reduced the risk of passwords being intercepted. There were several aspects to this change:

- Different format of password values produced by the `PASSWORD()` function
- Widening of the `Password` column
- Control over the default hashing method
- Control over the permitted hashing methods for clients attempting to connect to the server

The changes in MySQL 4.1 took place in two stages:

- MySQL 4.1.0 used a preliminary version of the 4.1 hashing method. This method was short lived and the following discussion says nothing more about it.
- In MySQL 4.1.1, the hashing method was modified to produce a longer 41-byte hash value:

```
mysql> SELECT PASSWORD('mypass');
+-------------------------------------------+
| PASSWORD('mypass')                        |
+-------------------------------------------+
| *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 |
+-------------------------------------------+
```

The longer password hash format has better cryptographic properties, and client authentication based on long hashes is more secure than that based on the older short hashes.

To accommodate longer password hashes, the `Password` column in the `user` table was changed at this point to be 41 bytes, its current length.

A widened `Password` column can store password hashes in both the pre-4.1 and 4.1 formats. The format of any given hash value can be determined two ways:

- The length: 4.1 and pre-4.1 hashes are 41 and 16 bytes, respectively.
- Password hashes in the 4.1 format always begin with a `*` character, whereas passwords in the pre-4.1 format never do.

To permit explicit generation of pre-4.1 password hashes, two additional changes were made:

- The `OLD_PASSWORD()` function was added, which returns hash values in the 16-byte format.
- For compatibility purposes, the `old_passwords` system variable was added, to enable DBAs and applications control over the hashing method. The default `old_passwords` value of 0 causes hashing to use the 4.1 method (41-byte hash values), but setting `old_passwords=1` causes hashing to use the pre-4.1 method. In this case, `PASSWORD()` produces 16-byte values and is equivalent to `OLD_PASSWORD()`

To permit DBAs control over how clients are permitted to connect, the `secure_auth` system variable was added. Starting the server with this variable disabled or enabled permits or prohibits clients to connect using the older pre-4.1 password hashing method. Before MySQL 5.6.5, `secure_auth` is disabled by default. As of 5.6.5, `secure_auth` is enabled by default to

promote a more secure default configuration DBAs can disable it at their discretion, but this is not recommended, and pre-4.1 password hashes are deprecated and should be avoided. (For account upgrade instructions, see Section 6.4.1.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”.)

In addition, the `mysql` client supports a `--secure-auth` option that is analogous to `secure_auth`, but from the client side. It can be used to prevent connections to less secure accounts that use pre-4.1 password hashing. This option is disabled by default before MySQL 5.6.7, enabled thereafter.

Compatibility Issues Related to Hashing Methods

The widening of the `Password` column in MySQL 4.1 from 16 bytes to 41 bytes affects installation or upgrade operations as follows:

- If you perform a new installation of MySQL, the `Password` column is made 41 bytes long automatically.
- Upgrades from MySQL 4.1 or later to current versions of MySQL should not give rise to any issues in regard to the `Password` column because both versions use the same column length and password hashing method.
- For upgrades from a pre-4.1 release to 4.1 or later, you must upgrade the system tables after upgrading. (See Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”.)

The 4.1 hashing method is understood only by MySQL 4.1 (and higher) servers and clients, which can result in some compatibility problems. A 4.1 or higher client can connect to a pre-4.1 server, because the client understands both the pre-4.1 and 4.1 password hashing methods. However, a pre-4.1 client that attempts to connect to a 4.1 or higher server may run into difficulties. For example, a 4.0 `mysql` client may fail with the following error message:

```
shell> mysql -h localhost -u root
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

This phenomenon also occurs for attempts to use the older PHP `mysql` extension after upgrading to MySQL 4.1 or higher. (See Common Problems with MySQL and PHP.)

The following discussion describes the differences between the pre-4.1 and 4.1 hashing methods, and what you should do if you upgrade your server but need to maintain backward compatibility with pre-4.1 clients. (However, permitting connections by old clients is not recommended and should be avoided if possible.) Additional information can be found in Section B.3.2.4, “Client does not support authentication protocol”. This information is of particular importance to PHP programmers migrating MySQL databases from versions older than 4.1 to 4.1 or higher.

The differences between short and long password hashes are relevant both for how the server uses passwords during authentication and for how it generates password hashes for connected clients that perform password-changing operations.

The way in which the server uses password hashes during authentication is affected by the width of the `Password` column:

- If the column is short, only short-hash authentication is used.
- If the column is long, it can hold either short or long hashes, and the server can use either format:
  - Pre-4.1 clients can connect, but because they know only about the pre-4.1 hashing method, they can authenticate only using accounts that have short hashes.
  - 4.1 and later clients can authenticate using accounts that have short or long hashes.

Even for short-hash accounts, the authentication process is actually a bit more secure for 4.1 and later clients than for older clients. In terms of security, the gradient from least to most secure is:

- Pre-4.1 client authenticating with short password hash
- 4.1 or later client authenticating with short password hash
- 4.1 or later client authenticating with long password hash

The way in which the server generates password hashes for connected clients is affected by the width of the `Password` column and by the `old_passwords` system variable. A 4.1 or later server generates long hashes only if certain conditions are met: The `Password` column must be wide enough to hold long values and `old_passwords` must not be set to 1.

Those conditions apply as follows:

- The `Password` column must be wide enough to hold long hashes (41 bytes). If the column has not been updated and still has the pre-4.1 width of 16 bytes, the server notices that long hashes cannot fit into it and generates only short hashes when a client performs password-changing operations using the `PASSWORD()` function or a password-generating statement. This is the behavior that occurs if you have upgraded from a version of MySQL older than 4.1 to 4.1 or later but have not yet run the `mysql_upgrade` program to widen the `Password` column.
- If the `Password` column is wide, it can store either short or long password hashes. In this case, the `PASSWORD()` function and password-generating statements generate long hashes unless the server was started with the `old_passwords` system variable set to 1 to force the server to generate short password hashes instead.

The purpose of the `old_passwords` system variable is to permit backward compatibility with pre-4.1 clients under circumstances where the server would otherwise generate long password hashes. The option does not affect authentication (4.1 and later clients can still use accounts that have long password hashes), but it does prevent creation of a long password hash in the `user` table as the result of a password-changing operation. Were that permitted to occur, the account could no longer be used by pre-4.1 clients. With `old_passwords` disabled, the following undesirable scenario is possible:

- An old pre-4.1 client connects to an account that has a short password hash.
- The client changes its own password. With `old_passwords` disabled, this results in the account having a long password hash.
- The next time the old client attempts to connect to the account, it cannot, because the account has a long password hash that requires the 4.1 hashing method during authentication. (Once an account has a long password hash in the user table, only 4.1 and later clients can authenticate for it because pre-4.1 clients do not understand long hashes.)

This scenario illustrates that, if you must support older pre-4.1 clients, it is problematic to run a 4.1 or higher server without `old_passwords` set to 1. By running the server with `old_passwords=1`, password-changing operations do not generate long password hashes and thus do not cause accounts to become inaccessible to older clients. (Those clients cannot inadvertently lock themselves out by changing their password and ending up with a long password hash.)

The downside of `old_passwords=1` is that any passwords created or changed use short hashes, even for 4.1 or later clients. Thus, you lose the additional security provided by long password hashes. To create an account that has a long hash (for example, for use by 4.1 clients) or to change an existing account to use a long password hash, an administrator can set the session value of `old_passwords` set to 0 while leaving the global value set to 1:

```
mysql> SET @@SESSION.old_passwords = 0;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@SESSION.old_passwords, @@GLOBAL.old_passwords;
+--------------------------+------------------------+
| @@SESSION.old_passwords | @@GLOBAL.old_passwords |
+--------------------------+------------------------+
|                        0 |                      1 |
```

```
+------------------------+-----------------------+
1 row in set (0.00 sec)

mysql> CREATE USER 'newuser'@'localhost' IDENTIFIED BY 'newpass';
Query OK, 0 rows affected (0.03 sec)

mysql> SET PASSWORD FOR 'existinguser'@'localhost' = PASSWORD('existingpass');
Query OK, 0 rows affected (0.00 sec)
```

The following scenarios are possible in MySQL 4.1 or later. The factors are whether the `Password` column is short or long, and, if long, whether the server is started with `old_passwords` enabled or disabled.

Scenario 1: Short `Password` column in user table:

- Only short hashes can be stored in the `Password` column.
- The server uses only short hashes during client authentication.
- For connected clients, password hash-generating operations involving the `PASSWORD()` function or password-generating statements use short hashes exclusively. Any change to an account's password results in that account having a short password hash.
- The value of `old_passwords` is irrelevant because with a short `Password` column, the server generates only short password hashes anyway.

This scenario occurs when a pre-4.1 MySQL installation has been upgraded to 4.1 or later but `mysql_upgrade` has not been run to upgrade the system tables in the `mysql` database. (This is not a recommended configuration because it does not permit use of more secure 4.1 password hashing.)

Scenario 2: Long `Password` column; server started with `old_passwords=1`:

- Short or long hashes can be stored in the `Password` column.
- 4.1 and later clients can authenticate for accounts that have short or long hashes.
- Pre-4.1 clients can authenticate only for accounts that have short hashes.
- For connected clients, password hash-generating operations involving the `PASSWORD()` function or password-generating statements use short hashes exclusively. Any change to an account's password results in that account having a short password hash.

In this scenario, newly created accounts have short password hashes because `old_passwords=1` prevents generation of long hashes. Also, if you create an account with a long hash before setting `old_passwords` to 1, changing the account's password while `old_passwords=1` results in the account being given a short password, causing it to lose the security benefits of a longer hash.

To create a new account that has a long password hash, or to change the password of any existing account to use a long hash, first set the session value of `old_passwords` set to 0 while leaving the global value set to 1, as described previously.

In this scenario, the server has an up to date `Password` column, but is running with the default password hashing method set to generate pre-4.1 hash values. This is not a recommended configuration but may be useful during a transitional period in which pre-4.1 clients and passwords are upgraded to 4.1 or later. When that has been done, it is preferable to run the server with `old_passwords=0` and `secure_auth=1`.

Scenario 3: Long `Password` column; server started with `old_passwords=0`:

- Short or long hashes can be stored in the `Password` column.
- 4.1 and later clients can authenticate using accounts that have short or long hashes.
- Pre-4.1 clients can authenticate only using accounts that have short hashes.

- For connected clients, password hash-generating operations involving the `PASSWORD()` function or password-generating statements use long hashes exclusively. A change to an account's password results in that account having a long password hash.

As indicated earlier, a danger in this scenario is that it is possible for accounts that have a short password hash to become inaccessible to pre-4.1 clients. A change to such an account's password made using the `PASSWORD()` function or a password-generating statement results in the account being given a long password hash. From that point on, no pre-4.1 client can connect to the server using that account. The client must upgrade to 4.1 or later.

If this is a problem, you can change a password in a special way. For example, normally you use `SET PASSWORD` as follows to change an account password:

```
SET PASSWORD FOR 'some_user'@'some_host' = PASSWORD('password');
```

To change the password but create a short hash, use the `OLD_PASSWORD()` function instead:

```
SET PASSWORD FOR 'some_user'@'some_host' = OLD_PASSWORD('password');
```

`OLD_PASSWORD()` is useful for situations in which you explicitly want to generate a short hash.

The disadvantages for each of the preceding scenarios may be summarized as follows:

In scenario 1, you cannot take advantage of longer hashes that provide more secure authentication.

In scenario 2, `old_passwords=1` prevents accounts with short hashes from becoming inaccessible, but password-changing operations cause accounts with long hashes to revert to short hashes unless you take care to change the session value of `old_passwords` to 0 first.

In scenario 3, accounts with short hashes become inaccessible to pre-4.1 clients if you change their passwords without explicitly using `OLD_PASSWORD()`.

The best way to avoid compatibility problems related to short password hashes is to not use them:

- Upgrade all client programs to MySQL 4.1 or later.
- Run the server with `old_passwords=0`.
- Reset the password for any account with a short password hash to use a long password hash.
- For additional security, run the server with `secure_auth=1`.

6.1.2.5 Implications of Password Hashing Changes in MySQL 4.1 for Application Programs

An upgrade to MySQL version 4.1 or later can cause compatibility issues for applications that use `PASSWORD()` to generate passwords for their own purposes. Applications really should not do this, because `PASSWORD()` should be used only to manage passwords for MySQL accounts. But some applications use `PASSWORD()` for their own purposes anyway.

If you upgrade to 4.1 or later from a pre-4.1 version of MySQL and run the server under conditions where it generates long password hashes, an application using `PASSWORD()` for its own passwords breaks. The recommended course of action in such cases is to modify the application to use another function, such as `SHA2()`, `SHA1()`, or `MD5()`, to produce hashed values. If that is not possible, you can use the `OLD_PASSWORD()` function, which is provided for generate short hashes in the old format. However, you should note that `OLD_PASSWORD()` may one day no longer be supported.

If the server is running with `old_passwords=1`, it generates short hashes and `OLD_PASSWORD()` is equivalent to `PASSWORD()`.

PHP programmers migrating their MySQL databases from version 4.0 or lower to version 4.1 or higher should see MySQL and PHP.

6.1.3 Making MySQL Secure Against Attackers

When you connect to a MySQL server, you should use a password. The password is not transmitted as cleartext over the connection. Password handling during the client connection sequence was upgraded in MySQL 4.1.1 to be very secure. If you are still using pre-4.1.1-style passwords, the encryption algorithm is not as strong as the newer algorithm. With some effort, a clever attacker who can sniff the traffic between the client and the server can crack the password. (See Section 6.1.2.4, “Password Hashing in MySQL”, for a discussion of the different password handling methods.)

All other information is transferred as text, and can be read by anyone who is able to watch the connection. If the connection between the client and the server goes through an untrusted network, and you are concerned about this, you can use the compressed protocol to make traffic much more difficult to decipher. You can also use MySQL's internal SSL support to make the connection even more secure. See Section 6.3, “Using Encrypted Connections”. Alternatively, use SSH to get an encrypted TCP/IP connection between a MySQL server and a MySQL client. You can find an Open Source SSH client at http://www.openssh.org/, and a comparison of both Open Source and Commercial SSH clients at http://en.wikipedia.org/wiki/Comparison_of_SSH_clients.

To make a MySQL system secure, you should strongly consider the following suggestions:

- Require all MySQL accounts to have a password. A client program does not necessarily know the identity of the person running it. It is common for client/server applications that the user can specify any user name to the client program. For example, anyone can use the `mysql` program to connect as any other person simply by invoking it as `mysql -u other_user db_name` if *`other_user`* has no password. If all accounts have a password, connecting using another user's account becomes much more difficult.

  For a discussion of methods for setting passwords, see Section 6.2.9, “Assigning Account Passwords”.

- Make sure that the only Unix user account with read or write privileges in the database directories is the account that is used for running `mysqld`.

- Never run the MySQL server as the Unix `root` user. This is extremely dangerous, because any user with the `FILE` privilege is able to cause the server to create files as `root` (for example, `~root/.bashrc`). To prevent this, `mysqld` refuses to run as `root` unless that is specified explicitly using the `--user=root` option.

  `mysqld` can (and should) be run as an ordinary, unprivileged user instead. You can create a separate Unix account named `mysql` to make everything even more secure. Use this account only for administering MySQL. To start `mysqld` as a different Unix user, add a `user` option that specifies the user name in the `[mysqld]` group of the `my.cnf` option file where you specify server options. For example:

  ```
  [mysqld]
  user=mysql
  ```

  This causes the server to start as the designated user whether you start it manually or by using `mysqld_safe` or `mysql.server`. For more details, see Section 6.1.5, “How to Run MySQL as a Normal User”.

  Running `mysqld` as a Unix user other than `root` does not mean that you need to change the `root` user name in the `user` table. *User names for MySQL accounts have nothing to do with user names for Unix accounts*.

- Do not grant the `FILE` privilege to nonadministrative users. Any user that has this privilege can write a file anywhere in the file system with the privileges of the `mysqld` daemon. This includes the server's data directory containing the files that implement the privilege tables. To make `FILE`-privilege operations a bit safer, files generated with `SELECT ... INTO OUTFILE` do not overwrite existing files and are writable by everyone.

  The `FILE` privilege may also be used to read any file that is world-readable or accessible to the Unix user that the server runs as. With this privilege, you can read any file into a database table. This

could be abused, for example, by using `LOAD DATA` to load `/etc/passwd` into a table, which then can be displayed with `SELECT`.

To limit the location in which files can be read and written, set the `secure_file_priv` system to a specific directory. See Section 5.1.7, “Server System Variables”.

- Do not grant the `PROCESS` or `SUPER` privilege to nonadministrative users. The output of `mysqladmin processlist` and `SHOW PROCESSLIST` shows the text of any statements currently being executed, so any user who is permitted to see the server process list might be able to see statements issued by other users such as `UPDATE user SET password=PASSWORD('not_secure')`.

  `mysqld` reserves an extra connection for users who have the `SUPER` privilege, so that a MySQL `root` user can log in and check server activity even if all normal connections are in use.

  The `SUPER` privilege can be used to terminate client connections, change server operation by changing the value of system variables, and control replication servers.

- Do not permit the use of symlinks to tables. (This capability can be disabled with the `--skip-symbolic-links` option.) This is especially important if you run `mysqld` as `root`, because anyone that has write access to the server's data directory then could delete any file in the system! See Section 8.12.3.2, “Using Symbolic Links for MyISAM Tables on Unix”.

- Stored programs and views should be written using the security guidelines discussed in Section 20.6, “Stored Object Access Control”.

- If you do not trust your DNS, you should use IP addresses rather than host names in the grant tables. In any case, you should be very careful about creating grant table entries using host name values that contain wildcards.

- If you want to restrict the number of connections permitted to a single account, you can do so by setting the `max_user_connections` variable in `mysqld`. The `GRANT` statement also supports resource control options for limiting the extent of server use permitted to an account. See Section 13.7.1.4, “GRANT Statement”.

- If the plugin directory is writable by the server, it may be possible for a user to write executable code to a file in the directory using `SELECT ... INTO DUMPFILE`. This can be prevented by making `plugin_dir` read only to the server or by setting `secure_file_priv` to a directory where `SELECT` writes can be made safely.

6.1.4 Security-Related mysqld Options and Variables

The following table shows `mysqld` options and system variables that affect security. For descriptions of each of these, see Section 5.1.6, “Server Command Options”, and Section 5.1.7, “Server System Variables”.

Table 6.1 Security Option and Variable Summary

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
allow-suspicious-udfs	Yes	Yes				
automatic_sp_privileges	Yes	Yes	Yes		Global	Yes
chroot	Yes	Yes				
des-key-file	Yes	Yes				
local_infile	Yes	Yes	Yes		Global	Yes
old_passwords	Yes	Yes	Yes		Both	Yes
safe-user-create	Yes	Yes				
secure_auth	Yes	Yes	Yes		Global	Yes
secure_file_priv	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
skip-grant-tables	Yes	Yes				
skip_name_resolve	Yes	Yes	Yes		Global	No
skip_networking	Yes	Yes	Yes		Global	No
skip_show_database	Yes	Yes	Yes		Global	No

6.1.5 How to Run MySQL as a Normal User

On Windows, you can run the server as a Windows service using a normal user account.

On Linux, for installations performed using a MySQL repository, RPM packages, or Debian packages, the MySQL server `mysqld` should be started by the local `mysql` operating system user. Starting by another operating system user is not supported by the init scripts that are included as part of the installation.

On Unix (or Linux for installations performed using `tar` or `tar.gz` packages) , the MySQL server `mysqld` can be started and run by any user. However, you should avoid running the server as the Unix `root` user for security reasons. To change `mysqld` to run as a normal unprivileged Unix user *`user_name`*, you must do the following:

1. Stop the server if it is running (use `mysqladmin shutdown`).
2. Change the database directories and files so that *`user_name`* has privileges to read and write files in them (you might need to do this as the Unix `root` user):

   ```
   shell> chown -R user_name /path/to/mysql/datadir
   ```

   If you do not do this, the server cannot access databases or tables when it runs as *`user_name`*.

   If directories or files within the MySQL data directory are symbolic links, `chown -R` might not follow symbolic links for you. If it does not, you must also follow those links and change the directories and files they point to.
3. Start the server as user *`user_name`*. Another alternative is to start `mysqld` as the Unix `root` user and use the `--user=`*`user_name`* option. `mysqld` starts, then switches to run as the Unix user *`user_name`* before accepting any connections.
4. To start the server as the given user automatically at system startup time, specify the user name by adding a `user` option to the `[mysqld]` group of the `/etc/my.cnf` option file or the `my.cnf` option file in the server's data directory. For example:

   ```
   [mysqld]
   user=user_name
   ```

If your Unix machine itself is not secured, you should assign passwords to the MySQL `root` accounts in the grant tables. Otherwise, any user with a login account on that machine can run the `mysql` client with a `--user=root` option and perform any operation. (It is a good idea to assign passwords to MySQL accounts in any case, but especially so when other login accounts exist on the server host.) See Section 2.10.4, “Securing the Initial MySQL Accounts”.

6.1.6 Security Considerations for LOAD DATA LOCAL

The `LOAD DATA` statement loads a data file into a table. The statement can load a file located on the server host, or, if the `LOCAL` keyword is specified, on the client host.

The `LOCAL` version of `LOAD DATA` has two potential security issues:

- Because `LOAD DATA LOCAL` is an SQL statement, parsing occurs on the server side, and transfer of the file from the client host to the server host is initiated by the MySQL server, which tells the client the file named in the statement. In theory, a patched server could tell the client program to

transfer a file of the server's choosing rather than the file named in the statement. Such a server could access any file on the client host to which the client user has read access. (A patched server could in fact reply with a file-transfer request to any statement, not just `LOAD DATA LOCAL`, so a more fundamental issue is that clients should not connect to untrusted servers.)

- In a Web environment where the clients are connecting from a Web server, a user could use `LOAD DATA LOCAL` to read any files that the Web server process has read access to (assuming that a user could run any statement against the SQL server). In this environment, the client with respect to the MySQL server actually is the Web server, not a remote program being run by users who connect to the Web server.

To avoid connecting to untrusted servers, clients can establish a secure connection and verify the server identity by connecting using the `--ssl-verify-server-cert` option and the appropriate CA certificate.

To avoid `LOAD DATA` issues, clients should avoid using `LOCAL`.

Adminstrators and applications can configure whether to permit local data loading as follows:

- On the server side:
  - The `local_infile` system variable controls server-side `LOCAL` capability. Depending on the `local_infile` setting, the server refuses or permits local data loading by clients that request local data loading.
  - By default, `local_infile` is disabled. To explicitly cause the server to refuse or permit `LOAD DATA LOCAL` statements (regardless of how client programs and libraries are configured at build time or runtime), start `mysqld` with `local_infile` disabled or enabled. `local_infile` can also be set at runtime.
- On the client side:
  - The `ENABLED_LOCAL_INFILE` `CMake` option controls the compiled-in default `LOCAL` capability for the MySQL client library (see Section 2.9.7, “MySQL Source-Configuration Options”). Clients that make no explicit arrangements therefore have `LOCAL` capability disabled or enabled according to the `ENABLED_LOCAL_INFILE` setting specified at MySQL build time.
  - By default, the client library in MySQL binary distributions is compiled with `ENABLED_LOCAL_INFILE` enabled. If you compile MySQL from source, configure it with `ENABLED_LOCAL_INFILE` disabled or enabled based on whether clients that make no explicit arrangements should have `LOCAL` capability disabled or enabled.
  - For client programs that use the C API, local data loading capability is determined by the default compiled into the MySQL client library. To enable or disable it explicitly, invoke the `mysql_options()` C API function to disable or enable the `MYSQL_OPT_LOCAL_INFILE` option. See mysql_options().
  - For the `mysql` client, local data loading capability is determined by the default compiled into the MySQL client library. To disable or enable it explicitly, use the `--local-infile=0` or `--local-infile[=1]` option.
  - For the `mysqlimport` client, local data loading is not used by default. To disable or enable it explicitly, use the `--local=0` or `--local[=1]` option.
  - If you use `LOAD DATA LOCAL` in Perl scripts or other programs that read the `[client]` group from option files, you can add a `local-infile` option setting to that group. To prevent problems for programs that do not understand this option, specify it using the `loose-` prefix:

    ```
    [client]
    loose-local-infile=0
    ```

    or:

```
[client]
loose-local-infile=1
```

- In all cases, successful use of a `LOCAL` load operation by a client also requires that the server permits local loading.

If `LOCAL` capability is disabled, on either the server or client side, a client that attempts to issue a `LOAD DATA LOCAL` statement receives the following error message:

```
ERROR 1148: The used command is not allowed with this MySQL version
```

6.1.7 Client Programming Security Guidelines

Applications that access MySQL should not trust any data entered by users, who can try to trick your code by entering special or escaped character sequences in Web forms, URLs, or whatever application you have built. Be sure that your application remains secure if a user enters something like `; DROP DATABASE mysql;`. This is an extreme example, but large security leaks and data loss might occur as a result of hackers using similar techniques, if you do not prepare for them.

A common mistake is to protect only string data values. Remember to check numeric data as well. If an application generates a query such as `SELECT * FROM table WHERE ID=234` when a user enters the value `234`, the user can enter the value `234 OR 1=1` to cause the application to generate the query `SELECT * FROM table WHERE ID=234 OR 1=1`. As a result, the server retrieves every row in the table. This exposes every row and causes excessive server load. The simplest way to protect from this type of attack is to use single quotation marks around the numeric constants: `SELECT * FROM table WHERE ID='234'`. If the user enters extra information, it all becomes part of the string. In a numeric context, MySQL automatically converts this string to a number and strips any trailing nonnumeric characters from it.

Sometimes people think that if a database contains only publicly available data, it need not be protected. This is incorrect. Even if it is permissible to display any row in the database, you should still protect against denial of service attacks (for example, those that are based on the technique in the preceding paragraph that causes the server to waste resources). Otherwise, your server becomes unresponsive to legitimate users.

Checklist:

- Enable strict SQL mode to tell the server to be more restrictive of what data values it accepts. See Section 5.1.10, “Server SQL Modes”.
- Try to enter single and double quotation marks (`'` and `"`) in all of your Web forms. If you get any kind of MySQL error, investigate the problem right away.
- Try to modify dynamic URLs by adding `%22` (`"`), `%23` (`#`), and `%27` (`'`) to them.
- Try to modify data types in dynamic URLs from numeric to character types using the characters shown in the previous examples. Your application should be safe against these and similar attacks.
- Try to enter characters, spaces, and special symbols rather than numbers in numeric fields. Your application should remove them before passing them to MySQL or else generate an error. Passing unchecked values to MySQL is very dangerous!
- Check the size of data before passing it to MySQL.
- Have your application connect to the database using a user name different from the one you use for administrative purposes. Do not give your applications any access privileges they do not need.

Many application programming interfaces provide a means of escaping special characters in data values. Properly used, this prevents application users from entering values that cause the application to generate statements that have a different effect than you intend:

- MySQL C API: Use the `mysql_real_escape_string()` API call.
- MySQL++: Use the `escape` and `quote` modifiers for query streams.
- PHP: Use either the `mysqli` or `pdo_mysql` extensions, and not the older `ext/mysql` extension. The preferred API's support the improved MySQL authentication protocol and passwords, as well as prepared statements with placeholders. See also Choosing an API.

  If the older `ext/mysql` extension must be used, then for escaping use the `mysql_real_escape_string()` function and not `mysql_escape_string()` or `addslashes()` because only `mysql_real_escape_string()` is character set-aware; the other functions can be "bypassed" when using (invalid) multibyte character sets.
- Perl DBI: Use placeholders or the `quote()` method.
- Ruby DBI: Use placeholders or the `quote()` method.
- Java JDBC: Use a `PreparedStatement` object and placeholders.

Other programming interfaces might have similar capabilities.

6.2 Access Control and Account Management

MySQL enables the creation of accounts that permit client users to connect to the server and access data managed by the server. The primary function of the MySQL privilege system is to authenticate a user who connects from a given host and to associate that user with privileges on a database such as `SELECT`, `INSERT`, `UPDATE`, and `DELETE`. Additional functionality includes the ability to grant privileges for administrative operations.

To control which users can connect, each account can be assigned authentication credentials such as a password. The user interface to MySQL accounts consists of SQL statements such as `CREATE USER`, `GRANT`, and `REVOKE`. See Section 13.7.1, "Account Management Statements".

The MySQL privilege system ensures that all users may perform only the operations permitted to them. As a user, when you connect to a MySQL server, your identity is determined by *the host from which you connect* and *the user name you specify*. When you issue requests after connecting, the system grants privileges according to your identity and *what you want to do*.

MySQL considers both your host name and user name in identifying you because there is no reason to assume that a given user name belongs to the same person on all hosts. For example, the user `joe` who connects from `office.example.com` need not be the same person as the user `joe` who connects from `home.example.com`. MySQL handles this by enabling you to distinguish users on different hosts that happen to have the same name: You can grant one set of privileges for connections by `joe` from `office.example.com`, and a different set of privileges for connections by `joe` from `home.example.com`. To see what privileges a given account has, use the `SHOW GRANTS` statement. For example:

```
SHOW GRANTS FOR 'joe'@'office.example.com';
SHOW GRANTS FOR 'joe'@'home.example.com';
```

Internally, the server stores privilege information in the grant tables of the `mysql` system database. The MySQL server reads the contents of these tables into memory when it starts and bases access-control decisions on the in-memory copies of the grant tables.

MySQL access control involves two stages when you run a client program that connects to the server:

Stage 1: The server accepts or rejects the connection based on your identity and whether you can verify your identity by supplying the correct password.

Stage 2: Assuming that you can connect, the server checks each statement you issue to determine whether you have sufficient privileges to perform it. For example, if you try to select rows from a table

in a database or drop a table from the database, the server verifies that you have the `SELECT` privilege for the table or the `DROP` privilege for the database.

For a more detailed description of what happens during each stage, see Section 6.2.5, “Access Control, Stage 1: Connection Verification”, and Section 6.2.6, “Access Control, Stage 2: Request Verification”. For help in diagnosing privilege-related problems, see Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”.

If your privileges are changed (either by yourself or someone else) while you are connected, those changes do not necessarily take effect immediately for the next statement that you issue. For details about the conditions under which the server reloads the grant tables, see Section 6.2.8, “When Privilege Changes Take Effect”.

There are some things that you cannot do with the MySQL privilege system:

- You cannot explicitly specify that a given user should be denied access. That is, you cannot explicitly match a user and then refuse the connection.
- You cannot specify that a user has privileges to create or drop tables in a database but not to create or drop the database itself.
- A password applies globally to an account. You cannot associate a password with a specific object such as a database, table, or routine.

6.2.1 Account User Names and Passwords

MySQL stores accounts in the `user` table of the `mysql` system database. An account is defined in terms of a user name and the client host or hosts from which the user can connect to the server. For information about account representation in the `user` table, see Section 6.2.3, “Grant Tables”.

An account may also have authentication credentials such as a password. The credentials are handled by the account authentication plugin. MySQL supports multiple authentication plugins. Some of them use built-in authentication methods, whereas others enable authentication using external authentication methods. See Section 6.2.11, “Pluggable Authentication”.

There are several distinctions between the way user names and passwords are used by MySQL and your operating system:

- User names, as used by MySQL for authentication purposes, have nothing to do with user names (login names) as used by Windows or Unix. On Unix, most MySQL clients by default try to log in using the current Unix user name as the MySQL user name, but that is for convenience only. The default can be overridden easily, because client programs permit any user name to be specified with a `-u` or `--user` option. This means that anyone can attempt to connect to the server using any user name, so you cannot make a database secure in any way unless all MySQL accounts have passwords. Anyone who specifies a user name for an account that has no password can connect successfully to the server.
- MySQL user names are up to 16 characters long. Operating system user names may have a different maximum length.

Warning

The MySQL user name length limit is hardcoded in MySQL servers and clients, and trying to circumvent it by modifying the definitions of the tables in the `mysql` database *does not work*.

You should never alter the structure of tables in the `mysql` database in any manner whatsoever except by means of the procedure that is described in Section 2.11, “Upgrading MySQL”. Attempting to redefine MySQL's system tables in any other fashion results in undefined and unsupported behavior.

> The server is free to ignore rows that become malformed as a result of such modifications.

- To authenticate client connections for accounts that use built-in authentication methods, the server uses passwords stored in the `user` table. These passwords are distinct from passwords for logging in to your operating system. There is no necessary connection between the “external” password you use to log in to a Windows or Unix machine and the password you use to access the MySQL server on that machine.

  If the server authenticates a client using some other plugin, the authentication method that the plugin implements may or may not use a password stored in the `user` table. In this case, it is possible that an external password is also used to authenticate to the MySQL server.

- Passwords stored in the `user` table are encrypted using plugin-specific algorithms. For information about MySQL native password hashing, see Section 6.1.2.4, “Password Hashing in MySQL”.

- If the user name and password contain only ASCII characters, it is possible to connect to the server regardless of character set settings. To enable connections when the user name or password contain non-ASCII characters, client applications should call the `mysql_options()` C API function with the `MYSQL_SET_CHARSET_NAME` option and appropriate character set name as arguments. This causes authentication to take place using the specified character set. Otherwise, authentication fails unless the server default character set is the same as the encoding in the authentication defaults.

  Standard MySQL client programs support a `--default-character-set` option that causes `mysql_options()` to be called as just described. In addition, character set autodetection is supported as described in Section 10.4, “Connection Character Sets and Collations”. For programs that use a connector that is not based on the C API, the connector may provide an equivalent to `mysql_options()` that can be used instead. Check the connector documentation.

  The preceding notes do not apply for `ucs2`, `utf16`, and `utf32`, which are not permitted as client character sets.

The MySQL installation process populates the grant tables with an initial account or accounts. The names and access privileges for these accounts are described in Section 2.10.4, “Securing the Initial MySQL Accounts”, which also discusses how to assign passwords to them. Thereafter, you normally set up, modify, and remove MySQL accounts using statements such as `CREATE USER`, `DROP USER`, `GRANT`, and `REVOKE`. See Section 6.2.7, “Adding Accounts, Assigning Privileges, and Dropping Accounts”, and Section 13.7.1, “Account Management Statements”.

To connect to a MySQL server with a command-line client, specify user name and password options as necessary for the account that you want to use:

```
shell> mysql --user=finley --password db_name
```

If you prefer short options, the command looks like this:

```
shell> mysql -u finley -p db_name
```

If you omit the password value following the `--password` or `-p` option on the command line (as just shown), the client prompts for one. Alternatively, the password can be specified on the command line:

```
shell> mysql --user=finley --password=password db_name
shell> mysql -u finley -ppassword db_name
```

If you use the `-p` option, there must be *no space* between `-p` and the following password value.

Specifying a password on the command line should be considered insecure. See Section 6.1.2.1, “End-User Guidelines for Password Security”. To avoid giving the password on the command line, use an option file or a login path file. See Section 4.2.2.2, “Using Option Files”, and Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”.

For additional information about specifying user names, passwords, and other connection parameters, see Section 4.2.4, “Connecting to the MySQL Server Using Command Options”.

6.2.2 Privileges Provided by MySQL

The privileges granted to a MySQL account determine which operations the account can perform. MySQL privileges differ in the contexts in which they apply and at different levels of operation:

- Administrative privileges enable users to manage operation of the MySQL server. These privileges are global because they are not specific to a particular database.
- Database privileges apply to a database and to all objects within it. These privileges can be granted for specific databases, or globally so that they apply to all databases.
- Privileges for database objects such as tables, indexes, views, and stored routines can be granted for specific objects within a database, for all objects of a given type within a database (for example, all tables in a database), or globally for all objects of a given type in all databases.

Information about account privileges is stored in the grant tables in the `mysql` system database. For a description of the structure and contents of these tables, see Section 6.2.3, “Grant Tables”. The MySQL server reads the contents of the grant tables into memory when it starts, and reloads them under the circumstances indicated in Section 6.2.8, “When Privilege Changes Take Effect”. The server bases access-control decisions on the in-memory copies of the grant tables.

Important

Some MySQL releases introduce changes to the grant tables to add new privileges or features. To make sure that you can take advantage of any new capabilities, update your grant tables to the current structure whenever you upgrade MySQL. See Section 2.11, “Upgrading MySQL”.

The following sections summarize the available privileges, provide more detailed descriptions of each privilege, and offer usage guidelines.

- Summary of Available Privileges
- Privilege Descriptions
- Privilege-Granting Guidelines

Summary of Available Privileges

The following table shows the privilege names used in `GRANT` and `REVOKE` statements, along with the column name associated with each privilege in the grant tables and the context in which the privilege applies.

Table 6.2 Permissible Privileges for GRANT and REVOKE

Privilege	Grant Table Column	Context
`ALL [PRIVILEGES]`	Synonym for “all privileges”	Server administration
`ALTER`	`Alter_priv`	Tables
`ALTER ROUTINE`	`Alter_routine_priv`	Stored routines
`CREATE`	`Create_priv`	Databases, tables, or indexes
`CREATE ROUTINE`	`Create_routine_priv`	Stored routines
`CREATE TABLESPACE`	`Create_tablespace_priv`	Server administration
`CREATE TEMPORARY TABLES`	`Create_tmp_table_priv`	Tables

Privilege	Grant Table Column	Context
CREATE USER	Create_user_priv	Server administration
CREATE VIEW	Create_view_priv	Views
DELETE	Delete_priv	Tables
DROP	Drop_priv	Databases, tables, or views
EVENT	Event_priv	Databases
EXECUTE	Execute_priv	Stored routines
FILE	File_priv	File access on server host
GRANT OPTION	Grant_priv	Databases, tables, or stored routines
INDEX	Index_priv	Tables
INSERT	Insert_priv	Tables or columns
LOCK TABLES	Lock_tables_priv	Databases
PROCESS	Process_priv	Server administration
PROXY	See proxies_priv table	Server administration
REFERENCES	References_priv	Databases or tables
RELOAD	Reload_priv	Server administration
REPLICATION CLIENT	Repl_client_priv	Server administration
REPLICATION SLAVE	Repl_slave_priv	Server administration
SELECT	Select_priv	Tables or columns
SHOW DATABASES	Show_db_priv	Server administration
SHOW VIEW	Show_view_priv	Views
SHUTDOWN	Shutdown_priv	Server administration
SUPER	Super_priv	Server administration
TRIGGER	Trigger_priv	Tables
UPDATE	Update_priv	Tables or columns
USAGE	Synonym for “no privileges”	Server administration

Privilege Descriptions

The following list provides general descriptions of each privilege available in MySQL. Particular SQL statements might have more specific privilege requirements than indicated here. If so, the description for the statement in question provides the details.

- ALL, ALL PRIVILEGES

  These privilege specifiers are shorthand for “all privileges available at a given privilege level” (except GRANT OPTION). For example, granting ALL at the global or table level grants all global privileges or all table-level privileges, respectively.

- ALTER

  Enables use of the ALTER TABLE statement to change the structure of tables. ALTER TABLE also requires the CREATE and INSERT privileges. Renaming a table requires ALTER and DROP on the old table, CREATE, and INSERT on the new table.

- ALTER ROUTINE

  Enables use of statements that alter or drop stored routines (stored procedures and functions).

- CREATE

Enables use of statements that create new databases and tables.

- `CREATE ROUTINE`

  Enables use of statements that create stored routines (stored procedures and functions).

- `CREATE TABLESPACE`

  Enables use of statements that create, alter, or drop tablespaces and log file groups.

- `CREATE TEMPORARY TABLES`

  Enables the creation of temporary tables using the `CREATE TEMPORARY TABLE` statement.

  After a session has created a temporary table, the server performs no further privilege checks on the table. The creating session can perform any operation on the table, such as `DROP TABLE`, `INSERT`, `UPDATE`, or `SELECT`. For more information, see Section 13.1.17.2, “CREATE TEMPORARY TABLE Statement”.

- `CREATE USER`

  Enables use of the `ALTER USER`, `CREATE USER`, `DROP USER`, `RENAME USER`, and `REVOKE ALL PRIVILEGES` statements.

- `CREATE VIEW`

  Enables use of the `CREATE VIEW` statement.

- `DELETE`

  Enables rows to be deleted from tables in a database.

- `DROP`

  Enables use of statements that drop (remove) existing databases, tables, and views. The `DROP` privilege is required to use the `ALTER TABLE ... DROP PARTITION` statement on a partitioned table. The `DROP` privilege is also required for `TRUNCATE TABLE`.

- `EVENT`

  Enables use of statements that create, alter, drop, or display events for the Event Scheduler.

- `EXECUTE`

  Enables use of statements that execute stored routines (stored procedures and functions).

- `FILE`

  Affects the following operations and server behaviors:

  - Enables reading and writing files on the server host using the `LOAD DATA` and `SELECT ... INTO OUTFILE` statements and the `LOAD_FILE()` function. A user who has the `FILE` privilege can read any file on the server host that is either world-readable or readable by the MySQL server. (This implies the user can read any file in any database directory, because the server can access any of those files.)
  - Enables creating new files in any directory where the MySQL server has write access. This includes the server's data directory containing the files that implement the privilege tables.
  - As of MySQL 5.6.35, enables use of the `DATA DIRECTORY` or `INDEX DIRECTORY` table option for the `CREATE TABLE` statement.

  As a security measure, the server does not overwrite existing files.

To limit the location in which files can be read and written, set the `secure_file_priv` system variable to a specific directory. See Section 5.1.7, “Server System Variables”.

- `GRANT OPTION`

  Enables you to grant to or revoke from other users those privileges that you yourself possess.

- `INDEX`

  Enables use of statements that create or drop (remove) indexes. `INDEX` applies to existing tables. If you have the `CREATE` privilege for a table, you can include index definitions in the `CREATE TABLE` statement.

- `INSERT`

  Enables rows to be inserted into tables in a database. `INSERT` is also required for the `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` table-maintenance statements.

- `LOCK TABLES`

  Enables use of explicit `LOCK TABLES` statements to lock tables for which you have the `SELECT` privilege. This includes use of write locks, which prevents other sessions from reading the locked table.

- `PROCESS`

  The `PROCESS` privilege controls access to information about threads executing within the server (that is, information about statements being executed by sessions). Thread information available using the `SHOW PROCESSLIST` statement, the `mysqladmin processlist` command, and the `INFORMATION_SCHEMA.PROCESSLIST` table is accessible as follows:

  - With the `PROCESS` privilege, a user has access to information about all threads, even those belonging to other users.
  - Without the `PROCESS` privilege, nonanonymous users have access to information about their own threads but not threads for other users, and anonymous users have no access to thread information.

  Note

  The Performance Schema `threads` table also provides thread information, but table access uses a different privilege model. See Section 22.12.10.3, “The threads Table”.

  The `PROCESS` privilege also enables use of the `SHOW ENGINE` statement, access to the `INFORMATION_SCHEMA InnoDB` tables (tables with names that begin with `INNODB_`), and (as of MySQL 5.6.49) access to the `INFORMATION_SCHEMA FILES` table.

- `PROXY`

  Enables one user to impersonate or become known as another user. See Section 6.2.12, “Proxy Users”.

- `REFERENCES`

  This privilege is unused before MySQL 5.6.22. As of 5.6.22, creation of a foreign key constraint requires at least one of the `SELECT`, `INSERT`, `UPDATE`, `DELETE`, or `REFERENCES` privileges for the parent table.

- `RELOAD`

  The `RELOAD` enables the following operations:

  - Use of the `FLUSH` statement.
  - Use of `mysqladmin` commands that are equivalent to `FLUSH` operations: `flush-hosts`, `flush-logs`, `flush-privileges`, `flush-status`, `flush-tables`, `flush-threads`, `refresh`, and `reload`.

    The `reload` command tells the server to reload the grant tables into memory. `flush-privileges` is a synonym for `reload`. The `refresh` command closes and reopens the log files and flushes all tables. The other `flush-xxx` commands perform functions similar to `refresh`, but are more specific and may be preferable in some instances. For example, if you want to flush just the log files, `flush-logs` is a better choice than `refresh`.
  - Use of `mysqldump` options that perform various `FLUSH` operations: `--flush-logs` and `--master-data`.
  - Use of the `RESET` statement.

- `REPLICATION CLIENT`

  Enables use of the `SHOW MASTER STATUS`, `SHOW SLAVE STATUS`, and `SHOW BINARY LOGS` statements. Grant this privilege to accounts that are used by replica servers to connect to the current server as their source.

- `REPLICATION SLAVE`

  Enables the account to request updates that have been made to databases on the source server, using the `SHOW SLAVE HOSTS`, `SHOW RELAYLOG EVENTS`, and `SHOW BINLOG EVENTS` statements. This privilege is also required to use the `mysqlbinlog` options `--read-from-remote-server` (`-R`) and `--read-from-remote-master`. Grant this privilege to accounts that are used by replica servers to connect to the current server as their source.

- `SELECT`

  Enables rows to be selected from tables in a database. `SELECT` statements require the `SELECT` privilege only if they actually access tables. Some `SELECT` statements do not access tables and can be executed without permission for any database. For example, you can use `SELECT` as a simple calculator to evaluate expressions that make no reference to tables:

  ```
  SELECT 1+1;
  SELECT PI()*2;
  ```

  The `SELECT` privilege is also needed for other statements that read column values. For example, `SELECT` is needed for columns referenced on the right hand side of *`col_name`*=*`expr`* assignment in `UPDATE` statements or for columns named in the `WHERE` clause of `DELETE` or `UPDATE` statements.

  The `SELECT` privilege is needed for tables or views used with `EXPLAIN`, including any underlying tables in view definitions.

- `SHOW DATABASES`

  Enables the account to see database names by issuing the `SHOW DATABASE` statement. Accounts that do not have this privilege see only databases for which they have some privileges, and cannot use the statement at all if the server was started with the `--skip-show-database` option.

  Caution

  Because a global privilege is considered a privilege for all databases, *any* global privilege enables a user to see all database names with `SHOW DATABASES` or by examining the `INFORMATION_SCHEMA SCHEMATA` table.

- `SHOW VIEW`

Enables use of the `SHOW CREATE VIEW` statement. This privilege is also needed for views used with `EXPLAIN`.

- `SHUTDOWN`

  Enables use of the `mysqladmin shutdown` command and the `mysql_shutdown()` C API function. There is no corresponding SQL statement.

- `SUPER`

  Affects the following operations and server behaviors:

  - Enables server configuration changes by modifying global system variables. For some system variables, setting the session value also requires the `SUPER` privilege. If a system variable is restricted and requires a special privilege to set the session value, the variable description indicates that restriction. Examples include `binlog_format`, `sql_log_bin`, and `sql_log_off`. See also Section 5.1.8.1, “System Variable Privileges”.
  - Enables changes to global transaction characteristics (see Section 13.3.6, “SET TRANSACTION Statement”).
  - Enables the account to start and stop replication.
  - Enables use of the `CHANGE MASTER TO` statement.
  - Enables binary log control by means of the `PURGE BINARY LOGS` and `BINLOG` statements.
  - Enables setting the effective authorization ID when executing a view or stored program. A user with this privilege can specify any account in the `DEFINER` attribute of a view or stored program.
  - Enables use of the `CREATE SERVER`, `ALTER SERVER`, and `DROP SERVER` statements.
  - Enables use of the `mysqladmin debug` command.
  - Enables reading the DES key file by the `DES_ENCRYPT()` function.
  - Enables control over client connections not permitted to non-`SUPER` accounts:
    - Enables use of the `KILL` statement or `mysqladmin kill` command to kill threads belonging to other accounts. (An account can always kill its own threads.)
    - The server does not execute `init_connect` system variable content when `SUPER` clients connect.
    - The server accepts one connection from a `SUPER` client even if the connection limit configured by the `max_connections` system variable is reached.
    - Updates can be performed even when the `read_only` system variable is enabled. This applies to explicit table updates, and to use of account-management statements such as `GRANT` and `REVOKE` that update tables implicitly.

  You may also need the `SUPER` privilege to create or alter stored functions if binary logging is enabled, as described in Section 20.7, “Stored Program Binary Logging”.

- `TRIGGER`

  Enables trigger operations. You must have this privilege for a table to create, drop, execute, or display triggers for that table.

  When a trigger is activated (by a user who has privileges to execute `INSERT`, `UPDATE`, or `DELETE` statements for the table associated with the trigger), trigger execution requires that the user who defined the trigger still have the `TRIGGER` privilege for the table.

- `UPDATE`

  Enables rows to be updated in tables in a database.

- `USAGE`

  This privilege specifier stands for “no privileges.” It is used at the global level with `GRANT` to modify account attributes such as resource limits or SSL characteristics without naming specific account privileges in the privilege list. `SHOW GRANTS` displays `USAGE` to indicate that an account has no privileges at a privilege level.

Privilege-Granting Guidelines

It is a good idea to grant to an account only those privileges that it needs. You should exercise particular caution in granting the `FILE` and administrative privileges:

- `FILE` can be abused to read into a database table any files that the MySQL server can read on the server host. This includes all world-readable files and files in the server's data directory. The table can then be accessed using `SELECT` to transfer its contents to the client host.
- `GRANT OPTION` enables users to give their privileges to other users. Two users that have different privileges and with the `GRANT OPTION` privilege are able to combine privileges.
- `ALTER` may be used to subvert the privilege system by renaming tables.
- `SHUTDOWN` can be abused to deny service to other users entirely by terminating the server.
- `PROCESS` can be used to view the plain text of currently executing statements, including statements that set or change passwords.
- `SUPER` can be used to terminate other sessions or change how the server operates.
- Privileges granted for the `mysql` system database itself can be used to change passwords and other access privilege information:
  - Passwords are stored encrypted, so a malicious user cannot simply read them to know the plain text password. However, a user with write access to the `mysql.user` system table `Password` column can change an account's password, and then connect to the MySQL server using that account.
  - `INSERT` or `UPDATE` granted for the `mysql` system database enable a user to add privileges or modify existing privileges, respectively.
  - `DROP` for the `mysql` system database enables a user to remote privilege tables, or even the database itself.

6.2.3 Grant Tables

The `mysql` system database includes several grant tables that contain information about user accounts and the privileges held by them. This section describes those tables. For information about other tables in the system database, see Section 5.3, “The mysql System Database”.

The discussion here describes the underlying structure of the grant tables and how the server uses their contents when interacting with clients. However, normally you do not modify the grant tables

directly. Modifications occur indirectly when you use account-management statements such as `CREATE USER`, `GRANT`, and `REVOKE` to set up accounts and control the privileges available to each one. See Section 13.7.1, “Account Management Statements”. When you use such statements to perform account manipulations, the server modifies the grant tables on your behalf.

Note

Direct modification of grant tables using statements such as `INSERT`, `UPDATE`, or `DELETE` is discouraged and done at your own risk. The server is free to ignore rows that become malformed as a result of such modifications.

As of MySQL 5.6.36, for any operation that modifies a grant table, the server checks whether the table has the expected structure and produces an error if not. `mysql_upgrade` must be run to update the tables to the expected structure.

- Grant Table Overview
- The user and db Grant Tables
- The tables_priv and columns_priv Grant Tables
- The procs_priv Grant Table
- The proxies_priv Grant Table
- Grant Table Scope Column Properties
- Grant Table Privilege Column Properties

Grant Table Overview

These `mysql` database tables contain grant information:

- `user`: User accounts, global privileges, and other nonprivilege columns.
- `db`: Database-level privileges.
- `tables_priv`: Table-level privileges.
- `columns_priv`: Column-level privileges.
- `procs_priv`: Stored procedure and function privileges.
- `proxies_priv`: Proxy-user privileges.

Each grant table contains scope columns and privilege columns:

- Scope columns determine the scope of each row in the tables; that is, the context in which the row applies. For example, a `user` table row with `Host` and `User` values of `'h1.example.net'` and `'bob'` applies to authenticating connections made to the server from the host `h1.example.net` by a client that specifies a user name of `bob`. Similarly, a `db` table row with `Host`, `User`, and `Db` column values of `'h1.example.net'`, `'bob'` and `'reports'` applies when `bob` connects from the host `h1.example.net` to access the `reports` database. The `tables_priv` and `columns_priv` tables contain scope columns indicating tables or table/column combinations to which each row applies. The `procs_priv` scope columns indicate the stored routine to which each row applies.
- Privilege columns indicate which privileges a table row grants; that is, which operations it permits to be performed. The server combines the information in the various grant tables to form a complete description of a user's privileges. Section 6.2.6, “Access Control, Stage 2: Request Verification”, describes the rules for this.

In addition, a grant table may contain columns used for purposes other than scope or privilege assessment.

The server uses the grant tables in the following manner:

- The `user` table scope columns determine whether to reject or permit incoming connections. For permitted connections, any privileges granted in the `user` table indicate the user's global privileges. Any privileges granted in this table apply to *all* databases on the server.

  Caution

  Because a global privilege is considered a privilege for all databases, *any* global privilege enables a user to see all database names with `SHOW DATABASES` or by examining the `INFORMATION_SCHEMA SCHEMATA` table.

- The `db` table scope columns determine which users can access which databases from which hosts. The privilege columns determine the permitted operations. A privilege granted at the database level applies to the database and to all objects in the database, such as tables and stored programs.

- The `tables_priv` and `columns_priv` tables are similar to the `db` table, but are more fine-grained: They apply at the table and column levels rather than at the database level. A privilege granted at the table level applies to the table and to all its columns. A privilege granted at the column level applies only to a specific column.

- The `procs_priv` table applies to stored routines (stored procedures and functions). A privilege granted at the routine level applies only to a single procedure or function.

- The `proxies_priv` table indicates which users can act as proxies for other users and whether a user can grant the `PROXY` privilege to other users.

The server reads the contents of the grant tables into memory when it starts. You can tell it to reload the tables by issuing a `FLUSH PRIVILEGES` statement or executing a `mysqladmin flush-privileges` or `mysqladmin reload` command. Changes to the grant tables take effect as indicated in Section 6.2.8, “When Privilege Changes Take Effect”.

When you modify an account, it is a good idea to verify that your changes have the intended effect. To check the privileges for a given account, use the `SHOW GRANTS` statement. For example, to determine the privileges that are granted to an account with user name and host name values of `bob` and `pc84.example.com`, use this statement:

```
SHOW GRANTS FOR 'bob'@'pc84.example.com';
```

The user and db Grant Tables

The server uses the `user` and `db` tables in the `mysql` database at both the first and second stages of access control (see Section 6.2, “Access Control and Account Management”). The columns in the `user` and `db` tables are shown here.

Table 6.3 user and db Table Columns

Table Name	`user`	`db`
Scope columns	`Host`	`Host`
	`User`	`Db`
	`Password`	`User`
Privilege columns	`Select_priv`	`Select_priv`
	`Insert_priv`	`Insert_priv`
	`Update_priv`	`Update_priv`
	`Delete_priv`	`Delete_priv`
	`Index_priv`	`Index_priv`
	`Alter_priv`	`Alter_priv`
	`Create_priv`	`Create_priv`

Table Name	user	db
	Drop_priv	Drop_priv
	Grant_priv	Grant_priv
	Create_view_priv	Create_view_priv
	Show_view_priv	Show_view_priv
	Create_routine_priv	Create_routine_priv
	Alter_routine_priv	Alter_routine_priv
	Execute_priv	Execute_priv
	Trigger_priv	Trigger_priv
	Event_priv	Event_priv
	Create_tmp_table_priv	Create_tmp_table_priv
	Lock_tables_priv	Lock_tables_priv
	References_priv	References_priv
	Reload_priv	
	Shutdown_priv	
	Process_priv	
	File_priv	
	Show_db_priv	
	Super_priv	
	Repl_slave_priv	
	Repl_client_priv	
	Create_user_priv	
	Create_tablespace_priv	
Security columns	ssl_type	
	ssl_cipher	
	x509_issuer	
	x509_subject	
	plugin	
	authentication_string	
	password_expired	
Resource control columns	max_questions	
	max_updates	
	max_connections	
	max_user_connections	

The user table plugin, Password, and authentication_string columns store authentication plugin and credential information.

If an account row names a plugin in the plugin column, the server uses it to authenticate connection attempts for the account. It is up to the plugin whether it uses the Password and authentication_string column values.

If the plugin column for an account row is empty, the server authenticates the account using the mysql_native_password or mysql_old_password plugin implicitly, depending on the format of the password hash in the Password column. If the Password value is empty or a 4.1 password hash (41 characters), the server uses mysql_native_password. If the password value is a pre-4.1

password hash (16 characters), the server uses `mysql_old_password`. (For additional information about these hash formats, see Section 6.1.2.4, “Password Hashing in MySQL”.) Clients must match the password in the `Password` column of the account row.

The `password_expired` column permits DBAs to expire account passwords and require users to reset their password. The default `password_expired` value is `'N'`, but can be set to `'Y'` with the `ALTER USER` statement. After an account's password has been expired, all operations performed by the account in subsequent connections to the server result in an error until the user issues a `SET PASSWORD` statement to establish a new account password.

Note

Although it is possible to “reset” an expired password by setting it to its current value, it is preferable, as a matter of good policy, to choose a different password.

The tables_priv and columns_priv Grant Tables

During the second stage of access control, the server performs request verification to ensure that each client has sufficient privileges for each request that it issues. In addition to the `user` and `db` grant tables, the server may also consult the `tables_priv` and `columns_priv` tables for requests that involve tables. The latter tables provide finer privilege control at the table and column levels. They have the columns shown in the following table.

Table 6.4 tables_priv and columns_priv Table Columns

Table Name	`tables_priv`	`columns_priv`
Scope columns	`Host`	`Host`
	`Db`	`Db`
	`User`	`User`
	`Table_name`	`Table_name`
		`Column_name`
Privilege columns	`Table_priv`	`Column_priv`
	`Column_priv`	
Other columns	`Timestamp`	`Timestamp`
	`Grantor`	

The `Timestamp` and `Grantor` columns are set to the current timestamp and the `CURRENT_USER` value, respectively, but are otherwise unused.

The procs_priv Grant Table

For verification of requests that involve stored routines, the server may consult the `procs_priv` table, which has the columns shown in the following table.

Table 6.5 procs_priv Table Columns

Table Name	`procs_priv`
Scope columns	`Host`
	`Db`
	`User`
	`Routine_name`
	`Routine_type`

Table Name	`procs_priv`
Privilege columns	`Proc_priv`
Other columns	`Timestamp`
	`Grantor`

The `Routine_type` column is an `ENUM` column with values of `'FUNCTION'` or `'PROCEDURE'` to indicate the type of routine the row refers to. This column enables privileges to be granted separately for a function and a procedure with the same name.

The `Timestamp` and `Grantor` columns are unused.

The proxies_priv Grant Table

The `proxies_priv` table records information about proxy accounts. It has these columns:

- `Host`, `User`: The proxy account; that is, the account that has the `PROXY` privilege for the proxied account.
- `Proxied_host`, `Proxied_user`: The proxied account.
- `Grantor`, `Timestamp`: Unused.
- `With_grant`: Whether the proxy account can grant the `PROXY` privilege to other accounts.

For an account to be able to grant the `PROXY` privilege to other accounts, it must have a row in the `proxies_priv` table with `With_grant` set to 1 and `Proxied_host` and `Proxied_user` set to indicate the account or accounts for which the privilege can be granted. For example, the `'root'@'localhost'` account created during MySQL installation has a row in the `proxies_priv` table that enables granting the `PROXY` privilege for `''@''`, that is, for all users and all hosts. This enables `root` to set up proxy users, as well as to delegate to other accounts the authority to set up proxy users. See Section 6.2.12, “Proxy Users”.

Grant Table Scope Column Properties

Scope columns in the grant tables contain strings. The default value for each is the empty string. The following table shows the number of characters permitted in each column.

Table 6.6 Grant Table Scope Column Lengths

Column Name	Maximum Permitted Characters
`Host`, `Proxied_host`	60
`User`, `Proxied_user`	16
`Password`	41
`Db`	64
`Table_name`	64
`Column_name`	64
`Routine_name`	64

`Host` and `Proxied_host` values are converted to lowercase before being stored in the grant tables.

For access-checking purposes, comparisons of `User`, `Proxied_user`, `Password`, `Db`, and `Table_name` values are case-sensitive. Comparisons of `Host`, `Proxied_host`, `Column_name`, and `Routine_name` values are not case-sensitive.

Grant Table Privilege Column Properties

The `user` and `db` tables list each privilege in a separate column that is declared as `ENUM('N','Y') DEFAULT 'N'`. In other words, each privilege can be disabled or enabled, with the default being disabled.

The `tables_priv`, `columns_priv`, and `procs_priv` tables declare the privilege columns as `SET` columns. Values in these columns can contain any combination of the privileges controlled by the table. Only those privileges listed in the column value are enabled.

Table 6.7 Set-Type Privilege Column Values

Table Name	Column Name	Possible Set Elements
`tables_priv`	`Table_priv`	`'Select', 'Insert', 'Update', 'Delete', 'Create', 'Drop', 'Grant', 'References', 'Index', 'Alter', 'Create View', 'Show view', 'Trigger'`
`tables_priv`	`Column_priv`	`'Select', 'Insert', 'Update', 'References'`
`columns_priv`	`Column_priv`	`'Select', 'Insert', 'Update', 'References'`
`procs_priv`	`Proc_priv`	`'Execute', 'Alter Routine', 'Grant'`

Only the `user` table specifies administrative privileges, such as `RELOAD` and `SHUTDOWN`. Administrative operations are operations on the server itself and are not database-specific, so there is no reason to list these privileges in the other grant tables. Consequently, the server need consult only the `user` table to determine whether a user can perform an administrative operation.

The `FILE` privilege also is specified only in the `user` table. It is not an administrative privilege as such, but a user's ability to read or write files on the server host is independent of the database being accessed.

6.2.4 Specifying Account Names

MySQL account names consist of a user name and a host name, which enables creation of distinct accounts for users with the same user name who connect from different hosts. This section describes the syntax for account names, including special values and wildcard rules.

Account names appear in SQL statements such as `CREATE USER`, `GRANT`, and `SET PASSWORD` and follow these rules:

- Account name syntax is `'user_name'@'host_name'`.
- The `@'host_name'` part is optional. An account name consisting only of a user name is equivalent to `'user_name'@'%'`. For example, `'me'` is equivalent to `'me'@'%'`.
- The user name and host name need not be quoted if they are legal as unquoted identifiers. Quotes must be used if a *`user_name`* string contains special characters (such as space or `-`), or a *`host_name`* string contains special characters or wildcard characters (such as `.` or `%`). For example, in the account name `'test-user'@'%.com'`, both the user name and host name parts require quotes.
- Quote user names and host names as identifiers or as strings, using either backticks (`` ` ``), single quotation marks (`'`), or double quotation marks (`"`). For string-quoting and identifier-quoting guidelines, see Section 9.1.1, “String Literals”, and Section 9.2, “Schema Object Names”.

- The user name and host name parts, if quoted, must be quoted separately. That is, write `'me'@'localhost'`, not `'me@localhost'`. The latter is actually equivalent to `'me@localhost'@'%'`.
- A reference to the `CURRENT_USER` or `CURRENT_USER()` function is equivalent to specifying the current client's user name and host name literally.

MySQL stores account names in grant tables in the `mysql` system database using separate columns for the user name and host name parts:

- The `user` table contains one row for each account. The `User` and `Host` columns store the user name and host name. This table also indicates which global privileges the account has.
- Other grant tables indicate privileges an account has for databases and objects within databases. These tables have `User` and `Host` columns to store the account name. Each row in these tables associates with the account in the `user` table that has the same `User` and `Host` values.
- For access-checking purposes, comparisons of User values are case-sensitive. Comparisons of Host values are not case-sensitive.

For additional detail about the properties of user names and host names as stored in the grant tables, such as maximum length, see Grant Table Scope Column Properties.

User names and host names have certain special values or wildcard conventions, as described following.

The user name part of an account name is either a nonblank value that literally matches the user name for incoming connection attempts, or a blank value (the empty string) that matches any user name. An account with a blank user name is an anonymous user. To specify an anonymous user in SQL statements, use a quoted empty user name part, such as `''@'localhost'`.

The host name part of an account name can take many forms, and wildcards are permitted:

- A host value can be a host name or an IP address (IPv4 or IPv6). The name `'localhost'` indicates the local host. The IP address `'127.0.0.1'` indicates the IPv4 loopback interface. The IP address `'::1'` indicates the IPv6 loopback interface.
- The `%` and `_` wildcard characters are permitted in host name or IP address values. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator. For example, a host value of `'%'` matches any host name, whereas a value of `'%.mysql.com'` matches any host in the `mysql.com` domain. `'198.51.100.%'` matches any host in the 198.51.100 class C network.

  Because IP wildcard values are permitted in host values (for example, `'198.51.100.%'` to match every host on a subnet), someone could try to exploit this capability by naming a host `198.51.100.somewhere.com`. To foil such attempts, MySQL does not perform matching on host names that start with digits and a dot. For example, if a host is named `1.2.example.com`, its name never matches the host part of account names. An IP wildcard value can match only IP addresses, not host names.
- For a host value specified as an IPv4 address, a netmask can be given to indicate how many address bits to use for the network number. Netmask notation cannot be used for IPv6 addresses.

  The syntax is *`host_ip/netmask`*. For example:

  ```
  CREATE USER 'david'@'198.51.100.0/255.255.255.0';
  ```

  This enables `david` to connect from any client host having an IP address *`client_ip`* for which the following condition is true:

  ```
  client_ip & netmask = host_ip
  ```

  That is, for the `CREATE USER` statement just shown:

```
client_ip & 255.255.255.0 = 198.51.100.0
```

IP addresses that satisfy this condition range from `198.51.100.0` to `198.51.100.255`.

A netmask typically begins with bits set to 1, followed by bits set to 0. Examples:

- `198.0.0.0/255.0.0.0`: Any host on the 198 class A network
- `198.51.0.0/255.255.0.0`: Any host on the 198.51 class B network
- `198.51.100.0/255.255.255.0`: Any host on the 198.51.100 class C network
- `198.51.100.1`: Only the host with this specific IP address

The server performs matching of host values in account names against the client host using the value returned by the system DNS resolver for the client host name or IP address. Except in the case that the account host value is specified using netmask notation, the server performs this comparison as a string match, even for an account host value given as an IP address. This means that you should specify account host values in the same format used by DNS. Here are examples of problems to watch out for:

- Suppose that a host on the local network has a fully qualified name of `host1.example.com`. If DNS returns name lookups for this host as `host1.example.com`, use that name in account host values. If DNS returns just `host1`, use `host1` instead.
- If DNS returns the IP address for a given host as `198.51.100.2`, that matches an account host value of `198.51.100.2` but not `198.051.100.2`. Similarly, it matches an account host pattern like `198.51.100.%` but not `198.051.100.%`.

To avoid problems like these, it is advisable to check the format in which your DNS returns host names and addresses. Use values in the same format in MySQL account names.

6.2.5 Access Control, Stage 1: Connection Verification

When you attempt to connect to a MySQL server, the server accepts or rejects the connection based on your identity and whether you can verify it identity by supplying the correct password. If not, the server denies access to you completely. Otherwise, the server accepts the connection, and then enters Stage 2 and waits for requests.

The server performs identity and credentials checking using columns in the `user` table, accepting the connection only if these conditions are satisfied:

- The client host name and user name match the `Host` and `User` columns in some `user` table row. For the rules governing permissible `Host` and `User` values, see Section 6.2.4, “Specifying Account Names”.
- The client supplies the password specified in the row, as indicated by the `Password` column.

Your identity is based on two pieces of information:

- Your MySQL user name.
- The client host from which you connect.

If the `User` column value is nonblank, the user name in an incoming connection must match exactly. If the `User` value is blank, it matches any user name. If the `user` table row that matches an incoming connection has a blank user name, the user is considered to be an anonymous user with no name, not a user with the name that the client actually specified. This means that a blank user name is used for all further access checking for the duration of the connection (that is, during Stage 2).

The `Password` column can be blank. This is not a wildcard and does not mean that any password matches. It means that the user must connect without specifying a password. If the server

authenticates a client using a plugin, the authentication method that the plugin implements may or may not use the password in the `Password` column. In this case, it is possible that an external password is also used to authenticate to the MySQL server.

Nonblank password values stored in the `Password` column of the `user` table are encrypted. MySQL does not store passwords as cleartext for anyone to see. Rather, the password supplied by a user who is attempting to connect is encrypted (using the `PASSWORD()` function). The encrypted password then is used during the connection process when checking whether the password is correct. This is done without the encrypted password ever traveling over the connection. See Section 6.2.1, “Account User Names and Passwords”.

From MySQL's point of view, the encrypted password is the *real* password, so you should never give anyone access to it. In particular, *do not give nonadministrative users read access to tables in the `mysql` system database*.

The following table shows how various combinations of `User` and `Host` values in the `user` table apply to incoming connections.

`User` Value	`Host` Value	Permissible Connections
`'fred'`	`'h1.example.net'`	`fred`, connecting from `h1.example.net`
`''`	`'h1.example.net'`	Any user, connecting from `h1.example.net`
`'fred'`	`'%'`	`fred`, connecting from any host
`''`	`'%'`	Any user, connecting from any host
`'fred'`	`'%.example.net'`	`fred`, connecting from any host in the `example.net` domain
`'fred'`	`'x.example.%'`	`fred`, connecting from `x.example.net`, `x.example.com`, `x.example.edu`, and so on; this is probably not useful
`'fred'`	`'198.51.100.177'`	`fred`, connecting from the host with IP address `198.51.100.177`
`'fred'`	`'198.51.100.%'`	`fred`, connecting from any host in the `198.51.100` class C subnet
`'fred'`	`'198.51.100.0/255.255.255.0'`	Same as previous example

It is possible for the client host name and user name of an incoming connection to match more than one row in the `user` table. The preceding set of examples demonstrates this: Several of the entries shown match a connection from `h1.example.net` by `fred`.

When multiple matches are possible, the server must determine which of them to use. It resolves this issue as follows:

- Whenever the server reads the `user` table into memory, it sorts the rows.
- When a client attempts to connect, the server looks through the rows in sorted order.
- The server uses the first row that matches the client host name and user name.

The server uses sorting rules that order rows with the most-specific `Host` values first:

- Literal IP addresses and host names are the most specific.
- The specificity of a literal IP address is not affected by whether it has a netmask, so `198.51.100.13` and `198.51.100.0/255.255.255.0` are considered equally specific.
- The pattern `'%'` means “any host” and is least specific.

- The empty string `''` also means “any host” but sorts after `'%'`.

Non-TCP (socket file, named pipe, and shared memory) connections are treated as local connections and match a host part of `localhost` if there are any such accounts, or host parts with wildcards that match `localhost` otherwise (for example, `local%`, `l%`, `%`).

Rows with the same `Host` value are ordered with the most-specific `User` values first. A blank `User` value means “any user” and is least specific, so for rows with the same `Host` value, nonanonymous users sort before anonymous users.

For rows with equally-specific `Host` and `User` values, the order is nondeterministic.

To see how this works, suppose that the `user` table looks like this:

```
+-----------+----------+-
| Host      | User     | ...
+-----------+----------+-
| %         | root     | ...
| %         | jeffrey  | ...
| localhost | root     | ...
| localhost |          | ...
+-----------+----------+-
```

When the server reads the table into memory, it sorts the rows using the rules just described. The result after sorting looks like this:

```
+-----------+----------+-
| Host      | User     | ...
+-----------+----------+-
| localhost | root     | ...
| localhost |          | ...
| %         | jeffrey  | ...
| %         | root     | ...
+-----------+----------+-
```

When a client attempts to connect, the server looks through the sorted rows and uses the first match found. For a connection from `localhost` by `jeffrey`, two of the rows from the table match: the one with `Host` and `User` values of `'localhost'` and `''`, and the one with values of `'%'` and `'jeffrey'`. The `'localhost'` row appears first in sorted order, so that is the one the server uses.

Here is another example. Suppose that the `user` table looks like this:

```
+----------------+----------+-
| Host           | User     | ...
+----------------+----------+-
| %              | jeffrey  | ...
| h1.example.net |          | ...
+----------------+----------+-
```

The sorted table looks like this:

```
+----------------+----------+-
| Host           | User     | ...
+----------------+----------+-
| h1.example.net |          | ...
| %              | jeffrey  | ...
+----------------+----------+-
```

The first row matches a connection by any user from `h1.example.net`, whereas the second row matches a connection by `jeffrey` from any host.

Note

It is a common misconception to think that, for a given user name, all rows that explicitly name that user are used first when the server attempts to find a

> match for the connection. This is not true. The preceding example illustrates this, where a connection from `h1.example.net` by `jeffrey` is first matched not by the row containing `'jeffrey'` as the `User` column value, but by the row with no user name. As a result, `jeffrey` is authenticated as an anonymous user, even though he specified a user name when connecting.

If you are able to connect to the server, but your privileges are not what you expect, you probably are being authenticated as some other account. To find out what account the server used to authenticate you, use the `CURRENT_USER()` function. (See Section 12.16, “Information Functions”.) It returns a value in *`user_name@host_name`* format that indicates the `User` and `Host` values from the matching `user` table row. Suppose that `jeffrey` connects and issues the following query:

```
mysql> SELECT CURRENT_USER();
+----------------+
| CURRENT_USER() |
+----------------+
| @localhost     |
+----------------+
```

The result shown here indicates that the matching `user` table row had a blank `User` column value. In other words, the server is treating `jeffrey` as an anonymous user.

Another way to diagnose authentication problems is to print out the `user` table and sort it by hand to see where the first match is being made.

6.2.6 Access Control, Stage 2: Request Verification

After the server accepts a connection, it enters Stage 2 of access control. For each request that you issue through the connection, the server determines what operation you want to perform, then checks whether your privileges are sufficient. This is where the privilege columns in the grant tables come into play. These privileges can come from any of the `user`, `db`, `tables_priv`, `columns_priv`, or `procs_priv` tables. (You may find it helpful to refer to Section 6.2.3, “Grant Tables”, which lists the columns present in each grant table.)

The `user` table grants global privileges. The `user` table row for an account indicates the account privileges that apply on a global basis no matter what the default database is. For example, if the `user` table grants you the `DELETE` privilege, you can delete rows from any table in any database on the server host. It is wise to grant privileges in the `user` table only to people who need them, such as database administrators. For other users, leave all privileges in the `user` table set to `'N'` and grant privileges at more specific levels only (for particular databases, tables, columns, or routines).

The `db` table grants database-specific privileges. Values in the scope columns of this table can take the following forms:

- A blank `User` value matches the anonymous user. A nonblank value matches literally; there are no wildcards in user names.
- The wildcard characters `%` and `_` can be used in the `Host` and `Db` columns. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator. If you want to use either character literally when granting privileges, you must escape it with a backslash. For example, to include the underscore character (`_`) as part of a database name, specify it as `\_` in the `GRANT` statement.
- A `'%'` or blank `Host` value means “any host.”
- A `'%'` or blank `Db` value means “any database.”

The server reads the `db` table into memory and sorts it at the same time that it reads the `user` table. The server sorts the `db` table based on the `Host`, `Db`, and `User` scope columns. As with the `user`

table, sorting puts the most-specific values first and least-specific values last, and when the server looks for matching rows, it uses the first match that it finds.

The `tables_priv`, `columns_priv`, and `procs_priv` tables grant table-specific, column-specific, and routine-specific privileges. Values in the scope columns of these tables can take the following forms:

- The wildcard characters `%` and `_` can be used in the `Host` column. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator.
- A `'%'` or blank `Host` value means “any host.”
- The `Db`, `Table_name`, `Column_name`, and `Routine_name` columns cannot contain wildcards or be blank.

The server sorts the `tables_priv`, `columns_priv`, and `procs_priv` tables based on the `Host`, `Db`, and `User` columns. This is similar to `db` table sorting, but simpler because only the `Host` column can contain wildcards.

The server uses the sorted tables to verify each request that it receives. For requests that require administrative privileges such as `SHUTDOWN` or `RELOAD`, the server checks only the `user` table row because that is the only table that specifies administrative privileges. The server grants access if the row permits the requested operation and denies access otherwise. For example, if you want to execute `mysqladmin shutdown` but your `user` table row does not grant the `SHUTDOWN` privilege to you, the server denies access without even checking the `db` table. (The latter table contains no `Shutdown_priv` column, so there is no need to check it.)

For database-related requests (`INSERT`, `UPDATE`, and so on), the server first checks the user's global privileges in the `user` table row. If the row permits the requested operation, access is granted. If the global privileges in the `user` table are insufficient, the server determines the user's database-specific privileges from the `db` table:

- The server looks in the `db` table for a match on the `Host`, `Db`, and `User` columns.
- The `Host` and `User` columns are matched to the connecting user's host name and MySQL user name.
- The `Db` column is matched to the database that the user wants to access.
- If there is no row for the `Host` and `User`, access is denied.

After determining the database-specific privileges granted by the `db` table rows, the server adds them to the global privileges granted by the `user` table. If the result permits the requested operation, access is granted. Otherwise, the server successively checks the user's table and column privileges in the `tables_priv` and `columns_priv` tables, adds those to the user's privileges, and permits or denies access based on the result. For stored-routine operations, the server uses the `procs_priv` table rather than `tables_priv` and `columns_priv`.

Expressed in boolean terms, the preceding description of how a user's privileges are calculated may be summarized like this:

```
global privileges
OR database privileges
OR table privileges
OR column privileges
OR routine privileges
```

It may not be apparent why, if the global privileges are initially found to be insufficient for the requested operation, the server adds those privileges to the database, table, and column privileges later. The reason is that a request might require more than one type of privilege. For example, if you execute an `INSERT INTO ... SELECT` statement, you need both the `INSERT` and the `SELECT` privileges.

Your privileges might be such that the `user` table row grants one privilege global and the `db` table row grants the other specifically for the relevant database. In this case, you have the necessary privileges to perform the request, but the server cannot tell that from either your global or database privileges alone. It must make an access-control decision based on the combined privileges.

6.2.7 Adding Accounts, Assigning Privileges, and Dropping Accounts

To manage MySQL accounts, use the SQL statements intended for that purpose:

- `CREATE USER` and `DROP USER` create and remove accounts.
- `GRANT` and `REVOKE` assign privileges to and revoke privileges from accounts.
- `SHOW GRANTS` displays account privilege assignments.

Account-management statements cause the server to make appropriate modifications to the underlying grant tables, which are discussed in Section 6.2.3, “Grant Tables”.

Note

Direct modification of grant tables using statements such as `INSERT`, `UPDATE`, or `DELETE` is discouraged and done at your own risk. The server is free to ignore rows that become malformed as a result of such modifications.

As of MySQL 5.6.36, for any operation that modifies a grant table, the server checks whether the table has the expected structure and produces an error if not. `mysql_upgrade` must be run to update the tables to the expected structure.

Another option for creating accounts is to use the GUI tool MySQL Workbench. Also, several third-party programs offer capabilities for MySQL account administration. `phpMyAdmin` is one such program.

This section discusses the following topics:

- Creating Accounts and Granting Privileges
- Checking Account Privileges
- Revoking Account Privileges
- Dropping Accounts

For additional information about the statements discussed here, see Section 13.7.1, “Account Management Statements”.

Creating Accounts and Granting Privileges

The following examples show how to use the `mysql` client program to set up new accounts. These examples assume that the MySQL `root` account has the `CREATE USER` privilege and all privileges that it grants to other accounts.

At the command line, connect to the server as the MySQL `root` user, supplying the appropriate password at the password prompt:

```
shell> mysql -u root -p
Enter password: (enter root password here)
```

After connecting to the server, you can add new accounts. The following example uses `CREATE USER` and `GRANT` statements to set up four accounts (where you see `'password'`, substitute an appropriate password):

```
CREATE USER 'finley'@'localhost'
  IDENTIFIED BY 'password';
GRANT ALL
  ON *.*
  TO 'finley'@'localhost'
  WITH GRANT OPTION;

CREATE USER 'finley'@'%.example.com'
  IDENTIFIED BY 'password';
GRANT ALL
  ON *.*
  TO 'finley'@'%.example.com'
  WITH GRANT OPTION;

CREATE USER 'admin'@'localhost'
  IDENTIFIED BY 'password';
GRANT RELOAD,PROCESS
  ON *.*
  TO 'admin'@'localhost';

CREATE USER 'dummy'@'localhost';
```

The accounts created by those statements have the following properties:

- Two accounts have a user name of `finley`. Both are superuser accounts with full global privileges to do anything. The `'finley'@'localhost'` account can be used only when connecting from the local host. The `'finley'@'%.example.com'` account uses the `'%'` wildcard in the host part, so it can be used to connect from any host in the `example.com` domain.

  The `'finley'@'localhost'` account is necessary if there is an anonymous-user account for `localhost`. Without the `'finley'@'localhost'` account, that anonymous-user account takes precedence when `finley` connects from the local host and `finley` is treated as an anonymous user. The reason for this is that the anonymous-user account has a more specific `Host` column value than the `'finley'@'%'` account and thus comes earlier in the `user` table sort order. (For information about `user` table sorting, see Section 6.2.5, “Access Control, Stage 1: Connection Verification”.)

- The `'admin'@'localhost'` account can be used only by `admin` to connect from the local host. It is granted the global `RELOAD` and `PROCESS` administrative privileges. These privileges enable the `admin` user to execute the `mysqladmin reload`, `mysqladmin refresh`, and `mysqladmin flush-xxx` commands, as well as `mysqladmin processlist` . No privileges are granted for accessing any databases. You could add such privileges using `GRANT` statements.

- The `'dummy'@'localhost'` account has no password (which is insecure and not recommended). This account can be used only to connect from the local host. No privileges are granted. It is assumed that you grant specific privileges to the account using `GRANT` statements.

The previous example grants privileges at the global level. The next example creates three accounts and grants them access at lower levels; that is, to specific databases or objects within databases. Each account has a user name of `custom`, but the host name parts differ:

```
CREATE USER 'custom'@'localhost'
  IDENTIFIED BY 'password';
GRANT ALL
  ON bankaccount.*
  TO 'custom'@'localhost';

CREATE USER 'custom'@'host47.example.com'
  IDENTIFIED BY 'password';
GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
  ON expenses.*
  TO 'custom'@'host47.example.com';

CREATE USER 'custom'@'%.example.com'
  IDENTIFIED BY 'password';
GRANT SELECT,INSERT,UPDATE,DELETE,CREATE,DROP
  ON customer.addresses
```

```
  TO 'custom'@'%.example.com';
```

The three accounts can be used as follows:

- The `'custom'@'localhost'` account has all database-level privileges to access the `bankaccount` database. The account can be used to connect to the server only from the local host.
- The `'custom'@'host47.example.com'` account has specific database-level privileges to access the `expenses` database. The account can be used to connect to the server only from the host `host47.example.com`.
- The `'custom'@'%.example.com'` account has specific table-level privileges to access the `addresses` table in the `customer` database, from any host in the `example.com` domain. The account can be used to connect to the server from all machines in the domain due to use of the `%` wildcard character in the host part of the account name.

Checking Account Privileges

To see the privileges for an account, use `SHOW GRANTS`:

```
mysql> SHOW GRANTS FOR 'admin'@'localhost';
+-----------------------------------------------------+
| Grants for admin@localhost                          |
+-----------------------------------------------------+
| GRANT RELOAD, PROCESS ON *.* TO 'admin'@'localhost' |
+-----------------------------------------------------+
```

Revoking Account Privileges

To revoke account privileges, use the `REVOKE` statement. Privileges can be revoked at different levels, just as they can be granted at different levels.

Revoke global privileges:

```
REVOKE ALL
  ON *.*
  FROM 'finley'@'%.example.com';

REVOKE RELOAD
  ON *.*
  FROM 'admin'@'localhost';
```

Revoke database-level privileges:

```
REVOKE CREATE,DROP
  ON expenses.*
  FROM 'custom'@'host47.example.com';
```

Revoke table-level privileges:

```
REVOKE INSERT,UPDATE,DELETE
  ON customer.addresses
  FROM 'custom'@'%.example.com';
```

To check the effect of privilege revocation, use `SHOW GRANTS`:

```
mysql> SHOW GRANTS FOR 'admin'@'localhost';
+-------------------------------------------+
| Grants for admin@localhost                |
+-------------------------------------------+
| GRANT PROCESS ON *.* TO 'admin'@'localhost' |
+-------------------------------------------+
```

Dropping Accounts

To remove an account, use the `DROP USER` statement. For example, to drop some of the accounts created previously:

```
DROP USER 'finley'@'localhost';
DROP USER 'finley'@'%.example.com';
DROP USER 'admin'@'localhost';
DROP USER 'dummy'@'localhost';
```

6.2.8 When Privilege Changes Take Effect

If the `mysqld` server is started without the `--skip-grant-tables` option, it reads all grant table contents into memory during its startup sequence. The in-memory tables become effective for access control at that point.

If you modify the grant tables indirectly using an account-management statement, the server notices these changes and loads the grant tables into memory again immediately. Account-management statements are described in Section 13.7.1, “Account Management Statements”. Examples include `GRANT`, `REVOKE`, `SET PASSWORD`, and `RENAME USER`.

If you modify the grant tables directly using statements such as `INSERT`, `UPDATE`, or `DELETE` (which is not recommended), the changes have no effect on privilege checking until you either tell the server to reload the tables or restart it. Thus, if you change the grant tables directly but forget to reload them, the changes have *no effect* until you restart the server. This may leave you wondering why your changes seem to make no difference!

To tell the server to reload the grant tables, perform a flush-privileges operation. This can be done by issuing a `FLUSH PRIVILEGES` statement or by executing a `mysqladmin flush-privileges` or `mysqladmin reload` command.

A grant table reload affects privileges for each existing client session as follows:

- Table and column privilege changes take effect with the client's next request.
- Database privilege changes take effect the next time the client executes a `USE db_name` statement.

Note

Client applications may cache the database name; thus, this effect may not be visible to them without actually changing to a different database.

- Global privileges and passwords are unaffected for a connected client. These changes take effect only in sessions for subsequent connections.

If the server is started with the `--skip-grant-tables` option, it does not read the grant tables or implement any access control. Any user can connect and perform any operation, *which is insecure.* To cause a server thus started to read the tables and enable access checking, flush the privileges.

6.2.9 Assigning Account Passwords

Required credentials for clients that connect to the MySQL server can include a password. This section describes how to assign passwords for MySQL accounts.

MySQL stores credentials in the `user` table in the `mysql` system database. Operations that assign or modify passwords are permitted only to users with the `CREATE USER` privilege, or, alternatively, privileges for the `mysql` database (`INSERT` privilege to create new accounts, `UPDATE` privilege to modify existing accounts). If the `read_only` system variable is enabled, use of account-modification statements such as `CREATE USER` or `SET PASSWORD` additionally requires the `SUPER` privilege.

The discussion here summarizes syntax only for the most common password-assignment statements. For complete details on other possibilities, see Section 13.7.1.2, “CREATE USER Statement”, Section 13.7.1.4, “GRANT Statement”, and Section 13.7.1.7, “SET PASSWORD Statement”.

MySQL uses plugins to perform client authentication; see Section 6.2.11, “Pluggable Authentication”. In password-assigning statements, the authentication plugin associated with an account performs any hashing required of a cleartext password specified. This enables MySQL to obfuscate passwords prior to storing them in the `mysql.user` system table. For most statements described here, MySQL automatically hashes the password specified. An exception is `SET PASSWORD ... = PASSWORD('auth_string')`, for which you use the `PASSWORD()` function explicitly to hash the password. There is also syntax for `CREATE USER`, `GRANT`, and `SET PASSWORD` that permits hashed values to be specified literally. For details, see the descriptions of those statements.

To assign a password when you create a new account, use `CREATE USER` and include an `IDENTIFIED BY` clause:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'password';
```

`CREATE USER` also supports syntax for specifying the account authentication plugin. See Section 13.7.1.2, “CREATE USER Statement”.

To assign or change a password for an existing account, use `SET PASSWORD` with the `PASSWORD()` function:

```
SET PASSWORD FOR 'jeffrey'@'localhost' = PASSWORD('password');
```

If you are not connected as an anonymous user, you can change your own password by omitting the `FOR` clause:

```
SET PASSWORD = PASSWORD('password');
```

The `PASSWORD()` function hashes the password using the hashing method determined by the value of the `old_passwords` system variable value. If `SET PASSWORD` rejects the hashed password value returned by `PASSWORD()` as not being in the correct format, it may be necessary to change `old_passwords` to change the hashing method. See Section 13.7.1.7, “SET PASSWORD Statement”.

Use a `GRANT USAGE` statement at the global level (`ON *.*`) to change an account password without affecting the account's current privileges:

```
GRANT USAGE ON *.* TO 'jeffrey'@'localhost' IDENTIFIED BY 'password';
```

To change an account password from the command line, use the `mysqladmin` command:

```
mysqladmin -u user_name -h host_name password "password"
```

The account for which this command sets the password is the one with a row in the `mysql.user` system table that matches *`user_name`* in the `User` column and the client host *from which you connect* in the `Host` column.

Warning

Setting a password using `mysqladmin` should be considered *insecure*. On some systems, your password becomes visible to system status programs such as `ps` that may be invoked by other users to display command lines. MySQL clients typically overwrite the command-line password argument with zeros during their initialization sequence. However, there is still a brief interval during which the value is visible. Also, on some systems this overwriting strategy is ineffective and the password remains visible to `ps`. (SystemV Unix systems and perhaps others are subject to this problem.)

If you are using MySQL Replication, be aware that, currently, a password used by a replica as part of a `CHANGE MASTER TO` statement is effectively limited to 32 characters in length; if the password is longer, any excess characters are truncated. This is not due to any limit imposed by the MySQL

Server generally, but rather is an issue specific to MySQL Replication. (For more information, see Bug #43439.)

6.2.10 Server Handling of Expired Passwords

MySQL 5.6 introduces password-expiration capability, which enables database administrators to require that users reset their password. The immediately following discussion describes how password expiration works currently. Later, the development of this capability is detailed as it occurred over several versions, as background to help you understand what features are available when. However, to ensure that you can take advantage of all features, use the most recent available version of MySQL if possible.

- How Password Expiration Works
- Development of Password-Expiration Capability

How Password Expiration Works

The `ALTER USER` statement enables account password expiration. For example:

```
ALTER USER 'myuser'@'localhost' PASSWORD EXPIRE;
```

For each connection that uses an account with an expired password, the server either disconnects the client or restricts the client to “sandbox mode,” in which the server permits the client to perform only those operations necessary to reset the expired password. Which action is taken by the server depends on both client and server settings, as discussed later.

If the server disconnects the client, it returns an `ER_MUST_CHANGE_PASSWORD_LOGIN` error:

```
shell> mysql -u myuser -p
Password: ******
ERROR 1862 (HY000): Your password has expired. To log in you must
change it using a client that supports expired passwords.
```

If the server restricts the client to sandbox mode, these operations are permitted within the client session:

- The client can reset the account password with `SET PASSWORD`. After that has been done, the server restores normal access for the session, as well as for subsequent connections that use the account.

  Note

  Although it is possible to “reset” an expired password by setting it to its current value, it is preferable, as a matter of good policy, to choose a different password.

- The client can use the `SET` statement. This might be necessary prior to using `SET PASSWORD` to reset the password if the account uses an authentication plugin for which the `old_passwords` system variable must first be set to a nondefault value to perform password hashing in a specific way.

For any operation not permitted within the session, the server returns an `ER_MUST_CHANGE_PASSWORD` error:

```
mysql> USE performance_schema;
ERROR 1820 (HY000): You must SET PASSWORD before executing this statement

mysql> SELECT 1;
ERROR 1820 (HY000): You must SET PASSWORD before executing this statement
```

That is what normally happens for interactive invocations of the `mysql` client because by default such invocations are put in sandbox mode. To resume normal functioning, select a new password.

For noninteractive invocations of the `mysql` client (for example, in batch mode), the server normally disconnects the client if the password is expired. To permit noninteractive `mysql` invocations to stay connected so that the password can be changed (using the statements permitted in sandbox mode), add the `--connect-expired-password` option to the `mysql` command.

As mentioned previously, whether the server disconnects an expired-password client or restricts it to sandbox mode depends on a combination of client and server settings. The following discussion describes the relevant settings and how they interact.

Note

This discussion applies only for accounts with expired passwords. If a client connects using a nonexpired password, the server handles the client normally.

On the client side, a given client indicates whether it can handle sandbox mode for expired passwords. For clients that use the C client library, there are two ways to do this:

- Pass the `MYSQL_OPT_CAN_HANDLE_EXPIRED_PASSWORDS` flag to `mysql_options()` prior to connecting:

```
my_bool arg = 1;
mysql_options(mysql,
              MYSQL_OPT_CAN_HANDLE_EXPIRED_PASSWORDS,
              &arg);
```

  This is the technique used within the `mysql` client, which enables `MYSQL_OPT_CAN_HANDLE_EXPIRED_PASSWORDS` if invoked interactively or with the `--connect-expired-password` option.

- Pass the `CLIENT_CAN_HANDLE_EXPIRED_PASSWORDS` flag to `mysql_real_connect()` at connect time:

```
MYSQL mysql;
mysql_init(&mysql);
if (!mysql_real_connect(&mysql,
                        host, user, password, db,
                        port, unix_socket,
                        CLIENT_CAN_HANDLE_EXPIRED_PASSWORDS))
{
  ... handle error ...
}
```

Other MySQL Connectors have their own conventions for indicating readiness to handle sandbox mode. See the documentation for the Connector in which you are interested.

On the server side, if a client indicates that it can handle expired passwords, the server puts it in sandbox mode.

If a client does not indicate that it can handle expired passwords (or uses an older version of the client library that cannot so indicate), the server action depends on the value of the `disconnect_on_expired_password` system variable:

- If `disconnect_on_expired_password` is enabled (the default), the server disconnects the client with an `ER_MUST_CHANGE_PASSWORD_LOGIN` error.
- If `disconnect_on_expired_password` is disabled, the server puts the client in sandbox mode.

Development of Password-Expiration Capability

The following timeline describes the versions in which various password-expiration features were added.

- MySQL 5.6.6: Initial implementation of password expiration.

The `password_expired` column is introduced in the `mysql.user` system table to enable DBAs to expire account passwords. The column default value is `'N'` (not expired).

The `ALTER USER ... PASSWORD EXPIRE` statement is introduced as the SQL interface for setting the `password_expired` column to `'Y'`.

Connections that use an account with an expired password enter “sandbox mode” that permits only `SET PASSWORD` statements. For other statements, the server returns an `ER_MUST_CHANGE_PASSWORD` error. The intent is to force the client to reset the password before the server permits any other operations. `SET PASSWORD` resets the account password and sets `password_expired` to `'N'`.

A bug in the initial implementation is that `ALTER USER` sets the `Password` column in the `mysql.user` system table to the empty string. The implication is that users should wait until MySQL 5.6.7 to use this statement.

- MySQL 5.6.7: `ALTER USER` is fixed to not set the `Password` column to the empty string.
- MySQL 5.6.8: `ALTER USER` can be used as a prepared statement.

  `mysqladmin password` is made capable of resetting expired passwords for accounts that use the `mysql_native_password` or `mysql_old_password` authentication plugin.

  Sandbox mode is changed to permit clients to execute `SET` statements in addition to `SET PASSWORD` Prohibiting `SET` prevented clients that needed to set `old_passwords` from resetting their password. It also broke some Connectors, which use `SET` extensively at connect time to initialize the session environment.
- MySQL 5.6.9: Sandbox mode is changed to permit `SET PASSWORD` only if the account named in the statement matches the account the client authenticated as.
- MySQL 5.6.10: Sandbox mode is changed to permit better control over how the server handles client connections for accounts with expired passwords, and to permit clients to signal whether they are capable of handling expired passwords:
  - The `disconnect_on_expired_password` system variable is added, which controls how the server treats expired-password accounts.
  - Two flags are added to the C API client library: `MYSQL_OPT_CAN_HANDLE_EXPIRED_PASSWORDS` for `mysql_options()` and `CLIENT_CAN_HANDLE_EXPIRED_PASSWORDS` for `mysql_real_connect()`. Each flag enables a client program to indicate whether it can handle sandbox mode for accounts with expired passwords.

    `MYSQL_OPT_CAN_HANDLE_EXPIRED_PASSWORDS` is enabled for `mysqltest` unconditionally, for `mysql` in interactive mode, and for `mysqladmin` if the first command is `password`.
  - The `ER_MUST_CHANGE_PASSWORD_LOGIN` error is added. The server returns this error when it disconnects a client that has an expired password.
  - MySQL 5.6.12: The `--connect-expired-password` option is added to the `mysql` client to enable password-change statement execution in batch mode for accounts with an expired password.

Concurrent with these changes to sandbox mode in MySQL Server and the C API client library, work begins to modify Connectors for conformance to the changes.

6.2.11 Pluggable Authentication

When a client connects to the MySQL server, the server uses the user name provided by the client and the client host to select the appropriate account row from the `mysql.user` system table. The server

then authenticates the client, determining from the account row which authentication plugin applies to the client:

- If the server cannot find the plugin, an error occurs and the connection attempt is rejected. Otherwise, if the account row specifies a plugin, the server invokes it to authenticate the user.
- If the account row specifies no plugin name, the server authenticates the account using either the `mysql_native_password` or `mysql_old_password` plugin, depending on whether the password hash value in the `Password` column used native hashing or the older pre-4.1 hashing method. Clients must match the password in the `Password` column of the account row.

The plugin returns a status to the server indicating whether the user provided the correct password and is permitted to connect.

Pluggable authentication enables these important capabilities:

- **Choice of authentication methods.** Pluggable authentication makes it easy for DBAs to choose and change the authentication method used for individual MySQL accounts.
- **External authentication.** Pluggable authentication makes it possible for clients to connect to the MySQL server with credentials appropriate for authentication methods that store credentials elsewhere than in the `mysql.user` system table. For example, plugins can be created to use external authentication methods such as PAM, Windows login IDs, LDAP, or Kerberos.
- **Proxy users:** If a user is permitted to connect, an authentication plugin can return to the server a user name different from the name of the connecting user, to indicate that the connecting user is a proxy for another user (the proxied user). While the connection lasts, the proxy user is treated, for purposes of access control, as having the privileges of the proxied user. In effect, one user impersonates another. For more information, see Section 6.2.12, “Proxy Users”.

Note

If you start the server with the `--skip-grant-tables` option, authentication plugins are not used even if loaded because the server performs no client authentication and permits any client to connect. Because this is insecure, you might want to use `--skip-grant-tables` in conjunction with enabling the `skip_networking` system variable to prevent remote clients from connecting.

- Available Authentication Plugins
- Authentication Plugin Usage
- Restrictions on Pluggable Authentication

Available Authentication Plugins

MySQL 5.6 provides these authentication plugins:

- Plugins that perform native authentication; that is, authentication based on the password hashing methods in use from before the introduction of pluggable authentication in MySQL. The `mysql_native_password` plugin implements authentication based on the native password hashing method. The `mysql_old_password` plugin implements native authentication based on the older (pre-4.1) password hashing method (and is now deprecated). See Section 6.4.1.1, “Native Pluggable Authentication”, and Section 6.4.1.2, “Old Native Pluggable Authentication”. Native authentication using `mysql_native_password` is the default for new accounts, unless the `--default-authentication-plugin` option is set otherwise at server startup.
- A plugin that performs authentication using SHA-256 password hashing. This is stronger encryption than that available with native authentication. See Section 6.4.1.4, “SHA-256 Pluggable Authentication”.

- A client-side plugin that sends the password to the server without hashing or encryption. This plugin is used in conjunction with server-side plugins that require access to the password exactly as provided by the client user. See Section 6.4.1.5, “Client-Side Cleartext Pluggable Authentication”.
- A plugin that performs external authentication using PAM (Pluggable Authentication Modules), enabling MySQL Server to use PAM to authenticate MySQL users. This plugin supports proxy users as well. See Section 6.4.1.6, “PAM Pluggable Authentication”.
- A plugin that performs external authentication on Windows, enabling MySQL Server to use native Windows services to authenticate client connections. Users who have logged in to Windows can connect from MySQL client programs to the server based on the information in their environment without specifying an additional password. This plugin supports proxy users as well. See Section 6.4.1.7, “Windows Pluggable Authentication”.
- A plugin that authenticates clients that connect from the local host through the Unix socket file. See Section 6.4.1.8, “Socket Peer-Credential Pluggable Authentication”.
- A test plugin that checks account credentials and logs success or failure to the server error log. This plugin is intended for testing and development purposes, and as an example of how to write an authentication plugin. See Section 6.4.1.9, “Test Pluggable Authentication”.

Note

For information about current restrictions on the use of pluggable authentication, including which connectors support which plugins, see Restrictions on Pluggable Authentication.

Third-party connector developers should read that section to determine the extent to which a connector can take advantage of pluggable authentication capabilities and what steps to take to become more compliant.

If you are interested in writing your own authentication plugins, see Writing Authentication Plugins.

Authentication Plugin Usage

This section provides general instructions for installing and using authentication plugins. For instructions specific to a given plugin, see the section that describes that plugin under Section 6.4.1, “Authentication Plugins”.

In general, pluggable authentication uses a pair of corresponding plugins on the server and client sides, so you use a given authentication method like this:

- If necessary, install the plugin library or libraries containing the appropriate plugins. On the server host, install the library containing the server-side plugin, so that the server can use it to authenticate client connections. Similarly, on each client host, install the library containing the client-side plugin for use by client programs. Authentication plugins that are built in need not be installed.
- For each MySQL account that you create, specify the appropriate server-side plugin to use for authentication. If the account is to use the default authentication plugin, the account-creation statement need not specify the plugin explicitly. The `--default-authentication-plugin` option configures the default authentication plugin.
- When a client connects, the server-side plugin tells the client program which client-side plugin to use for authentication.

In the case that an account uses an authentication method that is the default for both the server and the client program, the server need not communicate to the client which client-side plugin to use, and a round trip in client/server negotiation can be avoided. This is true for accounts that use native MySQL authentication.

For standard MySQL clients such as `mysql` and `mysqladmin`, the `--default-auth=`*`plugin_name`* option can be specified on the command line as a hint about which client-side plugin the program can

expect to use, although the server overrides this if the server-side plugin associated with the user account requires a different client-side plugin.

If the client program does not find the client-side plugin library file, specify a `--plugin-dir=dir_name` option to indicate the plugin library directory location.

Restrictions on Pluggable Authentication

The first part of this section describes general restrictions on the applicability of the pluggable authentication framework described at Section 6.2.11, “Pluggable Authentication”. The second part describes how third-party connector developers can determine the extent to which a connector can take advantage of pluggable authentication capabilities and what steps to take to become more compliant.

The term “native authentication” used here refers to authentication against passwords stored in the `Password` column of the `mysql.user` system table. This is the same authentication method provided by older MySQL servers, before pluggable authentication was implemented. “Windows native authentication” refers to authentication using the credentials of a user who has already logged in to Windows, as implemented by the Windows Native Authentication plugin (“Windows plugin” for short).

- General Pluggable Authentication Restrictions
- Pluggable Authentication and Third-Party Connectors

General Pluggable Authentication Restrictions

- **Connector/C++:** Clients that use this connector can connect to the server only through accounts that use native authentication.

  Exception: A connector supports pluggable authentication if it was built to link to `libmysqlclient` dynamically (rather than statically) and it loads the current version of `libmysqlclient` if that version is installed, or if the connector is recompiled from source to link against the current `libmysqlclient`.

- **Connector/NET:** Clients that use Connector/NET can connect to the server through accounts that use native authentication or Windows native authentication.

- **Connector/PHP:** Clients that use this connector can connect to the server only through accounts that use native authentication, when compiled using the MySQL native driver for PHP (`mysqlnd`).

- **Windows native authentication:** Connecting through an account that uses the Windows plugin requires Windows Domain setup. Without it, NTLM authentication is used and then only local connections are possible; that is, the client and server must run on the same computer.

- **Proxy users:** Proxy user support is available to the extent that clients can connect through accounts authenticated with plugins that implement proxy user capability (that is, plugins that can return a user name different from that of the connecting user). For example, the PAM and Windows plugins support proxy users. The native authentication plugins do not.

- **Replication:** Before MySQL 5.6.4, replicas can connect to the source server only through source accounts that use native authentication. As of 5.6.4, replicas can also connect through source accounts that use nonnative authentication if the required client-side plugin is available. If the plugin is built into `libmysqlclient`, it is available by default. Otherwise, the plugin must be installed on the replica side in the directory named by the replica `plugin_dir` system variable.

- **`FEDERATED` tables:** A `FEDERATED` table can access the remote table only through accounts on the remote server that use native authentication.

Pluggable Authentication and Third-Party Connectors

Third-party connector developers can use the following guidelines to determine readiness of a connector to take advantage of pluggable authentication capabilities and what steps to take to become more compliant:

- An existing connector to which no changes have been made uses native authentication and clients that use the connector can connect to the server only through accounts that use native authentication. *However, you should test the connector against a recent version of the server to verify that such connections still work without problem.*

  Exception: A connector might work with pluggable authentication without any changes if it links to `libmysqlclient` dynamically (rather than statically) and it loads the current version of `libmysqlclient` if that version is installed.

- To take advantage of pluggable authentication capabilities, a connector that is `libmysqlclient`-based should be relinked against the current version of `libmysqlclient`. This enables the connector to support connections though accounts that require client-side plugins now built into `libmysqlclient` (such as the cleartext plugin needed for PAM authentication and the Windows plugin needed for Windows native authentication). Linking with a current `libmysqlclient` also enables the connector to access client-side plugins installed in the default MySQL plugin directory (typically the directory named by the default value of the local server's `plugin_dir` system variable).

  If a connector links to `libmysqlclient` dynamically, it must be ensured that the newer version of `libmysqlclient` is installed on the client host and that the connector loads it at runtime.

- Another way for a connector to support a given authentication method is to implement it directly in the client/server protocol. Connector/NET uses this approach to provide support for Windows native authentication.

- If a connector should be able to load client-side plugins from a directory different from the default plugin directory, it must implement some means for client users to specify the directory. Possibilities for this include a command-line option or environment variable from which the connector can obtain the directory name. Standard MySQL client programs such as `mysql` and `mysqladmin` implement a `--plugin-dir` option. See also C API Client Plugin Functions.

- Proxy user support by a connector depends, as described earlier in this section, on whether the authentication methods that it supports permit proxy users.

6.2.12 Proxy Users

The MySQL server authenticates client connections using authentication plugins. The plugin that authenticates a given connection may request that the connecting (external) user be treated as a different user for privilege-checking purposes. This enables the external user to be a proxy for the second user; that is, to assume the privileges of the second user:

- The external user is a “proxy user” (a user who can impersonate or become known as another user).
- The second user is a “proxied user” (a user whose identity and privileges can be assumed by a proxy user).

This section describes how the proxy user capability works. For general information about authentication plugins, see Section 6.2.11, “Pluggable Authentication”. For information about specific plugins, see Section 6.4.1, “Authentication Plugins”. For information about writing authentication plugins that support proxy users, see Implementing Proxy User Support in Authentication Plugins.

- Requirements for Proxy User Support
- Simple Proxy User Example
- Preventing Direct Login to Proxied Accounts
- Granting and Revoking the PROXY Privilege
- Default Proxy Users

- Default Proxy User and Anonymous User Conflicts
- Proxy User System Variables

Requirements for Proxy User Support

For proxying to occur for a given authentication plugin, these conditions must be satisfied:

- The plugin must support proxying.
- The account for the external proxy user must be set up to be authenticated by the plugin. Use the `CREATE USER` or `GRANT` statement to associate an account with an authentication plugin.
- The account for the proxied user must exist and be granted the privileges to be assumed by the proxy user. Use the `CREATE USER` and `GRANT` statements for this.
- Normally, the proxied user is configured so that it can be used only in proxying scenarios and not for direct logins.
- The proxy user account must have the `PROXY` privilege for the proxied account. Use the `GRANT` statement for this.
- For a client connecting to the proxy account to be treated as a proxy user, the authentication plugin must return a user name different from the client user name, to indicate the user name of the proxied account that defines the privileges to be assumed by the proxy user.

The proxy mechanism permits mapping only the external client user name to the proxied user name. There is no provision for mapping host names:

- When a client connects to the server, the server determines the proper account based on the user name passed by the client program and the host from which the client connects.
- If that account is a proxy account, the server attempts to determine the appropriate proxied account by finding a match for a proxied account using the user name returned by the authentication plugin and the host name of the proxy account. The host name in the proxied account is ignored.

Simple Proxy User Example

Consider the following account definitions:

```
-- create proxy account
CREATE USER 'employee_ext'@'localhost'
  IDENTIFIED WITH my_auth_plugin
  AS 'my_auth_string';

-- create proxied account and grant its privileges
CREATE USER 'employee'@'localhost'
  IDENTIFIED BY 'employee_password';
GRANT ALL
  ON employees.*
  TO 'employee'@'localhost';

-- grant to proxy account the PROXY privilege for proxied account
GRANT PROXY
  ON 'employee'@'localhost'
  TO 'employee_ext'@'localhost';
```

When a client connects as `employee_ext` from the local host, MySQL uses the plugin named `my_auth_plugin` to perform authentication. Suppose that `my_auth_plugin` returns a user name of `employee` to the server, based on the content of `'my_auth_string'` and perhaps by consulting some external authentication system. The name `employee` differs from `employee_ext`, so returning `employee` serves as a request to the server to treat the `employee_ext` external user, for purposes of privilege checking, as the `employee` local user.

In this case, `employee_ext` is the proxy user and `employee` is the proxied user.

The server verifies that proxy authentication for `employee` is possible for the `employee_ext` user by checking whether `employee_ext` (the proxy user) has the `PROXY` privilege for `employee` (the proxied user). If this privilege has not been granted, an error occurs. Otherwise, `employee_ext` assumes the privileges of `employee`. The server checks statements executed during the client session by `employee_ext` against the privileges granted to `employee`. In this case, `employee_ext` can access tables in the `employees` database.

To ensure that the proxied account, `employee`, cannot be used directly, do not tell anyone else its password. If you do not let anyone know the password for the account, clients cannot use it to connect directly to the MySQL server.

When proxying occurs, the `USER()` and `CURRENT_USER()` functions can be used to see the difference between the connecting user (the proxy user) and the account whose privileges apply during the current session (the proxied user). For the example just described, those functions return these values:

```
mysql> SELECT USER(), CURRENT_USER();
+------------------------+--------------------+
| USER()                 | CURRENT_USER()     |
+------------------------+--------------------+
| employee_ext@localhost | employee@localhost |
+------------------------+--------------------+
```

In the `CREATE USER` statement that creates the proxy user account, the `IDENTIFIED WITH` clause that names the proxy-supporting authentication plugin is optionally followed by an `AS 'auth_string'` clause specifying a string that the server passes to the plugin when the user connects. If present, the string provides information that helps the plugin determine how to map the proxy (external) client user name to a proxied user name. It is up to each plugin whether it requires the `AS` clause. If so, the format of the authentication string depends on how the plugin intends to use it. Consult the documentation for a given plugin for information about the authentication string values it accepts.

Preventing Direct Login to Proxied Accounts

Proxied accounts generally are intended to be used only by means of proxy accounts. That is, clients connect using a proxy account, then are mapped onto and assume the privileges of the appropriate proxied user.

To ensure that a proxied account cannot be used directly, create the account with a password but do not tell anyone else the password. If you do not let anyone know the password for the account, clients cannot use it to connect directly to the MySQL server.

Granting and Revoking the PROXY Privilege

The `PROXY` privilege is needed to enable an external user to connect as and have the privileges of another user. To grant this privilege, use the `GRANT` statement. For example:

```
GRANT PROXY ON 'proxied_user' TO 'proxy_user';
```

The statement creates a row in the `mysql.proxies_priv` grant table.

At connect time, *`proxy_user`* must represent a valid externally authenticated MySQL user, and *`proxied_user`* must represent a valid locally authenticated user. Otherwise, the connection attempt fails.

The corresponding `REVOKE` syntax is:

```
REVOKE PROXY ON 'proxied_user' FROM 'proxy_user';
```

MySQL `GRANT` and `REVOKE` syntax extensions work as usual. Examples:

```
-- grant PROXY to multiple accounts
GRANT PROXY ON 'a' TO 'b', 'c', 'd';

-- revoke PROXY from multiple accounts
REVOKE PROXY ON 'a' FROM 'b', 'c', 'd';

-- grant PROXY to an account and enable the account to grant
-- PROXY to the proxied account
GRANT PROXY ON 'a' TO 'd' WITH GRANT OPTION;

-- grant PROXY to default proxy account
GRANT PROXY ON 'a' TO ''@'';
```

The `PROXY` privilege can be granted in these cases:

- By a user that has `GRANT PROXY ... WITH GRANT OPTION` for *`proxied_user`*.
- By *`proxied_user`* for itself: The value of `USER()` must exactly match `CURRENT_USER()` and *`proxied_user`*, for both the user name and host name parts of the account name.

The initial `root` account created during MySQL installation has the `PROXY ... WITH GRANT OPTION` privilege for `''@''`, that is, for all users and all hosts. This enables `root` to set up proxy users, as well as to delegate to other accounts the authority to set up proxy users. For example, `root` can do this:

```
CREATE USER 'admin'@'localhost'
  IDENTIFIED BY 'admin_password';
GRANT PROXY
  ON ''@''
  TO 'admin'@'localhost'
  WITH GRANT OPTION;
```

Those statements create an `admin` user that can manage all `GRANT PROXY` mappings. For example, `admin` can do this:

```
GRANT PROXY ON sally TO joe;
```

Default Proxy Users

To specify that some or all users should connect using a given authentication plugin, create a “blank” MySQL account with an empty user name and host name (`''@''`), associate it with that plugin, and let the plugin return the real authenticated user name (if different from the blank user). Suppose that there exists a plugin named `ldap_auth` that implements LDAP authentication and maps connecting users onto either a developer or manager account. To set up proxying of users onto these accounts, use the following statements:

```
-- create default proxy account
CREATE USER ''@''
  IDENTIFIED WITH ldap_auth
  AS 'O=Oracle, OU=MySQL';

-- create proxied accounts
CREATE USER 'developer'@'localhost'
  IDENTIFIED BY 'developer_password';
CREATE USER 'manager'@'localhost'
  IDENTIFIED BY 'manager_password';

-- grant to default proxy account the
-- PROXY privilege for proxied accounts
GRANT PROXY
  ON 'manager'@'localhost'
  TO ''@'';
GRANT PROXY
  ON 'developer'@'localhost'
  TO ''@'';
```

As with proxied accounts created in previous examples, the password should be kept secret so that clients cannot use the accounts to log in directly to the MySQL server.

Now assume that a client connects as follows:

```
shell> mysql --user=myuser --password ...
Enter password: myuser_password
```

The server does not find `myuser` defined as a MySQL user. But because there is a blank user account (`''@''`) that matches the client user name and host name, the server authenticates the client against that account: The server invokes the `ldap_auth` authentication plugin and passes `myuser` and *`myuser_password`* to it as the user name and password.

If the `ldap_auth` plugin finds in the LDAP directory that *`myuser_password`* is not the correct password for `myuser`, authentication fails and the server rejects the connection.

If the password is correct and `ldap_auth` finds that `myuser` is a developer, it returns the user name `developer` to the MySQL server, rather than `myuser`. Returning a user name different from the client user name of `myuser` signals to the server that it should treat `myuser` as a proxy. The server verifies that `''@''` can authenticate as `developer` (because `''@''` has the `PROXY` privilege to do so) and accepts the connection. The session proceeds with `myuser` having the privileges of the `developer` proxied user. (These privileges should be set up by the DBA using `GRANT` statements, not shown.) The `USER()` and `CURRENT_USER()` functions return these values:

```
mysql> SELECT USER(), CURRENT_USER();
+------------------+--------------------+
| USER()           | CURRENT_USER()     |
+------------------+--------------------+
| myuser@localhost | developer@localhost |
+------------------+--------------------+
```

If the plugin instead finds in the LDAP directory that `myuser` is a manager, it returns `manager` as the user name and the session proceeds with `myuser` having the privileges of the `manager` proxied user.

```
mysql> SELECT USER(), CURRENT_USER();
+------------------+------------------+
| USER()           | CURRENT_USER()   |
+------------------+------------------+
| myuser@localhost | manager@localhost |
+------------------+------------------+
```

For simplicity, external authentication cannot be multilevel: Neither the credentials for `developer` nor those for `manager` are taken into account in the preceding example. However, they are still used if a client tries to connect and authenticate directly as the `developer` or `manager` account, which is why those proxied accounts should be protected against direct login (see Preventing Direct Login to Proxied Accounts).

Default Proxy User and Anonymous User Conflicts

If you intend to create a default proxy user, check for other existing “match any user” accounts that take precedence over the default proxy user because they can prevent that user from working as intended.

In the preceding discussion, the default proxy user account has `''` in the host part, which matches any host. If you set up a default proxy user, take care to also check whether nonproxy accounts exist with the same user part and `'%'` in the host part, because `'%'` also matches any host, but has precedence over `''` by the rules that the server uses to sort account rows internally (see Section 6.2.5, “Access Control, Stage 1: Connection Verification”).

Suppose that a MySQL installation includes these two accounts:

```
-- create default proxy account
CREATE USER ''@''
  IDENTIFIED WITH some_plugin
  AS 'some_auth_string';
-- create anonymous account
```

```
CREATE USER ''@'%'
  IDENTIFIED BY 'anon_user_password';
```

The first account (`''@''`) is intended as the default proxy user, used to authenticate connections for users who do not otherwise match a more-specific account. The second account (`''@'%'`) is an anonymous-user account, which might have been created, for example, to enable users without their own account to connect anonymously.

Both accounts have the same user part (`''`), which matches any user. And each account has a host part that matches any host. Nevertheless, there is a priority in account matching for connection attempts because the matching rules sort a host of `'%'` ahead of `''`. For accounts that do not match any more-specific account, the server attempts to authenticate them against `''@'%'` (the anonymous user) rather than `''@''` (the default proxy user). As a result, the default proxy account is never used.

To avoid this problem, use one of the following strategies:

- Remove the anonymous account so that it does not conflict with the default proxy user.
- Use a more-specific default proxy user that matches ahead of the anonymous user. For example, to permit only `localhost` proxy connections, use `''@'localhost'`:

  ```
  CREATE USER ''@'localhost'
    IDENTIFIED WITH some_plugin
    AS 'some_auth_string';
  ```

  In addition, modify any `GRANT PROXY` statements to name `''@'localhost'` rather than `''@''` as the proxy user.

  Be aware that this strategy prevents anonymous-user connections from `localhost`.

- Use a named default account rather than an anonymous default account. For an example of this technique, consult the instructions for using the `authentication_windows` plugin. See Section 6.4.1.7, “Windows Pluggable Authentication”.
- Create multiple proxy users, one for local connections and one for “everything else” (remote connections). This can be useful particularly when local users should have different privileges from remote users.

  Create the proxy users:

  ```
  -- create proxy user for local connections
  CREATE USER ''@'localhost'
    IDENTIFIED WITH some_plugin
    AS 'some_auth_string';
  -- create proxy user for remote connections
  CREATE USER ''@'%'
    IDENTIFIED WITH some_plugin
    AS 'some_auth_string';
  ```

  Create the proxied users:

  ```
  -- create proxied user for local connections
  CREATE USER 'developer'@'localhost'
    IDENTIFIED BY 'some_password';
  -- create proxied user for remote connections
  CREATE USER 'developer'@'%'
    IDENTIFIED BY 'some_password';
  ```

  Grant to each proxy account the `PROXY` privilege for the corresponding proxied account:

  ```
  GRANT PROXY
    ON 'developer'@'localhost'
    TO ''@'localhost';
  GRANT PROXY
    ON 'developer'@'%'
    TO ''@'%';
  ```

Finally, grant appropriate privileges to the local and remote proxied users (not shown).

Assume that the `some_plugin`/`'some_auth_string'` combination causes `some_plugin` to map the client user name to `developer`. Local connections match the `''@'localhost'` proxy user, which maps to the `'developer'@'localhost'` proxied user. Remote connections match the `''@'%'` proxy user, which maps to the `'developer'@'%'` proxied user.

Proxy User System Variables

Two system variables help trace the proxy login process:

- `proxy_user`: This value is `NULL` if proxying is not used. Otherwise, it indicates the proxy user account. For example, if a client authenticates through the `''@''` proxy account, this variable is set as follows:

```
mysql> SELECT @@proxy_user;
+--------------+
| @@proxy_user |
+--------------+
| ''@''        |
+--------------+
```

- `external_user`: Sometimes the authentication plugin may use an external user to authenticate to the MySQL server. For example, when using Windows native authentication, a plugin that authenticates using the windows API does not need the login ID passed to it. However, it still uses a Windows user ID to authenticate. The plugin may return this external user ID (or the first 512 UTF-8 bytes of it) to the server using the `external_user` read-only session variable. If the plugin does not set this variable, its value is `NULL`.

6.2.13 Setting Account Resource Limits

One means of restricting client use of MySQL server resources is to set the global `max_user_connections` system variable to a nonzero value. This limits the number of simultaneous connections that can be made by any given account, but places no limits on what a client can do once connected. In addition, setting `max_user_connections` does not enable management of individual accounts. Both types of control are of interest to MySQL administrators.

To address such concerns, MySQL permits limits for individual accounts on use of these server resources:

- The number of queries an account can issue per hour
- The number of updates an account can issue per hour
- The number of times an account can connect to the server per hour
- The number of simultaneous connections to the server by an account

Any statement that a client can issue counts against the query limit, unless its results are served from the query cache. Only statements that modify databases or tables count against the update limit.

An “account” in this context corresponds to a row in the `mysql.user` system table. That is, a connection is assessed against the `User` and `Host` values in the `user` table row that applies to the connection. For example, an account `'usera'@'%.example.com'` corresponds to a row in the `user` table that has `User` and `Host` values of `usera` and `%.example.com`, to permit `usera` to connect from any host in the `example.com` domain. In this case, the server applies resource limits in this row collectively to all connections by `usera` from any host in the `example.com` domain because all such connections use the same account.

Before MySQL 5.0, an “account” was assessed against the actual host from which a user connects. This older method of accounting may be selected by starting the server with the `--old-style-`

`user-limits` option. In this case, if `usera` connects simultaneously from `host1.example.com` and `host2.example.com`, the server applies the account resource limits separately to each connection. If `usera` connects again from `host1.example.com`, the server applies the limits for that connection together with the existing connection from that host.

To establish resource limits for an account, use the `GRANT` statement (see Section 13.7.1.4, "GRANT Statement"). Provide a `WITH` clause that names each resource to be limited. The default value for each limit is zero (no limit). For example, to create a new account that can access the `customer` database, but only in a limited fashion, issue these statements:

```
mysql> CREATE USER 'francis'@'localhost' IDENTIFIED BY 'frank';
mysql> GRANT ALL ON customer.* TO 'francis'@'localhost'
    ->     WITH MAX_QUERIES_PER_HOUR 20
    ->          MAX_UPDATES_PER_HOUR 10
    ->          MAX_CONNECTIONS_PER_HOUR 5
    ->          MAX_USER_CONNECTIONS 2;
```

The limit types need not all be named in the `WITH` clause, but those named can be present in any order. The value for each per-hour limit should be an integer representing a count per hour. For `MAX_USER_CONNECTIONS`, the limit is an integer representing the maximum number of simultaneous connections by the account. If this limit is set to zero, the global `max_user_connections` system variable value determines the number of simultaneous connections. If `max_user_connections` is also zero, there is no limit for the account.

To modify limits for an existing account, use a `GRANT USAGE` statement at the global level (`ON *.*`). The following statement changes the query limit for `francis` to 100:

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'
    ->     WITH MAX_QUERIES_PER_HOUR 100;
```

The statement modifies only the limit value specified and leaves the account otherwise unchanged.

To remove a limit, set its value to zero. For example, to remove the limit on how many times per hour `francis` can connect, use this statement:

```
mysql> GRANT USAGE ON *.* TO 'francis'@'localhost'
    ->     WITH MAX_CONNECTIONS_PER_HOUR 0;
```

As mentioned previously, the simultaneous-connection limit for an account is determined from the `MAX_USER_CONNECTIONS` limit and the `max_user_connections` system variable. Suppose that the global `max_user_connections` value is 10 and three accounts have individual resource limits specified as follows:

```
GRANT ... TO 'user1'@'localhost' WITH MAX_USER_CONNECTIONS 0;
GRANT ... TO 'user2'@'localhost' WITH MAX_USER_CONNECTIONS 5;
GRANT ... TO 'user3'@'localhost' WITH MAX_USER_CONNECTIONS 20;
```

`user1` has a connection limit of 10 (the global `max_user_connections` value) because it has a `MAX_USER_CONNECTIONS` limit of zero. `user2` and `user3` have connection limits of 5 and 20, respectively, because they have nonzero `MAX_USER_CONNECTIONS` limits.

The server stores resource limits for an account in the `user` table row corresponding to the account. The `max_questions`, `max_updates`, and `max_connections` columns store the per-hour limits, and the `max_user_connections` column stores the `MAX_USER_CONNECTIONS` limit. (See Section 6.2.3, "Grant Tables".)

Resource-use counting takes place when any account has a nonzero limit placed on its use of any of the resources.

As the server runs, it counts the number of times each account uses resources. If an account reaches its limit on number of connections within the last hour, the server rejects further connections for the account until that hour is up. Similarly, if the account reaches its limit on the number of queries or updates, the server rejects further queries or updates until the hour is up. In all such cases, the server issues appropriate error messages.

Resource counting occurs per account, not per client. For example, if your account has a query limit of 50, you cannot increase your limit to 100 by making two simultaneous client connections to the server. Queries issued on both connections are counted together.

The current per-hour resource-use counts can be reset globally for all accounts, or individually for a given account:

- To reset the current counts to zero for all accounts, issue a `FLUSH USER_RESOURCES` statement. The counts also can be reset by reloading the grant tables (for example, with a `FLUSH PRIVILEGES` statement or a `mysqladmin reload` command).
- The counts for an individual account can be reset to zero by setting any of its limits again. Specify a limit value equal to the value currently assigned to the account.

Per-hour counter resets do not affect the `MAX_USER_CONNECTIONS` limit.

All counts begin at zero when the server starts. Counts do not carry over through server restarts.

For the `MAX_USER_CONNECTIONS` limit, an edge case can occur if the account currently has open the maximum number of connections permitted to it: A disconnect followed quickly by a connect can result in an error (`ER_TOO_MANY_USER_CONNECTIONS` or `ER_USER_LIMIT_REACHED`) if the server has not fully processed the disconnect by the time the connect occurs. When the server finishes disconnect processing, another connection is once more permitted.

6.2.14 Troubleshooting Problems Connecting to MySQL

If you encounter problems when you try to connect to the MySQL server, the following items describe some courses of action you can take to correct the problem.

- Make sure that the server is running. If it is not, clients cannot connect to it. For example, if an attempt to connect to the server fails with a message such as one of those following, one cause might be that the server is not running:

```
shell> mysql
ERROR 2003: Can't connect to MySQL server on 'host_name' (111)
shell> mysql
ERROR 2002: Can't connect to local MySQL server through socket
'/tmp/mysql.sock' (111)
```

- It might be that the server is running, but you are trying to connect using a TCP/IP port, named pipe, or Unix socket file different from the one on which the server is listening. To correct this when you invoke a client program, specify a `--port` option to indicate the proper port number, or a `--socket` option to indicate the proper named pipe or Unix socket file. To find out where the socket file is, you can use this command:

```
shell> netstat -ln | grep mysql
```

- Make sure that the server has not been configured to ignore network connections or (if you are attempting to connect remotely) that it has not been configured to listen only locally on its network interfaces. If the server was started with the `skip_networking` system variable enabled, it does not accept TCP/IP connections at all. If the server was started with the `bind_address` system variable set to `127.0.0.1`, it listens for TCP/IP connections only locally on the loopback interface and does not accept remote connections.
- Check to make sure that there is no firewall blocking access to MySQL. Your firewall may be configured on the basis of the application being executed, or the port number used by MySQL for communication (3306 by default). Under Linux or Unix, check your IP tables (or similar) configuration to ensure that the port has not been blocked. Under Windows, applications such as ZoneAlarm or Windows Firewall may need to be configured not to block the MySQL port.
- The grant tables must be properly set up so that the server can use them for access control. For some distribution types (such as binary distributions on Windows, or RPM distributions on Linux),

the installation process initializes the MySQL data directory, including the `mysql` system database containing the grant tables. For distributions that do not do this, you must initialize the grant tables manually. For details, see Section 2.10, “Postinstallation Setup and Testing”.

To determine whether you need to initialize the grant tables, look for a `mysql` directory under the data directory. (The data directory normally is named `data` or `var` and is located under your MySQL installation directory.) Make sure that you have a file named `user.MYD` in the `mysql` database directory. If not, execute the `mysql_install_db` program. After running this program and starting the server, test the initial privileges by executing this command:

```
shell> mysql -u root
```

The server should let you connect without error.

- After a fresh installation, you should connect to the server and set up your users and their access permissions:

```
shell> mysql -u root mysql
```

The server should let you connect because the MySQL `root` user has no password initially. That is also a security risk, so setting the password for the `root` accounts is something you should do while you're setting up your other MySQL accounts. For instructions on setting the initial passwords, see Section 2.10.4, “Securing the Initial MySQL Accounts”.

- If you have updated an existing MySQL installation to a newer version, did you perform the MySQL upgrade procedure? If not, do so. The structure of the grant tables changes occasionally when new capabilities are added, so after an upgrade you should always make sure that your tables have the current structure. For instructions, see Section 2.11, “Upgrading MySQL”.

- If a client program receives the following error message when it tries to connect, it means that the server expects passwords in a newer format than the client is capable of generating:

```
shell> mysql
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

For information on how to deal with this, see Section 6.1.2.4, “Password Hashing in MySQL”, and Section B.3.2.4, “Client does not support authentication protocol”.

- Remember that client programs use connection parameters specified in option files or environment variables. If a client program seems to be sending incorrect default connection parameters when you have not specified them on the command line, check any applicable option files and your environment. For example, if you get `Access denied` when you run a client without any options, make sure that you have not specified an old password in any of your option files!

  You can suppress the use of option files by a client program by invoking it with the `--no-defaults` option. For example:

```
shell> mysqladmin --no-defaults -u root version
```

The option files that clients use are listed in Section 4.2.2.2, “Using Option Files”. Environment variables are listed in Section 4.9, “Environment Variables”.

- If you get the following error, it means that you are using an incorrect `root` password:

```
shell> mysqladmin -u root -pxxxx ver
Access denied for user 'root'@'localhost' (using password: YES)
```

If the preceding error occurs even when you have not specified a password, it means that you have an incorrect password listed in some option file. Try the `--no-defaults` option as described in the previous item.

For information on changing passwords, see Section 6.2.9, “Assigning Account Passwords”.

If you have lost or forgotten the `root` password, see Section B.3.3.2, “How to Reset the Root Password”.

- If you change a password by using `SET PASSWORD`, `INSERT`, or `UPDATE`, you must encrypt the password using the `PASSWORD()` function. If you do not use `PASSWORD()` for these statements, the password does not work. For example, the following statement assigns a password, but fails to encrypt it, so the user is not able to connect afterward:

```
SET PASSWORD FOR 'abe'@'host_name' = 'eagle';
```

Instead, set the password like this:

```
SET PASSWORD FOR 'abe'@'host_name' = PASSWORD('eagle');
```

The `PASSWORD()` function is unnecessary when you specify a password using the `CREATE USER` or `GRANT` statements or the `mysqladmin password` command. Each of those automatically uses `PASSWORD()` to encrypt the password. See Section 6.2.9, “Assigning Account Passwords”, and Section 13.7.1.2, “CREATE USER Statement”.

- `localhost` is a synonym for your local host name, and is also the default host to which clients try to connect if you specify no host explicitly.

  You can use a `--host=127.0.0.1` option to name the server host explicitly. This makes a TCP/IP connection to the local `mysqld` server. You can also use TCP/IP by specifying a `--host` option that uses the actual host name of the local host. In this case, the host name must be specified in a `user` table row on the server host, even though you are running the client program on the same host as the server.

- The `Access denied` error message tells you who you are trying to log in as, the client host from which you are trying to connect, and whether you were using a password. Normally, you should have one row in the `user` table that exactly matches the host name and user name that were given in the error message. For example, if you get an error message that contains `using password: NO`, it means that you tried to log in without a password.

- If you get an `Access denied` error when trying to connect to the database with `mysql -u user_name`, you may have a problem with the `user` table. Check this by executing `mysql -u root mysql` and issuing this SQL statement:

```
SELECT * FROM user;
```

The result should include a row with the `Host` and `User` columns matching your client's host name and your MySQL user name.

- If the following error occurs when you try to connect from a host other than the one on which the MySQL server is running, it means that there is no row in the `user` table with a `Host` value that matches the client host:

```
Host ... is not allowed to connect to this MySQL server
```

You can fix this by setting up an account for the combination of client host name and user name that you are using when trying to connect.

If you do not know the IP address or host name of the machine from which you are connecting, you should put a row with `'%'` as the `Host` column value in the `user` table. After trying to connect from the client machine, use a `SELECT USER()` query to see how you really did connect. Then change the `'%'` in the `user` table row to the actual host name that shows up in the log. Otherwise, your system is left insecure because it permits connections from any host for the given user name.

On Linux, another reason that this error might occur is that you are using a binary MySQL version that is compiled with a different version of the `glibc` library than the one you are using. In this case, you should either upgrade your operating system or `glibc`, or download a source distribution of

MySQL version and compile it yourself. A source RPM is normally trivial to compile and install, so this is not a big problem.

- If you specify a host name when trying to connect, but get an error message where the host name is not shown or is an IP address, it means that the MySQL server got an error when trying to resolve the IP address of the client host to a name:

```
shell> mysqladmin -u root -pxxxx -h some_hostname ver
Access denied for user 'root'@'' (using password: YES)
```

If you try to connect as `root` and get the following error, it means that you do not have a row in the `user` table with a `User` column value of `'root'` and that `mysqld` cannot resolve the host name for your client:

```
Access denied for user ''@'unknown'
```

These errors indicate a DNS problem. To fix it, execute `mysqladmin flush-hosts` to reset the internal DNS host cache. See Section 5.1.11.2, “DNS Lookups and the Host Cache”.

Some permanent solutions are:

- Determine what is wrong with your DNS server and fix it.
- Specify IP addresses rather than host names in the MySQL grant tables.
- Put an entry for the client machine name in `/etc/hosts` on Unix or `\windows\hosts` on Windows.
- Start `mysqld` with the `skip_name_resolve` system variable enabled.
- Start `mysqld` with the `--skip-host-cache` option.
- On Unix, if you are running the server and the client on the same machine, connect to `localhost`. For connections to `localhost`, MySQL programs attempt to connect to the local server by using a Unix socket file, unless there are connection parameters specified to ensure that the client makes a TCP/IP connection. For more information, see Section 4.2.4, “Connecting to the MySQL Server Using Command Options”.
- On Windows, if you are running the server and the client on the same machine and the server supports named pipe connections, connect to the host name `.` (period). Connections to `.` use a named pipe rather than TCP/IP.

- If `mysql -u root` works but `mysql -h your_hostname -u root` results in `Access denied` (where *`your_hostname`* is the actual host name of the local host), you may not have the correct name for your host in the `user` table. A common problem here is that the `Host` value in the `user` table row specifies an unqualified host name, but your system's name resolution routines return a fully qualified domain name (or vice versa). For example, if you have a row with host `'pluto'` in the `user` table, but your DNS tells MySQL that your host name is `'pluto.example.com'`, the row does not work. Try adding a row to the `user` table that contains the IP address of your host as the `Host` column value. (Alternatively, you could add a row to the `user` table with a `Host` value that contains a wildcard (for example, `'pluto.%'`). However, use of `Host` values ending with `%` is *insecure* and is *not* recommended!)
- If `mysql -u user_name` works but `mysql -u user_name some_db` does not, you have not granted access to the given user for the database named *`some_db`*.
- If `mysql -u user_name` works when executed on the server host, but `mysql -h host_name -u user_name` does not work when executed on a remote client host, you have not enabled access to the server for the given user name from the remote host.
- If you cannot figure out why you get `Access denied`, remove from the `user` table all rows that have `Host` values containing wildcards (rows that contain `'%'` or `'_'` characters). A very common

error is to insert a new row with `Host='%'` and `User='some_user'`, thinking that this enables you to specify `localhost` to connect from the same machine. The reason that this does not work is that the default privileges include a row with `Host='localhost'` and `User=''`. Because that row has a `Host` value `'localhost'` that is more specific than `'%'`, it is used in preference to the new row when connecting from `localhost`! The correct procedure is to insert a second row with `Host='localhost'` and `User='some_user'`, or to delete the row with `Host='localhost'` and `User=''`. After deleting the row, remember to issue a `FLUSH PRIVILEGES` statement to reload the grant tables. See also Section 6.2.5, "Access Control, Stage 1: Connection Verification".

- If you are able to connect to the MySQL server, but get an `Access denied` message whenever you issue a `SELECT ... INTO OUTFILE` or `LOAD DATA` statement, your row in the `user` table does not have the `FILE` privilege enabled.
- If you change the grant tables directly (for example, by using `INSERT`, `UPDATE`, or `DELETE` statements) and your changes seem to be ignored, remember that you must execute a `FLUSH PRIVILEGES` statement or a `mysqladmin flush-privileges` command to cause the server to reload the privilege tables. Otherwise, your changes have no effect until the next time the server is restarted. Remember that after you change the `root` password with an `UPDATE` statement, you do not need to specify the new password until after you flush the privileges, because the server does not yet know that you have changed the password.
- If your privileges seem to have changed in the middle of a session, it may be that a MySQL administrator has changed them. Reloading the grant tables affects new client connections, but it also affects existing connections as indicated in Section 6.2.8, "When Privilege Changes Take Effect".
- If you have access problems with a Perl, PHP, Python, or ODBC program, try to connect to the server with `mysql -u user_name db_name` or `mysql -u user_name -ppassword db_name`. If you are able to connect using the `mysql` client, the problem lies with your program, not with the access privileges. (There is no space between `-p` and the password; you can also use the `--password=password` syntax to specify the password. If you use the `-p` or `--password` option with no password value, MySQL prompts you for the password.)
- For testing purposes, start the `mysqld` server with the `--skip-grant-tables` option. Then you can change the MySQL grant tables and use the `SHOW GRANTS` statement to check whether your modifications have the desired effect. When you are satisfied with your changes, execute `mysqladmin flush-privileges` to tell the `mysqld` server to reload the privileges. This enables you to begin using the new grant table contents without stopping and restarting the server.
- If everything else fails, start the `mysqld` server with a debugging option (for example, `--debug=d,general,query`). This prints host and user information about attempted connections, as well as information about each command issued. See Section 5.8.3, "The DBUG Package".
- If you have any other problems with the MySQL grant tables and ask on the MySQL Community Slack, always provide a dump of the MySQL grant tables. You can dump the tables with the `mysqldump mysql` command. To file a bug report, see the instructions at Section 1.6, "How to Report Bugs or Problems". In some cases, you may need to restart `mysqld` with `--skip-grant-tables` to run `mysqldump`.

6.2.15 SQL-Based Account Activity Auditing

Applications can use the following guidelines to perform SQL-based auditing that ties database activity to MySQL accounts.

MySQL accounts correspond to rows in the `mysql.user` system table. When a client connects successfully, the server authenticates the client to a particular row in this table. The `User` and `Host` column values in this row uniquely identify the account and correspond to the `'user_name'@'host_name'` format in which account names are written in SQL statements.

The account used to authenticate a client determines which privileges the client has. Normally, the `CURRENT_USER()` function can be invoked to determine which account this is for the client user. Its value is constructed from the `User` and `Host` columns of the `user` table row for the account.

However, there are circumstances under which the `CURRENT_USER()` value corresponds not to the client user but to a different account. This occurs in contexts when privilege checking is not based the client's account:

- Stored routines (procedures and functions) defined with the `SQL SECURITY DEFINER` characteristic
- Views defined with the `SQL SECURITY DEFINER` characteristic
- Triggers and events

In those contexts, privilege checking is done against the `DEFINER` account and `CURRENT_USER()` refers to that account, not to the account for the client who invoked the stored routine or view or who caused the trigger to activate. To determine the invoking user, you can call the `USER()` function, which returns a value indicating the actual user name provided by the client and the host from which the client connected. However, this value does not necessarily correspond directly to an account in the `user` table, because the `USER()` value never contains wildcards, whereas account values (as returned by `CURRENT_USER()`) may contain user name and host name wildcards.

For example, a blank user name matches any user, so an account of `''@'localhost'` enables clients to connect as an anonymous user from the local host with any user name. In this case, if a client connects as `user1` from the local host, `USER()` and `CURRENT_USER()` return different values:

```
mysql> SELECT USER(), CURRENT_USER();
+----------------+----------------+
| USER()         | CURRENT_USER() |
+----------------+----------------+
| user1@localhost | @localhost     |
+----------------+----------------+
```

The host name part of an account can contain wildcards, too. If the host name contains a `'%'` or `'_'` pattern character or uses netmask notation, the account can be used for clients connecting from multiple hosts and the `CURRENT_USER()` value does not indicate which one. For example, the account `'user2'@'%.example.com'` can be used by `user2` to connect from any host in the `example.com` domain. If `user2` connects from `remote.example.com`, `USER()` and `CURRENT_USER()` return different values:

```
mysql> SELECT USER(), CURRENT_USER();
+--------------------------+--------------------+
| USER()                   | CURRENT_USER()     |
+--------------------------+--------------------+
| user2@remote.example.com | user2@%.example.com |
+--------------------------+--------------------+
```

If an application must invoke `USER()` for user auditing (for example, if it does auditing from within triggers) but must also be able to associate the `USER()` value with an account in the `user` table, it is necessary to avoid accounts that contain wildcards in the `User` or `Host` column. Specifically, do not permit `User` to be empty (which creates an anonymous-user account), and do not permit pattern characters or netmask notation in `Host` values. All accounts must have a nonempty `User` value and literal `Host` value.

With respect to the previous examples, the `''@'localhost'` and `'user2'@'%.example.com'` accounts should be changed not to use wildcards:

```
RENAME USER ''@'localhost' TO 'user1'@'localhost';
RENAME USER 'user2'@'%.example.com' TO 'user2'@'remote.example.com';
```

If `user2` must be able to connect from several hosts in the `example.com` domain, there should be a separate account for each host.

To extract the user name or host name part from a `CURRENT_USER()` or `USER()` value, use the `SUBSTRING_INDEX()` function:

```
mysql> SELECT SUBSTRING_INDEX(CURRENT_USER(),'@',1);
+---------------------------------------+
| SUBSTRING_INDEX(CURRENT_USER(),'@',1) |
+---------------------------------------+
| user1                                 |
+---------------------------------------+

mysql> SELECT SUBSTRING_INDEX(CURRENT_USER(),'@',-1);
+----------------------------------------+
| SUBSTRING_INDEX(CURRENT_USER(),'@',-1) |
+----------------------------------------+
| localhost                              |
+----------------------------------------+
```

6.3 Using Encrypted Connections

With an unencrypted connection between the MySQL client and the server, someone with access to the network could watch all your traffic and inspect the data being sent or received between client and server.

When you must move information over a network in a secure fashion, an unencrypted connection is unacceptable. To make any kind of data unreadable, use encryption. Encryption algorithms must include security elements to resist many kinds of known attacks such as changing the order of encrypted messages or replaying data twice.

MySQL supports encrypted connections between clients and the server using the TLS (Transport Layer Security) protocol. TLS is sometimes referred to as SSL (Secure Sockets Layer) but MySQL does not actually use the SSL protocol for encrypted connections because its encryption is weak (see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”).

TLS uses encryption algorithms to ensure that data received over a public network can be trusted. It has mechanisms to detect data change, loss, or replay. TLS also incorporates algorithms that provide identity verification using the X.509 standard.

X.509 makes it possible to identify someone on the Internet. In basic terms, there should be some entity called a “Certificate Authority” (or CA) that assigns electronic certificates to anyone who needs them. Certificates rely on asymmetric encryption algorithms that have two encryption keys (a public key and a secret key). A certificate owner can present the certificate to another party as proof of identity. A certificate consists of its owner's public key. Any data encrypted using this public key can be decrypted only using the corresponding secret key, which is held by the owner of the certificate.

MySQL can be compiled for encrypted-connection support using OpenSSL or yaSSL. For a comparison of the two packages, see Section 6.3.4, “SSL Library-Dependent Capabilities” For information about the encryption protocols and ciphers each package supports, see Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.

Note

It is possible to compile MySQL using yaSSL as an alternative to OpenSSL only prior to MySQL 5.6.46. As of MySQL 5.6.46, support for yaSSL is removed and all MySQL builds use OpenSSL.

MySQL programs attempt to connect using encryption if the proper options are given and the server supports encrypted connections. For information about options that affect use of encrypted connections, see Section 6.3.1, “Configuring MySQL to Use Encrypted Connections” and Command Options for Encrypted Connections.

MySQL performs encryption on a per-connection basis, and use of encryption for a given user can be optional or mandatory. This enables you to choose an encrypted or unencrypted connection according to the requirements of individual applications. For information on how to require users to use encrypted connections, see the discussion of the `REQUIRE` clause of the `GRANT` statement in Section 13.7.1.4, “GRANT Statement”.

Encrypted connections are not used by default. For applications that require the security provided by encrypted connections, the extra computation to encrypt the data is worthwhile.

Encrypted connections can be used between source and replica replication servers. See Section 17.3.8, “Setting Up Replication to Use Encrypted Connections”.

For information about using encrypted connections from the MySQL C API, see C API Support for Encrypted Connections.

It is also possible to connect using encryption from within an SSH connection to the MySQL server host. For an example, see Section 6.3.5, “Connecting to MySQL Remotely from Windows with SSH”.

6.3.1 Configuring MySQL to Use Encrypted Connections

To enable encrypted connections, your MySQL distribution must be built with SSL support, as described in Section 2.9.6, “Configuring SSL Library Support”. In addition, several configuration parameters are available to indicate whether to use encrypted connections, and to specify the appropriate certificate and key files. This section provides general guidance about configuring the server and clients for encrypted connections:

- Server-Side Startup Configuration for Encrypted Connections
- Client-Side Configuration for Encrypted Connections

Encrypted connections also can be used in these contexts:

- Between source and replica servers. See Section 17.3.8, “Setting Up Replication to Use Encrypted Connections”.
- By client programs that are based on the MySQL C API. See C API Support for Encrypted Connections.

Note

If the server is compiled against OpenSSL, clients from MySQL 5.6 versions older than 5.6.17 are not able to connect to the server using encrypted connections if the client library is compiled using yaSSL. Either use a client and server compiled using the same SSL package, or upgrade to clients compiled against a client library version from MySQL 5.6.17 or higher.

Instructions for creating any required certificate and key files are available in Section 6.3.3, “Creating SSL and RSA Certificates and Keys”.

Server-Side Startup Configuration for Encrypted Connections

These system variables on the server side specify the certificate and key files the server uses when permitting clients to establish encrypted connections:

- `ssl_ca`: The path name of the Certificate Authority (CA) certificate file. (`ssl_capath` is similar but specifies the path name of a directory of CA certificate files.)
- `ssl_cert`: The path name of the server public key certificate file. This certificate can be sent to the client and authenticated against the CA certificate that it has.
- `ssl_key`: The path name of the server private key file.

For example, to enable the server for encrypted connections, start it with these lines in the `my.cnf` file, changing the file names as necessary:

```
[mysqld]
ssl_ca=ca.pem
ssl_cert=server-cert.pem
ssl_key=server-key.pem
```

Each certificate and key system variable names a file in PEM format. If you have a MySQL source distribution, you can test your setup using the demonstration certificate and key files in its `mysql-test/std_data` directory.

MySQL also provides these system variables for server-side encrypted-connection control:

- `ssl_cipher`: The list of permissible ciphers for connection encryption.
- `ssl_crl`: The path name of the file containing certificate revocation lists. (`ssl_crlpath` is similar but specifies the path name of a directory of certificate revocation-list files.)

Client-Side Configuration for Encrypted Connections

For a complete list of client options related to establishment of encrypted connections, see Command Options for Encrypted Connections.

These options on the client side identify the certificate and key files clients use when establishing encrypted connections to the server. They are similar to the `ssl_ca`, `ssl_cert`, and `ssl_key` system variables used on the server side, but `--ssl-cert` and `--ssl-key` identify the client public and private key:

- `--ssl-ca`: The path name of the Certificate Authority (CA) certificate file. This option, if used, must specify the same certificate used by the server. (`--ssl-capath` is similar but specifies the path name of a directory of CA certificate files.)
- `--ssl-cert`: The path name of the client public key certificate file.
- `--ssl-key`: The path name of the client private key file.

For additional security relative to that provided by the default encryption, clients can supply a CA certificate matching the one used by the server and enable host name identity verification. In this way, the server and client place their trust in the same CA certificate and the client verifies that the host to which it connected is the one intended:

- To specify the CA certificate, use `--ssl-ca` (or `--ssl-capath`).
- To enable host name identity verification as well, specify `--ssl-verify-server-cert`.
- To require an encrypted connection, specify `--ssl-mode=REQUIRED`.
- `--ssl-cipher`: The list of permissible ciphers for connection encryption.
- `--ssl-crl`: The path name of the file containing certificate revocation lists. (`--ssl-crlpath` is similar but specifies the path name of a directory of certificate revocation-list files.)

Depending on the encryption requirements of the MySQL account used by a client, the client may be required to specify certain options to connect using encryption to the MySQL server.

Suppose that you want to connect using an account that has no special encryption requirements or that was created using a `GRANT` statement that included the `REQUIRE SSL` clause. As a recommended set of encrypted-connection options, start the server with at least the `ssl_cert` and `ssl_key` system variables set, and invoke the client with the `--ssl-ca` (or `--ssl-capath`) option. A client can connect using encryption like this:

```
mysql --ssl-ca=ca.pem
```

To require that a client certificate also be specified, create the account using a `REQUIRE X509` clause. Then the client must also specify the proper client key and certificate files or the server rejects the connection (enter the command on a single line):

```
mysql --ssl-ca=ca.pem
      --ssl-cert=client-cert.pem
      --ssl-key=client-key.pem
```

For additional information about the `REQUIRE` clause, see Section 13.7.1.4, “GRANT Statement”.

To prevent use of encryption and override other `--ssl-xxx` options, invoke the client program with `--ssl=0` or a synonym (`--skip-ssl`, `--disable-ssl`):

```
mysql --ssl=0
```

To determine whether the current connection with the server uses encryption, check the session value of the `Ssl_cipher` status variable. If the value is empty, the connection is not encrypted. Otherwise, the connection is encrypted and the value indicates the encryption cipher. For example:

```
mysql> SHOW SESSION STATUS LIKE 'Ssl_cipher';
+---------------+--------------------+
| Variable_name | Value              |
+---------------+--------------------+
| Ssl_cipher    | DHE-RSA-AES256-SHA |
+---------------+--------------------+
```

For the `mysql` client, an alternative is to use the `STATUS` or `\s` command and check the `SSL` line:

```
mysql> \s
...
SSL: Not in use
...
```

Or:

```
mysql> \s
...
SSL: Cipher in use is DHE-RSA-AES256-SHA
...
```

6.3.2 Encrypted Connection TLS Protocols and Ciphers

MySQL supports multiple TLS protocols and ciphers, and enables configuring which ciphers to permit for encrypted connections. It is also possible to determine which protocol and cipher the current session uses.

- Supported Connection TLS Protocols
- Connection TLS Protocol Configuration
- Connection Cipher Configuration
- Connection TLS Protocol Negotiation
- Monitoring Current Client Session TLS Protocol and Cipher

Supported Connection TLS Protocols

MySQL supports encrypted connections using the TLSv1 protocol and (as of MySQL 5.6.46) TLSv1.1 and TLSv1.2, listed in order from less secure to more secure.

As of MySQL 5.6.23, SSL 2.0 and SSL 3.0 are explicitly disabled because they provide weak encryption.

The set of protocols actually permitted for connections is subject to multiple factors:

- MySQL configuration. Permitted TLS protocols on the server side and client side must include at least one protocol in common or connection attempts cannot negotiate a protocol to use. For details, see Connection TLS Protocol Negotiation.
- System-wide host configuration. The host system may permit only certain TLS protocols, which means that MySQL connections cannot use nonpermitted protocols even if MySQL itself permits them:

- Suppose that MySQL permits TLSv1, TLSv1.1, and TLSv1.2, but your host system configuration permits only connections that use TLSv1.2 or higher. In this case, you cannot establish MySQL connections that use TLSv1 or TLSv1.1, even though MySQL permits them, because the host system does not permit them.
- If MySQL permits TLSv1, TLSv1.1, and TLSv1.2, but your host system configuration permits only connections that use TLSv1.3 or higher, you cannot establish MySQL connections at all, because no protocol permitted by MySQL is permitted by the host system.

Workarounds for this issue include:

- Change the system-wide host configuration to permit additional TLS protocols. Consult your operating system documentation for instructions. For example, your system may have an `/etc/ssl/openssl.cnf` file that contains these lines to restrict TLS protocols to TLSv1.2 or higher:

  ```
  [system_default_sect]
  MinProtocol = TLSv1.2
  ```

  Changing the value to a lower protocol version or `None` makes the system more permissive. This workaround has the disadvantage that permitting lower (less secure) protocols may have adverse security consequences.
- If you cannot or prefer not to change the host system TLS configuration, change MySQL applications to use higher (more secure) TLS protocols that are permitted by the host system. This may not be possible for older versions of MySQL that support only lower protocol versions. For example, TLSv1 is the only supported protocol prior to MySQL 5.6.46, so attempts to connect to a pre-5.6.46 server fail even if the client is from a newer MySQL version that supports higher protocol versions. In such cases, an upgrade to a version of MySQL that supports additional TLS versions may be required.

- The SSL library. If the SSL library does not support a particular protocol, neither does MySQL, and any parts of the following discussion that specify that protocol do not apply.
  - When compiled using OpenSSL 1.0.1 or higher, MySQL supports the TLSv1, TLSv1.1, and TLSv1.2 protocols as of MySQL 5.6.46, and TLS1v1 prior to 5.6.46.
  - When compiled using yaSSL, MySQL supports the TLSv1 protocol.

Note

It is possible to compile MySQL using yaSSL as an alternative to OpenSSL only prior to MySQL 5.6.46. As of MySQL 5.6.46, support for yaSSL is removed and all MySQL builds use OpenSSL.

Connection TLS Protocol Configuration

MySQL supports encrypted connections using the TLSv1 protocol and (as of MySQL 5.6.46) TLSv1.1 and TLSv1.2. Protocol support is built in and cannot be configured as is possible in MySQL 5.7 and higher. The protocol chosen for a given connection depends on the negotiation process. See Connection TLS Protocol Negotiation.

Connection Cipher Configuration

A default set of ciphers applies to encrypted connections, which can be overridden by explicitly configuring the permitted ciphers. During connection establishment, both sides of a connection must permit some cipher in common or the connection fails. Of the permitted ciphers common to both sides, the SSL library chooses the one supported by the provided certificate that has the highest priority.

To specify a cipher or ciphers for encrypted connections, set the `ssl_cipher` system variable on the server side, and use the `--ssl-cipher` option for client programs.

For source/replica replication, the `MASTER_SSL_CIPHER` option for the `CHANGE MASTER TO` statement specifies which ciphers a replica server permits for connections to the source.

To determine which ciphers a given server supports, check the session value of the `Ssl_cipher_list` status variable:

```
SHOW SESSION STATUS LIKE 'Ssl_cipher_list';
```

The `Ssl_cipher_list` status variable lists the possible SSL ciphers (empty for non-SSL connections). The set of available ciphers depends on your MySQL version and whether MySQL was compiled using OpenSSL or yaSSL, and (for OpenSSL) the library version used to compile MySQL.

MySQL passes a default cipher list to the SSL library.

MySQL passes this default cipher list to OpenSSL:

```
AES256-GCM-SHA384
AES256-SHA
AES256-SHA256
CAMELLIA256-SHA
DES-CBC3-SHA
DHE-DSS-AES256-GCM-SHA384
DHE-DSS-AES256-SHA
DHE-DSS-AES256-SHA256
DHE-DSS-CAMELLIA256-SHA
DHE-RSA-AES256-GCM-SHA384
DHE-RSA-AES256-SHA
DHE-RSA-AES256-SHA256
DHE-RSA-CAMELLIA256-SHA
ECDH-ECDSA-AES256-GCM-SHA384
ECDH-ECDSA-AES256-SHA
ECDH-ECDSA-AES256-SHA384
ECDH-ECDSA-DES-CBC3-SHA
ECDH-RSA-AES256-GCM-SHA384
ECDH-RSA-AES256-SHA
ECDH-RSA-AES256-SHA384
ECDH-RSA-DES-CBC3-SHA
ECDHE-ECDSA-AES128-GCM-SHA256
ECDHE-ECDSA-AES128-SHA
ECDHE-ECDSA-AES128-SHA256
ECDHE-ECDSA-AES256-GCM-SHA384
ECDHE-ECDSA-AES256-SHA
ECDHE-ECDSA-AES256-SHA384
ECDHE-ECDSA-DES-CBC3-SHA
ECDHE-RSA-AES128-GCM-SHA256
ECDHE-RSA-AES128-SHA
ECDHE-RSA-AES128-SHA256
ECDHE-RSA-AES256-GCM-SHA384
ECDHE-RSA-AES256-SHA
ECDHE-RSA-AES256-SHA384
ECDHE-RSA-DES-CBC3-SHA
EDH-DSS-DES-CBC3-SHA
EDH-RSA-DES-CBC3-SHA
PSK-3DES-EDE-CBC-SHA
PSK-AES256-CBC-SHA
SRP-DSS-3DES-EDE-CBC-SHA
SRP-DSS-AES-128-CBC-SHA
SRP-DSS-AES-256-CBC-SHA
SRP-RSA-3DES-EDE-CBC-SHA
SRP-RSA-AES-128-CBC-S
SRP-RSA-AES-256-CBC-SHA
```

MySQL passes this default cipher list to yaSSL:

```
AES128-RMD
AES128-SHA
AES256-RMD
AES256-SHA
DES-CBC-SHA
DES-CBC3-RMD
DES-CBC3-SHA
```

```
DHE-RSA-AES128-RMD
DHE-RSA-AES128-SHA
DHE-RSA-AES256-RMD
DHE-RSA-AES256-SHA
DHE-RSA-DES-CBC3-RMD
EDH-RSA-DES-CBC-SHA
EDH-RSA-DES-CBC3-SHA
RC4-MD5
RC4-SHA
```

Connection TLS Protocol Negotiation

Connection attempts in MySQL negotiate use of the highest TLS protocol version available on both sides for which a protocol-compatible encryption cipher is available on both sides. The negotiation process depends on factors such as the SSL library used to compile the server and client, the TLS protocol and encryption cipher configuration, and which key size is used:

- For a connection attempt to succeed, the server and client TLS protocol configuration must permit some protocol in common.
- Similarly, the server and client encryption cipher configuration must permit some cipher in common. A given cipher may work only with particular TLS protocols, so a protocol available to the negotiation process is not chosen unless there is also a compatible cipher.
- If the server and client are compiled using OpenSSL, TLSv1.2 is used if possible. If either or both the server and client are compiled using yaSSL, only TLSv1 is possible. (“Possible” means that server and client configuration both must permit the indicated protocol, and both must also permit some protocol-compatible encryption cipher.) Otherwise, MySQL continues through the list of available protocols, proceeding from more secure protocols to less secure.
- TLSv1.2 does not work with all ciphers that have a key size of 512 bits or less. To use this protocol with such a key, set the `ssl_cipher` system variable on the server side or use the `--ssl-cipher` client option to specify the cipher name explicitly:

  ```
  AES128-SHA
  AES128-SHA256
  AES256-SHA
  AES256-SHA256
  CAMELLIA128-SHA
  CAMELLIA256-SHA
  DES-CBC3-SHA
  DHE-RSA-AES256-SHA
  RC4-MD5
  RC4-SHA
  SEED-SHA
  ```

- For better security, use a certificate with an RSA key size of at least 2048 bits.

If the server and client do not have a permitted protocol in common, and a protocol-compatible cipher in common, the server terminates the connection request. Examples:

- If the server is from MySQL 5.7 or higher and is configured with `tls_version=TLSv1.1,TLSv1.2`:
  - Connection attempts fail for pre-MySQL 5.6.46 clients that support only TLSv1.
  - Similarly, connection attempts fail for pre-MySQL 5.6.46 replicas that support only TLSv1.
- If the server is a pre-MySQL 5.6.46 server that supports only TLSv1:
  - Connection attempts fail for MySQL 5.7 or higher clients invoked with `--tls-version=TLSv1.1,TLSv1.2`.
  - Similarly, connection attempts fail for MySQL 5.7 or higher replicas configured with `MASTER_TLS_VERSION = 'TLSv1.1,TLSv1.2'`.

Monitoring Current Client Session TLS Protocol and Cipher

To determine which encryption TLS protocol and cipher the current client session uses, check the session values of the `Ssl_version` and `Ssl_cipher` status variables:

```
mysql> SHOW SESSION STATUS LIKE 'Ssl_version';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Ssl_version   | TLSv1 |
+---------------+-------+
mysql> SHOW SESSION STATUS LIKE 'Ssl_cipher';
+---------------+--------------------+
| Variable_name | Value              |
+---------------+--------------------+
| Ssl_cipher    | DHE-RSA-AES256-SHA |
+---------------+--------------------+
```

If the connection is not encrypted, both variables have an empty value.

6.3.3 Creating SSL and RSA Certificates and Keys

The following discussion describes how to create the files required for SSL and RSA support in MySQL. File creation is done by invoking the `openssl` command.

SSL certificate and key files enable MySQL to support encrypted connections using SSL. See Section 6.3, “Using Encrypted Connections”.

RSA key files enable MySQL to support secure password exchange over unencrypted connections for accounts authenticated by the `sha256_password` plugin. See Section 6.4.1.4, “SHA-256 Pluggable Authentication”.

6.3.3.1 Creating SSL Certificates and Keys Using openssl

This section describes how to use the `openssl` command to set up SSL certificate and key files for use by MySQL servers and clients. The first example shows a simplified procedure such as you might use from the command line. The second shows a script that contains more detail. The first two examples are intended for use on Unix and both use the `openssl` command that is part of OpenSSL. The third example describes how to set up SSL files on Windows.

Important

Whatever method you use to generate the certificate and key files, the Common Name value used for the server and client certificates/keys must each differ from the Common Name value used for the CA certificate. Otherwise, the certificate and key files do not work for servers compiled using OpenSSL. A typical error in this case is:

```
ERROR 2026 (HY000): SSL connection error:
error:00000001:lib(0):func(0):reason(1)
```

- Example 1: Creating SSL Files from the Command Line on Unix
- Example 2: Creating SSL Files Using a Script on Unix
- Example 3: Creating SSL Files on Windows

Example 1: Creating SSL Files from the Command Line on Unix

The following example shows a set of commands to create MySQL server and client certificate and key files. You must respond to several prompts by the `openssl` commands. To generate test files, you can press Enter to all prompts. To generate files for production use, you should provide nonempty responses.

```
# Create clean environment
rm -rf newcerts
```

```
mkdir newcerts && cd newcerts

# Create CA certificate
openssl genrsa 2048 > ca-key.pem
openssl req -new -x509 -nodes -days 3600 \
        -key ca-key.pem -out ca.pem

# Create server certificate, remove passphrase, and sign it
# server-cert.pem = public key, server-key.pem = private key
openssl req -newkey rsa:2048 -days 3600 \
        -nodes -keyout server-key.pem -out server-req.pem
openssl rsa -in server-key.pem -out server-key.pem
openssl x509 -req -in server-req.pem -days 3600 \
        -CA ca.pem -CAkey ca-key.pem -set_serial 01 -out server-cert.pem

# Create client certificate, remove passphrase, and sign it
# client-cert.pem = public key, client-key.pem = private key
openssl req -newkey rsa:2048 -days 3600 \
        -nodes -keyout client-key.pem -out client-req.pem
openssl rsa -in client-key.pem -out client-key.pem
openssl x509 -req -in client-req.pem -days 3600 \
        -CA ca.pem -CAkey ca-key.pem -set_serial 01 -out client-cert.pem
```

After generating the certificates, verify them:

```
openssl verify -CAfile ca.pem server-cert.pem client-cert.pem
```

You should see a response like this:

```
server-cert.pem: OK
client-cert.pem: OK
```

Now you have a set of files that can be used as follows:

- `ca.pem`: Use this to set the `ssl_ca` system variable on the server side and the `--ssl-ca` option on the client side. (The CA certificate, if used, must be the same on both sides.)
- `server-cert.pem`, `server-key.pem`: Use these to set the `ssl_cert` and `ssl_key` system variables on the server side.
- `client-cert.pem`, `client-key.pem`: Use these as the arguments to the `--ssl-cert` and `--ssl-key` options on the client side.

For additional usage instructions, see Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

Example 2: Creating SSL Files Using a Script on Unix

Here is an example script that shows how to set up SSL certificate and key files for MySQL. After executing the script, use the files for SSL connections as described in Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

```
DIR=`pwd`/openssl
PRIV=$DIR/private

mkdir $DIR $PRIV $DIR/newcerts
cp /usr/share/ssl/openssl.cnf $DIR
replace ./demoCA $DIR -- $DIR/openssl.cnf

# Create necessary files: $database, $serial and $new_certs_dir
# directory (optional)

touch $DIR/index.txt
echo "01" > $DIR/serial

#
# Generation of Certificate Authority(CA)
#

openssl req -new -x509 -keyout $PRIV/cakey.pem -out $DIR/ca.pem \
```

```
    -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/finley/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ................++++++
# .........++++++
# writing new private key to '/home/finley/openssl/private/cakey.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL admin
# Email Address []:

#
# Create server request and key
#
openssl req -new -keyout $DIR/server-key.pem -out \
    $DIR/server-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/finley/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ..++++++
# ..........++++++
# writing new private key to '/home/finley/openssl/server-key.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL server
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove the passphrase from the key
#
openssl rsa -in $DIR/server-key.pem -out $DIR/server-key.pem

#
# Sign server cert
#
openssl ca -cert $DIR/ca.pem -policy policy_anything \
```

```
    -out $DIR/server-cert.pem -config $DIR/openssl.cnf \
    -infiles $DIR/server-req.pem

# Sample output:
# Using configuration from /home/finley/openssl/openssl.cnf
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName           :PRINTABLE:'FI'
# organizationName      :PRINTABLE:'MySQL AB'
# commonName            :PRINTABLE:'MySQL admin'
# Certificate is to be certified until Sep 13 14:22:46 2003 GMT
# (365 days)
# Sign the certificate? [y/n]:y
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create client request and key
#
openssl req -new -keyout $DIR/client-key.pem -out \
    $DIR/client-req.pem -days 3600 -config $DIR/openssl.cnf

# Sample output:
# Using configuration from /home/finley/openssl/openssl.cnf
# Generating a 1024 bit RSA private key
# ..................................++++++
# ..........................................++++++
# writing new private key to '/home/finley/openssl/client-key.pem'
# Enter PEM pass phrase:
# Verifying password - Enter PEM pass phrase:
# -----
# You are about to be asked to enter information that will be
# incorporated into your certificate request.
# What you are about to enter is what is called a Distinguished Name
# or a DN.
# There are quite a few fields but you can leave some blank
# For some fields there will be a default value,
# If you enter '.', the field will be left blank.
# -----
# Country Name (2 letter code) [AU]:FI
# State or Province Name (full name) [Some-State]:.
# Locality Name (eg, city) []:
# Organization Name (eg, company) [Internet Widgits Pty Ltd]:MySQL AB
# Organizational Unit Name (eg, section) []:
# Common Name (eg, YOUR name) []:MySQL user
# Email Address []:
#
# Please enter the following 'extra' attributes
# to be sent with your certificate request
# A challenge password []:
# An optional company name []:

#
# Remove the passphrase from the key
#
openssl rsa -in $DIR/client-key.pem -out $DIR/client-key.pem

#
# Sign client cert
#

openssl ca -cert $DIR/ca.pem -policy policy_anything \
    -out $DIR/client-cert.pem -config $DIR/openssl.cnf \
    -infiles $DIR/client-req.pem

# Sample output:
# Using configuration from /home/finley/openssl/openssl.cnf
```

```
# Enter PEM pass phrase:
# Check that the request matches the signature
# Signature ok
# The Subjects Distinguished Name is as follows
# countryName           :PRINTABLE:'FI'
# organizationName      :PRINTABLE:'MySQL AB'
# commonName            :PRINTABLE:'MySQL user'
# Certificate is to be certified until Sep 13 16:45:17 2003 GMT
# (365 days)
# Sign the certificate? [y/n]:y
#
#
# 1 out of 1 certificate requests certified, commit? [y/n]y
# Write out database with 1 new entries
# Data Base Updated

#
# Create a my.cnf file that you can use to test the certificates
#

cat <<EOF > $DIR/my.cnf
[client]
ssl-ca=$DIR/ca.pem
ssl-cert=$DIR/client-cert.pem
ssl-key=$DIR/client-key.pem
[mysqld]
ssl_ca=$DIR/ca.pem
ssl_cert=$DIR/server-cert.pem
ssl_key=$DIR/server-key.pem
EOF
```

Example 3: Creating SSL Files on Windows

Download OpenSSL for Windows if it is not installed on your system. An overview of available packages can be seen here:

```
http://www.slproweb.com/products/Win32OpenSSL.html
```

Choose the Win32 OpenSSL Light or Win64 OpenSSL Light package, depending on your architecture (32-bit or 64-bit). The default installation location is `C:\OpenSSL-Win32` or `C:\OpenSSL-Win64`, depending on which package you downloaded. The following instructions assume a default location of `C:\OpenSSL-Win32`. Modify this as necessary if you are using the 64-bit package.

If a message occurs during setup indicating `'...critical component is missing: Microsoft Visual C++ 2008 Redistributables'`, cancel the setup and download one of the following packages as well, again depending on your architecture (32-bit or 64-bit):

- Visual C++ 2008 Redistributables (x86), available at:

  ```
  http://www.microsoft.com/downloads/details.aspx?familyid=9B2DA534-3E03-4391-8A4D-074B9F2BC1BF
  ```

- Visual C++ 2008 Redistributables (x64), available at:

  ```
  http://www.microsoft.com/downloads/details.aspx?familyid=bd2a6171-e2d6-4230-b809-9a8d7548c1b6
  ```

After installing the additional package, restart the OpenSSL setup procedure.

During installation, leave the default `C:\OpenSSL-Win32` as the install path, and also leave the default option `'Copy OpenSSL DLL files to the Windows system directory'` selected.

When the installation has finished, add `C:\OpenSSL-Win32\bin` to the Windows System Path variable of your server (depending on your version of Windows, the following path-setting instructions might differ slightly):

1. On the Windows desktop, right-click the **My Computer** icon, and select **Properties**.
2. Select the **Advanced** tab from the **System Properties** menu that appears, and click the **Environment Variables** button.

3. Under **System Variables**, select **Path**, then click the **Edit** button. The **Edit System Variable** dialogue should appear.

4. Add `';C:\OpenSSL-Win32\bin'` to the end (notice the semicolon).

5. Press OK 3 times.

6. Check that OpenSSL was correctly integrated into the Path variable by opening a new command console (`Start>Run>cmd.exe`) and verifying that OpenSSL is available:

```
Microsoft Windows [Version ...]
Copyright (c) 2006 Microsoft Corporation. All rights reserved.

C:\Windows\system32>cd \

C:\>openssl
OpenSSL> exit <<< If you see the OpenSSL prompt, installation was successful.

C:\>
```

After OpenSSL has been installed, use instructions similar to those from Example 1 (shown earlier in this section), with the following changes:

- Change the following Unix commands:

```
# Create clean environment
rm -rf newcerts
mkdir newcerts && cd newcerts
```

  On Windows, use these commands instead:

```
# Create clean environment
md c:\newcerts
cd c:\newcerts
```

- When a `'\'` character is shown at the end of a command line, this `'\'` character must be removed and the command lines entered all on a single line.

After generating the certificate and key files, to use them for SSL connections, see Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

6.3.3.2 Creating RSA Keys Using openssl

This section describes how to use the `openssl` command to set up the RSA key files that enable MySQL to support secure password exchange over unencrypted connections for accounts authenticated by the `sha256_password` plugin.

To create the RSA private and public key-pair files, run these commands while logged into the system account used to run the MySQL server so the files are owned by that account:

```
openssl genrsa -out private_key.pem 2048
openssl rsa -in private_key.pem -pubout -out public_key.pem
```

Those commands create 2,048-bit keys. To create stronger keys, use a larger value.

Then set the access modes for the key files. The private key should be readable only by the server, whereas the public key can be freely distributed to client users:

```
chmod 400 private_key.pem
chmod 444 public_key.pem
```

6.3.4 SSL Library-Dependent Capabilities

MySQL can be compiled using OpenSSL or yaSSL, both of which enable encrypted connections based on the OpenSSL API:

- MySQL Enterprise Edition binary distributions are compiled using OpenSSL. It is not possible to use yaSSL with MySQL Enterprise Edition.
- MySQL Community Edition binary distributions are compiled using yaSSL.
- MySQL Community Edition source distributions can be compiled using either OpenSSL or yaSSL (see Section 2.9.6, “Configuring SSL Library Support”).

Note

It is possible to compile MySQL using yaSSL as an alternative to OpenSSL only prior to MySQL 5.6.46. As of MySQL 5.6.46, support for yaSSL is removed and all MySQL builds use OpenSSL.

OpenSSL and yaSSL offer the same basic functionality, but MySQL distributions compiled using OpenSSL have additional features:

- OpenSSL supports a more flexible syntax for specifying ciphers (for the `ssl_cipher` system variable and `--ssl-cipher` client option), and supports a wider range of encryption ciphers from which to choose. See Command Options for Encrypted Connections, and Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”.
- OpenSSL supports the `ssl_capath` system variable and `--ssl-capath` client option. MySQL distributions compiled using yaSSL do not because yaSSL does not look in any directory and do not follow a chained certificate tree. yaSSL requires that all components of the CA certificate tree be contained within a single CA certificate tree and that each certificate in the file has a unique SubjectName value. To work around this limitation, concatenate the individual certificate files comprising the certificate tree into a new file and specify that file as the value of the `ssl_ca` system variable and `--ssl-ca` option.
- OpenSSL supports certificate revocation-list capability (for the `ssl_crl` and `ssl_crlpath` system variables and `--ssl-crl` and `--ssl-crlpath` client options). Distributions compiled using yaSSL do not because revocation lists do not work with yaSSL. (yaSSL accepts these options but silently ignores them.)
- Accounts that authenticate using the `sha256_password` plugin can use RSA key files for secure password exchange over unencrypted connections. See Section 6.4.1.4, “SHA-256 Pluggable Authentication”.
- OpenSSL supports more encryption modes for the `AES_ENCRYPT()` and `AES_DECRYPT()` functions. See Section 12.14, “Encryption and Compression Functions”

Certain OpenSSL-related system and status variables are present only if MySQL was compiled using OpenSSL:

- `sha256_password_private_key_path`
- `sha256_password_public_key_path`
- `Rsa_public_key`

To determine whether a server was compiled using OpenSSL, test the existence of any of those variables. For example, this statement returns a row if OpenSSL was used and an empty result if yaSSL was used:

```
SHOW STATUS LIKE 'Rsa_public_key';
```

Such tests assume that your server version is not older than the first appearance of the variable tested. For example, you cannot test for `Rsa_public_key` before MySQL 5.6.6, when that variable was added.

6.3.5 Connecting to MySQL Remotely from Windows with SSH

This section describes how to get an encrypted connection to a remote MySQL server with SSH. The original information was provided by David Carlson <dcarlson@mplcomm.com>.

1. Install an SSH client on your Windows machine. For a comparison of SSH clients, see http://en.wikipedia.org/wiki/Comparison_of_SSH_clients.
2. Start your Windows SSH client. Set `Host_Name = yourmysqlserver_URL_or_IP`. Set `userid=your_userid` to log in to your server. This `userid` value might not be the same as the user name of your MySQL account.
3. Set up port forwarding. Either do a remote forward (Set `local_port: 3306`, `remote_host: yourmysqlservername_or_ip`, `remote_port: 3306`) or a local forward (Set `port: 3306`, `host: localhost`, `remote port: 3306`).
4. Save everything, otherwise you must redo it the next time.
5. Log in to your server with the SSH session you just created.
6. On your Windows machine, start some ODBC application (such as Access).
7. Create a new file in Windows and link to MySQL using the ODBC driver the same way you normally do, except type in `localhost` for the MySQL host server, not `yourmysqlservername`.

At this point, you should have an ODBC connection to MySQL, encrypted using SSH.

6.4 Security Plugins

MySQL includes several plugins that implement security features:

- Plugins for authenticating attempts by clients to connect to MySQL Server. Plugins are available for several authentication protocols. For general discussion of the authentication process, see Section 6.2.11, “Pluggable Authentication”. For characteristics of specific authentication plugins, see Section 6.4.1, “Authentication Plugins”.
- A password-validation plugin for implementing password strength policies and assessing the strength of potential passwords. See Section 6.4.3, “The Password Validation Plugin”.
- (MySQL Enterprise Edition only) MySQL Enterprise Audit, implemented using a server plugin, uses the open MySQL Audit API to enable standard, policy-based monitoring and logging of connection and query activity executed on specific MySQL servers. Designed to meet the Oracle audit specification, MySQL Enterprise Audit provides an out of box, easy to use auditing and compliance solution for applications that are governed by both internal and external regulatory guidelines. See Section 6.4.4, “MySQL Enterprise Audit”.
- (MySQL Enterprise Edition only) MySQL Enterprise Firewall, an application-level firewall that enables database administrators to permit or deny SQL statement execution based on matching against lists of accepted statement patterns. This helps harden MySQL Server against attacks such as SQL injection or attempts to exploit applications by using them outside of their legitimate query workload characteristics. See Section 6.4.5, “MySQL Enterprise Firewall”.

6.4.1 Authentication Plugins

The following sections describe pluggable authentication methods available in MySQL and the plugins that implement these methods. For general discussion of the authentication process, see Section 6.2.11, “Pluggable Authentication”.

The default plugin is `mysql_native_password` unless the `--default-authentication-plugin` option is set otherwise at server startup.

6.4.1.1 Native Pluggable Authentication

MySQL includes two plugins that implement native authentication; that is, authentication based on the password hashing methods in use from before the introduction of pluggable authentication.

This section describes `mysql_native_password`, which implements authentication against the `mysql.user` system table using the native password hashing method. For information about `mysql_old_password`, which implements authentication using the older (pre-4.1) native password hashing method, see Section 6.4.1.2, “Old Native Pluggable Authentication”. For information about these password hashing methods, see Section 6.1.2.4, “Password Hashing in MySQL”.

The following table shows the plugin names on the server and client sides.

Table 6.8 Plugin and Library Names for Native Password Authentication

Plugin or File	Plugin or File Name
Server-side plugin	`mysql_native_password`
Client-side plugin	`mysql_native_password`
Library file	None (plugins are built in)

The following sections provide installation and usage information specific to native pluggable authentication:

- Installing Native Pluggable Authentication
- Using Native Pluggable Authentication

For general information about pluggable authentication in MySQL, see Section 6.2.11, “Pluggable Authentication”.

Installing Native Pluggable Authentication

The `mysql_native_password` plugin exists in server and client forms:

- The server-side plugin is built into the server, need not be loaded explicitly, and cannot be disabled by unloading it.
- The client-side plugin is built into the `libmysqlclient` client library and is available to any program linked against `libmysqlclient`.

Using Native Pluggable Authentication

MySQL client programs use `mysql_native_password` by default. The `--default-auth` option can be used as a hint about which client-side plugin the program can expect to use:

```
shell> mysql --default-auth=mysql_native_password ...
```

If an account row specifies no plugin name, the server authenticates the account using either the `mysql_native_password` or `mysql_old_password` plugin, depending on whether the password hash value in the `Password` column used native hashing or the older pre-4.1 hashing method. Clients must match the password in the `Password` column of the account row.

6.4.1.2 Old Native Pluggable Authentication

MySQL includes two plugins that implement native authentication; that is, authentication based on the password hashing methods in use from before the introduction of pluggable authentication. This section describes `mysql_old_password`, which implements authentication against the `mysql.user` system table using the older (pre-4.1) native password hashing method. For information about `mysql_native_password`, which implements authentication using the native password hashing method, see Section 6.4.1.1, “Native Pluggable Authentication”. For information about these password hashing methods, see Section 6.1.2.4, “Password Hashing in MySQL”.

Note

Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be

> avoided. Pre-4.1 passwords are deprecated; expect support for them to be removed in a future MySQL release. For account upgrade instructions, see Section 6.4.1.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”.

The following table shows the plugin names on the server and client sides.

Table 6.9 Plugin and Library Names for Old Native Password Authentication

Plugin or File	Plugin or File Name
Server-side plugin	`mysql_old_password`
Client-side plugin	`mysql_old_password`
Library file	None (plugins are built in)

The following sections provide installation and usage information specific to old native pluggable authentication:

- Installing Old Native Pluggable Authentication
- Using Old Native Pluggable Authentication

For general information about pluggable authentication in MySQL, see Section 6.2.11, “Pluggable Authentication”.

Installing Old Native Pluggable Authentication

The `mysql_old_password` plugin exists in server and client forms:

- The server-side plugin is built into the server, need not be loaded explicitly, and cannot be disabled by unloading it.
- The client-side plugin is built into the `libmysqlclient` client library and is available to any program linked against `libmysqlclient`.

Using Old Native Pluggable Authentication

MySQL client programs can use the `--default-auth` option to specify the `mysql_old_password` plugin as a hint about which client-side plugin the program can expect to use:

```
shell> mysql --default-auth=mysql_old_password ...
```

If an account row specifies no plugin name, the server authenticates the account using either the `mysql_native_password` or `mysql_old_password` plugin, depending on whether the password hash value in the `Password` column used native hashing or the older pre-4.1 hashing method. Clients must match the password in the `Password` column of the account row.

6.4.1.3 Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin

The MySQL server authenticates connection attempts for each account listed in the `mysql.user` system table using the authentication plugin named in the `plugin` column. If the `plugin` column is empty, the server authenticates the account as follows:

- Before MySQL 5.7, the server uses the `mysql_native_password` or `mysql_old_password` plugin implicitly, depending on the format of the password hash in the `Password` column. If the `Password` value is empty or a 4.1 password hash (41 characters), the server uses `mysql_native_password`. If the password value is a pre-4.1 password hash (16 characters), the server uses `mysql_old_password`. (For additional information about these hash formats, see Section 6.1.2.4, “Password Hashing in MySQL”.)

- As of MySQL 5.7, the server requires the `plugin` column to be nonempty and disables accounts that have an empty `plugin` value.

Pre-4.1 password hashes and the `mysql_old_password` plugin are deprecated in MySQL 5.6 and support for them is removed in MySQL 5.7. They provide a level of security inferior to that offered by 4.1 password hashing and the `mysql_native_password` plugin.

Given the requirement in MySQL 5.7 that the `plugin` column must be nonempty, coupled with removal of `mysql_old_password` support, DBAs are advised to upgrade accounts as follows:

- Upgrade accounts that use `mysql_native_password` implicitly to use it explicitly
- Upgrade accounts that use `mysql_old_password` (either implicitly or explicitly) to use `mysql_native_password` explicitly

The instructions in this section describe how to perform those upgrades. The result is that no account has an empty `plugin` value and no account uses pre-4.1 password hashing or the `mysql_old_password` plugin.

As a variant on these instructions, DBAs might offer users the choice to upgrade to the `sha256_password` plugin, which authenticates using SHA-256 password hashes. For information about this plugin, see Section 6.4.1.4, “SHA-256 Pluggable Authentication”.

The following table lists the types of `mysql.user` accounts considered in this discussion.

`plugin` Column	**`Password` Column**	**Authentication Result**	**Upgrade Action**
Empty	Empty	Implicitly uses `mysql_native_password`	Assign plugin
Empty	4.1 hash	Implicitly uses `mysql_native_password`	Assign plugin
Empty	Pre-4.1 hash	Implicitly uses `mysql_old_password`	Assign plugin, rehash password
`mysql_native_password`	Empty	Explicitly uses `mysql_native_password`	None
`mysql_native_password`	4.1 hash	Explicitly uses `mysql_native_password`	None
`mysql_old_password`	Empty	Explicitly uses `mysql_old_password`	Upgrade plugin
`mysql_old_password`	Pre-4.1 hash	Explicitly uses `mysql_old_password`	Upgrade plugin, rehash password

Accounts corresponding to lines for the `mysql_native_password` plugin require no upgrade action (because no change of plugin or hash format is required). For accounts corresponding to lines for which the password is empty, consider asking the account owners to choose a password (or require it by using `ALTER USER` to expire empty account passwords).

Upgrading Accounts from Implicit to Explicit mysql_native_password Use

Accounts that have an empty plugin and a 4.1 password hash use `mysql_native_password` implicitly. To upgrade these accounts to use `mysql_native_password` explicitly, execute these statements:

```
UPDATE mysql.user SET plugin = 'mysql_native_password'
WHERE plugin = '' AND (Password = '' OR LENGTH(Password) = 41);
FLUSH PRIVILEGES;
```

Before MySQL 5.7, you can execute those statements to uprade accounts proactively. As of MySQL 5.7, you can run `mysql_upgrade`, which performs the same operation among its upgrade actions.

Notes:

- The upgrade operation just described is safe to execute at any time because it makes the `mysql_native_password` plugin explicit only for accounts that already use it implicitly.
- This operation requires no password changes, so it can be performed without affecting users or requiring their involvement in the upgrade process.

Upgrading Accounts from mysql_old_password to mysql_native_password

Accounts that use `mysql_old_password` (either implicitly or explicitly) should be upgraded to use `mysql_native_password` explicitly. This requires changing the plugin *and* changing the password from pre-4.1 to 4.1 hash format.

For the accounts covered in this step that must be upgraded, one of these conditions is true:

- The account uses `mysql_old_password` implicitly because the `plugin` column is empty and the password has the pre-4.1 hash format (16 characters).
- The account uses `mysql_old_password` explicitly.

To identify such accounts, use this query:

```
SELECT User, Host, Password FROM mysql.user
WHERE (plugin = '' AND LENGTH(Password) = 16)
OR plugin = 'mysql_old_password';
```

The following discussion provides two methods for updating that set of accounts. They have differing characteristics, so read both and decide which is most suitable for a given MySQL installation.

Method 1.

Characteristics of this method:

- It requires that server and clients be run with `secure_auth=0` until all users have been upgraded to `mysql_native_password`. (Otherwise, users cannot connect to the server using their old-format password hashes for the purpose of upgrading to a new-format hash.)
- It works for MySQL 5.5 and 5.6. In 5.7, it does not work because the server requires accounts to have a nonempty plugin and disables them otherwise. Therefore, if you have already upgraded to 5.7, choose Method 2, described later.

You should ensure that the server is running with `secure_auth=0`.

For all accounts that use `mysql_old_password` explicitly, set them to the empty plugin:

```
UPDATE mysql.user SET plugin = ''
WHERE plugin = 'mysql_old_password';
FLUSH PRIVILEGES;
```

To also expire the password for affected accounts, use these statements instead:

```
UPDATE mysql.user SET plugin = '', password_expired = 'Y'
WHERE plugin = 'mysql_old_password';
FLUSH PRIVILEGES;
```

Now affected users can reset their password to use 4.1 hashing. Ask each user who now has an empty plugin to connect to the server and execute these statements:

```
SET old_passwords = 0;
SET PASSWORD = PASSWORD('user-chosen-password');
```

Note

The client-side `--secure-auth` option is enabled by default, so remind users to disable it or they cannot connect:

```
shell> mysql -u user_name -p --secure-auth=0
```

After an affected user has executed those statements, you can set the corresponding account plugin to `mysql_native_password` to make the plugin explicit. Or you can periodically run these statements to find and fix any accounts for which affected users have reset their password:

```
UPDATE mysql.user SET plugin = 'mysql_native_password'
WHERE plugin = '' AND (Password = '' OR LENGTH(Password) = 41);
FLUSH PRIVILEGES;
```

When there are no more accounts with an empty plugin, this query returns an empty result:

```
SELECT User, Host, Password FROM mysql.user
WHERE plugin = '' AND LENGTH(Password) = 16;
```

At that point, all accounts have been migrated away from pre-4.1 password hashing and the server no longer need be run with `secure_auth=0`.

Method 2.

Characteristics of this method:

- It assigns each affected account a new password, so you must tell each such user the new password and ask the user to choose a new one. Communication of passwords to users is outside the scope of MySQL, but should be done carefully.
- It does not require server or clients to be run with `secure_auth=0`.
- It works for any version of MySQL 5.5 or later (and for 5.7 has an easier variant).

With this method, you update each account separately due to the need to set passwords individually. *Choose a different password for each account.*

Suppose that `'user1'@'localhost'` is one of the accounts to be upgraded. Modify it as follows:

- In MySQL 5.7, `ALTER USER` provides the capability of modifying both the account password and its authentication plugin, so you need not modify the `mysql.user` system table directly:

  ```
  ALTER USER 'user1'@'localhost'
  IDENTIFIED WITH mysql_native_password BY 'DBA-chosen-password';
  ```

  To also expire the account password, use this statement instead:

  ```
  ALTER USER 'user1'@'localhost'
  IDENTIFIED WITH mysql_native_password BY 'DBA-chosen-password'
  PASSWORD EXPIRE;
  ```

  Then tell the user the new password and ask the user to connect to the server with that password and execute this statement to choose a new password:

  ```
  ALTER USER USER() IDENTIFIED BY 'user-chosen-password';
  ```

- Before MySQL 5.7, you must modify the `mysql.user` system table directly using these statements:

  ```
  SET old_passwords = 0;
  UPDATE mysql.user SET plugin = 'mysql_native_password',
  Password = PASSWORD('DBA-chosen-password')
  WHERE (User, Host) = ('user1', 'localhost');
  FLUSH PRIVILEGES;
  ```

  To also expire the account password, use these statements instead:

  ```
  SET old_passwords = 0;
  UPDATE mysql.user SET plugin = 'mysql_native_password',
  Password = PASSWORD('DBA-chosen-password'), password_expired = 'Y'
  ```

```
WHERE (User, Host) = ('user1', 'localhost');
FLUSH PRIVILEGES;
```

Then tell the user the new password and ask the user to connect to the server with that password and execute these statements to choose a new password:

```
SET old_passwords = 0;
SET PASSWORD = PASSWORD('user-chosen-password');
```

Repeat for each account to be upgraded.

6.4.1.4 SHA-256 Pluggable Authentication

MySQL provides an authentication plugin that implements SHA-256 hashing for user account passwords.

Important

To connect to the server using an account that authenticates with the `sha256_password` plugin, you must use either a TLS connection or an unencrypted connection that supports password exchange using an RSA key pair, as described later in this section. Either way, use of the `sha256_password` plugin requires that MySQL be built with SSL capabilities. See Section 6.3, “Using Encrypted Connections”.

The following table shows the plugin names on the server and client sides.

Table 6.10 Plugin and Library Names for SHA-256 Authentication

Plugin or File	Plugin or File Name
Server-side plugin	`sha256_password`
Client-side plugin	`sha256_password`
Library file	None (plugins are built in)

The following sections provide installation and usage information specific to SHA-256 pluggable authentication:

- Installing SHA-256 Pluggable Authentication
- Using SHA-256 Pluggable Authentication

For general information about pluggable authentication in MySQL, see Section 6.2.11, “Pluggable Authentication”.

Installing SHA-256 Pluggable Authentication

The `sha256_password` plugin exists in server and client forms:

- The server-side plugin is built into the server, need not be loaded explicitly, and cannot be disabled by unloading it.
- The client-side plugin is built into the `libmysqlclient` client library and is available to any program linked against `libmysqlclient`.

Using SHA-256 Pluggable Authentication

To set up an account that uses the `sha256_password` plugin for SHA-256 password hashing, use the following procedure.

1. Create the account and specify that it authenticates using the `sha256_password` plugin:

```
CREATE USER 'sha256user'@'localhost' IDENTIFIED WITH sha256_password;
```

2. Set the `old_passwords` system variable to 2 to cause the `PASSWORD()` function to use SHA-256 hashing of password strings, then set the account password:

```
SET old_passwords = 2;
SET PASSWORD FOR 'sha256user'@'localhost' = PASSWORD('password');
```

The server assigns the `sha256_password` plugin to the account and uses it to encrypt the password using SHA-256, storing those values in the `plugin` and `authentication_string` columns of the `mysql.user` system table.

The preceding instructions do not assume that `sha256_password` is the default authentication plugin. If `sha256_password` is the default authentication plugin, a simpler `CREATE USER` syntax can be used.

To start the server with the default authentication plugin set to `sha256_password`, put these lines in the server option file:

```
[mysqld]
default-authentication-plugin=sha256_password
```

That causes the `sha256_password` plugin to be used by default for new accounts. As a result, it is possible to create the account and set its password without naming the plugin explicitly:

```
CREATE USER 'sha256user'@'localhost' IDENTIFIED BY 'password';
```

Another consequence of setting `default-authentication-plugin` to `sha256_password` is that, to use some other plugin for account creation, you must specify that plugin explicitly in the `CREATE USER` statement, then set `old_passwords` appropriately for the plugin before using `SET PASSWORD` to set the account password. For example, to use the `mysql_native_password` plugin, do this:

```
CREATE USER 'nativeuser'@'localhost' IDENTIFIED WITH mysql_native_password;
SET old_passwords = 0;
SET PASSWORD FOR 'nativeuser'@'localhost' = PASSWORD('N@tivePa33');
```

To set or change the password for any account that authenticates using the `sha256_password` plugin, be sure that the value of `old_passwords` is 2 before using `SET PASSWORD`. If `old_passwords` has a value other than 2, an error occurs for attempts to set the password:

```
mysql> SET old_passwords = 0;
mysql> SET PASSWORD FOR 'sha256user'@'localhost' = PASSWORD('password');
ERROR 1827 (HY000): The password hash doesn't have the expected format.
Check if the correct password algorithm is being used with the
PASSWORD() function.
```

For more information about `old_passwords` and `PASSWORD()`, see Section 5.1.7, “Server System Variables”, and Section 12.14, “Encryption and Compression Functions”.

MySQL can be compiled using either OpenSSL or yaSSL (see Section 6.3.4, “SSL Library-Dependent Capabilities”). The `sha256_password` plugin works with distributions compiled using either package, but if MySQL is compiled using OpenSSL, `sha256_password` supports the use of RSA encryption. (To enable this capability, you must follow the RSA configuration procedure given later in this section.)

Note

It is possible to compile MySQL using yaSSL as an alternative to OpenSSL only prior to MySQL 5.6.46. As of MySQL 5.6.46, support for yaSSL is removed and all MySQL builds use OpenSSL.

RSA support has these characteristics:

- On the server side, two system variables name the RSA private and public key-pair files: `sha256_password_private_key_path` and `sha256_password_public_key_path`. The database administrator must set these variables at server startup if the key files to use have names that differ from the system variable default values.

- The `Rsa_public_key` status variable displays the RSA public key value used by the `sha256_password` authentication plugin.
- Clients that have the RSA public key can perform RSA key pair-based password exchange with the server during the connection process, as described later.
- For connections by accounts that authenticate with `sha256_password` and RSA public key pair-based password exchange, the server sends the RSA public key to the client as needed. However, if a copy of the public key is available on the client host, the client can use it to save a round trip in the client/server protocol:
  - For these command-line clients, use the `--server-public-key-path` option to specify the RSA public key file: `mysql`, `mysqltest`.
  - For programs that use the C API, call `mysql_options()` to specify the RSA public key file by passing the `MYSQL_SERVER_PUBLIC_KEY` option and the name of the file.
  - For replicas, RSA key pair-based password exchange cannot be used to connect to source servers for accounts that authenticate with the `sha256_password` plugin. For such accounts, only secure connections can be used.

For clients that use the `sha256_password` plugin, passwords are never exposed as cleartext when connecting to the server. How password transmission occurs depends on whether a secure connection or RSA encryption is used:

- If the connection is secure, an RSA key pair is unnecessary and is not used. This applies to connections encrypted using TLS. The password is sent as cleartext but cannot be snooped because the connection is secure.

Note

`sha256_password` does not treat shared-memory connections as secure, even though share-memory transport is secure by default.

- If the connection is not secure, and an RSA key pair is available, the connection remains unencrypted. This applies to connections not encrypted using TLS. RSA is used only for password exchange between client and server, to prevent password snooping. When the server receives the encrypted password, it decrypts it. A scramble is used in the encryption to prevent repeat attacks.
- If a secure connection is not used and RSA encryption is not available, the connection attempt fails because the password cannot be sent without being exposed as cleartext.

As mentioned previously, RSA password encryption is available only if MySQL was compiled using OpenSSL. The implication for MySQL distributions compiled using yaSSL is that, to use SHA-256 passwords, clients *must* use an encrypted connection to access the server. See Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

Note

To use RSA password encryption with `sha256_password`, the client and server both must be compiled using OpenSSL, not just one of them.

Assuming that MySQL has been compiled using OpenSSL, use the following procedure to enable use of an RSA key pair for password exchange during the client connection process:

1. Create the RSA private and public key-pair files using the instructions in Section 6.3.3, “Creating SSL and RSA Certificates and Keys”.
2. If the private and public key files are located in the data directory and are named `private_key.pem` and `public_key.pem` (the default values of the `sha256_password_private_key_path` and `sha256_password_public_key_path` system variables), the server uses them automatically at startup.

Otherwise, to name the key files explicitly, set the system variables to the key file names in the server option file. If the files are located in the server data directory, you need not specify their full path names:

```
[mysqld]
sha256_password_private_key_path=myprivkey.pem
sha256_password_public_key_path=mypubkey.pem
```

If the key files are not located in the data directory, or to make their locations explicit in the system variable values, use full path names:

```
[mysqld]
sha256_password_private_key_path=/usr/local/mysql/myprivkey.pem
sha256_password_public_key_path=/usr/local/mysql/mypubkey.pem
```

3. Restart the server, then connect to it and check the `Rsa_public_key` status variable value. The value differs from that shown here, but should be nonempty:

```
mysql> SHOW STATUS LIKE 'Rsa_public_key'\G
*************************** 1. row ***************************
Variable_name: Rsa_public_key
        Value: -----BEGIN PUBLIC KEY-----
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDO9nRUDd+KvSZgY7cNBZMNpwX6
MvE1PbJFXO7u18nJ9lwc99Du/E7lw6CVXw7VKrXPeHbVQUzGyUNkf45Nz/ckaaJa
aLgJOBCIDmNVnyU54OT/1lcs2xiyfaDMe8fCJ64ZwTnKbY2gkt1IMjUAB5Ogd5kJ
g8aV7EtKwyhHb0c30QIDAQAB
-----END PUBLIC KEY-----
```

If the value is empty, the server found some problem with the key files. Check the error log for diagnostic information.

After the server has been configured with the RSA key files, accounts that authenticate with the `sha256_password` plugin have the option of using those key files to connect to the server. As mentioned previously, such accounts can use either a secure connection (in which case RSA is not used) or an unencrypted connection that performs password exchange using RSA. Suppose that an unencrypted connection is used. For example:

```
shell> mysql --ssl-mode=DISABLED -u sha256user -p
Enter password: password
```

For this connection attempt by `sha256user`, the server determines that `sha256_password` is the appropriate authentication plugin and invokes it (because that was the plugin specified at `CREATE USER` time). The plugin finds that the connection is not encrypted and thus requires the password to be transmitted using RSA encryption. In this case, the plugin sends the RSA public key to the client, which uses it to encrypt the password and returns the result to the server. The plugin uses the RSA private key on the server side to decrypt the password and accepts or rejects the connection based on whether the password is correct.

The server sends the RSA public key to the client as needed. However, if the client has a file containing a local copy of the RSA public key required by the server, it can specify the file using the `--server-public-key-path` option:

```
shell> mysql --ssl-mode=DISABLED -u sha256user -p --server-public-key-path=file_name
Enter password: password
```

The public key value in the file named by the `--server-public-key-path` option should be the same as the key value in the server-side file named by the `sha256_password_public_key_path` system variable. If the key file contains a valid public key value but the value is incorrect, an access-denied error occurs. If the key file does not contain a valid public key, the client program cannot use it. In this case, the `sha256_password` plugin sends the public key to the client as if no `--server-public-key-path` option had been specified.

Client users can obtain the RSA public key two ways:

- The database administrator can provide a copy of the public key file.
- A client user who can connect to the server some other way can use a `SHOW STATUS LIKE 'Rsa_public_key'` statement and save the returned key value in a file.

6.4.1.5 Client-Side Cleartext Pluggable Authentication

A client-side authentication plugin is available that enables clients to send passwords to the server as cleartext, without hashing or encryption. This plugin is built into the MySQL client library.

The following table shows the plugin name.

Table 6.11 Plugin and Library Names for Cleartext Authentication

Plugin or File	Plugin or File Name
Server-side plugin	None, see discussion
Client-side plugin	`mysql_clear_password`
Library file	None (plugin is built in)

Many client-side authentication plugins perform hashing or encryption of a password before the client sends it to the server. This enables clients to avoid sending passwords as cleartext.

Hashing or encryption cannot be done for authentication schemes that require the server to receive the password as entered on the client side. In such cases, the client-side `mysql_clear_password` plugin is used to send the password to the server as cleartext. There is no corresponding server-side plugin. Rather, the client-side plugin can be used by any server-side plugin that needs a cleartext

Hashing or encryption cannot be done for authentication schemes that require the server to receive the password as entered on the client side. In such cases, the client-side `mysql_clear_password` plugin is used, which enables the client to send the password to the server as cleartext. There is no corresponding server-side plugin. Rather, `mysql_clear_password` can be used on the client side in concert with any server-side plugin that needs a cleartext password. (The PAM authentication plugin is one such; see Section 6.4.1.6, “PAM Pluggable Authentication”.)

The following discussion provides usage information specific to cleartext pluggable authentication. For general information about pluggable authentication in MySQL, see Section 6.2.11, “Pluggable Authentication”.

Note

Sending passwords as cleartext may be a security problem in some configurations. To avoid problems if there is any possibility that the password would be intercepted, clients should connect to MySQL Server using a method that protects the password. Possibilities include SSL (see Section 6.3, “Using Encrypted Connections”), IPsec, or a private network.

To make inadvertent use of the `mysql_clear_password` plugin less likely, MySQL clients must explicitly enable it. This can be done in several ways:

- Set the `LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN` environment variable to a value that begins with `1`, `Y`, or `y`. This enables the plugin for all client connections.
- The `mysql`, `mysqladmin`, and `mysqlslap` client programs (also `mysqlcheck`, `mysqldump`, and `mysqlshow` for MySQL 5.6.28 and later) support an `--enable-cleartext-plugin` option that enables the plugin on a per-invocation basis.
- The `mysql_options()` C API function supports a `MYSQL_ENABLE_CLEARTEXT_PLUGIN` option that enables the plugin on a per-connection basis. Also, any program that uses `libmysqlclient` and reads option files can enable the plugin by including an `enable-cleartext-plugin` option in an option group read by the client library.

6.4.1.6 PAM Pluggable Authentication

Note

PAM pluggable authentication is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

MySQL Enterprise Edition supports an authentication method that enables MySQL Server to use PAM (Pluggable Authentication Modules) to authenticate MySQL users. PAM enables a system to use a standard interface to access various kinds of authentication methods, such as traditional Unix passwords or an LDAP directory.

PAM pluggable authentication provides these capabilities:

- External authentication: PAM authentication enables MySQL Server to accept connections from users defined outside the MySQL grant tables and that authenticate using methods supported by PAM.
- Proxy user support: PAM authentication can return to MySQL a user name different from the external user name passed by the client program, based on the PAM groups the external user is a member of and the authentication string provided. This means that the plugin can return the MySQL user that defines the privileges the external PAM-authenticated user should have. For example, an operating sytem user named `joe` can connect and have the privileges of a MySQL user named `developer`.

PAM pluggable authentication has been tested on Linux and macOS.

The following table shows the plugin and library file names. The file name suffix might differ on your system. The file must be located in the directory named by the `plugin_dir` system variable. For installation information, see Installing PAM Pluggable Authentication.

Table 6.12 Plugin and Library Names for PAM Authentication

Plugin or File	Plugin or File Name
Server-side plugin	`authentication_pam`
Client-side plugin	`mysql_clear_password`
Library file	`authentication_pam.so`

The client-side `mysql_clear_password` cleartext plugin that communicates with the server-side PAM plugin is built into the `libmysqlclient` client library and is included in all distributions, including community distributions. Inclusion of the client-side cleartext plugin in all MySQL distributions enables clients from any distribution to connect to a server that has the server-side PAM plugin loaded.

The following sections provide installation and usage information specific to PAM pluggable authentication:

- How PAM Authentication of MySQL Users Works
- Installing PAM Pluggable Authentication
- Uninstalling PAM Pluggable Authentication
- Using PAM Pluggable Authentication
- PAM Unix Password Authentication without Proxy Users
- PAM LDAP Authentication without Proxy Users
- PAM Unix Password Authentication with Proxy Users and Group Mapping

- PAM Authentication Access to Unix Password Store
- PAM Authentication Debugging

For general information about pluggable authentication in MySQL, see Section 6.2.11, “Pluggable Authentication”. For information about the `mysql_clear_password` plugin, see Section 6.4.1.5, “Client-Side Cleartext Pluggable Authentication”. For proxy user information, see Section 6.2.12, “Proxy Users”.

How PAM Authentication of MySQL Users Works

This section provides a general overview of how MySQL and PAM work together to authenticate MySQL users. For examples showing how to set up MySQL accounts to use specific PAM services, see Using PAM Pluggable Authentication.

1. The client program and the server communicate, with the client sending to the server the client user name (the operating system user name by default) and password:
   - The client user name is the external user name.
   - For accounts that use the PAM server-side authentication plugin, the corresponding client-side plugin is `mysql_clear_password`. This client-side plugin performs no password hashing, with the result that the client sends the password to the server as cleartext.
2. The server finds a matching MySQL account based on the external user name and the host from which the client connects. The PAM plugin uses the information passed to it by MySQL Server (such as user name, host name, password, and authentication string). When you define a MySQL account that authenticates using PAM, the authentication string contains:
   - A PAM service name, which is a name that the system administrator can use to refer to an authentication method for a particular application. There can be multiple applications associated with a single database server instance, so the choice of service name is left to the SQL application developer.
   - Optionally, if proxying is to be used, a mapping from PAM groups to MySQL user names.
3. The plugin uses the PAM service named in the authentication string to check the user credentials and returns `'Authentication succeeded, Username is user_name'` or `'Authentication failed'`. The password must be appropriate for the password store used by the PAM service. Examples:
   - For traditional Unix passwords, the service looks up passwords stored in the `/etc/shadow` file.
   - For LDAP, the service looks up passwords stored in an LDAP directory.

   If the credentials check fails, the server refuses the connection.
4. Otherwise, the authentication string indicates whether proxying occurs. If the string contains no PAM group mapping, proxying does not occur. In this case, the MySQL user name is the same as the external user name.
5. Otherwise, proxying is indicated based on the PAM group mapping, with the MySQL user name determined based on the first matching group in the mapping list. The meaning of “PAM group” depends on the PAM service. Examples:
   - For traditional Unix passwords, groups are Unix groups defined in the `/etc/group` file, possibly supplemented with additional PAM information in a file such as `/etc/security/group.conf`.
   - For LDAP, groups are LDAP groups defined in an LDAP directory.

   If the proxy user (the external user) has the `PROXY` privilege for the proxied MySQL user name, proxying occurs, with the proxy user assuming the privileges of the proxied user.

Installing PAM Pluggable Authentication

This section describes how to install the PAM authentication plugin. For general information about installing plugins, see Section 5.5.1, “Installing and Uninstalling Plugins”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base name is `authentication_pam`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To load the plugin at server startup, use the `--plugin-load-add` option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in the server `my.cnf` file, adjusting the `.so` suffix for your platform as necessary:

```
[mysqld]
plugin-load-add=authentication_pam.so
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugin at runtime, use this statement, adjusting the `.so` suffix for your platform as necessary:

```
INSTALL PLUGIN authentication_pam SONAME 'authentication_pam.so';
```

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.5.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE '%pam%';
+-------------------+---------------+
| PLUGIN_NAME       | PLUGIN_STATUS |
+-------------------+---------------+
| authentication_pam | ACTIVE        |
+-------------------+---------------+
```

If the plugin failed to initialize, check the server error log for diagnostic messages.

To associate MySQL accounts with the PAM plugin, see Using PAM Pluggable Authentication.

Uninstalling PAM Pluggable Authentication

The method used to uninstall the PAM authentication plugin depends on how you installed it:

- If you installed the plugin at server startup using a `--plugin-load-add` option, restart the server without the option.
- If you installed the plugin at runtime using an `INSTALL PLUGIN` statement, it remains installed across server restarts. To uninstall it, use `UNINSTALL PLUGIN`:

  ```
  UNINSTALL PLUGIN authentication_pam;
  ```

Using PAM Pluggable Authentication

This section describes in general terms how to use the PAM authentication plugin to connect from MySQL client programs to the server. The following sections provide instructions for using PAM

authentication in specific ways. It is assumed that the server is running with the server-side PAM plugin enabled, as described in Installing PAM Pluggable Authentication.

To refer to the PAM authentication plugin in the `IDENTIFIED WITH` clause of a `CREATE USER` or `GRANT` statement, use the name `authentication_pam`. For example:

```
CREATE USER user
  IDENTIFIED WITH authentication_pam
  AS 'auth_string';
```

The authentication string specifies the following types of information:

- The PAM service name (see How PAM Authentication of MySQL Users Works). Examples in the following discussion use a service name of `mysql-unix` for authentication using traditional Unix passwords, and `mysql-ldap` for authentication using LDAP.
- For proxy support, PAM provides a way for a PAM module to return to the server a MySQL user name other than the external user name passed by the client program when it connects to the server. Use the authentication string to control the mapping from external user names to MySQL user names. If you want to take advantage of proxy user capabilities, the authentication string must include this kind of mapping.

For example, if an account uses the `mysql-unix` PAM service name and should map operating system users in the `root` and `users` PAM groups to the `developer` and `data_entry` MySQL users, respectively, use a statement like this:

```
CREATE USER user
  IDENTIFIED WITH authentication_pam
  AS 'mysql-unix, root=developer, users=data_entry';
```

Authentication string syntax for the PAM authentication plugin follows these rules:

- The string consists of a PAM service name, optionally followed by a PAM group mapping list consisting of one or more keyword/value pairs each specifying a PAM group name and a MySQL user name:

  ```
  pam_service_name[,pam_group_name=mysql_user_name]...
  ```

  The plugin parses the authentication string for each connection attempt that uses the account. To minimize overhead, keep the string as short as possible.

- Each *`pam_group_name=mysql_user_name`* pair must be preceded by a comma.
- Leading and trailing spaces not inside double quotation marks are ignored.
- Unquoted *`pam_service_name`*, *`pam_group_name`*, and *`mysql_user_name`* values can contain anything except equal sign, comma, or space.
- If a *`pam_service_name`*, *`pam_group_name`*, or *`mysql_user_name`* value is quoted with double quotation marks, everything between the quotation marks is part of the value. This is necessary, for example, if the value contains space characters. All characters are legal except double quotation mark and backslash (`\`). To include either character, escape it with a backslash.

If the plugin successfully authenticates the external user name (the name passed by the client), it looks for a PAM group mapping list in the authentication string and, if present, uses it to return a different MySQL user name to the MySQL server based on which PAM groups the external user is a member of:

- If the authentication string contains no PAM group mapping list, the plugin returns the external name.
- If the authentication string does contain a PAM group mapping list, the plugin examines each *`pam_group_name=mysql_user_name`* pair in the list from left to right and tries to find a match for the *`pam_group_name`* value in a non-MySQL directory of the groups assigned to the authenticated user and returns *`mysql_user_name`* for the first match it finds. If the plugin finds no match for

any PAM group, it returns the external name. If the plugin is not capable of looking up a group in a directory, it ignores the PAM group mapping list and returns the external name.

The following sections describe how to set up several authentication scenarios that use the PAM authentication plugin:

- No proxy users. This uses PAM only to check login names and passwords. Every external user permitted to connect to MySQL Server should have a matching MySQL account that is defined to use PAM authentication. (For a MySQL account of `'user_name'@'host_name'` to match the external user, *`user_name`* must be the external user name and *`host_name`* must match the host from which the client connects.) Authentication can be performed by various PAM-supported methods. Later discussion shows how to authenticate client credentials using traditional Unix passwords, and passwords in LDAP.

  PAM authentication, when not done through proxy users or PAM groups, requires the MySQL user name to be same as the operating system user name. MySQL user names are limited to 16 characters (see Section 6.2.3, “Grant Tables”), which limits PAM nonproxy authentication to Unix accounts with names of at most 16 characters.

- Proxy users only, with PAM group mapping. For this scenario, create one or more MySQL accounts that define different sets of privileges. (Ideally, nobody should connect using those accounts directly.) Then define a default user authenticating through PAM that uses some mapping scheme (usually based on the external PAM groups the users are members of) to map all the external user names to the few MySQL accounts holding the privilege sets. Any client who connects and specifies an external user name as the client user name is mapped to one of the MySQL accounts and uses its privileges. The discussion shows how to set this up using traditional Unix passwords, but other PAM methods such as LDAP could be used instead.

Variations on these scenarios are possible:

- You can permit some users to log in directly (without proxying) but require others to connect through proxy accounts.
- You can use one PAM authentication method for some users, and another method for other users, by using differing PAM service names among your PAM-authenticated accounts. For example, you can use the `mysql-unix` PAM service for some users, and `mysql-ldap` for others.

The examples make the following assumptions. You might need to make some adjustments if your system is set up differently.

- The login name and password are `antonio` and *`antonio_password`*, respectively. Change these to correspond to the user you want to authenticate.
- The PAM configuration directory is `/etc/pam.d`.
- The PAM service name corresponds to the authentication method (`mysql-unix` or `mysql-ldap` in this discussion). To use a given PAM service, you must set up a PAM file with the same name in the PAM configuration directory (creating the file if it does not exist). In addition, you must use the PAM service name in the authentication string of `CREATE USER` or `GRANT` statements for any account that authenticates using that service.

The PAM authentication plugin checks at initialization time whether the `AUTHENTICATION_PAM_LOG` environment value is set in the server's startup environment. If so, the plugin enables logging of diagnostic messages to the standard output. Depending on how your server is started, the message might appear on the console or in the error log. These messages can be helpful for debugging PAM-related issues that occur when the plugin performs authentication. For more information, see PAM Authentication Debugging.

PAM Unix Password Authentication without Proxy Users

This authentication scenario uses PAM to check external users defined in terms of operating system user names and Unix passwords, without proxying. Every such external user permitted to connect to

MySQL Server should have a matching MySQL account that is defined to use PAM authentication through traditional Unix password store.

Note

Traditional Unix passwords are checked using the `/etc/shadow` file. For information regarding possible issues related to this file, see PAM Authentication Access to Unix Password Store.

1. Verify that Unix authentication permits logins to the operating system with the user name `antonio` and password *`antonio_password`*.

2. Set up PAM to authenticate MySQL connections using traditional Unix passwords by creating a `mysql-unix` PAM service file named `/etc/pam.d/mysql-unix`. The file contents are system dependent, so check existing login-related files in the `/etc/pam.d` directory to see what they look like. On Linux, the `mysql-unix` file might look like this:

   ```
   #%PAM-1.0
   auth            include         password-auth
   account         include         password-auth
   ```

   For macOS, use `login` rather than `password-auth`.

   The PAM file format might differ on some systems. For example, on Ubuntu and other Debian-based systems, use these file contents instead:

   ```
   @include common-auth
   @include common-account
   @include common-session-noninteractive
   ```

3. Create a MySQL account with the same user name as the operating system user name and define it to authenticate using the PAM plugin and the `mysql-unix` PAM service:

   ```
   CREATE USER 'antonio'@'localhost'
     IDENTIFIED WITH authentication_pam
     AS 'mysql-unix';
   GRANT ALL PRIVILEGES
     ON mydb.*
     TO 'antonio'@'localhost';
   ```

   Here, the authentication string contains only the PAM service name, `mysql-unix`, which authenticates Unix passwords.

4. Use the `mysql` command-line client to connect to the MySQL server as `antonio`. For example:

   ```
   shell> mysql --user=antonio --password --enable-cleartext-plugin
   Enter password: antonio_password
   ```

   The server should permit the connection and the following query returns output as shown:

   ```
   mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
   +-------------------+-------------------+--------------+
   | USER()            | CURRENT_USER()    | @@proxy_user |
   +-------------------+-------------------+--------------+
   | antonio@localhost | antonio@localhost | NULL         |
   +-------------------+-------------------+--------------+
   ```

   This demonstrates that the `antonio` operating system user is authenticated to have the privileges granted to the `antonio` MySQL user, and that no proxying has occurred.

Note

The client-side `mysql_clear_password` authentication plugin leaves the password untouched, so client programs send it to the MySQL server as cleartext. This enables the password to be passed as is to PAM. A cleartext

password is necessary to use the server-side PAM library, but may be a security problem in some configurations. These measures minimize the risk:

- To make inadvertent use of the `mysql_clear_password` plugin less likely, MySQL clients must explicitly enable it (for example, with the `--enable-cleartext-plugin` option). See Section 6.4.1.5, “Client-Side Cleartext Pluggable Authentication”.
- To avoid password exposure with the `mysql_clear_password` plugin enabled, MySQL clients should connect to the MySQL server using an encrypted connection. See Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

PAM LDAP Authentication without Proxy Users

This authentication scenario uses PAM to check external users defined in terms of operating system user names and LDAP passwords, without proxying. Every such external user permitted to connect to MySQL Server should have a matching MySQL account that is defined to use PAM authentication through LDAP.

To use PAM LDAP pluggable authentication for MySQL, these prerequisites must be satisfied:

- An LDAP server must be available for the PAM LDAP service to communicate with.
- LDAP users to be authenticated by MySQL must be present in the directory managed by the LDAP server.

Configure MySQL for PAM LDAP authentication as follows:

1. Verify that Unix authentication permits logins to the operating system with the user name `antonio` and password *`antonio_password`*.
2. Set up PAM to authenticate MySQL connections using LDAP by creating a `mysql-ldap` PAM service file named `/etc/pam.d/mysql-ldap`. The file contents are system dependent, so check existing login-related files in the `/etc/pam.d` directory to see what they look like. On Linux, the `mysql-ldap` file might look like this:

   ```
   #%PAM-1.0
   auth        required    pam_ldap.so
   account     required    pam_ldap.so
   ```

   If PAM object files have a suffix different from `.so` on your system, substitute the correct suffix.

   The PAM file format might differ on some systems.
3. Create a MySQL account with the same user name as the operating system user name and define it to authenticate using the PAM plugin and the `mysql-ldap` PAM service:

   ```
   CREATE USER 'antonio'@'localhost'
     IDENTIFIED WITH authentication_pam
     AS 'mysql-ldap';
   GRANT ALL PRIVILEGES
     ON mydb.*
     TO 'antonio'@'localhost';
   ```

   Here, the authentication string contains only the PAM service name, `mysql-ldap`, which authenticates using LDAP.
4. Connecting to the server is the same as described in PAM Unix Password Authentication without Proxy Users.

PAM Unix Password Authentication with Proxy Users and Group Mapping

The authentication scheme described here uses proxying and PAM group mapping to map connecting MySQL users who authenticate using PAM onto other MySQL accounts that define different sets of privileges. Users do not connect directly through the accounts that define the privileges. Instead, they connect through a default proxy account authenticated using PAM, such that all the external users are mapped to the MySQL accounts that hold the privileges. Any user who connects using the proxy account is mapped to one of those MySQL accounts, the privileges for which determine the database operations permitted to the external user.

The procedure shown here uses Unix password authentication. To use LDAP instead, see the early steps of PAM LDAP Authentication without Proxy Users.

Note

Traditional Unix passwords are checked using the `/etc/shadow` file. For information regarding possible issues related to this file, see PAM Authentication Access to Unix Password Store.

1. Verify that Unix authentication permits logins to the operating system with the user name `antonio` and password *`antonio_password`*.

2. Verify that `antonio` is a member of the `root` or `users` PAM group.

3. Set up PAM to authenticate the `mysql-unix` PAM service through operating system users by creating a file named `/etc/pam.d/mysql-unix`. The file contents are system dependent, so check existing login-related files in the `/etc/pam.d` directory to see what they look like. On Linux, the `mysql-unix` file might look like this:

   ```
   #%PAM-1.0
   auth            include         password-auth
   account         include         password-auth
   ```

   For macOS, use `login` rather than `password-auth`.

   The PAM file format might differ on some systems. For example, on Ubuntu and other Debian-based systems, use these file contents instead:

   ```
   @include common-auth
   @include common-account
   @include common-session-noninteractive
   ```

4. Create a default proxy user (`''@''`) that maps external PAM users to the proxied accounts:

   ```
   CREATE USER ''@''
     IDENTIFIED WITH authentication_pam
     AS 'mysql-unix, root=developer, users=data_entry';
   ```

   Here, the authentication string contains the PAM service name, `mysql-unix`, which authenticates Unix passwords. The authentication string also maps external users in the `root` and `users` PAM groups to the `developer` and `data_entry` MySQL user names, respectively.

   The PAM group mapping list following the PAM service name is required when you set up proxy users. Otherwise, the plugin cannot tell how to perform mapping from external user names to the proper proxied MySQL user names.

   Note

   If your MySQL installation has anonymous users, they might conflict with the default proxy user. For more information about this issue, and ways of dealing with it, see Default Proxy User and Anonymous User Conflicts.

5. Create the proxied accounts and grant to each one the privileges it should have:

   ```
   CREATE USER 'developer'@'localhost'
     IDENTIFIED BY 'very secret password';
   ```

```
CREATE USER 'data_entry'@'localhost'
  IDENTIFIED BY 'very secret password';

GRANT ALL PRIVILEGES
  ON mydevdb.*
  TO 'developer'@'localhost';
GRANT ALL PRIVILEGES
  ON mydb.*
  TO 'data_entry'@'localhost';
```

If you do not let anyone know the passwords for these accounts, clients cannot use them to connect directly to the MySQL server. Instead, it is expected that users who authenticate using PAM use the `developer` or `data_entry` account by proxy based on their PAM group.

6. Grant to the proxy account the `PROXY` privilege for each proxied account:

```
GRANT PROXY
  ON 'developer'@'localhost'
  TO ''@'';
GRANT PROXY
  ON 'data_entry'@'localhost'
  TO ''@'';
```

7. Use the `mysql` command-line client to connect to the MySQL server as `antonio`.

```
shell> mysql --user=antonio --password --enable-cleartext-plugin
Enter password: antonio_password
```

The server authenticates the connection using the default `''@''` proxy account. The resulting privileges for `antonio` depend on which PAM groups `antonio` is a member of. If `antonio` is a member of the `root` PAM group, the PAM plugin maps `root` to the `developer` MySQL user name and returns that name to the server. The server verifies that `''@''` has the `PROXY` privilege for `developer` and permits the connection. The following query returns output as shown:

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
+-------------------+--------------------+--------------+
| USER()            | CURRENT_USER()     | @@proxy_user |
+-------------------+--------------------+--------------+
| antonio@localhost | developer@localhost | ''@''        |
+-------------------+--------------------+--------------+
```

This demonstrates that the `antonio` operating system user is authenticated to have the privileges granted to the `developer` MySQL user, and that proxying occurred through the default proxy account.

If `antonio` is not a member of the `root` PAM group but is a member of the `users` PAM group, a similar process occurs, but the plugin maps `user` PAM group membership to the `data_entry` MySQL user name and returns that name to the server:

```
mysql> SELECT USER(), CURRENT_USER(), @@proxy_user;
+-------------------+---------------------+--------------+
| USER()            | CURRENT_USER()      | @@proxy_user |
+-------------------+---------------------+--------------+
| antonio@localhost | data_entry@localhost | ''@''        |
+-------------------+---------------------+--------------+
```

This demonstrates that the `antonio` operating system user is authenticated to have the privileges of the `data_entry` MySQL user, and that proxying occurred through the default proxy account.

Note

The client-side `mysql_clear_password` authentication plugin leaves the password untouched, so client programs send it to the MySQL server as cleartext. This enables the password to be passed as is to PAM. A cleartext password is necessary to use the server-side PAM library, but may be a security problem in some configurations. These measures minimize the risk:

- To make inadvertent use of the `mysql_clear_password` plugin less likely, MySQL clients must explicitly enable it (for example, with the `--enable-cleartext-plugin` option). See Section 6.4.1.5, “Client-Side Cleartext Pluggable Authentication”.
- To avoid password exposure with the `mysql_clear_password` plugin enabled, MySQL clients should connect to the MySQL server using an encrypted connection. See Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

PAM Authentication Access to Unix Password Store

On some systems, Unix authentication uses a password store such as `/etc/shadow`, a file that typically has restricted access permissions. This can cause MySQL PAM-based authentication to fail. Unfortunately, the PAM implementation does not permit distinguishing “password could not be checked” (due, for example, to inability to read `/etc/shadow`) from “password does not match.” If you are using Unix password store for PAM authentication, you may be able to enable access to it from MySQL using one of the following methods:

- Assuming that the MySQL server is run from the `mysql` operating system account, put that account in the `shadow` group that has `/etc/shadow` access:
  1. Create a `shadow` group in `/etc/group`.
  2. Add the `mysql` operating system user to the `shadow` group in `/etc/group`.
  3. Assign `/etc/group` to the `shadow` group and enable the group read permission:

     ```
     chgrp shadow /etc/shadow
     chmod g+r /etc/shadow
     ```

  4. Restart the MySQL server.
- If you are using the `pam_unix` module and the `unix_chkpwd` utility, enable password store access as follows:

  ```
  chmod u-s /usr/sbin/unix_chkpwd
  setcap cap_dac_read_search+ep /usr/sbin/unix_chkpwd
  ```

  Adjust the path to `unix_chkpwd` as necessary for your platform.

PAM Authentication Debugging

The PAM authentication plugin checks at initialization time whether the `AUTHENTICATION_PAM_LOG` environment value is set (the value does not matter). If so, the plugin enables logging of diagnostic messages to the standard output. These messages may be helpful for debugging PAM-related issues that occur when the plugin performs authentication.

Some messages include reference to PAM plugin source files and line numbers, which enables plugin actions to be tied more closely to the location in the code where they occur.

Another technique for debugging connection failures and determining what is happening during connection attempts is to configure PAM authentication to permit all connections, then check the system log files. This technique should be used only on a *temporary* basis, and not on a production server.

Configure a PAM service file named `/etc/pam.d/mysql-any-password` with these contents (the format may differ on some systems):

```
#%PAM-1.0
auth        required    pam_permit.so
account     required    pam_permit.so
```

Create an account that uses the PAM plugin and names the `mysql-any-password` PAM service:

```
CREATE USER 'testuser'@'localhost'
  IDENTIFIED WITH authentication_pam
  AS 'mysql-any-password';
```

The `mysql-any-password` service file causes any authentication attempt to return true, even for incorrect passwords. If an authentication attempt fails, that tells you the configuration problem is on the MySQL side. Otherwise, the problem is on the operating system/PAM side. To see what might be happening, check system log files such as `/var/log/secure`, `/var/log/audit.log`, `/var/log/syslog`, or `/var/log/messages`.

After determining what the problem is, remove the `mysql-any-password` PAM service file to disable any-password access.

6.4.1.7 Windows Pluggable Authentication

Note

Windows pluggable authentication is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

MySQL Enterprise Edition for Windows supports an authentication method that performs external authentication on Windows, enabling MySQL Server to use native Windows services to authenticate client connections. Users who have logged in to Windows can connect from MySQL client programs to the server based on the information in their environment without specifying an additional password.

The client and server exchange data packets in the authentication handshake. As a result of this exchange, the server creates a security context object that represents the identity of the client in the Windows OS. This identity includes the name of the client account. Windows pluggable authentication uses the identity of the client to check whether it is a given account or a member of a group. By default, negotiation uses Kerberos to authenticate, then NTLM if Kerberos is unavailable.

Windows pluggable authentication provides these capabilities:

- External authentication: Windows authentication enables MySQL Server to accept connections from users defined outside the MySQL grant tables who have logged in to Windows.
- Proxy user support: Windows authentication can return to MySQL a user name different from the external user name passed by the client program. This means that the plugin can return the MySQL user that defines the privileges the external Windows-authenticated user should have. For example, a Windows user named `joe` can connect and have the privileges of a MySQL user named `developer`.

The following table shows the plugin and library file names. The file must be located in the directory named by the `plugin_dir` system variable.

Table 6.13 Plugin and Library Names for Windows Authentication

Plugin or File	Plugin or File Name
Server-side plugin	`authentication_windows`
Client-side plugin	`authentication_windows_client`
Library file	`authentication_windows.dll`

The library file includes only the server-side plugin. The client-side plugin is built into the `libmysqlclient` client library.

The server-side Windows authentication plugin is included only in MySQL Enterprise Edition. It is not included in MySQL community distributions. The client-side plugin is included in all distributions, including community distributions. This permits clients from any distribution to connect to a server that has the server-side plugin loaded.

The Windows authentication plugin is supported on any version of Windows supported by MySQL 5.6 (see https://www.mysql.com/support/supportedplatforms/database.html).

The following sections provide installation and usage information specific to Windows pluggable authentication:

- Installing Windows Pluggable Authentication
- Uninstalling Windows Pluggable Authentication
- Using Windows Pluggable Authentication

For general information about pluggable authentication in MySQL, see Section 6.2.11, “Pluggable Authentication”. For proxy user information, see Section 6.2.12, “Proxy Users”.

Installing Windows Pluggable Authentication

This section describes how to install the Windows authentication plugin. For general information about installing plugins, see Section 5.5.1, “Installing and Uninstalling Plugins”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

To load the plugin at server startup, use the `--plugin-load-add` option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in the server `my.cnf` file:

```
[mysqld]
plugin-load-add=authentication_windows.dll
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugin at runtime, use this statement:

```
INSTALL PLUGIN authentication_windows SONAME 'authentication_windows.dll';
```

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.5.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE '%windows%';
+-----------------------+---------------+
| PLUGIN_NAME           | PLUGIN_STATUS |
+-----------------------+---------------+
| authentication_windows | ACTIVE        |
+-----------------------+---------------+
```

If the plugin fails to initialize, check the server error log for diagnostic messages.

To associate MySQL accounts with the Windows authentication plugin, see Using Windows Pluggable Authentication. Additional plugin control is provided by the `authentication_windows_use_principal_name` and `authentication_windows_log_level` system variables. See Section 5.1.7, “Server System Variables”.

Uninstalling Windows Pluggable Authentication

The method used to uninstall the Windows authentication plugin depends on how you installed it:

- If you installed the plugin at server startup using a `--plugin-load-add` option, restart the server without the option.
- If you installed the plugin at runtime using an `INSTALL PLUGIN` statement, it remains installed across server restarts. To uninstall it, use `UNINSTALL PLUGIN`:

```
UNINSTALL PLUGIN authentication_windows;
```

In addition, remove any startup options that set Windows plugin-related system variables.

Using Windows Pluggable Authentication

The Windows authentication plugin supports the use of MySQL accounts such that users who have logged in to Windows can connect to the MySQL server without having to specify an additional password. It is assumed that the server is running with the server-side plugin enabled, as described in Installing Windows Pluggable Authentication. Once the DBA has enabled the server-side plugin and set up accounts to use it, clients can connect using those accounts with no other setup required on their part.

To refer to the Windows authentication plugin in the `IDENTIFIED WITH` clause of a `CREATE USER` or `GRANT` statement, use the name `authentication_windows`. Suppose that the Windows users `Rafal` and `Tasha` should be permitted to connect to MySQL, as well as any users in the `Administrators` or `Power Users` group. To set this up, create a MySQL account named `sql_admin` that uses the Windows plugin for authentication:

```
CREATE USER sql_admin
  IDENTIFIED WITH authentication_windows
  AS 'Rafal, Tasha, Administrators, "Power Users"';
```

The plugin name is `authentication_windows`. The string following the `AS` keyword is the authentication string. It specifies that the Windows users named `Rafal` or `Tasha` are permitted to authenticate to the server as the MySQL user `sql_admin`, as are any Windows users in the `Administrators` or `Power Users` group. The latter group name contains a space, so it must be quoted with double quote characters.

After you create the `sql_admin` account, a user who has logged in to Windows can attempt to connect to the server using that account:

```
C:\> mysql --user=sql_admin
```

No password is required here. The `authentication_windows` plugin uses the Windows security API to check which Windows user is connecting. If that user is named `Rafal` or `Tasha`, or is a member of the `Administrators` or `Power Users` group, the server grants access and the client is authenticated as `sql_admin` and has whatever privileges are granted to the `sql_admin` account. Otherwise, the server denies access.

Authentication string syntax for the Windows authentication plugin follows these rules:

- The string consists of one or more user mappings separated by commas.
- Each user mapping associates a Windows user or group name with a MySQL user name:

```
win_user_or_group_name=mysql_user_name
win_user_or_group_name
```

For the latter syntax, with no *`mysql_user_name`* value given, the implicit value is the MySQL user created by the `CREATE USER` statement. Thus, these statements are equivalent:

```
CREATE USER sql_admin
  IDENTIFIED WITH authentication_windows
  AS 'Rafal, Tasha, Administrators, "Power Users"';

CREATE USER sql_admin
```

```
  IDENTIFIED WITH authentication_windows
  AS 'Rafal=sql_admin, Tasha=sql_admin, Administrators=sql_admin,
      "Power Users"=sql_admin';
```

- Each backslash character (`\`) in a value must be doubled because backslash is the escape character in MySQL strings.
- Leading and trailing spaces not inside double quotation marks are ignored.
- Unquoted *`win_user_or_group_name`* and *`mysql_user_name`* values can contain anything except equal sign, comma, or space.
- If a *`win_user_or_group_name`* and or *`mysql_user_name`* value is quoted with double quotation marks, everything between the quotation marks is part of the value. This is necessary, for example, if the name contains space characters. All characters within double quotes are legal except double quotation mark and backslash. To include either character, escape it with a backslash.
- *`win_user_or_group_name`* values use conventional syntax for Windows principals, either local or in a domain. Examples (note the doubling of backslashes):

```
domain\\user
.\\user
domain\\group
.\\group
BUILTIN\\WellKnownGroup
```

When invoked by the server to authenticate a client, the plugin scans the authentication string left to right for a user or group match to the Windows user. If there is a match, the plugin returns the corresponding *`mysql_user_name`* to the MySQL server. If there is no match, authentication fails.

A user name match takes preference over a group name match. Suppose that the Windows user named `win_user` is a member of `win_group` and the authentication string looks like this:

```
'win_group = sql_user1, win_user = sql_user2'
```

When `win_user` connects to the MySQL server, there is a match both to `win_group` and to `win_user`. The plugin authenticates the user as `sql_user2` because the more-specific user match takes precedence over the group match, even though the group is listed first in the authentication string.

Windows authentication always works for connections from the same computer on which the server is running. For cross-computer connections, both computers must be registered with Windows Active Directory. If they are in the same Windows domain, it is unnecessary to specify a domain name. It is also possible to permit connections from a different domain, as in this example:

```
CREATE USER sql_accounting
  IDENTIFIED WITH authentication_windows
  AS 'SomeDomain\\Accounting';
```

Here `SomeDomain` is the name of the other domain. The backslash character is doubled because it is the MySQL escape character within strings.

MySQL supports the concept of proxy users whereby a client can connect and authenticate to the MySQL server using one account but while connected has the privileges of another account (see Section 6.2.12, “Proxy Users”). Suppose that you want Windows users to connect using a single user name but be mapped based on their Windows user and group names onto specific MySQL accounts as follows:

- The `local_user` and `MyDomain\domain_user` local and domain Windows users should map to the `local_wlad` MySQL account.
- Users in the `MyDomain\Developers` domain group should map to the `local_dev` MySQL account.

- Local machine administrators should map to the `local_admin` MySQL account.

To set this up, create a proxy account for Windows users to connect to, and configure this account so that users and groups map to the appropriate MySQL accounts (`local_wlad`, `local_dev`, `local_admin`). In addition, grant the MySQL accounts the privileges appropriate to the operations they need to perform. The following instructions use `win_proxy` as the proxy account, and `local_wlad`, `local_dev`, and `local_admin` as the proxied accounts.

1. Create the proxy MySQL account:

```
CREATE USER win_proxy
  IDENTIFIED WITH  authentication_windows
  AS 'local_user = local_wlad,
      MyDomain\\domain_user = local_wlad,
      MyDomain\\Developers = local_dev,
      BUILTIN\\Administrators = local_admin';
```

2. For proxying to work, the proxied accounts must exist, so create them:

```
CREATE USER local_wlad IDENTIFIED BY 'secret_wlad_pass';
CREATE USER local_dev IDENTIFIED BY 'secret_dev_pass';
CREATE USER local_admin IDENTIFIED BY  'secret_admin_pass';
```

   If you do not let anyone know the passwords for these accounts, clients cannot use them to connect directly to the MySQL server. Instead, it is expected that users who authenticate using Windows use the `win_proxy` proxy account.

   You should also execute `GRANT` statements (not shown) that grant each proxied account the privileges required for MySQL access.

3. Grant to the proxy account the `PROXY` privilege for each proxied account:

```
GRANT PROXY ON local_wlad TO win_proxy;
GRANT PROXY ON local_dev TO win_proxy;
GRANT PROXY ON local_admin TO win_proxy;
```

Now the Windows users `local_user` and `MyDomain\domain_user` can connect to the MySQL server as `win_proxy` and when authenticated have the privileges of the account given in the authentication string (in this case, `local_wlad`). A user in the `MyDomain\Developers` group who connects as `win_proxy` has the privileges of the `local_dev` account. A user in the `BUILTIN\Administrators` group has the privileges of the `local_admin` account.

To configure authentication so that all Windows users who do not have their own MySQL account go through a proxy account, substitute the default proxy account (`''@''`) for `win_proxy` in the preceding instructions. For information about default proxy accounts, see Section 6.2.12, “Proxy Users”.

Note

If your MySQL installation has anonymous users, they might conflict with the default proxy user. For more information about this issue, and ways of dealing with it, see Default Proxy User and Anonymous User Conflicts.

To use the Windows authentication plugin with Connector/NET connection strings in Connector/NET 6.4.4 and higher, see Using the Windows Native Authentication Plugin.

6.4.1.8 Socket Peer-Credential Pluggable Authentication

The server-side `auth_socket` authentication plugin authenticates clients that connect from the local host through the Unix socket file. The plugin uses the `SO_PEERCRED` socket option to obtain information about the user running the client program. Thus, the plugin can be used only on systems that support the `SO_PEERCRED` option, such as Linux.

The source code for this plugin can be examined as a relatively simple example demonstrating how to write a loadable authentication plugin.

The following table shows the plugin and library file names. The file must be located in the directory named by the `plugin_dir` system variable.

Table 6.14 Plugin and Library Names for Socket Peer-Credential Authentication

Plugin or File	Plugin or File Name
Server-side plugin	`auth_socket`
Client-side plugin	None, see discussion
Library file	`auth_socket.so`

The following sections provide installation and usage information specific to socket pluggable authentication:

- Installing Socket Pluggable Authentication
- Uninstalling Socket Pluggable Authentication
- Using Socket Pluggable Authentication

For general information about pluggable authentication in MySQL, see Section 6.2.11, “Pluggable Authentication”.

Installing Socket Pluggable Authentication

This section describes how to install the socket authentication plugin. For general information about installing plugins, see Section 5.5.1, “Installing and Uninstalling Plugins”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

To load the plugin at server startup, use the `--plugin-load-add` option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in the server `my.cnf` file:

```
[mysqld]
plugin-load-add=auth_socket.so
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugin at runtime, use this statement:

```
INSTALL PLUGIN auth_socket SONAME 'auth_socket.so';
```

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.5.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE '%socket%';
+-------------+---------------+
| PLUGIN_NAME | PLUGIN_STATUS |
+-------------+---------------+
| auth_socket | ACTIVE        |
+-------------+---------------+
```

If the plugin fails to initialize, check the server error log for diagnostic messages.

To associate MySQL accounts with the socket plugin, see Using Socket Pluggable Authentication.

Uninstalling Socket Pluggable Authentication

The method used to uninstall the socket authentication plugin depends on how you installed it:

- If you installed the plugin at server startup using a `--plugin-load-add` option, restart the server without the option.
- If you installed the plugin at runtime using an `INSTALL PLUGIN` statement, it remains installed across server restarts. To uninstall it, use `UNINSTALL PLUGIN`:

```
UNINSTALL PLUGIN auth_socket;
```

Using Socket Pluggable Authentication

The socket plugin checks whether the socket user name (the operating system user name) matches the MySQL user name specified by the client program to the server, and permits the connection only if the names match.

Suppose that a MySQL account is created for an operating system user named `valerie` who is to be authenticated by the `auth_socket` plugin for connections from the local host through the socket file:

```
CREATE USER 'valerie'@'localhost' IDENTIFIED WITH auth_socket;
```

If a user on the local host with a login name of `stefanie` invokes `mysql` with the option `--user=valerie` to connect through the socket file, the server uses `auth_socket` to authenticate the client. The plugin determines that the `--user` option value (`valerie`) differs from the client user's name (`stephanie`) and refuses the connection. If a user named `valerie` tries the same thing, the plugin finds that the user name and the MySQL user name are both `valerie` and permits the connection. However, the plugin refuses the connection even for `valerie` if the connection is made using a different protocol, such as TCP/IP.

6.4.1.9 Test Pluggable Authentication

MySQL includes a test plugin that checks account credentials and logs success or failure to the server error log. This is a loadable plugin (not built in) and must be installed prior to use.

The test plugin source code is separate from the server source, unlike the built-in native plugin, so it can be examined as a relatively simple example demonstrating how to write a loadable authentication plugin.

Note

This plugin is intended for testing and development purposes, and is not for use in production environments or on servers that are exposed to public networks.

The following table shows the plugin and library file names. The file name suffix might differ on your system. The file must be located in the directory named by the `plugin_dir` system variable.

Table 6.15 Plugin and Library Names for Test Authentication

Plugin or File	Plugin or File Name
Server-side plugin	`test_plugin_server`
Client-side plugin	`auth_test_plugin`
Library file	`auth_test_plugin.so`

The following sections provide installation and usage information specific to test pluggable authentication:

- Installing Test Pluggable Authentication
- Uninstalling Test Pluggable Authentication
- Using Test Pluggable Authentication

For general information about pluggable authentication in MySQL, see Section 6.2.11, “Pluggable Authentication”.

Installing Test Pluggable Authentication

This section describes how to install the test authentication plugin. For general information about installing plugins, see Section 5.5.1, “Installing and Uninstalling Plugins”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

To load the plugin at server startup, use the `--plugin-load-add` option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in the server `my.cnf` file, adjusting the `.so` suffix for your platform as necessary:

```
[mysqld]
plugin-load-add=auth_test_plugin.so
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugin at runtime, use this statement, adjusting the `.so` suffix for your platform as necessary:

```
INSTALL PLUGIN test_plugin_server SONAME 'auth_test_plugin.so';
```

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.5.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE '%test_plugin%';
+--------------------+---------------+
| PLUGIN_NAME        | PLUGIN_STATUS |
+--------------------+---------------+
| test_plugin_server | ACTIVE        |
+--------------------+---------------+
```

If the plugin fails to initialize, check the server error log for diagnostic messages.

To associate MySQL accounts with the test plugin, see Using Test Pluggable Authentication.

Uninstalling Test Pluggable Authentication

The method used to uninstall the test authentication plugin depends on how you installed it:

- If you installed the plugin at server startup using a `--plugin-load-add` option, restart the server without the option.
- If you installed the plugin at runtime using an `INSTALL PLUGIN` statement, it remains installed across server restarts. To uninstall it, use `UNINSTALL PLUGIN`:

  ```
  UNINSTALL PLUGIN test_plugin_server;
  ```

Using Test Pluggable Authentication

To use the test authentication plugin, create an account and name that plugin in the `IDENTIFIED WITH` clause:

```
CREATE USER 'testuser'@'localhost' IDENTIFIED WITH test_plugin_server;
SET PASSWORD FOR 'testuser'@'localhost' = PASSWORD('testpassword');
```

Then provide the `--user` and `--password` options for that account when you connect to the server. For example:

```
shell> mysql --user=testuser --password
Enter password: testpassword
```

The plugin fetches the password as received from the client and compares it with the value stored in the `authentication_string` column of the account row in the `mysql.user` system table. If the two values match, the plugin returns the `authentication_string` value as the new effective user ID.

You can look in the server error log for a message indicating whether authentication succeeded (notice that the password is reported as the “user”):

```
[Note] Plugin test_plugin_server reported:
'successfully authenticated user testpassword'
```

6.4.2 The Connection-Control Plugins

As of MySQL 5.6.35, MySQL Server includes a plugin library that enables administrators to introduce an increasing delay in server response to connection attempts after a configurable number of consecutive failed attempts. This capability provides a deterrent that slows down brute force attacks against MySQL user accounts. The plugin library contains two plugins:

- `CONNECTION_CONTROL` checks incoming connection attempts and adds a delay to server responses as necessary. This plugin also exposes system variables that enable its operation to be configured and a status variable that provides rudimentary monitoring information.

  The `CONNECTION_CONTROL` plugin uses the audit plugin interface (see Writing Audit Plugins). To collect information, it subscribes to the `MYSQL_AUDIT_CONNECTION_CLASSMASK` event class, and processes `MYSQL_AUDIT_CONNECTION_CONNECT` and `MYSQL_AUDIT_CONNECTION_CHANGE_USER` subevents to check whether the server should introduce a delay before responding to connection attempts.

- `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` implements an `INFORMATION_SCHEMA` table that exposes more detailed monitoring information for failed connection attempts.

The following sections provide information about connection-control plugin installation and configuration. For information about the `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` table, see Section 21.33.1, “The INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS Table”.

6.4.2.1 Connection-Control Plugin Installation

This section describes how to install the connection-control plugins, `CONNECTION_CONTROL` and `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS`. For general information about installing plugins, see Section 5.5.1, “Installing and Uninstalling Plugins”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base name is `connection_control`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To load the plugins at server startup, use the `--plugin-load-add` option to name the library file that contains them. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in the server `my.cnf` file, adjusting the `.so` suffix for your platform as necessary:

```
[mysqld]
plugin-load-add=connection_control.so
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugins at runtime, use these statements, adjusting the `.so` suffix for your platform as necessary:

```
INSTALL PLUGIN CONNECTION_CONTROL
  SONAME 'connection_control.so';
INSTALL PLUGIN CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS
  SONAME 'connection_control.so';
```

`INSTALL PLUGIN` loads the plugin immediately, and also registers it in the `mysql.plugins` system table to cause the server to load it for each subsequent normal startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.5.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE 'connection%';
+---------------------------------------+---------------+
| PLUGIN_NAME                           | PLUGIN_STATUS |
+---------------------------------------+---------------+
| CONNECTION_CONTROL                    | ACTIVE        |
| CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS | ACTIVE        |
+---------------------------------------+---------------+
```

If a plugin failed to initialize, check the server error log for diagnostic messages.

If the plugins have been previously registered with `INSTALL PLUGIN` or are loaded with `--plugin-load-add`, you can use the `--connection-control` and `--connection-control-failed-login-attempts` options at server startup to control plugin activation. For example, to load the plugins at startup and prevent them from being removed at runtime, use these options:

```
[mysqld]
plugin-load-add=connection_control.so
connection-control=FORCE_PLUS_PERMANENT
connection-control-failed-login-attempts=FORCE_PLUS_PERMANENT
```

If it is desired to prevent the server from running without a given connection-control plugin, use an option value of `FORCE` or `FORCE_PLUS_PERMANENT` to force server startup to fail if the plugin does not initialize successfully.

Note

It is possible to install one plugin without the other, but both must be installed for full connection-control capability. In particular, installing only the `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` plugin is of little use because without the `CONNECTION_CONTROL` plugin to provide the data that populates the `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` table, retrievals from the table are always empty.

- Connection Delay Configuration
- Connection Failure Assessment
- Connection Failure Monitoring

Connection Delay Configuration

To enable configuring its operation, the `CONNECTION_CONTROL` plugin exposes these system variables:

- `connection_control_failed_connections_threshold`: The number of consecutive failed connection attempts permitted to accounts before the server adds a delay for subsequent connection attempts. To disable failed-connection counting, set `connection_control_failed_connections_threshold` to zero.

- `connection_control_min_connection_delay`: The minimum delay in milliseconds for connection failures above the threshold.
- `connection_control_max_connection_delay`: The maximum delay in milliseconds for connection failures above the threshold.

If `connection_control_failed_connections_threshold` is nonzero, failed-connection counting is enabled and has these properties:

- The delay is zero up through `connection_control_failed_connections_threshold` consecutive failed connection attempts.
- Thereafter, the server adds an increasing delay for subsequent consecutive attempts, until a successful connection occurs. The initial unadjusted delays begin at 1000 milliseconds (1 second) and increase by 1000 milliseconds per attempt. That is, once delay has been activated for an account, the unadjusted delays for subsequent failed attempts are 1000 milliseconds, 2000 milliseconds, 3000 milliseconds, and so forth.
- The actual delay experienced by a client is the unadjusted delay, adjusted to lie within the values of the `connection_control_min_connection_delay` and `connection_control_max_connection_delay` system variables, inclusive.
- Once delay has been activated for an account, the first successful connection thereafter by the account also experiences a delay, but failure counting is reset for subsequent connections.

For example, with the default `connection_control_failed_connections_threshold` value of 3, there is no delay for the first three consecutive failed connection attempts by an account. The actual adjusted delays experienced by the account for the fourth and subsequent failed connections depend on the `connection_control_min_connection_delay` and `connection_control_max_connection_delay` values:

- If `connection_control_min_connection_delay` and `connection_control_max_connection_delay` are 1000 and 20000, the adjusted delays are the same as the unadjusted delays, up to a maximum of 20000 milliseconds. The fourth and subsequent failed connections are delayed by 1000 milliseconds, 2000 milliseconds, 3000 milliseconds, and so forth.
- If `connection_control_min_connection_delay` and `connection_control_max_connection_delay` are 1500 and 20000, the adjusted delays for the fourth and subsequent failed connections are 1500 milliseconds, 2000 milliseconds, 3000 milliseconds, and so forth, up to a maximum of 20000 milliseconds.
- If `connection_control_min_connection_delay` and `connection_control_max_connection_delay` are 2000 and 3000, the adjusted delays for the fourth and subsequent failed connections are 2000 milliseconds, 2000 milliseconds, and 3000 milliseconds, with all subsequent failed connections also delayed by 3000 milliseconds.

You can set the `CONNECTION_CONTROL` system variables at server startup or runtime. Suppose that you want to permit four consecutive failed connection attempts before the server starts delaying its responses, with a minimum delay of 2000 milliseconds. To set the relevant variables at server startup, put these lines in the server `my.cnf` file:

```
[mysqld]
plugin-load-add=connection_control.so
connection_control_failed_connections_threshold=4
connection_control_min_connection_delay=2000
```

To set the variables at runtime, use these statements:

```
SET GLOBAL connection_control_failed_connections_threshold = 4;
SET GLOBAL connection_control_min_connection_delay = 1500;
```

`SET GLOBAL` sets the value for the running MySQL instance. To make the change permanent, add a line in your `my.cnf` file, as shown previously.

The `connection_control_min_connection_delay` and `connection_control_max_connection_delay` system variables both have minimum and maximum values of 1000 and 2147483647. In addition, the permitted range of values of each variable also depends on the current value of the other:

- `connection_control_min_connection_delay` cannot be set greater than the current value of `connection_control_max_connection_delay`.
- `connection_control_max_connection_delay` cannot be set less than the current value of `connection_control_min_connection_delay`.

Thus, to make the changes required for some configurations, you might need to set the variables in a specific order. Suppose that the current minimum and maximum delays are 1000 and 2000, and that you want to set them to 3000 and 5000. You cannot first set `connection_control_min_connection_delay` to 3000 because that is greater than the current `connection_control_max_connection_delay` value of 2000. Instead, set `connection_control_max_connection_delay` to 5000, then set `connection_control_min_connection_delay` to 3000.

Connection Failure Assessment

When the `CONNECTION_CONTROL` plugin is installed, it checks connection attempts and tracks whether they fail or succeed. For this purpose, a failed connection attempt is one for which the client user and host match a known MySQL account but the provided credentials are incorrect, or do not match any known account.

Failed-connection counting is based on the user/host combination for each connection attempt. Determination of the applicable user name and host name takes proxying into account and occurs as follows:

- If the client user proxies another user, the account for failed-connection counting is the proxying user, not the proxied user. For example, if `external_user@example.com` proxies `proxy_user@example.com`, connection counting uses the proxying user, `external_user@example.com`, rather than the proxied user, `proxy_user@example.com`. Both `external_user@example.com` and `proxy_user@example.com` must have valid entries in the `mysql.user` system table and a proxy relationship between them must be defined in the `mysql.proxies_priv` system table (see Section 6.2.12, “Proxy Users”).
- If the client user does not proxy another user, but does match a `mysql.user` entry, counting uses the `CURRENT_USER()` value corresponding to that entry. For example, if a user `user1` connecting from a host `host1.example.com` matches a `user1@host1.example.com` entry, counting uses `user1@host1.example.com`. If the user matches a `user1@%.example.com`, `user1@%.com`, or `user1@%` entry instead, counting uses `user1@%.example.com`, `user1@%.com`, or `user1@%`, respectively.

For the cases just described, the connection attempt matches some `mysql.user` entry, and whether the request succeeds or fails depends on whether the client provides the correct authentication credentials. For example, if the client presents an incorrect password, the connection attempt fails.

If the connection attempt matches no `mysql.user` entry, the attempt fails. In this case, no `CURRENT_USER()` value is available and connection-failure counting uses the user name provided by the client and the client host as determined by the server. For example, if a client attempts to connect as user `user2` from host `host2.example.com`, the user name part is available in the client request and the server determines the host information. The user/host combination used for counting is `user2@host2.example.com`.

Note

The server maintains information about which client hosts can possibly connect to the server (essentially the union of host values for `mysql.user` entries). If a

> client attempts to connect from any other host, the server rejects the attempt at an early stage of connection setup:
>
> ```
> ERROR 1130 (HY000): Host 'host_name' is not
> allowed to connect to this MySQL server
> ```
>
> Because this type of rejection occurs so early, `CONNECTION_CONTROL` does not see it, and does not count it.

Connection Failure Monitoring

To monitor failed connections, use these information sources:

- The `Connection_control_delay_generated` status variable indicates the number of times the server added a delay to its response to a failed connection attempt. This does not count attempts that occur before reaching the threshold defined by the `connection_control_failed_connections_threshold` system variable.
- The `INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` table provides information about the current number of consecutive failed connection attempts per account (user/host combination). This counts all failed attempts, regardless of whether they were delayed.

Assigning a value to `connection_control_failed_connections_threshold` at runtime has these effects:

- All accumulated failed-connection counters are reset to zero.
- The `Connection_control_delay_generated` status variable is reset to zero.
- The `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` table becomes empty.

6.4.2.2 Connection-Control System and Status Variables

This section describes the system and status variables that the `CONNECTION_CONTROL` plugin provides to enable its operation to be configured and monitored.

- Connection-Control System Variables
- Connection-Control Status Variables

Connection-Control System Variables

If the `CONNECTION_CONTROL` plugin is installed, it exposes these system variables:

- `connection_control_failed_connections_threshold`

Command-Line Format	`--connection-control-failed-connections-threshold=#`
Introduced	5.6.35
System Variable	`connection_control_failed_connections_threshold`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`3`
Minimum Value	`0`
Maximum Value	`2147483647`

The number of consecutive failed connection attempts permitted to accounts before the server adds a delay for subsequent connection attempts:

- If the variable has a nonzero value *N*, the server adds a delay beginning with consecutive failed attempt *N*+1. If an account has reached the point where connection responses are delayed, a delay also occurs for the next subsequent successful connection.
- Setting this variable to zero disables failed-connection counting. In this case, the server never adds delays.

For information about how `connection_control_failed_connections_threshold` interacts with other connection-control system and status variables, see Section 6.4.2.1, "Connection-Control Plugin Installation".

- `connection_control_max_connection_delay`

Command-Line Format	`--connection-control-max-connection-delay=#`
Introduced	5.6.35
System Variable	`connection_control_max_connection_delay`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`2147483647`
Minimum Value	`1000`
Maximum Value	`2147483647`
Unit	`milliseconds`

The maximum delay in milliseconds for server response to failed connection attempts, if `connection_control_failed_connections_threshold` is greater than zero.

For information about how `connection_control_max_connection_delay` interacts with other connection-control system and status variables, see Section 6.4.2.1, "Connection-Control Plugin Installation".

- `connection_control_min_connection_delay`

Command-Line Format	`--connection-control-min-connection-delay=#`
Introduced	5.6.35
System Variable	`connection_control_min_connection_delay`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`1000`
Minimum Value	`1000`
Maximum Value	`2147483647`
Unit	`milliseconds`

The minimum delay in milliseconds for server response to failed connection attempts, if `connection_control_failed_connections_threshold` is greater than zero.

For information about how `connection_control_min_connection_delay` interacts with other connection-control system and status variables, see Section 6.4.2.1, "Connection-Control Plugin Installation".

Connection-Control Status Variables

If the CONNECTION_CONTROL plugin is installed, it exposes this status variable:

- Connection_control_delay_generated

  The number of times the server added a delay to its response to a failed connection attempt. This does not count attempts that occur before reaching the threshold defined by the connection_control_failed_connections_threshold system variable.

  This variable provides a simple counter. For more detailed connection-control monitoring information, examine the INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS table; see Section 21.33.1, “The INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS Table”.

  Assigning a value to connection_control_failed_connections_threshold at runtime resets Connection_control_delay_generated to zero.

  This variable was added in MySQL 5.6.35.

6.4.3 The Password Validation Plugin

The validate_password plugin serves to improve security by requiring account passwords and enabling strength testing of potential passwords. This plugin exposes a set of system variables that enable you to configure password policy.

The validate_password plugin implements these capabilities:

- For SQL statements that assign a password supplied as a cleartext value, validate_password checks the password against the current password policy and rejects the password if it is weak (the statement returns an ER_NOT_VALID_PASSWORD error). This applies to the CREATE USER, GRANT, and SET PASSWORD statements, and passwords given as arguments to the PASSWORD() and OLD_PASSWORD() functions.
- For CREATE USER statements, validate_password requires that a password be given, and that it satisfies the password policy.
- validate_password implements a VALIDATE_PASSWORD_STRENGTH() SQL function that assesses the strength of potential passwords. This function takes a password argument and returns an integer from 0 (weak) to 100 (strong).

Note

For statements that assign, modify, or generate account passwords (CREATE USER, GRANT, and SET PASSWORD; statements that use PASSWORD() and OLD_PASSWORD()), the validate_password capabilities described here apply only to accounts that use an authentication plugin that stores credentials internally in the mysql.user system table (mysql_native_password or sha256_password). For accounts that use plugins that perform authentication against an external credential system, password management must be handled externally against that system as well.

The preceding restriction does not apply to use of the VALIDATE_PASSWORD_STRENGTH() function because it does not affect accounts directly.

Examples:

- validate_password checks the cleartext password in the following statement. Under the default password policy, which requires passwords to be at least 8 characters long, the password is weak and the statement produces an error:

```
mysql> SET PASSWORD = PASSWORD('abc');
ERROR 1819 (HY000): Your password does not satisfy the current
```

```
policy requirements
```

- Passwords specified as hashed values are not checked because the original password value is not available for checking:

```
mysql> SET PASSWORD = '*0D3CED9BEC10A777AEC23CCC353A8C08A633045E';
Query OK, 0 rows affected (0.01 sec)
```

- To check a password, use the `VALIDATE_PASSWORD_STRENGTH()` function:

```
mysql> SELECT VALIDATE_PASSWORD_STRENGTH('weak');
+----------------------------------+
| VALIDATE_PASSWORD_STRENGTH('weak') |
+----------------------------------+
|                               25 |
+----------------------------------+
mysql> SELECT VALIDATE_PASSWORD_STRENGTH('lessweak$_@123');
+--------------------------------------------+
| VALIDATE_PASSWORD_STRENGTH('lessweak$_@123') |
+--------------------------------------------+
|                                         50 |
+--------------------------------------------+
mysql> SELECT VALIDATE_PASSWORD_STRENGTH('N0Tweak$_@123!');
+-----------------------------------------+
| VALIDATE_PASSWORD_STRENGTH('N0Tweak$_@123!') |
+-----------------------------------------+
|                                     100 |
+-----------------------------------------+
```

To configure password checking, modify the system variables having names of the form `validate_password_xxx`; these are the parameters that control password policy. See Section 6.4.3.2, “Password Validation Plugin Options and Variables”.

If `validate_password` is not installed, the `validate_password_xxx` system variables are not available, passwords in statements are not checked, and the `VALIDATE_PASSWORD_STRENGTH()` function always returns 0. For example, without the plugin installed, accounts can be assigned passwords shorter than 8 characters, or no password at all.

Assuming that `validate_password` is installed, it implements three levels of password checking: `LOW`, `MEDIUM`, and `STRONG`. The default is `MEDIUM`; to change this, modify the value of `validate_password_policy`. The policies implement increasingly strict password tests. The following descriptions refer to default parameter values, which can be modified by changing the appropriate system variables.

- `LOW` policy tests password length only. Passwords must be at least 8 characters long. To change this length, modify `validate_password_length`.
- `MEDIUM` policy adds the conditions that passwords must contain at least 1 numeric character, 1 lowercase character, 1 uppercase character, and 1 special (nonalphanumeric) character. To change these values, modify `validate_password_number_count`, `validate_password_mixed_case_count`, and `validate_password_special_char_count`.
- `STRONG` policy adds the condition that password substrings of length 4 or longer must not match words in the dictionary file, if one has been specified. To specify the dictionary file, modify `validate_password_dictionary_file`.

6.4.3.1 Password Validation Plugin Installation

This section describes how to install the `validate_password` password-validation plugin. For general information about installing plugins, see Section 5.5.1, “Installing and Uninstalling Plugins”.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base name is `validate_password`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To load the plugin at server startup, use the `--plugin-load-add` option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts. For example, put these lines in the server `my.cnf` file, adjusting the `.so` suffix for your platform as necessary:

```
[mysqld]
plugin-load-add=validate_password.so
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugin at runtime, use this statement, adjusting the `.so` suffix for your platform as necessary:

```
INSTALL PLUGIN validate_password SONAME 'validate_password.so';
```

`INSTALL PLUGIN` loads the plugin, and also registers it in the `mysql.plugins` system table to cause the plugin to be loaded for each subsequent normal server startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.5.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE 'validate%';
+-------------------+---------------+
| PLUGIN_NAME       | PLUGIN_STATUS |
+-------------------+---------------+
| validate_password | ACTIVE        |
+-------------------+---------------+
```

If the plugin failed to initialize, check the server error log for diagnostic messages.

If the plugin has been previously registered with `INSTALL PLUGIN` or is loaded with `--plugin-load-add`, you can use the `--validate-password` option at server startup to control plugin activation. For example, to load the plugin at startup and prevent it from being removed at runtime, use these options:

```
[mysqld]
plugin-load-add=validate_password.so
validate-password=FORCE_PLUS_PERMANENT
```

If it is desired to prevent the server from running without the password-validation plugin, use `--validate-password` with a value of `FORCE` or `FORCE_PLUS_PERMANENT` to force server startup to fail if the plugin does not initialize successfully.

6.4.3.2 Password Validation Plugin Options and Variables

This section describes the options, system variables, and status variables that `validate_password` provides to enable its operation to be configured and monitored.

- Password Validation Plugin Options
- Password Validation Plugin System Variables
- Password Validation Plugin Status Variables

Password Validation Plugin Options

To control activation of the `validate_password` plugin, use this option:

- `--validate-password[=value]`

Command-Line Format	`--validate-password[=value]`
Type	Enumeration
Default Value	`ON`

Valid Values	`ON`
	`OFF`
	`FORCE`
	`FORCE_PLUS_PERMANENT`

This option controls how the server loads the `validate_password` plugin at startup. The value should be one of those available for plugin-loading options, as described in Section 5.5.1, “Installing and Uninstalling Plugins”. For example, `--validate-password=FORCE_PLUS_PERMANENT` tells the server to load the plugin at startup and prevents it from being removed while the server is running.

This option is available only if the `validate_password` plugin has been previously registered with `INSTALL PLUGIN` or is loaded with `--plugin-load-add`. See Section 6.4.3.1, “Password Validation Plugin Installation”.

Password Validation Plugin System Variables

If the `validate_password` plugin is enabled, it exposes several system variables that enable configuration of password checking:

```
mysql> SHOW VARIABLES LIKE 'validate_password%';
+--------------------------------------+--------+
| Variable_name                        | Value  |
+--------------------------------------+--------+
| validate_password_dictionary_file    |        |
| validate_password_length             | 8      |
| validate_password_mixed_case_count   | 1      |
| validate_password_number_count       | 1      |
| validate_password_policy             | MEDIUM |
| validate_password_special_char_count | 1      |
+--------------------------------------+--------+
```

To change how passwords are checked, you can set these system variables at server startup or at runtime. The following list describes the meaning of each variable.

- `validate_password_dictionary_file`

Command-Line Format	`--validate-password-dictionary-file=file_name`
System Variable	`validate_password_dictionary_file`
Scope	Global
Dynamic (≥ 5.6.26)	Yes
Dynamic (≤ 5.6.25)	No
Type	File name

The path name of the dictionary file that `validate_password` uses for checking passwords. This variable is unavailable unless `validate_password` is installed.

By default, this variable has an empty value and dictionary checks are not performed. For dictionary checks to occur, the variable value must be nonempty. If the file is named as a relative path, it is interpreted relative to the server data directory. File contents should be lowercase, one word per line. Contents are treated as having a character set of `utf8`. The maximum permitted file size is 1MB.

For the dictionary file to be used during password checking, the password policy must be set to 2 (`STRONG`); see the description of the `validate_password_policy` system variable. Assuming that is true, each substring of the password of length 4 up to 100 is compared to the words in the dictionary file. Any match causes the password to be rejected. Comparisons are not case-sensitive.

For `VALIDATE_PASSWORD_STRENGTH()`, the password is checked against all policies, including `STRONG`, so the strength assessment includes the dictionary check regardless of the `validate_password_policy` value.

Before MySQL 5.6.26, changes to the dictionary file while the server is running require a restart for the server to recognize the changes. As of MySQL 5.6.26, `validate_password_dictionary_file` can be set at runtime and assigning a value causes the named file to be read without a server restart.

- `validate_password_length`

Command-Line Format	`--validate-password-length=#`
System Variable	`validate_password_length`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`8`
Minimum Value	`0`

The minimum number of characters that `validate_password` requires passwords to have. This variable is unavailable unless `validate_password` is installed.

The `validate_password_length` minimum value is a function of several other related system variables. The value cannot be set less than the value of this expression:

```
validate_password_number_count
+ validate_password_special_char_count
+ (2 * validate_password_mixed_case_count)
```

If `validate_password` adjusts the value of `validate_password_length` due to the preceding constraint, it writes a message to the error log.

- `validate_password_mixed_case_count`

Command-Line Format	`--validate-password-mixed-case-count=#`
System Variable	`validate_password_mixed_case_count`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`1`
Minimum Value	`0`

The minimum number of lowercase and uppercase characters that `validate_password` requires passwords to have if the password policy is `MEDIUM` or stronger. This variable is unavailable unless `validate_password` is installed.

For a given `validate_password_mixed_case_count` value, the password must have that many lowercase characters, and that many uppercase characters.

- `validate_password_number_count`

Command-Line Format	`--validate-password-number-count=#`
System Variable	`validate_password_number_count`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`1`
Minimum Value	`0`

The minimum number of numeric (digit) characters that `validate_password` requires passwords to have if the password policy is `MEDIUM` or stronger. This variable is unavailable unless `validate_password` is installed.

- `validate_password_policy`

Command-Line Format	`--validate-password-policy=value`
System Variable	`validate_password_policy`
Scope	Global
Dynamic	Yes
Type	Enumeration
Default Value	`1`

Valid Values	`0`
	`1`
	`2`

The password policy enforced by `validate_password`. This variable is unavailable unless `validate_password` is installed.

The `validate_password_policy` value can be specified using numeric values 0, 1, 2, or the corresponding symbolic values `LOW`, `MEDIUM`, `STRONG`. The following table describes the tests performed for each policy. For the length test, the required length is the value of the `validate_password_length` system variable. Similarly, the required values for the other tests are given by other `validate_password_xxx` variables.

Policy	Tests Performed
`0` or `LOW`	Length
`1` or `MEDIUM`	Length; numeric, lowercase/uppercase, and special characters
`2` or `STRONG`	Length; numeric, lowercase/uppercase, and special characters; dictionary file

- `validate_password_special_char_count`

Command-Line Format	`--validate-password-special-char-count=#`
System Variable	`validate_password_special_char_count`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`1`

Minimum Value	0

The minimum number of nonalphanumeric characters that validate_password requires passwords to have if the password policy is MEDIUM or stronger. This variable is unavailable unless validate_password is installed.

Password Validation Plugin Status Variables

If the validate_password plugin is enabled, it exposes status variables that provide operational information:

```
mysql> SHOW STATUS LIKE 'validate_password%';
+--------------------------------------------+---------------------+
| Variable_name                              | Value               |
+--------------------------------------------+---------------------+
| validate_password_dictionary_file_last_parsed | 2015-06-29 11:08:51 |
| validate_password_dictionary_file_words_count | 1902                |
+--------------------------------------------+---------------------+
```

The following list describes the meaning of each status variable.

- validate_password_dictionary_file_last_parsed

  When the dictionary file was last parsed.

  This variable was added in MySQL 5.6.26.

- validate_password_dictionary_file_words_count

  The number of words read from the dictionary file.

  This variable was added in MySQL 5.6.26.

6.4.4 MySQL Enterprise Audit

Note

MySQL Enterprise Audit is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

MySQL Enterprise Edition includes MySQL Enterprise Audit, implemented using a server plugin named audit_log. MySQL Enterprise Audit uses the open MySQL Audit API to enable standard, policy-based monitoring and logging of connection and query activity executed on specific MySQL servers. Designed to meet the Oracle audit specification, MySQL Enterprise Audit provides an out of box, easy to use auditing and compliance solution for applications that are governed by both internal and external regulatory guidelines.

When installed, the audit plugin enables MySQL Server to produce a log file containing an audit record of server activity. The log contents include when clients connect and disconnect, and what actions they perform while connected, such as which databases and tables they access.

After you install the audit plugin (see Section 6.4.4.1, “Installing MySQL Enterprise Audit”), it writes an audit log file. By default, the file is named audit.log in the server data directory. To change the name of the file, set the audit_log_file system variable at server startup.

Audit log file contents are not encrypted. See Section 6.4.4.2, “MySQL Enterprise Audit Security Considerations”.

The audit log file is written in XML, with auditable events encoded as <AUDIT_RECORD> elements. To select the file format, set the audit_log_format system variable at server startup. For details on file format and contents, see Section 6.4.4.3, “Audit Log File Formats”.

For more information about controlling how logging occurs, including audit log file naming and format selection, see Section 6.4.4.4, “Audit Log Logging Control”. To perform filtering of audited events, see Section 6.4.4.5, “Audit Log Filtering”. For descriptions of the parameters used to configure the audit log plugin, see Audit Log Options and System Variables.

If the audit log plugin is enabled, the Performance Schema (see Chapter 22, *MySQL Performance Schema*) has instrumentation for it. To identify the relevant instruments, use this query:

```
SELECT NAME FROM performance_schema.setup_instruments
WHERE NAME LIKE '%/alog/%';
```

Changes from Older MySQL Enterprise Audit Versions

Several changes were made to the audit log plugin in MySQL 5.6.14 for better compatibility with Oracle Audit Vault.

A new audit log file format was implemented. It is possible to select either the old or new format using the `audit_log_format` system variable, which has permitted values of `OLD` and `NEW` (default `OLD`). The two formats differ as follows:

- Information within `<AUDIT_RECORD>` elements written in the old format using attributes is written in the new format using subelements.
- The new format includes more information in `<AUDIT_RECORD>` elements. Every element includes a `RECORD_ID` value providing a unique identifier. The `TIMESTAMP` value includes time zone information. Query records include `HOST`, `IP`, `OS_LOGIN`, and `USER` information, as well as `COMMAND_CLASS` and `STATUS_CODE` values.

Example of old `<AUDIT_RECORD>` format:

```
<AUDIT_RECORD
 TIMESTAMP="2013-09-15T15:27:27"
 NAME="Query"
 CONNECTION_ID="3"
 STATUS="0"
 SQLTEXT="SELECT 1"
/>
```

Example of new `<AUDIT_RECORD>` format:

```
<AUDIT_RECORD>
 <TIMESTAMP>2013-09-15T15:27:27 UTC</TIMESTAMP>
 <RECORD_ID>3998_2013-09-15T15:27:27</RECORD_ID>
 <NAME>Query</NAME>
 <CONNECTION_ID>3</CONNECTION_ID>
 <STATUS>0</STATUS>
 <STATUS_CODE>0</STATUS_CODE>
 <USER>root[root] @ localhost [127.0.0.1]</USER>
 <OS_LOGIN></OS_LOGIN>
 <HOST>localhost</HOST>
 <IP>127.0.0.1</IP>
 <COMMAND_CLASS>select</COMMAND_CLASS>
 <SQLTEXT>SELECT 1</SQLTEXT>
</AUDIT_RECORD>
```

When the audit log plugin rotates the audit log file, it uses a different file name format. For a log file named `audit.log`, the plugin previously renamed the file to `audit.log.`*`TIMESTAMP`*. The plugin now renames the file to `audit.log.`*`TIMESTAMP`*`.xml` to indicate that it is an XML file.

If you change the value of `audit_log_format`, use this procedure to avoid writing log entries in one format to an existing log file that contains entries in a different format:

1. Stop the server.
2. Rename the current audit log file manually.

3. Restart the server with the new value of `audit_log_format`. The audit log plugin creates a new log file, which contains log entries in the selected format.

The API for writing audit plugins has also changed. The `mysql_event_general` structure has new members to represent client host name and IP address, command class, and external user. For more information, see Writing Audit Plugins.

6.4.4.1 Installing MySQL Enterprise Audit

This section describes how to install MySQL Enterprise Audit, which is implemented using the `audit_log` plugin. For general information about installing plugins, see Section 5.5.1, "Installing and Uninstalling Plugins".

Note

If installed, the `audit_log` plugin involves some minimal overhead even when disabled. To avoid this overhead, do not install MySQL Enterprise Audit unless you plan to use it.

To be usable by the server, the plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base name is `audit_log`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

To load the plugin at server startup, use the `--plugin-load-add` option to name the library file that contains it. With this plugin-loading method, the option must be given each time the server starts. For example, put the following lines in the server `my.cnf` file, adjusting the `.so` suffix for your platform as necessary:

```
[mysqld]
plugin-load-add=audit_log.so
```

After modifying `my.cnf`, restart the server to cause the new settings to take effect.

Alternatively, to load the plugin at runtime, use this statement, adjusting the `.so` suffix for your platform as necessary:

```
INSTALL PLUGIN audit_log SONAME 'audit_log.so';
```

`INSTALL PLUGIN` loads the plugin, and also registers it in the `mysql.plugins` system table to cause the plugin to be loaded for each subsequent normal server startup without the need for `--plugin-load-add`.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.5.2, "Obtaining Server Plugin Information"). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE 'audit%';
+-------------+---------------+
| PLUGIN_NAME | PLUGIN_STATUS |
+-------------+---------------+
| audit_log   | ACTIVE        |
+-------------+---------------+
```

If the plugin failed to initialize, check the server error log for diagnostic messages.

If the plugin has been previously registered with `INSTALL PLUGIN` or is loaded with `--plugin-load-add`, you can use the `--audit-log` option at server startup to control plugin activation. For example, to load the plugin at startup and prevent it from being removed at runtime, use these options:

```
[mysqld]
plugin-load-add=audit_log.so
audit-log=FORCE_PLUS_PERMANENT
```

If it is desired to prevent the server from running without the audit plugin, use `--audit-log` with a value of `FORCE` or `FORCE_PLUS_PERMANENT` to force server startup to fail if the plugin does not initialize successfully.

For additional information about the parameters used to configure operation of the `audit_log` plugin, see Audit Log Options and System Variables.

Audit log file contents are not encrypted. See Section 6.4.4.2, "MySQL Enterprise Audit Security Considerations".

6.4.4.2 MySQL Enterprise Audit Security Considerations

Contents of audit log files produced by the audit log plugin are not encrypted and may contain sensitive information, such as the text of SQL statements. For security reasons, audit log files should be written to a directory accessible only to the MySQL server and to users with a legitimate reason to view the log. The default file name is `audit.log` in the data directory. This can be changed by setting the `audit_log_file` system variable at server startup. Other audit log files may exist due to log rotation.

6.4.4.3 Audit Log File Formats

The MySQL server calls the audit log plugin to write an audit record to its log file whenever an auditable event occurs. Typically the first audit record written after plugin startup contains the server description and startup options. Elements following that one represent events such as client connect and disconnect events, executed SQL statements, and so forth. Only top-level statements are logged, not statements within stored programs such as triggers or stored procedures. Contents of files referenced by statements such as `LOAD DATA` are not logged.

To select the log format that the audit log plugin uses to write its log file, set the `audit_log_format` system variable at server startup. These formats are available:

- New-style XML format (`audit_log_format=NEW`): An XML format that has better compatibility with Oracle Audit Vault than old-style XML format. MySQL 5.7 introduced this format, which was backported to MySQL 5.6 as of MySQL 5.6.14.
- Old-style XML format (`audit_log_format=OLD`): The original audit log format used by default in older MySQL series. MySQL 5.6 uses old-style XML format by default.

Note

Changing the value of `audit_log_format` can result in writing log entries in one format to an existing log file that contains entries in a different format. To avoid this issue, use the procedure described at Audit Log File Format.

Audit log file contents are not encrypted. See Section 6.4.4.2, "MySQL Enterprise Audit Security Considerations".

The following sections describe the available audit logging formats:

- New-Style XML Audit Log File Format
- Old-Style XML Audit Log File Format

New-Style XML Audit Log File Format

Here is a sample log file in new-style XML format (`audit_log_format=NEW`), reformatted slightly for readability:

```
<?xml version="1.0" encoding="utf-8"?>
<AUDIT>
 <AUDIT_RECORD>
  <TIMESTAMP>2017-10-16T14:06:33 UTC</TIMESTAMP>
  <RECORD_ID>1_2017-10-16T14:06:33</RECORD_ID>
  <NAME>Audit</NAME>
  <SERVER_ID>1</SERVER_ID>
  <VERSION>1</VERSION>
```

```
  <STARTUP_OPTIONS>/usr/local/mysql/bin/mysqld
    --socket=/usr/local/mysql/mysql.sock
    --port=3306</STARTUP_OPTIONS>
  <OS_VERSION>i686-Linux</OS_VERSION>
  <MYSQL_VERSION>5.6.39-log</MYSQL_VERSION>
 </AUDIT_RECORD>
 <AUDIT_RECORD>
  <TIMESTAMP>2017-10-16T14:09:38 UTC</TIMESTAMP>
  <RECORD_ID>2_2017-10-16T14:06:33</RECORD_ID>
  <NAME>Connect</NAME>
  <CONNECTION_ID>5</CONNECTION_ID>
  <STATUS>0</STATUS>
  <STATUS_CODE>0</STATUS_CODE>
  <USER>root</USER>
  <OS_LOGIN/>
  <HOST>localhost</HOST>
  <IP>127.0.0.1</IP>
  <COMMAND_CLASS>connect</COMMAND_CLASS>
  <PRIV_USER>root</PRIV_USER>
  <PROXY_USER/>
  <DB>test</DB>
 </AUDIT_RECORD>

...

 <AUDIT_RECORD>
  <TIMESTAMP>2017-10-16T14:09:38 UTC</TIMESTAMP>
  <RECORD_ID>6_2017-10-16T14:06:33</RECORD_ID>
  <NAME>Query</NAME>
  <CONNECTION_ID>5</CONNECTION_ID>
  <STATUS>0</STATUS>
  <STATUS_CODE>0</STATUS_CODE>
  <USER>root[root] @ localhost [127.0.0.1]</USER>
  <OS_LOGIN/>
  <HOST>localhost</HOST>
  <IP>127.0.0.1</IP>
  <COMMAND_CLASS>drop_table</COMMAND_CLASS>
  <SQLTEXT>DROP TABLE IF EXISTS t</SQLTEXT>
 </AUDIT_RECORD>

...

 <AUDIT_RECORD>
  <TIMESTAMP>2017-10-16T14:09:39 UTC</TIMESTAMP>
  <RECORD_ID>8_2017-10-16T14:06:33</RECORD_ID>
  <NAME>Quit</NAME>
  <CONNECTION_ID>5</CONNECTION_ID>
  <STATUS>0</STATUS>
  <STATUS_CODE>0</STATUS_CODE>
  <USER>root</USER>
  <OS_LOGIN/>
  <HOST>localhost</HOST>
  <IP>127.0.0.1</IP>
  <COMMAND_CLASS>connect</COMMAND_CLASS>
 </AUDIT_RECORD>

...

 <AUDIT_RECORD>
  <TIMESTAMP>2017-10-16T14:09:43 UTC</TIMESTAMP>
  <RECORD_ID>11_2017-10-16T14:06:33</RECORD_ID>
  <NAME>Quit</NAME>
  <CONNECTION_ID>6</CONNECTION_ID>
  <STATUS>0</STATUS>
  <STATUS_CODE>0</STATUS_CODE>
  <USER>root</USER>
  <OS_LOGIN/>
  <HOST>localhost</HOST>
  <IP>127.0.0.1</IP>
  <COMMAND_CLASS>connect</COMMAND_CLASS>
 </AUDIT_RECORD>
 <AUDIT_RECORD>
```

```
  <TIMESTAMP>2017-10-16T14:09:45 UTC</TIMESTAMP>
  <RECORD_ID>12_2017-10-16T14:06:33</RECORD_ID>
  <NAME>NoAudit</NAME>
  <SERVER_ID>1</SERVER_ID>
 </AUDIT_RECORD>
</AUDIT>
```

The audit log file is written as XML, using UTF-8 (up to 4 bytes per character). The root element is `<AUDIT>`. The root element contains `<AUDIT_RECORD>` elements, each of which provides information about an audited event. When the audit log plugin begins writing a new log file, it writes the XML declaration and opening `<AUDIT>` root element tag. When the plugin closes a log file, it writes the closing `</AUDIT>` root element tag. The closing tag is not present while the file is open.

Elements within `<AUDIT_RECORD>` elements have these characteristics:

- Some elements appear in every `<AUDIT_RECORD>` element. Others are optional and may appear depending on the audit record type.
- Order of elements within an `<AUDIT_RECORD>` element is not guaranteed.
- Element values are not fixed length. Long values may be truncated as indicated in the element descriptions given later.
- The `<`, `>`, `"`, and `&` characters are encoded as `<`, `>`, `"`, and `&`, respectively. NUL bytes (U+00) are encoded as the `?` character.
- Characters not valid as XML characters are encoded using numeric character references. Valid XML characters are:

```
#x9 | #xA | #xD | [#x20-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]
```

The following elements are mandatory in every `<AUDIT_RECORD>` element:

- `<NAME>`

  A string representing the type of instruction that generated the audit event, such as a command that the server received from a client.

  Example:

  ```
  <NAME>Query</NAME>
  ```

  Some common `<NAME>` values:

  ```
  Audit    When auditing starts, which may be server startup time
  Connect  When a client connects, also known as logging in
  Query    An SQL statement (executed directly)
  Prepare  Preparation of an SQL statement; usually followed by Execute
  Execute  Execution of an SQL statement; usually follows Prepare
  Shutdown Server shutdown
  Quit     When a client disconnects
  NoAudit  Auditing has been turned off
  ```

  The possible values are `Audit`, `Binlog Dump`, `Change user`, `Close stmt`, `Connect Out`, `Connect`, `Create DB`, `Daemon`, `Debug`, `Delayed insert`, `Drop DB`, `Execute`, `Fetch`, `Field List`, `Init DB`, `Kill`, `Long Data`, `NoAudit`, `Ping`, `Prepare`, `Processlist`, `Query`, `Quit`, `Refresh`, `Register Slave`, `Reset stmt`, `Set option`, `Shutdown`, `Sleep`, `Statistics`, `Table Dump`, `Time`.

  With the exception of `Audit` and `NoAudit`, these values correspond to the `COM_`*`xxx`* command values listed in the `mysql_com.h` header file. For example, `Create DB` and `Change user` correspond to `COM_CREATE_DB` and `COM_CHANGE_USER`, respectively.

- `<RECORD_ID>`

A unique identifier for the audit record. The value is composed from a sequence number and timestamp, in the format *`SEQ_TIMESTAMP`*. When the audit log plugin opens the audit log file, it initializes the sequence number to the size of the audit log file, then increments the sequence by 1 for each record logged. The timestamp is a UTC value in *`YYYY-MM-DD`*`Thh:mm:ss` format indicating the date and time when the audit log plugin opened the file.

Example:

```
<RECORD_ID>12_2017-10-16T14:06:33</RECORD_ID>
```

- `<TIMESTAMP>`

  A string representing a UTC value in *`YYYY-MM-DD`*`Thh:mm:ss UTC` format indicating the date and time when the audit event was generated. For example, the event corresponding to execution of an SQL statement received from a client has a `<TIMESTAMP>` value occurring after the statement finishes, not when it was received.

  Example:

```
<TIMESTAMP>2017-10-16T14:09:45 UTC</TIMESTAMP>
```

The following elements are optional in `<AUDIT_RECORD>` elements. Many of them occur only with specific `<NAME>` element values.

- `<COMMAND_CLASS>`

  A string that indicates the type of action performed.

  Example:

```
<COMMAND_CLASS>drop_table</COMMAND_CLASS>
```

  The values correspond to the `Com_`*`xxx`* status variables that indicate command counts; for example `Com_drop_table` and `Com_select` count `DROP TABLE` and `SELECT` statements, respectively. The following statement displays the possible names:

```
SELECT LOWER(REPLACE(VARIABLE_NAME, 'COM_', '')) AS name
FROM INFORMATION_SCHEMA.GLOBAL_STATUS
WHERE VARIABLE_NAME LIKE 'COM%'
ORDER BY name;
```

- `<CONNECTION_ID>`

  An unsigned integer representing the client connection identifier. This is the same as the value returned by the `CONNECTION_ID()` function within the session.

  Example:

```
<CONNECTION_ID>127</CONNECTION_ID>
```

- `<DB>`

  A string representing the default database name.

  Example:

```
<DB>test</DB>
```

- `<HOST>`

  A string representing the client host name.

  Example:

```
<HOST>localhost</HOST>
```

- `<IP>`

  A string representing the client IP address.

  Example:

  ```
  <IP>127.0.0.1</IP>
  ```

- `<MYSQL_VERSION>`

  A string representing the MySQL server version. This is the same as the value of the `VERSION()` function or `version` system variable.

  Example:

  ```
  <MYSQL_VERSION>5.6.39-log</MYSQL_VERSION>
  ```

- `<OS_LOGIN>`

  A string representing the external user name used during the authentication process, as set by the plugin used to authenticate the client. With native (built-in) MySQL authentication, or if the plugin does not set the value, this element is empty. The value is the same as that of the `external_user` system variable (see Section 6.2.12, “Proxy Users”).

  Example:

  ```
  <OS_LOGIN>jeffrey</OS_LOGIN>
  ```

- `<OS_VERSION>`

  A string representing the operating system on which the server was built or is running.

  Example:

  ```
  <OS_VERSION>x86_64-Linux</OS_VERSION>
  ```

- `<PRIV_USER>`

  A string representing the user that the server authenticated the client as. This is the user name that the server uses for privilege checking, and may differ from the `<USER>` value.

  Example:

  ```
  <PRIV_USER>jeffrey</PRIV_USER>
  ```

- `<PROXY_USER>`

  A string representing the proxy user (see Section 6.2.12, “Proxy Users”). The value is empty if user proxying is not in effect.

  Example:

  ```
  <PROXY_USER>developer</PROXY_USER>
  ```

- `<SERVER_ID>`

  An unsigned integer representing the server ID. This is the same as the value of the `server_id` system variable.

  Example:

  ```
  <SERVER_ID>1</SERVER_ID>
  ```

- `<SQLTEXT>`

  A string representing the text of an SQL statement. The value can be empty. Long values may be truncated. The string, like the audit log file itself, is written using UTF-8 (up to 4 bytes per character), so the value may be the result of conversion. For example, the original statement might have been received from the client as an SJIS string.

  Example:

  ```
  <SQLTEXT>DELETE FROM t1</SQLTEXT>
  ```

- `<STARTUP_OPTIONS>`

  A string representing the options that were given on the command line or in option files when the MySQL server was started. The first option is the path to the server executable.

  Example:

  ```
  <STARTUP_OPTIONS>/usr/local/mysql/bin/mysqld
    --port=3306 --log_output=FILE</STARTUP_OPTIONS>
  ```

- `<STATUS>`

  An unsigned integer representing the command status: 0 for success, nonzero if an error occurred. This is the same as the value of the `mysql_errno()` C API function. See the description for `<STATUS_CODE>` for information about how it differs from `<STATUS>`.

  The audit log does not contain the SQLSTATE value or error message. To see the associations between error codes, SQLSTATE values, and messages, see Server Error Message Reference.

  Warnings are not logged.

  Example:

  ```
  <STATUS>1051</STATUS>
  ```

- `<STATUS_CODE>`

  An unsigned integer representing the command status: 0 for success, 1 if an error occurred.

  The `STATUS_CODE` value differs from the `STATUS` value: `STATUS_CODE` is 0 for success and 1 for error, which is compatible with the EZ_collector consumer for Audit Vault. `STATUS` is the value of the `mysql_errno()` C API function. This is 0 for success and nonzero for error, and thus is not necessarily 1 for error.

  Example:

  ```
  <STATUS_CODE>0</STATUS_CODE>
  ```

- `<USER>`

  A string representing the user name sent by the client. This may differ from the `<PRIV_USER>` value.

  Example:

  ```
  <USER>root[root] @ localhost [127.0.0.1]</USER>
  ```

- `<VERSION>`

  An unsigned integer representing the version of the audit log file format.

  Example:

  ```
  <VERSION>1</VERSION>
  ```

Old-Style XML Audit Log File Format

Here is a sample log file in old-style XML format (`audit_log_format=OLD`), reformatted slightly for readability:

```
<?xml version="1.0" encoding="utf-8"?>
<AUDIT>
  <AUDIT_RECORD
    TIMESTAMP="2017-10-16T14:25:00 UTC"
    RECORD_ID="1_2017-10-16T14:25:00"
    NAME="Audit"
    SERVER_ID="1"
    VERSION="1"
    STARTUP_OPTIONS="--port=3306"
    OS_VERSION="i686-Linux"
    MYSQL_VERSION="5.6.39-log"/>
  <AUDIT_RECORD
    TIMESTAMP="2017-10-16T14:25:24 UTC"
    RECORD_ID="2_2017-10-16T14:25:00"
    NAME="Connect"
    CONNECTION_ID="4"
    STATUS="0"
    STATUS_CODE="0"
    USER="root"
    OS_LOGIN=""
    HOST="localhost"
    IP="127.0.0.1"
    COMMAND_CLASS="connect"
    PRIV_USER="root"
    PROXY_USER=""
    DB="test"/>

...

  <AUDIT_RECORD
    TIMESTAMP="2017-10-16T14:25:24 UTC"
    RECORD_ID="6_2017-10-16T14:25:00"
    NAME="Query"
    CONNECTION_ID="4"
    STATUS="0"
    STATUS_CODE="0"
    USER="root[root] @ localhost [127.0.0.1]"
    OS_LOGIN=""
    HOST="localhost"
    IP="127.0.0.1"
    COMMAND_CLASS="drop_table"
    SQLTEXT="DROP TABLE IF EXISTS t"/>

...

  <AUDIT_RECORD
    TIMESTAMP="2017-10-16T14:25:24 UTC"
    RECORD_ID="8_2017-10-16T14:25:00"
    NAME="Quit"
    CONNECTION_ID="4"
    STATUS="0"
    STATUS_CODE="0"
    USER="root"
    OS_LOGIN=""
    HOST="localhost"
    IP="127.0.0.1"
    COMMAND_CLASS="connect"
  <AUDIT_RECORD
    TIMESTAMP="2017-10-16T14:25:32 UTC"
    RECORD_ID="12_2017-10-16T14:25:00"
    NAME="NoAudit"
    SERVER_ID="1"/>
</AUDIT>
```

The audit log file is written as XML, using UTF-8 (up to 4 bytes per character). The root element is `<AUDIT>`. The root element contains `<AUDIT_RECORD>` elements, each of which provides information

about an audited event. When the audit log plugin begins writing a new log file, it writes the XML declaration and opening `<AUDIT>` root element tag. When the plugin closes a log file, it writes the closing `</AUDIT>` root element tag. The closing tag is not present while the file is open.

Attributes of `<AUDIT_RECORD>` elements have these characteristics:

- Some attributes appear in every `<AUDIT_RECORD>` element. Others are optional and may appear depending on the audit record type.
- Order of attributes within an `<AUDIT_RECORD>` element is not guaranteed.
- Attribute values are not fixed length. Long values may be truncated as indicated in the attribute descriptions given later.
- The `<`, `>`, `"`, and `&` characters are encoded as `<`, `>`, `"`, and `&`, respectively. NUL bytes (U+00) are encoded as the `?` character.
- Characters not valid as XML characters are encoded using numeric character references. Valid XML characters are:

```
#x9 | #xA | #xD | [#x20-#xD7FF] | [#xE000-#xFFFD] | [#x10000-#x10FFFF]
```

The following attributes are mandatory in every `<AUDIT_RECORD>` element:

- `NAME`

  A string representing the type of instruction that generated the audit event, such as a command that the server received from a client.

  Example: `NAME="Query"`

  Some common `NAME` values:

```
Audit    When auditing starts, which may be server startup time
Connect  When a client connects, also known as logging in
Query    An SQL statement (executed directly)
Prepare  Preparation of an SQL statement; usually followed by Execute
Execute  Execution of an SQL statement; usually follows Prepare
Shutdown Server shutdown
Quit     When a client disconnects
NoAudit  Auditing has been turned off
```

  The possible values are `Audit`, `Binlog Dump`, `Change user`, `Close stmt`, `Connect Out`, `Connect`, `Create DB`, `Daemon`, `Debug`, `Delayed insert`, `Drop DB`, `Execute`, `Fetch`, `Field List`, `Init DB`, `Kill`, `Long Data`, `NoAudit`, `Ping`, `Prepare`, `Processlist`, `Query`, `Quit`, `Refresh`, `Register Slave`, `Reset stmt`, `Set option`, `Shutdown`, `Sleep`, `Statistics`, `Table Dump`, `Time`.

  With the exception of `"Audit"` and `"NoAudit"`, these values correspond to the `COM_`*`xxx`* command values listed in the `mysql_com.h` header file. For example, `"Create DB"` and `"Change user"` correspond to `COM_CREATE_DB` and `COM_CHANGE_USER`, respectively.

- `RECORD_ID`

  A unique identifier for the audit record. The value is composed from a sequence number and timestamp, in the format *`SEQ_TIMESTAMP`*. When the audit log plugin opens the audit log file, it initializes the sequence number to the size of the audit log file, then increments the sequence by 1 for each record logged. The timestamp is a UTC value in *`YYYY-MM-DD`*`Thh:mm:ss` format indicating the date and time when the audit log plugin opened the file.

  Example: `RECORD_ID="12_2017-10-16T14:25:00"`

- `TIMESTAMP`

A string representing a UTC value in *YYYY-MM-DD*T*hh:mm:ss* UTC format indicating the date and time when the audit event was generated. For example, the event corresponding to execution of an SQL statement received from a client has a TIMESTAMP value occurring after the statement finishes, not when it was received.

Example: TIMESTAMP="2017-10-16T14:25:32 UTC"

The following attributes are optional in <AUDIT_RECORD> elements. Many of them occur only for elements with specific values of the NAME attribute.

- COMMAND_CLASS

  A string that indicates the type of action performed.

  Example: COMMAND_CLASS="drop_table"

  The values correspond to the Com_*xxx* status variables that indicate command counts; for example Com_drop_table and Com_select count DROP TABLE and SELECT statements, respectively. The following statement displays the possible names:

```
SELECT LOWER(REPLACE(VARIABLE_NAME, 'COM_', '')) AS name
FROM INFORMATION_SCHEMA.GLOBAL_STATUS
WHERE VARIABLE_NAME LIKE 'COM%'
ORDER BY name;
```

- CONNECTION_ID

  An unsigned integer representing the client connection identifier. This is the same as the value returned by the CONNECTION_ID() function within the session.

  Example: CONNECTION_ID="127"

- DB

  A string representing the default database name.

  Example: DB="test"

- HOST

  A string representing the client host name.

  Example: HOST="localhost"

- IP

  A string representing the client IP address.

  Example: IP="127.0.0.1"

- MYSQL_VERSION

  A string representing the MySQL server version. This is the same as the value of the VERSION() function or version system variable.

  Example: MYSQL_VERSION="5.6.39-log"

- OS_LOGIN

  A string representing the external user name used during the authentication process, as set by the plugin used to authenticate the client. With native (built-in) MySQL authentication, or if the plugin does not set the value, this attribute is empty. The value is the same as that of the external_user system variable (see Section 6.2.12, “Proxy Users”).

Example: `OS_LOGIN="jeffrey"`

- `OS_VERSION`

  A string representing the operating system on which the server was built or is running.

  Example: `OS_VERSION="x86_64-Linux"`

- `PRIV_USER`

  A string representing the user that the server authenticated the client as. This is the user name that the server uses for privilege checking, and it may differ from the `USER` value.

  Example: `PRIV_USER="jeffrey"`

- `PROXY_USER`

  A string representing the proxy user (see Section 6.2.12, “Proxy Users”). The value is empty if user proxying is not in effect.

  Example: `PROXY_USER="developer"`

- `SERVER_ID`

  An unsigned integer representing the server ID. This is the same as the value of the `server_id` system variable.

  Example: `SERVER_ID="1"`

- `SQLTEXT`

  A string representing the text of an SQL statement. The value can be empty. Long values may be truncated. The string, like the audit log file itself, is written using UTF-8 (up to 4 bytes per character), so the value may be the result of conversion. For example, the original statement might have been received from the client as an SJIS string.

  Example: `SQLTEXT="DELETE FROM t1"`

- `STARTUP_OPTIONS`

  A string representing the options that were given on the command line or in option files when the MySQL server was started.

  Example: `STARTUP_OPTIONS="--port=3306 --log_output=FILE"`

- `STATUS`

  An unsigned integer representing the command status: 0 for success, nonzero if an error occurred. This is the same as the value of the `mysql_errno()` C API function. See the description for `STATUS_CODE` for information about how it differs from `STATUS`.

  The audit log does not contain the SQLSTATE value or error message. To see the associations between error codes, SQLSTATE values, and messages, see Server Error Message Reference.

  Warnings are not logged.

  Example: `STATUS="1051"`

- `STATUS_CODE`

  An unsigned integer representing the command status: 0 for success, 1 if an error occurred.

  The `STATUS_CODE` value differs from the `STATUS` value: `STATUS_CODE` is 0 for success and 1 for error, which is compatible with the EZ_collector consumer for Audit Vault. `STATUS` is the value of the `mysql_errno()` C API function. This is 0 for success and nonzero for error, and thus is not necessarily 1 for error.

  Example: `STATUS_CODE="0"`

- `USER`

  A string representing the user name sent by the client. This may differ from the `PRIV_USER` value.

- `VERSION`

  An unsigned integer representing the version of the audit log file format.

  Example: `VERSION="1"`

6.4.4.4 Audit Log Logging Control

This section describes how to control general characteristics of audit logging, such as the file to which the audit log plugin writes events and the format of written events.

- Audit Log File Naming Conventions
- Audit Log File Format
- Audit Logging Write Strategy
- Space Management and Rotation of Audit Log Files

For additional information about the system variables that affect audit logging, see Audit Log Options and System Variables.

The audit log plugin can also control which audited events are written to the audit log file, based on the account from which events originate or event status. See Section 6.4.4.5, “Audit Log Filtering”.

Audit Log File Naming Conventions

To control the audit log file name, set the `audit_log_file` system variable at server startup. The default name is `audit.log` in the server data directory. For best security, write the audit log to a directory accessible only to the MySQL server and to users with a legitimate reason to view the log.

When the audit plugin initializes, it checks whether a file with the audit log file name already exists. If so, the plugin checks whether the file ends with an `</AUDIT>` tag and truncates the tag before writing any `<AUDIT_RECORD>` elements. If the log file exists but does not end with `</AUDIT>` or the `</AUDIT>` tag cannot be truncated, the plugin considers the file malformed and fails to initialize. This can occur if the server exits unexpectedly with the audit log plugin running. No logging occurs until the problem is rectified. Check the error log for diagnostic information:

```
[ERROR] Plugin 'audit_log' init function returned error.
```

To deal with this problem, either remove or rename the malformed log file and restart the server.

Audit Log File Format

To control the audit log file format, set the `audit_log_format` system variable at server startup. By default, the format is `OLD` (old-style XML format). For information about available formats, see Section 6.4.4.3, “Audit Log File Formats”.

Note

Changing the value of `audit_log_format` can result in writing log entries in one format to an existing log file that contains entries in a different format. To avoid this issue, use the following procedure:

1. Stop the server.
2. Either change the value of the `audit_log_file` system variable so the plugin writes to a different file, or rename the current audit log file manually.
3. Restart the server with the new value of `audit_log_format`. The audit log plugin creates a new log file and writes entries to it in the selected format.

Audit Logging Write Strategy

The audit log plugin can use any of several strategies for log writes. Regardless of strategy, logging occurs on a best-effort basis, with no guarantee of consistency.

To specify a write strategy, set the `audit_log_strategy` system variable at server startup. By default, the strategy value is `ASYNCHRONOUS` and the plugin logs asynchronously to a buffer, waiting if the buffer is full. It's possible to tell the plugin not to wait (`PERFORMANCE`) or to log synchronously, either using file system caching (`SEMISYNCHRONOUS`) or forcing output with a `sync()` call after each write request (`SYNCHRONOUS`).

For asynchronous write strategy, the `audit_log_buffer_size` system variable is the buffer size in bytes. Set this variable at server startup to change the buffer size. The plugin uses a single buffer, which it allocates when it initializes and removes when it terminates. The plugin does not allocate this buffer for nonasynchronous write strategies.

Asynchronous logging strategy has these characteristics:

- Minimal impact on server performance and scalability.
- Blocking of threads that generate audit events for the shortest possible time; that is, time to allocate the buffer plus time to copy the event to the buffer.
- Output goes to the buffer. A separate thread handles writes from the buffer to the log file.

With asynchronous logging, the integrity of the log file may be compromised if a problem occurs during a write to the file or if the plugin does not shut down cleanly (for example, in the event that the server host exits unexpectedly). To reduce this risk, set `audit_log_strategy` to use synchronous logging.

If the file system to which the audit log is being written fills up, a “disk full” error is written to the error log. Audit logging continues until the audit log buffer is full. If free disk space has not been made available by the time the buffer fills, client sessions hang, and stopping the server at the time of client sessions hanging results in audit log corruption. To avoid this if client sessions are hung, ensure that free space is available on the audit logging file system before stopping the server.

A disadvantage of `PERFORMANCE` strategy is that it drops events when the buffer is full. For a heavily loaded server, the audit log may have events missing.

Space Management and Rotation of Audit Log Files

The audit log file has the potential to grow quite large and consume a great deal of disk space. To enable management of the space used by its log files, the audit log plugin provides the `audit_log_rotate_on_size` and `audit_log_flush` system variables, which control audit log file rotation and flushing. Rotation can be done manually, or automatically based on file size.

Manual audit log file rotation. If `audit_log_rotate_on_size` is 0 (the default), no log rotation occurs except that performed manually. In this case, the audit log plugin closes and reopens the log file when the `audit_log_flush` value changes from disabled to enabled. Log file renaming must be done externally to the server. Suppose that the log file name is `audit.log` and you want to maintain

the three most recent log files, cycling through the names `audit.log.1` through `audit.log.3`. On Unix, perform rotation manually like this:

1. From the command line, rename the current log files:

   ```
   mv audit.log.2 audit.log.3
   mv audit.log.1 audit.log.2
   mv audit.log audit.log.1
   ```

   At this point, the plugin is still writing to the current log file, which has been renamed to `audit.log.1`.

2. Connect to the server and flush the log file so the plugin closes it and reopens a new `audit.log` file:

   ```
   SET GLOBAL audit_log_flush = ON;
   ```

Automatic size-based audit log file rotation. If `audit_log_rotate_on_size` is greater than 0, setting `audit_log_flush` has no effect. Instead, whenever a write to the log file causes its size to exceed the `audit_log_rotate_on_size` value, the audit log plugin automatically closes the file, renames it, and opens a new log file.

The renamed file has a timestamp and `.xml` added to the end. For example, if the file name is `audit.log`, the plugin renames it to a value such as `audit.log.15081807937726520.xml`. The timestamp value is similar to a Unix timestamp, with the last 7 digits representing the fractional second part. By inserting a decimal point, the value can be interpreted using the `FROM_UNIXTIME()` function:

```
mysql> SELECT FROM_UNIXTIME(1508180793.7726520);
+----------------------------------+
| FROM_UNIXTIME(1508180793.7726520) |
+----------------------------------+
| 2017-10-16 14:06:33.772652       |
+----------------------------------+
```

Note

With size-based log file rotation, renamed log files do not rotate off the end of the name sequence. Instead, they have unique names and accumulate indefinitely. To avoid excessive space use, remove old files periodically, backing them up first as necessary.

6.4.4.5 Audit Log Filtering

The audit log plugin can filter audited events. This enables you to control whether audited events are written to the audit log file based on the account from which events originate or event status. Status filtering occurs separately for connection events and statement events.

Event Filtering by Account

As of MySQL 5.6.20, to filter audited events based on the originating account, set one of these system variables at server startup or runtime:

- `audit_log_include_accounts`: The accounts to include in audit logging. If this variable is set, only these accounts are audited.
- `audit_log_exclude_accounts`: The accounts to exclude from audit logging. If this variable is set, all but these accounts are audited.

The value for either variable can be `NULL` or a string containing one or more comma-separated account names, each in *`user_name@host_name`* format. By default, both variables are `NULL`, in which case, no account filtering is done and auditing occurs for all accounts.

Modifications to `audit_log_include_accounts` or `audit_log_exclude_accounts` affect only connections created subsequent to the modification, not existing connections.

Example: To enable audit logging only for the `user1` and `user2` local host account accounts, set the `audit_log_include_accounts` system variable like this:

```
SET GLOBAL audit_log_include_accounts = 'user1@localhost,user2@localhost';
```

Only one of `audit_log_include_accounts` or `audit_log_exclude_accounts` can be non-`NULL` at a time:

- If you set `audit_log_include_accounts`, the server sets `audit_log_exclude_accounts` to `NULL`.
- If you attempt to set `audit_log_exclude_accounts`, an error occurs unless `audit_log_include_accounts` is `NULL`. In this case, you must first clear `audit_log_include_accounts` by setting it to `NULL`.

```
-- This sets audit_log_exclude_accounts to NULL
SET GLOBAL audit_log_include_accounts = value;

-- This fails because audit_log_include_accounts is not NULL
SET GLOBAL audit_log_exclude_accounts = value;

-- To set audit_log_exclude_accounts, first set
-- audit_log_include_accounts to NULL
SET GLOBAL audit_log_include_accounts = NULL;
SET GLOBAL audit_log_exclude_accounts = value;
```

If you inspect the value of either variable, be aware that `SHOW VARIABLES` displays `NULL` as an empty string. To avoid this, use `SELECT` instead:

```
mysql> SHOW VARIABLES LIKE 'audit_log_include_accounts';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| audit_log_include_accounts |       |
+----------------------------+-------+
mysql> SELECT @@audit_log_include_accounts;
+-----------------------------+
| @@audit_log_include_accounts |
+-----------------------------+
| NULL                        |
+-----------------------------+
```

If a user name or host name requires quoting because it contains a comma, space, or other special character, quote it using single quotes. If the variable value itself is quoted with single quotes, double each inner single quote or escape it with a backslash. The following statements each enable audit logging for the local `root` account and are equivalent, even though the quoting styles differ:

```
SET GLOBAL audit_log_include_accounts = 'root@localhost';
SET GLOBAL audit_log_include_accounts = '''root''@''localhost''';
SET GLOBAL audit_log_include_accounts = '\'root\'@\'localhost\'';
SET GLOBAL audit_log_include_accounts = "'root'@'localhost'";
```

The last statement does not work if the `ANSI_QUOTES` SQL mode is enabled because in that mode double quotes signify identifier quoting, not string quoting.

Event Filtering by Status

As of MySQL 5.6.20, to filter audited events based on status, set these system variables at server startup or runtime:

- `audit_log_connection_policy`: Logging policy for connection events
- `audit_log_statement_policy`: Logging policy for statement events

Each variable takes a value of `ALL` (log all associated events; this is the default), `ERRORS` (log only failed events), or `NONE` (do not log events). For example, to log all statement events but only failed connection events, use these settings:

```
SET GLOBAL audit_log_statement_policy = ALL;
SET GLOBAL audit_log_connection_policy = ERRORS;
```

Before MySQL 5.6.20, `audit_log_connection_policy` and `audit_log_statement_policy` are not available. Instead, use `audit_log_policy` at server startup or runtime. It takes a value of `ALL` (log all events; this is the default), `LOGINS` (log connection events), `QUERIES` (log statement events), or `NONE` (do not log events). For any of those values, the audit log plugin logs all selected events without distinction as to success or failure.

As of MySQL 5.6.20, `audit_log_policy` is still available but can be set only at server startup. At runtime, it is a read-only variable. Its use at startup works as follows:

- If you do not set `audit_log_policy` or set it to its default of `ALL`, any explicit settings for `audit_log_connection_policy` or `audit_log_statement_policy` apply as specified. If not specified, they default to `ALL`.
- If you set `audit_log_policy` to a non-`ALL` value, that value takes precedence over and is used to set `audit_log_connection_policy` and `audit_log_statement_policy`, as indicated in the following table. If you also set either of those variables to a value other than their default of `ALL`, the server writes a message to the error log to indicate that their values are being overridden.

Startup audit_log_policy Value	Resulting audit_log_connection_policy Value	Resulting audit_log_statement_policy Value
`LOGINS`	`ALL`	`NONE`
`QUERIES`	`NONE`	`ALL`
`NONE`	`NONE`	`NONE`

6.4.4.6 Audit Log Reference

The following sections provide a reference to MySQL Enterprise Audit elements:

- Audit Log Option and Variable Reference
- Audit Log Options and System Variables
- Audit Log Plugin Status Variables

Audit Log Option and Variable Reference

Table 6.16 Audit Log Option and Variable Reference

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
audit-log	Yes	Yes				
audit_log_buffer_size	Yes	Yes	Yes		Global	No
audit_log_connection_policy	Yes	Yes	Yes		Global	Yes
audit_log_current_session			Yes		Both	No
Audit_log_current_size				Yes	Global	No
Audit_log_event_max_drop_size				Yes	Global	No
Audit_log_events				Yes	Global	No
Audit_log_events_filtered				Yes	Global	No
Audit_log_events_lost				Yes	Global	No
Audit_log_events_written				Yes	Global	No
audit_log_exclude_accounts	Yes	Yes	Yes		Global	Yes
audit_log_file	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
audit_log_flush			Yes		Global	Yes
audit_log_format	Yes	Yes	Yes		Global	No
audit_log_include_accounts	Yes	Yes	Yes		Global	Yes
audit_log_policy	Yes	Yes	Yes		Global	Yes
audit_log_rotate_on_size	Yes	Yes	Yes		Global	Yes
audit_log_statement_policy	Yes	Yes	Yes		Global	Yes
audit_log_strategy	Yes	Yes	Yes		Global	No
Audit_log_total_size				Yes	Global	No
Audit_log_write_waits				Yes	Global	No

Audit Log Options and System Variables

This section describes the command options and system variables that control operation of MySQL Enterprise Audit. If values specified at startup time are incorrect, the audit log plugin may fail to initialize properly and the server does not load it. In this case, the server may also produce error messages for other audit log settings because it does not recognize them.

To control activation of the audit log plugin, use this option:

- `--audit-log[=value]`

Command-Line Format	`--audit-log[=value]`
Type	Enumeration
Default Value	`ON`

Valid Values	`ON`
	`OFF`
	`FORCE`
	`FORCE_PLUS_PERMANENT`

This option controls how the server loads the `audit_log` plugin at startup. It is available only if the plugin has been previously registered with `INSTALL PLUGIN` or is loaded with `--plugin-load` or `--plugin-load-add`. See Section 6.4.4.1, “Installing MySQL Enterprise Audit”.

The option value should be one of those available for plugin-loading options, as described in Section 5.5.1, “Installing and Uninstalling Plugins”. For example, `--audit-log=FORCE_PLUS_PERMANENT` tells the server to load the plugin at startup and prevents it from being removed while the server is running.

If the audit log plugin is enabled, it exposes several system variables that permit control over logging:

```
mysql> SHOW VARIABLES LIKE 'audit_log%';
+-------------------------------+--------------+
| Variable_name                 | Value        |
+-------------------------------+--------------+
| audit_log_buffer_size         | 1048576      |
| audit_log_connection_policy   | ALL          |
| audit_log_current_session     | ON           |
| audit_log_exclude_accounts    |              |
| audit_log_file                | audit.log    |
| audit_log_flush               | OFF          |
| audit_log_format              | OLD          |
| audit_log_include_accounts    |              |
| audit_log_policy              | ALL          |
| audit_log_rotate_on_size      | 0            |
```

```
| audit_log_statement_policy  | ALL          |
| audit_log_strategy          | ASYNCHRONOUS |
+-----------------------------+--------------+
```

You can set any of these variables at server startup, and some of them at runtime.

- `audit_log_buffer_size`

Command-Line Format	`--audit-log-buffer-size=#`
System Variable	`audit_log_buffer_size`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`1048576`
Minimum Value	`4096`
Maximum Value (64-bit platforms)	`18446744073709547520`
Maximum Value (32-bit platforms)	`4294967295`

When the audit log plugin writes events to the log asynchronously, it uses a buffer to store event contents prior to writing them. This variable controls the size of that buffer, in bytes. The server adjusts the value to a multiple of 4096. The plugin uses a single buffer, which it allocates when it initializes and removes when it terminates. The plugin allocates this buffer only if logging is asynchronous.

- `audit_log_connection_policy`

Command-Line Format	`--audit-log-connection-policy=value`
Introduced	5.6.20
System Variable	`audit_log_connection_policy`
Scope	Global
Dynamic	Yes
Type	Enumeration
Default Value	`ALL`

Valid Values	`ALL`
	`ERRORS`
	`NONE`

The policy controlling how the audit log plugin writes connection events to its log file. The following table shows the permitted values.

Value	Description
`ALL`	Log all connection events
`ERRORS`	Log only failed connection events
`NONE`	Do not log connection events

Note

At server startup, any explicit value given for `audit_log_connection_policy` may be overridden if

> `audit_log_policy` is also specified, as described in Section 6.4.4.4, “Audit Log Logging Control”.

- `audit_log_current_session`

Introduced	5.6.20
System Variable	`audit_log_current_session`
Scope	Global, Session
Dynamic	No
Type	Boolean
Default Value	`depends on filtering policy`

Whether audit logging is enabled for the current session. The session value of this variable is read only. It is set when the session begins based on the values of the `audit_log_include_accounts` and `audit_log_exclude_accounts` system variables. The audit log plugin uses the session value to determine whether to audit events for the session. (There is a global value, but the plugin does not use it.)

- `audit_log_exclude_accounts`

Command-Line Format	`--audit-log-exclude-accounts=value`
Introduced	5.6.20
System Variable	`audit_log_exclude_accounts`
Scope	Global
Dynamic	Yes
Type	String
Default Value	`NULL`

The accounts for which events should not be logged. The value should be `NULL` or a string containing a list of one or more comma-separated account names. For more information, see Section 6.4.4.5, “Audit Log Filtering”.

Modifications to `audit_log_exclude_accounts` affect only connections created subsequent to the modification, not existing connections.

- `audit_log_file`

Command-Line Format	`--audit-log-file=file_name`
System Variable	`audit_log_file`
Scope	Global
Dynamic	No
Type	File name
Default Value	`audit.log`

The name of the file to which the audit log plugin writes events. The default value is `audit.log`. If the value of `audit_log_file` is a relative path name, the plugin interprets it relative to the data directory. If the value is a full path name, the plugin uses the value as is. A full path name may be useful if it is desirable to locate audit files on a separate file system or directory. For security reasons, the audit log file should be written to a directory accessible only to the MySQL server and to users with a legitimate reason to view the log. For more information, see Section 6.4.4.4, “Audit Log Logging Control”.

- `audit_log_flush`

System Variable	audit_log_flush
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	OFF

When this variable is set to enabled (1 or ON), the audit log plugin closes and reopens its log file to flush it. (The value remains OFF so that you need not disable it explicitly before enabling it again to perform another flush.) Enabling this variable has no effect unless audit_log_rotate_on_size is 0. For more information, see Section 6.4.4.4, “Audit Log Logging Control”.

- audit_log_format

Command-Line Format	--audit-log-format=value
Introduced	5.6.14
System Variable	audit_log_format
Scope	Global
Dynamic	No
Type	Enumeration
Default Value	OLD
Valid Values	OLD NEW

The audit log file format. Permitted values are OLD and NEW (default OLD). For details about each format, see Section 6.4.4.3, “Audit Log File Formats”.

Note

Changing the value of audit_log_format can result in writing log entries in one format to an existing log file that contains entries in a different format. To avoid this issue, use the procedure described at Audit Log File Format.

- audit_log_include_accounts

Command-Line Format	--audit-log-include-accounts=value
Introduced	5.6.20
System Variable	audit_log_include_accounts
Scope	Global
Dynamic	Yes
Type	String
Default Value	NULL

The accounts for which events should be logged. The value should be NULL or a string containing a list of one or more comma-separated account names. For more information, see Section 6.4.4.5, “Audit Log Filtering”.

Modifications to audit_log_include_accounts affect only connections created subsequent to the modification, not existing connections.

- `audit_log_policy`

Command-Line Format	`--audit-log-policy=value`
System Variable	`audit_log_policy`
Scope	Global
Dynamic (≥ 5.6.20)	No
Dynamic (≤ 5.6.19)	Yes
Type	Enumeration
Default Value	`ALL`

Valid Values	`ALL`
	`LOGINS`
	`QUERIES`
	`NONE`

The policy controlling how the audit log plugin writes events to its log file. The following table shows the permitted values.

Value	Description
`ALL`	Log all events
`LOGINS`	Log only login events
`QUERIES`	Log only query events
`NONE`	Log nothing (disable the audit stream)

As of MySQL 5.6.20, `audit_log_policy` can be set only at server startup. At runtime, it is a read-only variable. This is due to the introduction of two other system variables, `audit_log_connection_policy` and `audit_log_statement_policy`, that provide finer control over logging policy and that can be set either at startup or at runtime. If you continue to use `audit_log_policy` at startup instead of the other two variables, the server uses its value to set those variables. For more information about the policy variables and their interaction, see Section 6.4.4.4, “Audit Log Logging Control”.

Before MySQL 5.6.20, the `audit_log_connection_policy` and `audit_log_statement_policy` system variables do not exist. `audit_log_policy` is the only policy control variable and it can be set at server startup or runtime.

- `audit_log_rotate_on_size`

Command-Line Format	`--audit-log-rotate-on-size=#`
System Variable	`audit_log_rotate_on_size`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`0`

Unit	bytes

If audit_log_rotate_on_size is 0, the audit log plugin does not perform automatic size-based log file rotation. Instead, use audit_log_flush to close and reopen the log on demand. In this case, manually rename the file externally to the server before flushing it.

If audit_log_rotate_on_size is greater than 0, automatic size-based log file rotation occurs. Whenever a write to the log file causes its size to exceed the audit_log_rotate_on_size value, the audit log plugin closes the current log file, renames it, and opens a new log file.

If you set audit_log_rotate_on_size to a value that is not a multiple of 4096, it is truncated to the nearest multiple. (Thus, setting it to a value less than 4096 has the effect of setting it to 0 and no rotation occurs, except manually.)

For more information about audit log file rotation, see Space Management and Rotation of Audit Log Files.

- audit_log_statement_policy

Command-Line Format	--audit-log-statement-policy=value
Introduced	5.6.20
System Variable	audit_log_statement_policy
Scope	Global
Dynamic	Yes
Type	Enumeration
Default Value	ALL
Valid Values	ALL ERRORS NONE

The policy controlling how the audit log plugin writes statement events to its log file. The following table shows the permitted values.

Value	Description
ALL	Log all statement events
ERRORS	Log only failed statement events
NONE	Do not log statement events

Note

At server startup, any explicit value given for audit_log_statement_policy may be overridden if audit_log_policy is also specified, as described in Section 6.4.4.4, “Audit Log Logging Control”.

- audit_log_strategy

Command-Line Format	--audit-log-strategy=value
System Variable	audit_log_strategy
Scope	Global
Dynamic	No
Type	Enumeration

Default Value	ASYNCHRONOUS
Valid Values	ASYNCHRONOUS PERFORMANCE SEMISYNCHRONOUS SYNCHRONOUS

The logging method used by the audit log plugin. These strategy values are permitted:

- ASYNCHRONOUS: Log asynchronously. Wait for space in the output buffer.
- PERFORMANCE: Log asynchronously. Drop requests for which there is insufficient space in the output buffer.
- SEMISYNCHRONOUS: Log synchronously. Permit caching by the operating system.
- SYNCHRONOUS: Log synchronously. Call sync() after each request.

Audit Log Plugin Status Variables

If the audit log plugin is enabled, it exposes several status variables that provide operational information.

- Audit_log_current_size

  The size of the current audit log file. The value increases when an event is written to the log and is reset to 0 when the log is rotated.

- Audit_log_event_max_drop_size

  The size of the largest dropped event in performance logging mode. For a description of logging modes, see Section 6.4.4.4, “Audit Log Logging Control”.

- Audit_log_events

  The number of events handled by the audit log plugin, whether or not they were written to the log based on filtering policy (see Section 6.4.4.4, “Audit Log Logging Control”).

- Audit_log_events_filtered

  The number of events handled by the audit log plugin that were filtered (not written to the log) based on filtering policy (see Section 6.4.4.4, “Audit Log Logging Control”).

- Audit_log_events_lost

  The number of events lost in performance logging mode because an event was larger than than the available audit log buffer space. This value may be useful for assessing how to set audit_log_buffer_size to size the buffer for performance mode. For a description of logging modes, see Section 6.4.4.4, “Audit Log Logging Control”.

- Audit_log_events_written

  The number of events written to the audit log.

- Audit_log_total_size

  The total size of events written to all audit log files. Unlike Audit_log_current_size, the value of Audit_log_total_size increases even when the log is rotated.

- Audit_log_write_waits

The number of times an event had to wait for space in the audit log buffer in asynchronous logging mode. For a description of logging modes, see Section 6.4.4.4, “Audit Log Logging Control”.

6.4.4.7 Audit Log Restrictions

MySQL Enterprise Audit is subject to these general restrictions:

- Only SQL statements are logged. Changes made by no-SQL APIs, such as memcached, Node.JS, and the NDB API, are not logged.
- Only top-level statements are logged, not statements within stored programs such as triggers or stored procedures.
- Contents of files referenced by statements such as `LOAD DATA` are not logged.

NDB Cluster. It is possible to use MySQL Enterprise Audit with MySQL NDB Cluster, subject to the following conditions:

- All changes to be logged must be done using the SQL interface. Changes using no-SQL interfaces, such as those provided by the NDB API, memcached, or ClusterJ, are not logged.
- The plugin must be installed on each MySQL server that is used to execute SQL on the cluster.
- Audit plugin data must be aggregated amongst all MySQL servers used with the cluster. This aggregation is the responsibility of the application or user.

6.4.5 MySQL Enterprise Firewall

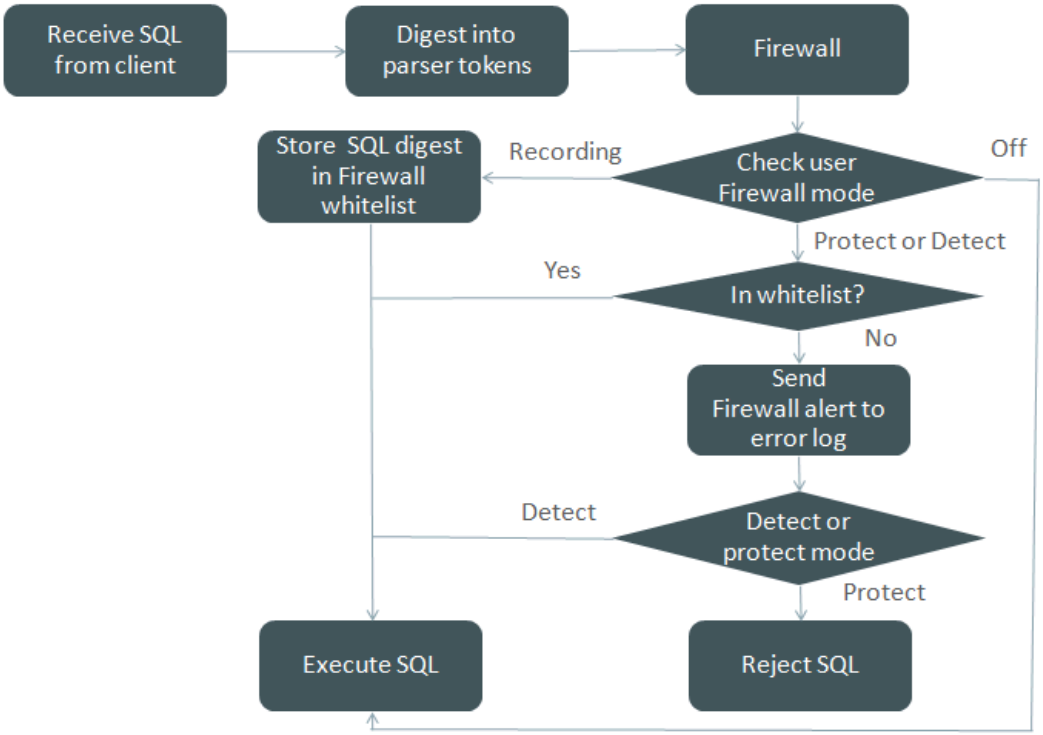
Note

MySQL Enterprise Firewall is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, see https://www.mysql.com/products/.

As of MySQL 5.6.24, MySQL Enterprise Edition includes MySQL Enterprise Firewall, an application-level firewall that enables database administrators to permit or deny SQL statement execution based on matching against lists of accepted statement patterns. This helps harden MySQL Server against attacks such as SQL injection or attempts to exploit applications by using them outside of their legitimate query workload characteristics.

Each MySQL account registered with the firewall has its own statement allowlist, enabling protection to be tailored per account. For a given account, the firewall can operate in recording, protecting, or detecting mode, for training in the accepted statement patterns, active protection against unacceptable statements, or passive detection of unacceptable statements. The diagram illustrates how the firewall processes incoming statements in each mode.

Figure 6.1 MySQL Enterprise Firewall Operation



The following sections describe the elements of MySQL Enterprise Firewall, discuss how to install and use it, and provide reference information for its elements.

6.4.5.1 Elements of MySQL Enterprise Firewall

MySQL Enterprise Firewall is based on a plugin library that includes these elements:

- A server-side plugin named `MYSQL_FIREWALL` examines SQL statements before they execute and, based on the registered firewall profiles, renders a decision whether to execute or reject each statement.
- Server-side plugins named `MYSQL_FIREWALL_USERS` and `MYSQL_FIREWALL_WHITELIST` implement `INFORMATION_SCHEMA` tables that provide views into the registered profiles.
- Profiles are cached in memory for better performance. Tables in the `mysql` system database provide persistent backing storage of firewall data.
- Stored procedures perform tasks such as registering firewall profiles, establishing their operational mode, and managing transfer of firewall data between the in-memory cache and persistent storage.
- User-defined functions provide an API for lower-level tasks such as synchronizing the cache with persistent storage.
- System variables enable firewall configuration and status variables provide runtime operational information.

6.4.5.2 Installing or Uninstalling MySQL Enterprise Firewall

MySQL Enterprise Firewall installation is a one-time operation that installs the elements described in Section 6.4.5.1, “Elements of MySQL Enterprise Firewall”. Installation can be performed using a graphical interface or manually:

- On Windows, MySQL Installer includes an option to enable MySQL Enterprise Firewall for you.

- MySQL Workbench 6.3.4 or higher can install MySQL Enterprise Firewall, enable or disable an installed firewall, or uninstall the firewall.
- Manual MySQL Enterprise Firewall installation involves running a script located in the `share` directory of your MySQL installation.

Note

If installed, MySQL Enterprise Firewall involves some minimal overhead even when disabled. To avoid this overhead, do not install the firewall unless you plan to use it.

Note

MySQL Enterprise Firewall does not work together with the query cache. If the query cache is enabled, disable it before installing the firewall (see Section 8.10.3.3, “Query Cache Configuration”).

For usage instructions, see Section 6.4.5.3, “Using MySQL Enterprise Firewall”. For reference information, see Section 6.4.5.4, “MySQL Enterprise Firewall Reference”.

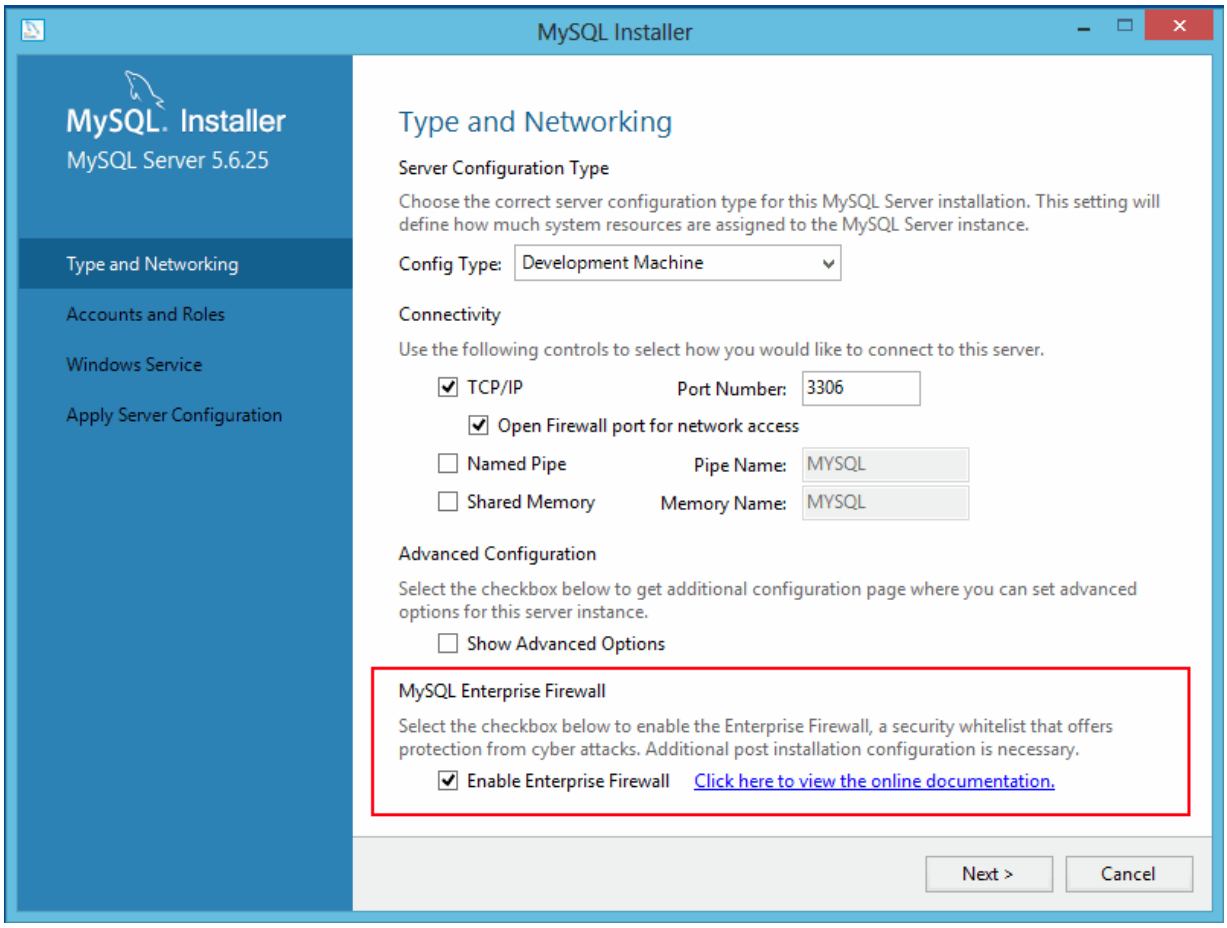
- Installing MySQL Enterprise Firewall
- Uninstalling MySQL Enterprise Firewall

Installing MySQL Enterprise Firewall

If MySQL Enterprise Firewall is already installed from an older version of MySQL, uninstall it using the instructions given later in this section and then restart your server before installing the current version. In this case, it is also necessary to register your configuration again.

On Windows, you can use MySQL Installer to install MySQL Enterprise Firewall, as shown in Figure 6.2, “MySQL Enterprise Firewall Installation on Windows”. Check the **Enable MySQL Enterprise Firewall** check box. (**Open Firewall port for network access** has a different purpose. It refers to Windows Firewall and controls whether Windows blocks the TCP/IP port on which the MySQL server listens for client connections.)

Figure 6.2 MySQL Enterprise Firewall Installation on Windows



To install MySQL Enterprise Firewall using MySQL Workbench 6.3.4 or higher, see MySQL Enterprise Firewall Interface.

To install MySQL Enterprise Firewall manually, look in the `share` directory of your MySQL installation and choose the script that is appropriate for your platform. The available scripts differ in the suffix used to refer to the plugin library file:

- `win_install_firewall.sql`: Choose this script for Windows systems that use `.dll` as the file name suffix.
- `linux_install_firewall.sql`: Choose this script for Linux and similar systems that use `.so` as the file name suffix.

The installation script creates stored procedures in the default database, so choose a database to use. Then run the script as follows, naming the chosen database on the command line. The example here uses the `mysql` system database and the Linux installation script. Make the appropriate substitutions for your system.

```
shell> mysql -u root -p mysql < linux_install_firewall.sql
Enter password: (enter root password here)
```

Installing MySQL Enterprise Firewall either using a graphical interface or manually should enable the firewall. To verify that, connect to the server and execute this statement:

```
mysql> SHOW GLOBAL VARIABLES LIKE 'mysql_firewall_mode';
+---------------------+-------+
| Variable_name       | Value |
+---------------------+-------+
| mysql_firewall_mode | ON    |
```

```
+--------------------+-------+
```

If the plugin fails to initialize, check the server error log for diagnostic messages.

Uninstalling MySQL Enterprise Firewall

MySQL Enterprise Firewall can be uninstalled using MySQL Workbench or manually.

To uninstall MySQL Enterprise Firewall using MySQL Workbench 6.3.4 or higher, see MySQL Enterprise Firewall Interface, in Chapter 25, *MySQL Workbench*.

To uninstall MySQL Enterprise Firewall manually, execute the following statements. Statements use `IF EXISTS` because, depending on the previously installed firewall version, some objects might not exist.

```
DROP TABLE IF EXISTS mysql.firewall_users;
DROP TABLE IF EXISTS mysql.firewall_whitelist;

UNINSTALL PLUGIN MYSQL_FIREWALL;
UNINSTALL PLUGIN MYSQL_FIREWALL_USERS;
UNINSTALL PLUGIN MYSQL_FIREWALL_WHITELIST;

DROP FUNCTION IF EXISTS mysql_firewall_flush_status;
DROP FUNCTION IF EXISTS normalize_statement;
DROP FUNCTION IF EXISTS read_firewall_users;
DROP FUNCTION IF EXISTS read_firewall_whitelist;
DROP FUNCTION IF EXISTS set_firewall_mode;

DROP PROCEDURE IF EXISTS mysql.sp_reload_firewall_rules;
DROP PROCEDURE IF EXISTS mysql.sp_set_firewall_mode;
```

6.4.5.3 Using MySQL Enterprise Firewall

Before using MySQL Enterprise Firewall, install it according to the instructions provided in Section 6.4.5.2, “Installing or Uninstalling MySQL Enterprise Firewall”. Also, MySQL Enterprise Firewall does not work together with the query cache; disable the query cache if it is enabled (see Section 8.10.3.3, “Query Cache Configuration”).

This section describes how to configure MySQL Enterprise Firewall using SQL statements. Alternatively, MySQL Workbench 6.3.4 or higher provides a graphical interface for firewall control. See MySQL Enterprise Firewall Interface.

- Enabling or Disabling the Firewall
- Assigning Firewall Privileges
- Firewall Operational Concepts
- Registering Firewall Account Profiles
- Monitoring the Firewall

Enabling or Disabling the Firewall

To enable or disable the firewall, set the `mysql_firewall_mode` system variable. By default, this variable is enabled when the firewall is installed. To control the initial firewall state explicitly, you can set the variable at server startup. For example, to enable the firewall in an option file, use these lines:

```
[mysqld]
mysql_firewall_mode=ON
```

After modifying `my.cnf`, restart the server to cause the new setting to take effect.

It is also possible to disable or enable the firewall at runtime:

```
SET GLOBAL mysql_firewall_mode = OFF;
SET GLOBAL mysql_firewall_mode = ON;
```

Assigning Firewall Privileges

With the firewall installed, grant the appropriate privileges to the account or accounts that are expected to administer it:

- Grant the `EXECUTE` privilege for the firewall stored procedures in the `mysql` system database. These may invoke UDFs, so stored procedure access also requires the privileges needed for those UDFs.
- Grant the `SUPER` privilege so that the firewall user-defined procedures can be executed.

Firewall Operational Concepts

The MySQL server permits clients to connect and receives from them SQL statements to be executed. The server passes to the firewall each incoming statement that does not immediately fail with a syntax error. Based on whether the firewall accepts the statement, the server executes it or returns an error to the client.

Firewall operation is based on a registry of profiles that enable statement execution protection to be applied. A profile has these attributes:

- Rules that define which statements are acceptable to the profile. This set of rules forms the profile allowlist.
- The current operational mode. The mode enables the profile to be used in different ways. For example: the profile can be placed in training mode to establish the allowlist; the allowlist can be used for restricting statement execution or intrusion detection; the profile can be disabled entirely.
- The scope of applicability, indicating which client connections the profile applies to.

  The firewall supports account-based profiles such that each profile matches a particular client account (client user name and host name combination). For example, you can register one account profile for which the allowlist applies to connections originating from `admin@localhost` and another account profile for which the allowlist applies to connections originating from `myapp@apphost.example.com`.

For each client connection, the firewall determines which profile applies, and accepts only statements the profile allowlist permits. (If the client matches no profile, the firewall ignores it and accepts all statements.)

By default, the firewall accepts all statements and has no effect on which statements MySQL accounts can execute. To apply firewall protective capabilities, you must take explicit action:

- Register one or more profiles with the firewall. (This is required because the firewall ignores clients that match no profile.)
- Train the firewall to establish the allowlist for each profile; that is, the types of statements the profile permits clients to execute.
- Tell the firewall to protect MySQL using each profile for which it has been trained; that is, to match incoming statements against the appropriate allowlist when clients connect.

Statement matching performed by the firewall does not use SQL statements as received from clients. Instead, the server converts incoming statements to normalized digest form and firewall operation uses these digests. The benefit of statement normalization is that it enables similar statements to be grouped and recognized using a single pattern. For example, these statements are distinct from each other:

```
SELECT first_name, last_name FROM customer WHERE customer_id = 1;
```

```
select first_name, last_name from customer where customer_id = 99;
SELECT first_name, last_name FROM customer WHERE customer_id = 143;
```

But all of them have the same normalized digest form:

```
SELECT `first_name` , `last_name` FROM `customer` WHERE `customer_id` = ?
```

By using normalization, the firewall can store in allowlists digests that each match many different statements received from clients. For more information about normalization and digests, see Section 22.10, “Performance Schema Statement Digests”.

Note

Before MySQL 5.6.25, MySQL Enterprise Firewall records prepared statements as they are received by the server, not as normalized digests. Thus, spaces, tabs, and lettercase are significant for comparison of allowlist rules against incoming statements.

Each profile registered with the firewall has its own operational mode, chosen from these values:

- `OFF`: This mode disables the profile. The firewall considers it inactive and ignores it.
- `RECORDING`: This is the firewall training mode. Incoming statements received from a client that matches the profile are considered acceptable for the profile and become part of its “fingerprint.” The firewall records the normalized digest form of each statement to learn the acceptable statement patterns for the profile. Each pattern is a rule, and the union of the rules is the profile allowlist.
- `PROTECTING`: In this mode, the profile allows or prevents statement execution. The firewall matches incoming statements against the profile allowlist, accepting only only statements that match and rejecting those that do not. After training the profile in `RECORDING` mode, switch it to `PROTECTING` mode to harden MySQL against access by statements that deviate from the allowlist.
- `DETECTING`: This mode detects but not does not block intrusions (statements that are suspicious because they match nothing in the profile allowlist). In `DETECTING` mode, the firewall writes suspicious statements to the error log but accepts them without denying access. This mode is available in MySQL 5.6.26 and higher.

When a profile is assigned any of the preceding mode values, the firewall stores the mode in the profile. Firewall mode-setting operations also permit a mode value of `RESET`, but this value is not stored: setting a profile to `RESET` mode causes the firewall to delete all rules for the profile and set its mode to `OFF`.

Registering Firewall Account Profiles

MySQL Enterprise Firewall enables profiles to be registered that correspond to individual accounts.

MySQL authenticates each client session for a specific user name and host name combination. This combination is the *session account*. The firewall matches the session account against registered account profiles to determine which profile applies to handling incoming statements from the session:

- The firewall ignores inactive profiles (profiles with a mode of `OFF`).
- The session account matches the active account profile having the same user and host, if there is one. There is at most one such account profile.

In other words, at most one active account profile is applicable to a given session, for which the firewall handles each incoming statement as follows:

- If there is no applicable profile, there are no restrictions. The firewall accepts the statement.
- If there is an applicable profile, its mode determines statement handling:

- In `RECORDING` mode, the firewall adds the statement to the profile allowlist rules and accepts it.
- In `PROTECTING` mode, the firewall compares the statement to the rules in the profile allowlist. The firewall accepts the statement if there is a match, and rejects it otherwise. As of MySQL 5.6.25, if the `mysql_firewall_trace` system variable is enabled, the firewall also writes rejected statements to the error log.
- In `DETECTING` mode, the firewall detects instrusions without denying access. The firewall accepts the statement, but also matches it to the profile allowlist, as in `PROTECTING` mode. If the statement is suspicious (nonmatching), the firewall writes it to the error log.

To use a firewall account profile to protect MySQL against incoming statements from a given account, follow these steps:

1. Register the account profile and put it in `RECORDING` mode.
2. Connect to the MySQL server using the account and execute statements to be learned. This trains the corresponding account profile and establishes the rules that form the profile allowlist.
3. Switch the account profile to `PROTECTING` mode. When a client connects to the server using the account, the account profile allowlist restricts statement execution.
4. Should additional training be necessary, switch the account profile to `RECORDING` mode again, update its allowlist with new statement patterns, then switch it back to `PROTECTING` mode.

By maintaining profiles, the firewall enables implementation of protection strategies such as these:

- If an application has unique protection requirements, configure it to use an account not used for any other purpose and set up a corresponding account profile.
- If related applications share protection requirements, configure them all to use the same account (and thus the same account profile).

Observe these guidelines for firewall-related account references:

- Take note of the context in which account references occur. To name an account for firewall operations, specify it as a single quoted string (`'user_name@host_name'`). This differs from the usual MySQL convention for statements such as `CREATE USER` and `GRANT`, for which you quote the user and host parts of an account name separately (`'user_name'@'host_name'`).

  The requirement for naming accounts as a single quoted string for firewall operations means that you cannot use accounts that have embedded `@` characters in the user name.

- The firewall assesses statements against accounts represented by actual user and host names as authenticated by the server. When registering account profiles, do not use wildcard characters or netmasks:
  - Suppose that an account named `me@%.example.org` exists and a client uses it to connect to the server from the host `abc.example.org`.
  - The account name contains a `%` wildcard character, but the server authenticates the client as having a user name of `me` and host name of `abc.example.com`, and that is what the firewall sees.
  - Consequently, the account name to use for firewall operations is `me@abc.example.org` rather than `me@%.example.org`.

The following example shows how to register an account profile with the firewall, teach the firewall the acceptable statements for that profile, and use the profile to protect MySQL against execution of unacceptable statements by the account. The example account, `fwuser@localhost`, is presumed for

use by an application that accesses tables in the `sakila` database (available at https://dev.mysql.com/doc/index-other.html).

Use an administrative MySQL account to perform the steps in this procedure, except those steps designated for execution by the `fwuser@localhost` account corresponding to the account profile registered with the firewall. For statements executed using this account, the default database should be `sakila`. (You can use a different database by adjusting the instructions accordingly.)

1. If necessary, create the account to use for executing statements (choose an appropriate password) and grant it privileges for the `sakila` database:

   ```
   CREATE USER 'fwuser'@'localhost' IDENTIFIED BY 'password';
   GRANT ALL ON sakila.* TO 'fwuser'@'localhost';
   ```

2. Use the `sp_set_firewall_mode()` stored procedure to register an account profile with the firewall and place the profile in `RECORDING` (training) mode:

   ```
   CALL mysql.sp_set_firewall_mode('fwuser@localhost', 'RECORDING');
   ```

   As it executes, the stored procedure invokes firewall user-defined functions, which may produce output of their own.

3. To train the registered account profile, connect to the server as `fwuser` from the server host so that the firewall sees a session account of `fwuser@localhost`. Then use the account to execute some statements to be considered legitimate for the profile. For example:

   ```
   SELECT first_name, last_name FROM customer WHERE customer_id = 1;
   UPDATE rental SET return_date = NOW() WHERE rental_id = 1;
   SELECT get_customer_balance(1, NOW());
   ```

   Because the profile is in `RECORDING` mode, the firewall records the normalized digest form of the statements as rules in the profile allowlist.

   Note

   Until the `fwuser@localhost` account profile receives statements in `RECORDING` mode, its allowlist is empty, which is equivalent to “deny all.” No statement can match an empty allowlist, which has these implications:

   - The account profile cannot be switched to `PROTECTING` mode. It would reject every statement, effectively prohibiting the account from executing any statement.
   - The account profile can be switched to `DETECTING` mode. In this case, the profile accepts every statement but logs it as suspicious.

4. At this point, the account profile information is cached. To see this information, query the firewall `INFORMATION_SCHEMA` tables:

   ```
   mysql> SELECT MODE FROM INFORMATION_SCHEMA.MYSQL_FIREWALL_USERS
          WHERE USERHOST = 'fwuser@localhost';
   +-----------+
   | MODE      |
   +-----------+
   | RECORDING |
   +-----------+
   mysql> SELECT RULE FROM INFORMATION_SCHEMA.MYSQL_FIREWALL_WHITELIST
          WHERE USERHOST = 'fwuser@localhost';
   +-----------------------------------------------------------------------------+
   | RULE                                                                        |
   +-----------------------------------------------------------------------------+
   | SELECT `first_name` , `last_name` FROM `customer` WHERE `customer_id` = ?   |
   | SELECT `get_customer_balance` ( ? , NOW ( ) )                               |
   | UPDATE `rental` SET `return_date` = NOW ( ) WHERE `rental_id` = ?           |
   | SELECT @@`version_comment` LIMIT ?                                          |
   +-----------------------------------------------------------------------------+
   ```

Note

The `@@version_comment` rule comes from a statement sent automatically by the `mysql` client when you connect to the server using the account corresponding to the account profile.

Important

Train the firewall under conditions matching application use. For example, to determine server characteristics and capabilities, a given MySQL connector might send statements to the server at the beginning of each session. If an application normally is used through that connector, train the firewall using the connector, too. That enables those initial statements to become part of the allowlist for the account profile associated with the application.

5. Invoke `sp_set_firewall_mode()` again, this time switching the account profile to `PROTECTING` mode:

```
CALL mysql.sp_set_firewall_mode('fwuser@localhost', 'PROTECTING');
```

Important

Switching the account profile out of `RECORDING` mode synchronizes its cached data to the `mysql` system database tables that provide persistent underlying storage. If you do not switch the mode for a profile that is being recorded, the cached data is not written to persistent storage and is lost when the server is restarted.

6. Test the account profile by using the account to execute some acceptable and unacceptable statements. The firewall matches each statement against the profile allowlist and accepts or rejects it:

   - This statement is not identical to a training statement but produces the same normalized statement as one of them, so the firewall accepts it:

```
mysql> SELECT first_name, last_name FROM customer WHERE customer_id = '48';
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| ANN        | EVANS     |
+------------+-----------+
```

   - These statements match nothing in the allowlist, so the firewall rejects each with an error:

```
mysql> SELECT first_name, last_name FROM customer WHERE customer_id = 1 OR TRUE;
ERROR 1045 (28000): Statement was blocked by Firewall
mysql> SHOW TABLES LIKE 'customer%';
ERROR 1045 (28000): Statement was blocked by Firewall
mysql> TRUNCATE TABLE mysql.slow_log;
ERROR 1045 (28000): Statement was blocked by Firewall
```

   - As of MySQL 5.6.25, if the `mysql_firewall_trace` system variable is enabled, the firewall also writes rejected statements to the error log. For example:

```
[Note] Plugin MYSQL_FIREWALL reported:
'ACCESS DENIED for fwuser@localhost. Reason: No match in whitelist.
Statement: TRUNCATE TABLE `mysql` . `slow_log` '
```

     These log messages may be helpful in identifying the source of attacks, should that be necessary.

The firewall account profile now is trained for the `fwuser@localhost` account. When clients connect using that account and attempt to execute statements, the profile protects MySQL against statements not matched by the profile allowlist.

As of MySQL 5.6.26, it is also possible to detect intrusions by logging nonmatching statements as suspicious without denying access. First, put the account profile in `DETECTING` mode:

```
CALL mysql.sp_set_firewall_mode('fwuser@localhost', 'DETECTING');
```

Then, using the account, execute a statement that does not match the account profile allowlist. In `DETECTING` mode, the firewall permits the nonmatching statement to execute:

```
mysql> SHOW TABLES LIKE 'customer%';
+------------------------------+
| Tables_in_sakila (customer%) |
+------------------------------+
| customer                     |
| customer_list                |
+------------------------------+
```

In addition, the firewall writes a message to the error log:

```
[Note] Plugin MYSQL_FIREWALL reported:
'SUSPICIOUS STATEMENT from 'fwuser@localhost'. Reason: No match in whitelist.
Statement: SHOW TABLES LIKE ? '
```

To disable an account profile, change its mode to `OFF`:

```
CALL mysql.sp_set_firewall_mode(user, 'OFF');
```

To forget all training for a profile and disable it, reset it:

```
CALL mysql.sp_set_firewall_mode(user, 'RESET');
```

The reset operation causes the firewall to delete all rules for the profile and set its mode to `OFF`.

Monitoring the Firewall

To assess firewall activity, examine its status variables. For example, after performing the procedure shown earlier to train and protect the `fwuser@localhost` account, the variables look like this:

```
mysql> SHOW GLOBAL STATUS LIKE 'Firewall%';
+----------------------------+-------+
| Variable_name              | Value |
+----------------------------+-------+
| Firewall_access_denied     | 3     |
| Firewall_access_granted    | 4     |
| Firewall_access_suspicious | 1     |
| Firewall_cached_entries    | 4     |
+----------------------------+-------+
```

The variables indicate the number of statements rejected, accepted, logged as suspicious, and added to the cache, respectively. The `Firewall_access_granted` count is 4 because of the `@@version_comment` statement sent by the `mysql` client each of the three times you connected using the registered account, plus the `SHOW TABLES` statement that was not blocked in `DETECTING` mode.

6.4.5.4 MySQL Enterprise Firewall Reference

The following sections provide a reference to MySQL Enterprise Firewall elements:

- MySQL Enterprise Firewall Tables
- MySQL Enterprise Firewall Stored Procedures
- MySQL Enterprise Firewall User-Defined Functions
- MySQL Enterprise Firewall System Variables

- MySQL Enterprise Firewall Status Variables

MySQL Enterprise Firewall Tables

MySQL Enterprise Firewall maintains profile information using tables in the `mysql` system database for persistent storage and `INFORMATION_SCHEMA` tables to provide views into in-memory cached data. When enabled, the firewall bases operational decisions on the cached data.

Each `mysql` system database table is accessible only by accounts that have the `SELECT` privilege for it. The `INFORMATION_SCHEMA` tables are accessible by anyone.

The `mysql.firewall_users` table lists names and operational modes of registered firewall account profiles. The table has the following columns (with the corresponding `INFORMATION_SCHEMA.MYSQL_FIREWALL_USERS` table having similar but not necessarily identical columns):

- `USERHOST`

  The account profile name. Each account name has the format *`user_name@host_name`*.

- `MODE`

  The current operational mode for the profile. Permitted mode values are `OFF`, `DETECTING` (as of MySQL 5.6.26), `PROTECTING`, `RECORDING`, and `RESET`. For details about their meanings, see Firewall Operational Concepts.

The `mysql.firewall_whitelist` table lists allowlist rules of registered firewall account profiles. The table has the following columns (with the corresponding `INFORMATION_SCHEMA.MYSQL_FIREWALL_WHITELIST` table having similar but not necessarily identical columns):

- `USERHOST`

  The account profile name. Each account name has the format *`user_name@host_name`*.

- `RULE`

  A normalized statement indicating an acceptable statement pattern for the profile. A profile allowlist is the union of its rules.

MySQL Enterprise Firewall Stored Procedures

MySQL Enterprise Firewall stored procedures perform tasks such as registering profiles with the firewall, establishing their operational mode, and managing transfer of firewall data between the cache and persistent storage. These procedures invoke user-defined functions (UDFs) that provide an API for lower-level tasks.

Firewall stored procedures are created in the `mysql` system database. To invoke a firewall stored procedure, either do so while `mysql` is the default database, or qualify the procedure name with the database name. For example:

```
CALL mysql.sp_set_firewall_mode(user, mode);
```

The following list describes each firewall stored procedure:

- `sp_reload_firewall_rules(`*`user`*`)`

  This stored procedure provides control over firewall operation for individual account profiles. The procedure uses firewall UDFs to reload the in-memory rules for an account profile from the rules stored in the `mysql.firewall_whitelist` table.

  Arguments:

- *user*: The name of the affected account profile, as a string in *user_name@host_name* format.

Example:

```
CALL mysql.sp_reload_firewall_rules('fwuser@localhost');
```

Warning

This procedure clears the account profile in-memory allowlist rules before reloading them from persistent storage, and sets the profile mode to `OFF`. If the profile mode was not `OFF` prior to the `sp_reload_firewall_rules()` call, use `sp_set_firewall_mode()` to restore its previous mode after reloading the rules. For example, if the profile was in `PROTECTING` mode, that is no longer true after calling `sp_reload_firewall_rules()` and you must set it to `PROTECTING` again explicitly.

- `sp_set_firewall_mode(`*user, mode*`)`

  This stored procedure establishes the operational mode for a firewall account profile, after registering the profile with the firewall if it was not already registered. The procedure also invokes firewall UDFs as necessary to transfer firewall data between the cache and persistent storage. This procedure may be called even if the `mysql_firewall_mode` system variable is `OFF`, although setting the mode for a profile has no operational effect until the firewall is enabled.

  Arguments:

  - *user*: The name of the affected account profile, as a string in *user_name@host_name* format.
  - *mode*: The operational mode for the profile, as a string. Permitted mode values are `OFF`, `DETECTING`, `PROTECTING`, `RECORDING`, and `RESET`. For details about their meanings, see Firewall Operational Concepts.

  Switching an account profile to any mode but `RECORDING` synchronizes its firewall cache data to the `mysql` system database tables that provide persistent underlying storage. Switching the mode from `OFF` to `RECORDING` reloads the allowlist from the `mysql.firewall_whitelist` table into the cache.

  If an account profile has an empty allowlist, its mode cannot be set to `PROTECTING` because the profile would reject every statement, effectively prohibiting the account from executing statements. In response to such a mode-setting attempt, the firewall produces a diagnostic message that is returned as a result set rather than as an SQL error:

```
mysql> CALL mysql.sp_set_firewall_mode('a@b','PROTECTING');
+-----------------------------------------------------------------------+
| set_firewall_mode(arg_userhost, arg_mode)                             |
+-----------------------------------------------------------------------+
| ERROR: PROTECTING mode requested for a@b but the whitelist is empty. |
+-----------------------------------------------------------------------+
1 row in set (0.02 sec)

Query OK, 0 rows affected (0.02 sec)
```

MySQL Enterprise Firewall User-Defined Functions

MySQL Enterprise Firewall user-defined functions (UDFs) provide an API for lower-level tasks such as synchronizing the firewall cache with the underlying system tables.

Under normal operation, these UDFs are invoked by the firewall stored procedures, not directly by users. For that reason, these UDF descriptions do not include the usual details, such as information about their arguments and return types.

- Firewall Account Profile User-Defined Functions

- Firewall Miscellaneous User-Defined Functions

Firewall Account Profile User-Defined Functions

These UDFs perform management operations on firewall account profiles:

- `read_firewall_users(user, mode)`

  This aggregate UDF updates the firewall account profile cache through a `SELECT` statement on the `mysql.firewall_users` table. It requires the `SUPER` privilege.

  Example:

```
SELECT read_firewall_users('fwuser@localhost', 'RECORDING')
FROM mysql.firewall_users;
```

- `read_firewall_whitelist(user, rule)`

  This aggregate UDF updates the recorded-statement cache for the named account profile through a `SELECT` statement on the `mysql.firewall_whitelist` table. It requires the `SUPER` privilege.

  Example:

```
SELECT read_firewall_whitelist('fwuser@localhost', fw.rule)
FROM mysql.firewall_whitelist AS fw
WHERE USERHOST = 'fwuser@localhost';
```

- `set_firewall_mode(user, mode)`

  This UDF manages the account profile cache and establishes the profile operational mode. It requires the `SUPER` privilege.

  Example:

```
SELECT set_firewall_mode('fwuser@localhost', 'RECORDING');
```

Firewall Miscellaneous User-Defined Functions

These UDFs perform miscellaneous firewall operations:

- `mysql_firewall_flush_status()`

  This UDF resets several firewall status variables to 0:

  - `Firewall_access_denied`
  - `Firewall_access_granted`
  - `Firewall_access_suspicious`

  This UDF requires the `SUPER` privilege.

  Example:

```
SELECT mysql_firewall_flush_status();
```

- `normalize_statement(stmt)`

  This UDF normalizes an SQL statement into the digest form used for allowlist rules. It requires the `SUPER` privilege.

  Example:

```
SELECT normalize_statement('SELECT * FROM t1 WHERE c1 > 2');
```

MySQL Enterprise Firewall System Variables

MySQL Enterprise Firewall supports the following system variables. Use them to configure firewall operation. These variables are unavailable unless the firewall is installed (see Section 6.4.5.2, "Installing or Uninstalling MySQL Enterprise Firewall").

- `mysql_firewall_max_query_size`

Command-Line Format	`--mysql-firewall-max-query-size=#`
Introduced	5.6.24
Removed	5.6.26
System Variable	`mysql_firewall_max_query_size`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`4096`
Minimum Value	`0`
Maximum Value	`4294967295`

The maximum size of a normalized statement that can be inserted in the MySQL Enterprise Firewall cache. Normalized statements longer than this size are truncated. Truncated statements are discarded if the firewall mode for the current user is `RECORDING` and rejected if the mode is `PROTECTING`.

`mysql_firewall_max_query_size` was removed in MySQL 5.6.26. `max_digest_length` should be set large enough to avoid statement truncation.

- `mysql_firewall_mode`

Command-Line Format	`--mysql-firewall-mode[={OFF|ON}]`
Introduced	5.6.24
System Variable	`mysql_firewall_mode`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`ON`

Whether MySQL Enterprise Firewall is enabled (the default) or disabled.

- `mysql_firewall_trace`

Command-Line Format	`--mysql-firewall-trace[={OFF|ON}]`
Introduced	5.6.24
System Variable	`mysql_firewall_trace`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

Whether the MySQL Enterprise Firewall trace is enabled or disabled (the default). When enabled, `mysql_firewall_trace` has this effect:

- In MySQL 5.6.24, the firewall writes a file named `firewall_trace.txt` in the data directory.
- In MySQL 5.6.25 and higher, for `PROTECTING` mode, the firewall writes rejected statements to the error log.

MySQL Enterprise Firewall Status Variables

MySQL Enterprise Firewall supports the following status variables. Use them to obtain information about firewall operational status. These variables are unavailable unless the firewall is installed (see Section 6.4.5.2, "Installing or Uninstalling MySQL Enterprise Firewall"). Firewall status variables are set to 0 whenever the `MYSQL_FIREWALL` plugin is installed or the server is started. Many of them are reset to zero by the `mysql_firewall_flush_status()` UDF (see MySQL Enterprise Firewall User-Defined Functions).

- `Firewall_access_denied`

  The number of statements rejected by MySQL Enterprise Firewall.

- `Firewall_access_granted`

  The number of statements accepted by MySQL Enterprise Firewall.

- `Firewall_access_suspicious`

  The number of statements logged by MySQL Enterprise Firewall as suspicious for users who are in `DETECTING` mode.

- `Firewall_cached_entries`

  The number of statements recorded by MySQL Enterprise Firewall, including duplicates.

6.5 MySQL Enterprise Encryption

Note

MySQL Enterprise Encryption is an extension included in MySQL Enterprise Edition, a commercial product. To learn more about commercial products, https://www.mysql.com/products/.

As of MySQL 5.6.21, MySQL Enterprise Edition includes a set of encryption functions based on the OpenSSL library that expose OpenSSL capabilities at the SQL level. These functions enable Enterprise applications to perform the following operations:

- Implement added data protection using public-key asymmetric cryptography
- Create public and private keys and digital signatures
- Perform asymmetric encryption and decryption
- Use cryptographic hashing for digital signing and data verification and validation

MySQL Enterprise Encryption supports the RSA, DSA, and DH cryptographic algorithms.

MySQL Enterprise Encryption is supplied as a user-defined function (UDF) library, from which individual functions can be installed individually.

6.5.1 MySQL Enterprise Encryption Installation

MySQL Enterprise Encryption functions are located in a user-defined function (UDF) library file installed in the plugin directory (the directory named by the `plugin_dir` system variable). The UDF library base name is `openssl_udf` and the suffix is platform dependent. For example, the file name on Linux or Windows is `openssl_udf.so` or `openssl_udf.dll`, respectively.

To install functions from the library file, use the `CREATE FUNCTION` statement. To load all functions from the library, use this set of statements, adjusting the file name suffix as necessary:

```
CREATE FUNCTION asymmetric_decrypt RETURNS STRING
  SONAME 'openssl_udf.so';
CREATE FUNCTION asymmetric_derive RETURNS STRING
  SONAME 'openssl_udf.so';
CREATE FUNCTION asymmetric_encrypt RETURNS STRING
  SONAME 'openssl_udf.so';
CREATE FUNCTION asymmetric_sign RETURNS STRING
  SONAME 'openssl_udf.so';
CREATE FUNCTION asymmetric_verify RETURNS INTEGER
  SONAME 'openssl_udf.so';
CREATE FUNCTION create_asymmetric_priv_key RETURNS STRING
  SONAME 'openssl_udf.so';
CREATE FUNCTION create_asymmetric_pub_key RETURNS STRING
  SONAME 'openssl_udf.so';
CREATE FUNCTION create_dh_parameters RETURNS STRING
  SONAME 'openssl_udf.so';
CREATE FUNCTION create_digest RETURNS STRING
  SONAME 'openssl_udf.so';
```

Once installed, UDFs remain installed across server restarts. To unload the UDFs, use the `DROP FUNCTION` statement:

```
DROP FUNCTION asymmetric_decrypt;
DROP FUNCTION asymmetric_derive;
DROP FUNCTION asymmetric_encrypt;
DROP FUNCTION asymmetric_sign;
DROP FUNCTION asymmetric_verify;
DROP FUNCTION create_asymmetric_priv_key;
DROP FUNCTION create_asymmetric_pub_key;
DROP FUNCTION create_dh_parameters;
DROP FUNCTION create_digest;
```

In the `CREATE FUNCTION` and `DROP FUNCTION` statements, the function names must be specified in lowercase. This differs from their use at function invocation time, for which you can use any lettercase.

The `CREATE FUNCTION` and `DROP FUNCTION` statements require the `INSERT` and `DROP` privilege, respectively, for the `mysql` database.

6.5.2 MySQL Enterprise Encryption Usage and Examples

To use MySQL Enterprise Encryption in applications, invoke the functions that are appropriate for the operations you wish to perform. This section demonstrates how to carry out some representative tasks:

- Create a private/public key pair using RSA encryption
- Use the private key to encrypt data and the public key to decrypt it
- Generate a digest from a string
- Use the digest with a key pair
- Create a symmetric key
- Limit CPU usage by key-generation operations

Create a private/public key pair using RSA encryption

```
-- Encryption algorithm; can be 'DSA' or 'DH' instead
SET @algo = 'RSA';
-- Key length in bits; make larger for stronger keys
SET @key_len = 1024;

-- Create private key
SET @priv = create_asymmetric_priv_key(@algo, @key_len);
-- Derive corresponding public key from private key, using same algorithm
```

```
SET @pub = create_asymmetric_pub_key(@algo, @priv);
```

Now you can use the key pair to encrypt and decrypt data, sign and verify data, or generate symmetric keys.

Use the private key to encrypt data and the public key to decrypt it

This requires that the members of the key pair be RSA keys.

```
SET @ciphertext = asymmetric_encrypt(@algo, 'My secret text', @priv);
SET @plaintext = asymmetric_decrypt(@algo, @ciphertext, @pub);
```

Conversely, you can encrypt using the public key and decrypt using the private key.

```
SET @ciphertext = asymmetric_encrypt(@algo, 'My secret text', @pub);
SET @plaintext = asymmetric_decrypt(@algo, @ciphertext, @priv);
```

In either case, the algorithm specified for the encryption and decryption functions must match that used to generate the keys.

Generate a digest from a string

```
-- Digest type; can be 'SHA256', 'SHA384', or 'SHA512' instead
SET @dig_type = 'SHA224';

-- Generate digest string
SET @dig = create_digest(@dig_type, 'My text to digest');
```

Use the digest with a key pair

The key pair can be used to sign data, then verify that the signature matches the digest.

```
-- Encryption algorithm; could be 'DSA' instead; keys must
-- have been created using same algorithm
SET @algo = 'RSA';

-- Generate signature for digest and verify signature against digest
SET @sig = asymmetric_sign(@algo, @dig, @priv, @dig_type);
-- Verify signature against digest
SET @verf = asymmetric_verify(@algo, @dig, @sig, @pub, @dig_type);
```

Create a symmetric key

This requires DH private/public keys as inputs, created using a shared symmetric secret. Create the secret by passing the key length to `create_dh_parameters()`, then pass the secret as the “key length” to `create_asymmetric_priv_key()`.

```
-- Generate DH shared symmetric secret
SET @dhp = create_dh_parameters(1024);
-- Generate DH key pairs
SET @algo = 'DH';
SET @priv1 = create_asymmetric_priv_key(@algo, @dhp);
SET @pub1 = create_asymmetric_pub_key(@algo, @priv1);
SET @priv2 = create_asymmetric_priv_key(@algo, @dhp);
SET @pub2 = create_asymmetric_pub_key(@algo, @priv2);

-- Generate symmetric key using public key of first party,
-- private key of second party
SET @sym1 = asymmetric_derive(@pub1, @priv2);

-- Or use public key of second party, private key of first party
SET @sym2 = asymmetric_derive(@pub2, @priv1);
```

Key string values can be created at runtime and stored into a variable or table using `SET`, `SELECT`, or `INSERT`:

```
SET @priv1 = create_asymmetric_priv_key('RSA', 1024);
SELECT create_asymmetric_priv_key('RSA', 1024) INTO @priv2;
INSERT INTO t (key_col) VALUES(create_asymmetric_priv_key('RSA', 1024));
```

Key string values stored in files can be read using the `LOAD_FILE()` function by users who have the `FILE` privilege.

Digest and signature strings can be handled similarly.

Limit CPU usage by key-generation operations

The `create_asymmetric_priv_key()` and `create_dh_parameters()` encryption functions take a key-length parameter, and the amount of CPU resources required by these functions increases as the key length increases. For some installations, this might result in unacceptable CPU usage if applications frequently generate excessively long keys.

OpenSSL imposes a minimum key length of 1,024 bits for all keys. OpenSSL also imposes a maximum key length of 10,000 bits and 16,384 bits for DSA and RSA keys, respectively, for `create_asymmetric_priv_key()`, and a maximum key length of 10,000 bits for `create_dh_parameters()`. If those maximum values are too high, three environment variables are available as of MySQL 5.6.35 to enable MySQL server administrators to set lower maximum lengths for key generation, and thereby to limit CPU usage:

- `MYSQL_OPENSSL_UDF_DSA_BITS_THRESHOLD`: Maximum DSA key length in bits for `create_asymmetric_priv_key()`. The minimum and maximum values for this variable are 1,024 and 10,000.
- `MYSQL_OPENSSL_UDF_RSA_BITS_THRESHOLD`: Maximum RSA key length in bits for `create_asymmetric_priv_key()`. The minimum and maximum values for this variable are 1,024 and 16,384.
- `MYSQL_OPENSSL_UDF_DH_BITS_THRESHOLD`: Maximum key length in bits for `create_dh_parameters()`. The minimum and maximum values for this variable are 1,024 and 10,000.

To use any of these environment variables, set them in the environment of the process that starts the server. If set, their values take precedence over the maximum key lengths imposed by OpenSSL. For example, to set a maximum key length of 4,096 bits for DSA and RSA keys for `create_asymmetric_priv_key()`, set these variables:

```
export MYSQL_OPENSSL_UDF_DSA_BITS_THRESHOLD=4096
export MYSQL_OPENSSL_UDF_RSA_BITS_THRESHOLD=4096
```

The example uses Bourne shell syntax. The syntax for other shells may differ.

6.5.3 MySQL Enterprise Encryption User-Defined Function Reference

MySQL Enterprise Encryption User-Defined Functions A reference that lists MySQL Enterprise Encryption user-defined functions.

6.5.4 MySQL Enterprise Encryption User-Defined Function Descriptions

MySQL Enterprise Encryption functions have these general characteristics:

- For arguments of the wrong type or an incorrect number of arguments, each function returns an error.
- If the arguments are not suitable to permit a function to perform the requested operation, it returns `NULL` or 0 as appropriate. This occurs, for example, if a function does not support a specified algorithm, a key length is too short or long, or a string expected to be a key string in PEM format is not a valid key. (OpenSSL imposes its own key-length limits, and server administrators can impose additional limits on maximum key length by setting environment variables. See Section 6.5.2, “MySQL Enterprise Encryption Usage and Examples”.)
- The underlying SSL library takes care of randomness initialization.

Several of the functions take an encryption algorithm argument. The following table summarizes the supported algorithms by function.

Table 6.17 Supported Algorithms by Function

Function	Supported Algorithms
`asymmetric_decrypt()`	RSA
`asymmetric_derive()`	DH
`asymmetric_encrypt()`	RSA
`asymmetric_sign()`	RSA, DSA
`asymmetric_verify()`	RSA, DSA
`create_asymmetric_priv_key()`	RSA, DSA, DH
`create_asymmetric_pub_key()`	RSA, DSA, DH
`create_dh_parameters()`	DH

Note

Although you can create keys using any of the RSA, DSA, or DH encryption algorithms, other functions that take key arguments might accept only certain types of keys. For example, `asymmetric_encrypt()` and `asymmetric_decrypt()` accept only RSA keys.

The following descriptions describe the calling sequences for MySQL Enterprise Encryption functions. For additional examples and discussion, see Section 6.5.2, “MySQL Enterprise Encryption Usage and Examples”.

- `asymmetric_decrypt(`*`algorithm`*`, `*`crypt_str`*`, `*`key_str`*`)`

  Decrypts an encrypted string using the given algorithm and key string, and returns the resulting plaintext as a binary string. If decryption fails, the result is `NULL`.

  `key_str` must be a valid key string in PEM format. For successful decryption, it must be the public or private key string corresponding to the private or public key string used with `asymmetric_encrypt()` to produce the encrypted string. *`algorithm`* indicates the encryption algorithm used to create the key.

  Supported *`algorithm`* values: `'RSA'`

  For a usage example, see the description of `asymmetric_encrypt()`.

- `asymmetric_derive(`*`pub_key_str`*`, `*`priv_key_str`*`)`

  Derives a symmetric key using the private key of one party and the public key of another, and returns the resulting key as a binary string. If key derivation fails, the result is `NULL`.

  `pub_key_str` and *`priv_key_str`* must be valid key strings in PEM format. They must be created using the DH algorithm.

  Suppose that you have two pairs of public and private keys:

```
SET @dhp = create_dh_parameters(1024);
SET @priv1 = create_asymmetric_priv_key('DH', @dhp);
SET @pub1 = create_asymmetric_pub_key('DH', @priv1);
SET @priv2 = create_asymmetric_priv_key('DH', @dhp);
SET @pub2 = create_asymmetric_pub_key('DH', @priv2);
```

  Suppose further that you use the private key from one pair and the public key from the other pair to create a symmetric key string. Then this symmetric key identity relationship holds:

```
asymmetric_derive(@pub1, @priv2) = asymmetric_derive(@pub2, @priv1)
```

- `asymmetric_encrypt(algorithm, str, key_str)`

  Encrypts a string using the given algorithm and key string, and returns the resulting ciphertext as a binary string. If encryption fails, the result is `NULL`.

  The *`str`* length cannot be greater than the *`key_str`* length − 11, in bytes

  `key_str` must be a valid key string in PEM format. *`algorithm`* indicates the encryption algorithm used to create the key.

  Supported *`algorithm`* values: `'RSA'`

  To encrypt a string, pass a private or public key string to `asymmetric_encrypt()`. To recover the original unencrypted string, pass the encrypted string to `asymmetric_decrypt()`, along with the public or private key string correponding to the private or public key string used for encryption.

  ```
  -- Generate private/public key pair
  SET @priv = create_asymmetric_priv_key('RSA', 1024);
  SET @pub = create_asymmetric_pub_key('RSA', @priv);

  -- Encrypt using private key, decrypt using public key
  SET @ciphertext = asymmetric_encrypt('RSA', 'The quick brown fox', @priv);
  SET @plaintext = asymmetric_decrypt('RSA', @ciphertext, @pub);

  -- Encrypt using public key, decrypt using private key
  SET @ciphertext = asymmetric_encrypt('RSA', 'The quick brown fox', @pub);
  SET @plaintext = asymmetric_decrypt('RSA', @ciphertext, @priv);
  ```

  Suppose that:

  ```
  SET @s = a string to be encrypted
  SET @priv = a valid private RSA key string in PEM format
  SET @pub = the corresponding public RSA key string in PEM format
  ```

  Then these identity relationships hold:

  ```
  asymmetric_decrypt('RSA', asymmetric_encrypt('RSA', @s, @priv), @pub) = @s
  asymmetric_decrypt('RSA', asymmetric_encrypt('RSA', @s, @pub), @priv) = @s
  ```

- `asymmetric_sign(algorithm, digest_str, priv_key_str, digest_type)`

  Signs a digest string using a private key string, and returns the signature as a binary string. If signing fails, the result is `NULL`.

  `digest_str` is the digest string. It can be generated by calling `create_digest()`. *`digest_type`* indicates the digest algorithm used to generate the digest string.

  `priv_key_str` is the private key string to use for signing the digest string. It must be a valid key string in PEM format. *`algorithm`* indicates the encryption algorithm used to create the key.

  Supported *`algorithm`* values: `'RSA'`, `'DSA'`

  Supported *`digest_type`* values: `'SHA224'`, `'SHA256'`, `'SHA384'`, `'SHA512'`

  For a usage example, see the description of `asymmetric_verify()`.

- `asymmetric_verify(algorithm, digest_str, sig_str, pub_key_str, digest_type)`

  Verifies whether the signature string matches the digest string, and returns 1 or 0 to indicate whether verification succeeded or failed.

  `digest_str` is the digest string. It can be generated by calling `create_digest()`. *`digest_type`* indicates the digest algorithm used to generate the digest string.

  `sig_str` is the signature string. It can be generated by calling `asymmetric_sign()`.

  `pub_key_str` is the public key string of the signer. It corresponds to the private key passed to `asymmetric_sign()` to generate the signature string and must be a valid key string in PEM format. *`algorithm`* indicates the encryption algorithm used to create the key.

  Supported *`algorithm`* values: `'RSA'`, `'DSA'`

  Supported *`digest_type`* values: `'SHA224'`, `'SHA256'`, `'SHA384'`, `'SHA512'`

```
-- Set the encryption algorithm and digest type
SET @algo = 'RSA';
SET @dig_type = 'SHA224';

-- Create private/public key pair
SET @priv = create_asymmetric_priv_key(@algo, 1024);
SET @pub = create_asymmetric_pub_key(@algo, @priv);

-- Generate digest from string
SET @dig = create_digest(@dig_type, 'The quick brown fox');

-- Generate signature for digest and verify signature against digest
SET @sig = asymmetric_sign(@algo, @dig, @priv, @dig_type);
SET @verf = asymmetric_verify(@algo, @dig, @sig, @pub, @dig_type);
```

- `create_asymmetric_priv_key(algorithm, {key_len|dh_secret})`

  Creates a private key using the given algorithm and key length or DH secret, and returns the key as a binary string in PEM format. If key generation fails, the result is `NULL`.

  Supported *`algorithm`* values: `'RSA'`, `'DSA'`, `'DH'`

  Supported *`key_len`* values: The minimum key length in bits is 1,024. The maximum key length depends on the algorithm: 16,384 for RSA and 10,000 for DSA. These key-length limits are constraints imposed by OpenSSL. Server administrators can impose additional limits on maximum key length by setting environment variables. See Section 6.5.2, “MySQL Enterprise Encryption Usage and Examples”.

  For DH keys, pass a shared DH secret instead of a key length. To create the secret, pass the key length to `create_dh_parameters()`.

  This example creates a 2,048-bit DSA private key, then derives a public key from the private key:

```
SET @priv = create_asymmetric_priv_key('DSA', 2048);
```

```
SET @pub = create_asymmetric_pub_key('DSA', @priv);
```

For an example showing DH key generation, see the description of `asymmetric_derive()`.

Some general considerations in choosing key lengths and encryption algorithms:

- The strength of encryption for private and public keys increases with the key size, but the time for key generation increases as well.
- Generation of DH keys takes much longer than RSA or RSA keys.
- Asymmetric encryption functions are slower than symmetric functions. If performance is an important factor and the functions are to be used very frequently, you are better off using symmetric encryption. For example, consider using `AES_ENCRYPT()` and `AES_DECRYPT()`.

- `create_asymmetric_pub_key(algorithm, priv_key_str)`

  Derives a public key from the given private key using the given algorithm, and returns the key as a binary string in PEM format. If key derivation fails, the result is `NULL`.

  `priv_key_str` must be a valid key string in PEM format. *`algorithm`* indicates the encryption algorithm used to create the key.

  Supported *`algorithm`* values: `'RSA'`, `'DSA'`, `'DH'`

  For a usage example, see the description of `create_asymmetric_priv_key()`.

- `create_dh_parameters(key_len)`

  Creates a shared secret for generating a DH private/public key pair and returns a binary string that can be passed to `create_asymmetric_priv_key()`. If secret generation fails, the result is null.

  Supported *`key_len`* values: The minimum and maximum key lengths in bits are 1,024 and 10,000. These key-length limits are constraints imposed by OpenSSL. Server administrators can impose additional limits on maximum key length by setting environment variables. See Section 6.5.2, “MySQL Enterprise Encryption Usage and Examples”.

  For an example showing how to use the return value for generating symmetric keys, see the description of `asymmetric_derive()`.

```
SET @dhp = create_dh_parameters(1024);
```

- `create_digest(digest_type, str)`

  Creates a digest from the given string using the given digest type, and returns the digest as a binary string. If digest generation fails, the result is `NULL`.

  Supported *`digest_type`* values: `'SHA224'`, `'SHA256'`, `'SHA384'`, `'SHA512'`

```
SET @dig = create_digest('SHA512', The quick brown fox');
```

The resulting digest string is suitable for use with `asymmetric_sign()` and `asymmetric_verify()`.

6.6 SELinux

Security-Enhanced Linux (SELinux) is a mandatory access control (MAC) system that implements access rights by applying a security label referred to as an *SELinux context* to each system object. SELinux policy modules use SELinux contexts to define rules for how processes, files, ports, and other system objects interact with each other. Interaction between system objects is only permitted if a policy rule allows it.

An SELinux context (the label applied to a system object) has the following fields: `user`, `role`, `type`, and `security level`. Type information rather than the entire SELinux context is used most commonly to define rules for how processes interact with other system objects. MySQL SELinux policy modules, for example, define policy rules using `type` information.

You can view SELinux contexts using operating system commands such as `ls` and `ps` with the `-Z` option. Assuming that SELinux is enabled and a MySQL Server is running, the following commands show the SELinux context for the `mysqld` process and MySQL data directory:

`mysqld` process:

```
shell> ps -eZ | grep mysqld
system_u:system_r:mysqld_t:s0    5924 ?        00:00:03 mysqld
```

MySQL data directory:

```
shell> cd /var/lib
shell> ls -Z | grep mysql
system_u:object_r:mysqld_db_t:s0 mysql
```

where:

- `system_u` is an SELinux user identity for system processes and objects.
- `system_r` is an SELinux role used for system processes.
- `objects_r` is an SELinux role used for system objects.
- `mysqld_t` is the type associated with the mysqld process.
- `mysqld_db_t` is the type associated with the MySQL data directory and its files.
- `s0` is the security level.

For more information about interpreting SELinux contexts, refer to your distribution's SELinux documentation.

6.6.1 Check if SELinux is Enabled

SELinux is enabled by default on some Linux distributions including Oracle Linux, RHEL, CentOS, and Fedora. Use the `sestatus` command to determine if SELinux is enabled on your distribution:

```
shell> sestatus
SELinux status:                 enabled
SELinuxfs mount:                /sys/fs/selinux
SELinux root directory:         /etc/selinux
Loaded policy name:             targeted
Current mode:                   enforcing
Mode from config file:          enforcing
Policy MLS status:              enabled
Policy deny_unknown status:     allowed
Memory protection checking:     actual (secure)
Max kernel policy version:      31
```

If SELinux is disabled or the `sestatus` command is not found, refer to your distribution's SELinux documentation for guidance before enabling SELinux.

6.6.2 Changing the SELinux Mode

SELinux supports enforcing, permissive, and disabled modes. Enforcing mode is the default. Permissive mode allows operations that are not permitted in enforcing mode and logs those operations to the SELinux audit log. Permissive mode is typically used when developing policies or troubleshooting. In disabled mode, polices are not enforced, and contexts are not applied to system objects, which makes it difficult to enable SELinux later.

To view the current SELinux mode, use the `sestatus` command mentioned previously or the `getenforce` utility.

```
shell> getenforce
Enforcing
```

To change the SELinux mode, use the `setenforce` utility:

```
shell> setenforce 0
shell> getenforce
Permissive
```

```
shell> setenforce 1
shell> getenforce
Enforcing
```

Changes made with `setenforce` are lost when you restart the system. To permanently change the SELinux mode, edit the `/etc/selinux/config` file and restart the system.

6.6.3 MySQL Server SELinux Policies

MySQL Server SELinux policy modules are typically installed by default. You can view installed modules using the `semodule -l` command. MySQL Server SELinux policy modules include:

- `myqld_selinux`
- `myqld_safe_selinux`

For information about MySQL Server SELinux policy modules, refer to the SELinux manual pages. The manual pages provide information about types and Booleans associated with the MySQL service. Manual pages are named in the *`service-name`*`_selinux` format.

```
man mysqld_selinux
```

If SELinux manual pages are not available, refer to your distribution's SELinux documentation for information about how to generate manual pages using the `sepolicy manpage` utility.

6.6.4 SELinux File Context

The MySQL Server reads from and writes to many files. If the SELinux context is not set correctly for these files, access to the files could be denied.

The instructions that follow use the `semanage` binary to manage file context; on RHEL, it's part of the `policycoreutils-python-utils` package:

```
yum install -y policycoreutils-python-utils
```

After installing the `semanage` binary, you can list MySQL file contexts using `semanage` with the `fcontext` option.

```
semanage fcontext -l | grep -i mysql
```

Setting the MySQL Data Directory Context

The default data directory location is `/var/lib/mysql/`; and the SELinux context used is `mysqld_db_t`.

If you edit the configuration file to use a different location for the data directory, or for any of the files normally in the data directory (such as the binary logs), you may need to set the context for the new location. For example:

```
semanage fcontext -a -t mysqld_db_t "/path/to/my/custom/datadir(/.*)?"
restorecon -Rv /path/to/my/custom/datadir
```

```
semanage fcontext -a -t mysqld_db_t "/path/to/my/custom/logdir(/.*)?"
restorecon -Rv /path/to/my/custom/logdir
```

Setting the MySQL Error Log File Context

The default location for RedHat RPMs is `/var/log/mysqld.log`; and the SELinux context type used is `mysqld_log_t`.

If you edit the configuration file to use a different location, you may need to set the context for the new location. For example:

```
semanage fcontext -a -t mysqld_log_t "/path/to/my/custom/error.log"
restorecon -Rv /path/to/my/custom/error.log
```

Setting the PID File Context

The default location for the PID file is `/var/run/mysqld/mysqld.pid`; and the SELinux context type used is `mysqld_var_run_t`.

If you edit the configuration file to use a different location, you may need to set the context for the new location. For example:

```
semanage fcontext -a -t mysqld_var_run_t "/path/to/my/custom/pidfile/directory/.*?"
restorecon -Rv /path/to/my/custom/pidfile/directory
```

Setting the Unix Domain Socket Context

The default location for the Unix domain socket is `/var/lib/mysql/mysql.sock`; and the SELinux context type used is `mysqld_var_run_t`.

If you edit the configuration file to use a different location, you may need to set the context for the new location. For example:

```
semanage fcontext -a -t mysqld_var_run_t "/path/to/my/custom/mysql\.sock"
restorecon -Rv /path/to/my/custom/mysql.sock
```

Setting the secure_file_priv Directory Context

For MySQL versions since 5.6.34, 5.7.16, and 8.0.11.

Installing the MySQL Server RPM creates a `/var/lib/mysql-files/` directory but does not set the SELinux context for it. The `/var/lib/mysql-files/` directory is intended to be used for operations such as `SELECT ... INTO OUTFILE`.

If you enabled the use of this directory by setting `secure_file_priv`, you may need to set the context like so:

```
semanage fcontext -a -t mysqld_db_t "/var/lib/mysql-files/(/.*)?"
restorecon -Rv /var/lib/mysql-files
```

Edit this path if you used a different location. For security purposes, this directory should never be within the data directory.

For more information about this variable, see the `secure_file_priv` documentation.

6.6.5 SELinux TCP Port Context

The instructions that follow use the `semanage` binary to manage port context; on RHEL, it's part of the `policycoreutils-python-utils` package:

```
yum install -y policycoreutils-python-utils
```

After installing the `semanage` binary, you can list ports defined with the `mysqld_port_t` context using `semanage` with the `port` option.

```
shell> semanage port -l | grep mysqld
mysqld_port_t                  tcp      1186, 3306, 63132-63164
```

6.6.5.1 Setting the TCP Port Context for mysqld

The default TCP port for `mysqld` is `3306`; and the SELinux context type used is `mysqld_port_t`.

If you configure `mysqld` to use a different TCP `port`, you may need to set the context for the new port. For example to define the SELinux context for a non-default port such as port 3307:

```
semanage port -a -t mysqld_port_t -p tcp 3307
```

To confirm that the port is added:

```
shell> semanage port -l | grep mysqld
mysqld_port_t                  tcp      3307, 1186, 3306, 63132-63164
```

6.6.5.2 Setting the TCP Port Context for MySQL Features

If you enable certain MySQL features, you may need to set the SELinux TCP port context for additional ports used by those features. If ports used by MySQL features do not have the correct SELinux context, the features may not function correctly.

Generally, the following method can be used to set the port context for other MySQL features:

1. View the ports currently enabled for MySQL:

   ```
   semanage port -l | grep mysqld
   ```

2. If the port used by the MySQL feature is not listed, enable the required port using the following command:

   ```
   semanage port -a -t mysqld_port_t -p tcp port_number
   ```

   where *`port_number`* is the port number you want to enable.

For information about ports used by MySQL features, refer to the MySQL Port Reference.

6.6.6 Troubleshooting SELinux

Troubleshooting SELinux typically involves placing SELinux into permissive mode, rerunning problematic operations, checking for access denial messages in the SELinux audit log, and placing SELinux back into enforcing mode after problems are resolved.

To avoid placing the entire system into permissive mode using `setenforce`, you can permit only the MySQL service to run permissively by placing its SELinux domain (`mysqld_t`) into permissive mode using the `semanage` command:

```
semanage permissive -a mysqld_t
```

When you are finished troubleshooting, use this command to place the `mysqld_t` domain back into enforcing mode:

```
semanage permissive -d mysqld_t
```

SELinux writes logs for denied operations to `/var/log/audit/audit.log`. You can check for denials by searching for “denied” messages.

```
grep "denied" /var/log/audit/audit.log
```

The following sections describes a few common areas where SELinux-related issues may be encountered.

File Contexts

If a MySQL directory or file has an incorrect SELinux context, access may be denied. This issue can occur if MySQL is configured to read from or write to a non-default directory or file. For example, if you configure MySQL to use a non-default data directory, the directory may not have the expected SELinux context.

Attempting to start the MySQL service on a non-default data directory with an invalid SELinux context causes the following startup failure.

```
shell> systemctl start mysql.service
Job for mysqld.service failed because the control process exited with error code.
See "systemctl status mysqld.service" and "journalctl -xe" for details.
```

In this case, a “denial” message is logged to `/var/log/audit/audit.log`:

```
shell> grep "denied" /var/log/audit/audit.log
type=AVC msg=audit(1587133719.786:194): avc:  denied  { write } for  pid=7133 comm="mysqld"
name="mysql" dev="dm-0" ino=51347078 scontext=system_u:system_r:mysqld_t:s0
tcontext=unconfined_u:object_r:default_t:s0 tclass=dir permissive=0
```

For information about setting the proper SELinux context for MySQL directories and files, see Section 6.6.4, “SELinux File Context”.

Port Access

SELinux expects services such as MySQL Server to use specific ports. Changing ports without updating the SELinux policies may cause a service failure.

The `mysqld_port_t` port type defines the ports that the MySQL listens on. If you configure the MySQL Server to use a non-default port, such as port 3307, and do not update the policy to reflect the change, the MySQL service fails to start:

```
shell> systemctl start mysqld.service
Job for mysqld.service failed because the control process exited with error code.
See "systemctl status mysqld.service" and "journalctl -xe" for details.
```

In this case, a denial message is logged to `/var/log/audit/audit.log`:

```
shell> grep "denied" /var/log/audit/audit.log
type=AVC msg=audit(1587134375.845:198): avc:  denied  { name_bind } for  pid=7340
comm="mysqld" src=3307 scontext=system_u:system_r:mysqld_t:s0
tcontext=system_u:object_r:unreserved_port_t:s0 tclass=tcp_socket permissive=0
```

For information about setting the proper SELinux port context for MySQL, see Section 6.6.5, “SELinux TCP Port Context”. Similar port access issues can occur when enabling MySQL features that use ports that are not defined with the required context. For more information, see Section 6.6.5.2, “Setting the TCP Port Context for MySQL Features”.

Application Changes

SELinux may not be aware of application changes. For example, a new release, an application extension, or a new feature may access system resources in a way that is not permitted by SELinux, resulting in access denials. In such cases, you can use the `audit2allow` utility to create custom policies to permit access where it is required. The typical method for creating custom policies is to change the SELinux mode to permissive, identify access denial messages in the SELinux audit log, and use the `audit2allow` utility to create custom policies to permit access.

For information about using the `audit2allow` utility, refer to your distribution's SELinux documentation.

If you encounter access issues for MySQL that you believe should be handled by standard MySQL SELinux policy modules, please open a bug report in your distribution's bug tracking system.

Chapter 7 Backup and Recovery

Table of Contents

7.1 Backup and Recovery Types 938
7.2 Database Backup Methods 941
7.3 Example Backup and Recovery Strategy 943
7.3.1 Establishing a Backup Policy 943
7.3.2 Using Backups for Recovery 945
7.3.3 Backup Strategy Summary 946
7.4 Using mysqldump for Backups 946
7.4.1 Dumping Data in SQL Format with mysqldump 946
7.4.2 Reloading SQL-Format Backups 947
7.4.3 Dumping Data in Delimited-Text Format with mysqldump 948
7.4.4 Reloading Delimited-Text Format Backups 949
7.4.5 mysqldump Tips 950
7.5 Point-in-Time (Incremental) Recovery Using the Binary Log 951
7.5.1 Point-in-Time Recovery Using Event Times 953
7.5.2 Point-in-Time Recovery Using Event Positions 953
7.6 MyISAM Table Maintenance and Crash Recovery 954
7.6.1 Using myisamchk for Crash Recovery 955
7.6.2 How to Check MyISAM Tables for Errors 955
7.6.3 How to Repair MyISAM Tables 956
7.6.4 MyISAM Table Optimization 958
7.6.5 Setting Up a MyISAM Table Maintenance Schedule 959

It is important to back up your databases so that you can recover your data and be up and running again in case problems occur, such as system crashes, hardware failures, or users deleting data by mistake. Backups are also essential as a safeguard before upgrading a MySQL installation, and they can be used to transfer a MySQL installation to another system or to set up replica servers.

MySQL offers a variety of backup strategies from which you can choose the methods that best suit the requirements for your installation. This chapter discusses several backup and recovery topics with which you should be familiar:

- Types of backups: Logical versus physical, full versus incremental, and so forth.
- Methods for creating backups.
- Recovery methods, including point-in-time recovery.
- Backup scheduling, compression, and encryption.
- Table maintenance, to enable recovery of corrupt tables.

Additional Resources

Resources related to backup or to maintaining data availability include the following:

- Customers of MySQL Enterprise Edition can use the MySQL Enterprise Backup product for backups. For an overview of the MySQL Enterprise Backup product, see Section 24.2, “MySQL Enterprise Backup Overview”.
- A forum dedicated to backup issues is available at https://forums.mysql.com/list.php?28.
- Details for `mysqldump`, `mysqlhotcopy`, and other MySQL backup programs can be found in Chapter 4, *MySQL Programs*.
- The syntax of the SQL statements described here is given in Chapter 13, *SQL Statements*.

- For additional information about InnoDB backup procedures, see Section 14.18.1, “InnoDB Backup”.
- Replication enables you to maintain identical data on multiple servers. This has several benefits, such as enabling client query load to be distributed over servers, availability of data even if a given server is taken offline or fails, and the ability to make backups with no impact on the source by using a replica server. See Chapter 17, *Replication*.
- NDB Cluster provides a high-availability, high-redundancy version of MySQL adapted for the distributed computing environment. See Chapter 18, *MySQL NDB Cluster 7.3 and NDB Cluster 7.4*, which provides information about MySQL NDB Cluster 7.4 (based on MySQL 5.6 but containing the latest improvements and fixes for the NDB storage engine).
- Distributed Replicated Block Device (DRBD) is another high-availability solution. It works by replicating a block device from a primary server to a secondary server at the block level. See Chapter 16, *High Availability and Scalability*.

7.1 Backup and Recovery Types

This section describes the characteristics of different types of backups.

Physical (Raw) Versus Logical Backups

Physical backups consist of raw copies of the directories and files that store database contents. This type of backup is suitable for large, important databases that need to be recovered quickly when problems occur.

Logical backups save information represented as logical database structure (CREATE DATABASE, CREATE TABLE statements) and content (INSERT statements or delimited-text files). This type of backup is suitable for smaller amounts of data where you might edit the data values or table structure, or recreate the data on a different machine architecture.

Physical backup methods have these characteristics:

- The backup consists of exact copies of database directories and files. Typically this is a copy of all or part of the MySQL data directory.
- Physical backup methods are faster than logical because they involve only file copying without conversion.
- Output is more compact than for logical backup.
- Because backup speed and compactness are important for busy, important databases, the MySQL Enterprise Backup product performs physical backups. For an overview of the MySQL Enterprise Backup product, see Section 24.2, “MySQL Enterprise Backup Overview”.
- Backup and restore granularity ranges from the level of the entire data directory down to the level of individual files. This may or may not provide for table-level granularity, depending on storage engine. For example, InnoDB tables can each be in a separate file, or share file storage with other InnoDB tables; each MyISAM table corresponds uniquely to a set of files.
- In addition to databases, the backup can include any related files such as log or configuration files.
- Data from MEMORY tables is tricky to back up this way because their contents are not stored on disk. (The MySQL Enterprise Backup product has a feature where you can retrieve data from MEMORY tables during a backup.)
- Backups are portable only to other machines that have identical or similar hardware characteristics.
- Backups can be performed while the MySQL server is not running. If the server is running, it is necessary to perform appropriate locking so that the server does not change database contents during the backup. MySQL Enterprise Backup does this locking automatically for tables that require it.

- Physical backup tools include the `mysqlbackup` of MySQL Enterprise Backup for `InnoDB` or any other tables, file system-level commands (such as `cp`, `scp`, `tar`, `rsync`), or `mysqlhotcopy` for `MyISAM` tables.
- For restore:
  - MySQL Enterprise Backup restores `InnoDB` and other tables that it backed up.
  - `ndb_restore` restores `NDB` tables.
  - Files copied at the file system level or with `mysqlhotcopy` can be copied back to their original locations with file system commands.

Logical backup methods have these characteristics:

- The backup is done by querying the MySQL server to obtain database structure and content information.
- Backup is slower than physical methods because the server must access database information and convert it to logical format. If the output is written on the client side, the server must also send it to the backup program.
- Output is larger than for physical backup, particularly when saved in text format.
- Backup and restore granularity is available at the server level (all databases), database level (all tables in a particular database), or table level. This is true regardless of storage engine.
- The backup does not include log or configuration files, or other database-related files that are not part of databases.
- Backups stored in logical format are machine independent and highly portable.
- Logical backups are performed with the MySQL server running. The server is not taken offline.
- Logical backup tools include the `mysqldump` program and the `SELECT ... INTO OUTFILE` statement. These work for any storage engine, even `MEMORY`.
- To restore logical backups, SQL-format dump files can be processed using the `mysql` client. To load delimited-text files, use the `LOAD DATA` statement or the `mysqlimport` client.

Online Versus Offline Backups

Online backups take place while the MySQL server is running so that the database information can be obtained from the server. Offline backups take place while the server is stopped. This distinction can also be described as “hot” versus “cold” backups; a “warm” backup is one where the server remains running but locked against modifying data while you access database files externally.

Online backup methods have these characteristics:

- The backup is less intrusive to other clients, which can connect to the MySQL server during the backup and may be able to access data depending on what operations they need to perform.
- Care must be taken to impose appropriate locking so that data modifications do not take place that would compromise backup integrity. The MySQL Enterprise Backup product does such locking automatically.

Offline backup methods have these characteristics:

- Clients can be affected adversely because the server is unavailable during backup. For that reason, such backups are often taken from a replica server that can be taken offline without harming availability.
- The backup procedure is simpler because there is no possibility of interference from client activity.

A similar distinction between online and offline applies for recovery operations, and similar characteristics apply. However, it is more likely that clients are affected for online recovery than for online backup because recovery requires stronger locking. During backup, clients might be able to read data while it is being backed up. Recovery modifies data and does not just read it, so clients must be prevented from accessing data while it is being restored.

Local Versus Remote Backups

A local backup is performed on the same host where the MySQL server runs, whereas a remote backup is done from a different host. For some types of backups, the backup can be initiated from a remote host even if the output is written locally on the server. host.

- `mysqldump` can connect to local or remote servers. For SQL output (`CREATE` and `INSERT` statements), local or remote dumps can be done and generate output on the client. For delimited-text output (with the `--tab` option), data files are created on the server host.
- `mysqlhotcopy` performs only local backups: It connects to the server to lock it against data modifications and then copies local table files.
- `SELECT ... INTO OUTFILE` can be initiated from a local or remote client host, but the output file is created on the server host.
- Physical backup methods typically are initiated locally on the MySQL server host so that the server can be taken offline, although the destination for copied files might be remote.

Snapshot Backups

Some file system implementations enable “snapshots” to be taken. These provide logical copies of the file system at a given point in time, without requiring a physical copy of the entire file system. (For example, the implementation may use copy-on-write techniques so that only parts of the file system modified after the snapshot time need be copied.) MySQL itself does not provide the capability for taking file system snapshots. It is available through third-party solutions such as Veritas, LVM, or ZFS.

Full Versus Incremental Backups

A full backup includes all data managed by a MySQL server at a given point in time. An incremental backup consists of the changes made to the data during a given time span (from one point in time to another). MySQL has different ways to perform full backups, such as those described earlier in this section. Incremental backups are made possible by enabling the server's binary log, which the server uses to record data changes.

Full Versus Point-in-Time (Incremental) Recovery

A full recovery restores all data from a full backup. This restores the server instance to the state that it had when the backup was made. If that state is not sufficiently current, a full recovery can be followed by recovery of incremental backups made since the full backup, to bring the server to a more up-to-date state.

Incremental recovery is recovery of changes made during a given time span. This is also called point-in-time recovery because it makes a server's state current up to a given time. Point-in-time recovery is based on the binary log and typically follows a full recovery from the backup files that restores the server to its state when the backup was made. Then the data changes written in the binary log files are applied as incremental recovery to redo data modifications and bring the server up to the desired point in time.

Table Maintenance

Data integrity can be compromised if tables become corrupt. For `InnoDB` tables, this is not a typical issue. For programs to check `MyISAM` tables and repair them if problems are found, see Section 7.6, “MyISAM Table Maintenance and Crash Recovery”.

Backup Scheduling, Compression, and Encryption

Backup scheduling is valuable for automating backup procedures. Compression of backup output reduces space requirements, and encryption of the output provides better security against unauthorized access of backed-up data. MySQL itself does not provide these capabilities. The MySQL Enterprise Backup product can compress `InnoDB` backups, and compression or encryption of backup output can be achieved using file system utilities. Other third-party solutions may be available.

7.2 Database Backup Methods

This section summarizes some general methods for making backups.

Making a Hot Backup with MySQL Enterprise Backup

Customers of MySQL Enterprise Edition can use the MySQL Enterprise Backup product to do physical backups of entire instances or selected databases, tables, or both. This product includes features for incremental and compressed backups. Backing up the physical database files makes restore much faster than logical techniques such as the `mysqldump` command. `InnoDB` tables are copied using a hot backup mechanism. (Ideally, the `InnoDB` tables should represent a substantial majority of the data.) Tables from other storage engines are copied using a warm backup mechanism. For an overview of the MySQL Enterprise Backup product, see Section 24.2, “MySQL Enterprise Backup Overview”.

Making Backups with mysqldump or mysqlhotcopy

The `mysqldump` program and the `mysqlhotcopy` script can make backups. `mysqldump` is more general because it can back up all kinds of tables. `mysqlhotcopy` works only with some storage engines. (See Section 7.4, “Using mysqldump for Backups”, and Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”.)

For `InnoDB` tables, it is possible to perform an online backup that takes no locks on tables using the `--single-transaction` option to `mysqldump`. See Section 7.3.1, “Establishing a Backup Policy”.

Making Backups by Copying Table Files

For storage engines that represent each table using its own files, tables can be backed up by copying those files. For example, `MyISAM` tables are stored as files, so it is easy to do a backup by copying files (`*.frm`, `*.MYD`, and `*.MYI` files). To get a consistent backup, stop the server or lock and flush the relevant tables:

```
FLUSH TABLES tbl_list WITH READ LOCK;
```

You need only a read lock; this enables other clients to continue to query the tables while you are making a copy of the files in the database directory. The flush is needed to ensure that the all active index pages are written to disk before you start the backup. See Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”, and Section 13.7.6.3, “FLUSH Statement”.

You can also create a binary backup simply by copying all table files, as long as the server isn't updating anything. The `mysqlhotcopy` script uses this method. (But note that table file copying methods do not work if your database contains `InnoDB` tables. `mysqlhotcopy` does not work for `InnoDB` tables because `InnoDB` does not necessarily store table contents in database directories. Also, even if the server is not actively updating data, `InnoDB` may still have modified data cached in memory and not flushed to disk.)

Making Delimited-Text File Backups

To create a text file containing a table's data, you can use `SELECT * INTO OUTFILE 'file_name' FROM tbl_name`. The file is created on the MySQL server host, not the client host. For this statement,

the output file cannot already exist because permitting files to be overwritten constitutes a security risk. See Section 13.2.9, “SELECT Statement”. This method works for any kind of data file, but saves only table data, not the table structure.

Another way to create text data files (along with files containing `CREATE TABLE` statements for the backed up tables) is to use `mysqldump` with the `--tab` option. See Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”.

To reload a delimited-text data file, use `LOAD DATA` or `mysqlimport`.

Making Incremental Backups by Enabling the Binary Log

MySQL supports incremental backups: You must start the server with the `--log-bin` option to enable binary logging; see Section 5.4.4, “The Binary Log”. The binary log files provide you with the information you need to replicate changes to the database that are made subsequent to the point at which you performed a backup. At the moment you want to make an incremental backup (containing all changes that happened since the last full or incremental backup), you should rotate the binary log by using `FLUSH LOGS`. This done, you need to copy to the backup location all binary logs which range from the one of the moment of the last full or incremental backup to the last but one. These binary logs are the incremental backup; at restore time, you apply them as explained in Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”. The next time you do a full backup, you should also rotate the binary log using `FLUSH LOGS`, `mysqldump --flush-logs`, or `mysqlhotcopy --flushlog`. See Section 4.5.4, “`mysqldump` — A Database Backup Program”, and Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”.

Making Backups Using Replicas

If you have performance problems with your source server while making backups, one strategy that can help is to set up replication and perform backups on the replica rather than on the source. See Section 17.3.1, “Using Replication for Backups”.

If you are backing up a replica server, you should back up its source info and relay log info repositories (see Section 17.2.2, “Relay Log and Replication Metadata Repositories”) when you back up the replica's databases, regardless of the backup method you choose. These information files are always needed to resume replication after you restore the replica's data. If your replica is replicating `LOAD DATA` statements, you should also back up any `SQL_LOAD-*` files that exist in the directory that the replica uses for this purpose. The replica needs these files to resume replication of any interrupted `LOAD DATA` operations. The location of this directory is the value of the `slave_load_tmpdir` system variable. If the server was not started with that variable set, the directory location is the value of the `tmpdir` system variable.

Recovering Corrupt Tables

If you have to restore `MyISAM` tables that have become corrupt, try to recover them using `REPAIR TABLE` or `myisamchk -r` first. That should work in 99.9% of all cases. If `myisamchk` fails, see Section 7.6, “MyISAM Table Maintenance and Crash Recovery”.

Making Backups Using a File System Snapshot

If you are using a Veritas file system, you can make a backup like this:

1. From a client program, execute `FLUSH TABLES WITH READ LOCK`.
2. From another shell, execute `mount vxfs snapshot`.
3. From the first client, execute `UNLOCK TABLES`.
4. Copy files from the snapshot.
5. Unmount the snapshot.

Similar snapshot capabilities may be available in other file systems, such as LVM or ZFS.

7.3 Example Backup and Recovery Strategy

This section discusses a procedure for performing backups that enables you to recover data after several types of crashes:

- Operating system crash
- Power failure
- File system crash
- Hardware problem (hard drive, motherboard, and so forth)

The example commands do not include options such as `--user` and `--password` for the `mysqldump` and `mysql` client programs. You should include such options as necessary to enable client programs to connect to the MySQL server.

Assume that data is stored in the `InnoDB` storage engine, which has support for transactions and automatic crash recovery. Assume also that the MySQL server is under load at the time of the crash. If it were not, no recovery would ever be needed.

For cases of operating system crashes or power failures, we can assume that MySQL's disk data is available after a restart. The `InnoDB` data files might not contain consistent data due to the crash, but `InnoDB` reads its logs and finds in them the list of pending committed and noncommitted transactions that have not been flushed to the data files. `InnoDB` automatically rolls back those transactions that were not committed, and flushes to its data files those that were committed. Information about this recovery process is conveyed to the user through the MySQL error log. The following is an example log excerpt:

```
InnoDB: Database was not shut down normally.
InnoDB: Starting recovery from log files...
InnoDB: Starting log scan based on checkpoint at
InnoDB: log sequence number 0 13674004
InnoDB: Doing recovery: scanned up to log sequence number 0 13739520
InnoDB: Doing recovery: scanned up to log sequence number 0 13805056
InnoDB: Doing recovery: scanned up to log sequence number 0 13870592
InnoDB: Doing recovery: scanned up to log sequence number 0 13936128
...
InnoDB: Doing recovery: scanned up to log sequence number 0 20555264
InnoDB: Doing recovery: scanned up to log sequence number 0 20620800
InnoDB: Doing recovery: scanned up to log sequence number 0 20664692
InnoDB: 1 uncommitted transaction(s) which must be rolled back
InnoDB: Starting rollback of uncommitted transactions
InnoDB: Rolling back trx no 16745
InnoDB: Rolling back of trx no 16745 completed
InnoDB: Rollback of uncommitted transactions completed
InnoDB: Starting an apply batch of log records to the database...
InnoDB: Apply batch completed
InnoDB: Started
mysqld: ready for connections
```

For the cases of file system crashes or hardware problems, we can assume that the MySQL disk data is *not* available after a restart. This means that MySQL fails to start successfully because some blocks of disk data are no longer readable. In this case, it is necessary to reformat the disk, install a new one, or otherwise correct the underlying problem. Then it is necessary to recover our MySQL data from backups, which means that backups must already have been made. To make sure that is the case, design and implement a backup policy.

7.3.1 Establishing a Backup Policy

To be useful, backups must be scheduled regularly. A full backup (a snapshot of the data at a point in time) can be done in MySQL with several tools. For example, MySQL Enterprise Backup can perform

a physical backup of an entire instance, with optimizations to minimize overhead and avoid disruption when backing up `InnoDB` data files; `mysqldump` provides online logical backup. This discussion uses `mysqldump`.

Assume that we make a full backup of all our `InnoDB` tables in all databases using the following command on Sunday at 1 p.m., when load is low:

```
shell> mysqldump --all-databases --master-data --single-transaction > backup_sunday_1_PM.sql
```

The resulting `.sql` file produced by `mysqldump` contains a set of SQL `INSERT` statements that can be used to reload the dumped tables at a later time.

This backup operation acquires a global read lock on all tables at the beginning of the dump (using `FLUSH TABLES WITH READ LOCK`). As soon as this lock has been acquired, the binary log coordinates are read and the lock is released. If long updating statements are running when the `FLUSH` statement is issued, the backup operation may stall until those statements finish. After that, the dump becomes lock-free and does not disturb reads and writes on the tables.

It was assumed earlier that the tables to back up are `InnoDB` tables, so `--single-transaction` uses a consistent read and guarantees that data seen by `mysqldump` does not change. (Changes made by other clients to `InnoDB` tables are not seen by the `mysqldump` process.) If the backup operation includes nontransactional tables, consistency requires that they do not change during the backup. For example, for the `MyISAM` tables in the `mysql` database, there must be no administrative changes to MySQL accounts during the backup.

Full backups are necessary, but it is not always convenient to create them. They produce large backup files and take time to generate. They are not optimal in the sense that each successive full backup includes all data, even that part that has not changed since the previous full backup. It is more efficient to make an initial full backup, and then to make incremental backups. The incremental backups are smaller and take less time to produce. The tradeoff is that, at recovery time, you cannot restore your data just by reloading the full backup. You must also process the incremental backups to recover the incremental changes.

To make incremental backups, we need to save the incremental changes. In MySQL, these changes are represented in the binary log, so the MySQL server should always be started with the `--log-bin` option to enable that log. With binary logging enabled, the server writes each data change into a file while it updates data. Looking at the data directory of a MySQL server that was started with the `--log-bin` option and that has been running for some days, we find these MySQL binary log files:

```
-rw-rw---- 1 guilhem  guilhem   1277324 Nov 10 23:59 gbichot2-bin.000001
-rw-rw---- 1 guilhem  guilhem         4 Nov 10 23:59 gbichot2-bin.000002
-rw-rw---- 1 guilhem  guilhem        79 Nov 11 11:06 gbichot2-bin.000003
-rw-rw---- 1 guilhem  guilhem       508 Nov 11 11:08 gbichot2-bin.000004
-rw-rw---- 1 guilhem  guilhem 220047446 Nov 12 16:47 gbichot2-bin.000005
-rw-rw---- 1 guilhem  guilhem    998412 Nov 14 10:08 gbichot2-bin.000006
-rw-rw---- 1 guilhem  guilhem       361 Nov 14 10:07 gbichot2-bin.index
```

Each time it restarts, the MySQL server creates a new binary log file using the next number in the sequence. While the server is running, you can also tell it to close the current binary log file and begin a new one manually by issuing a `FLUSH LOGS` SQL statement or with a `mysqladmin flush-logs` command. `mysqldump` also has an option to flush the logs. The `.index` file in the data directory contains the list of all MySQL binary logs in the directory.

The MySQL binary logs are important for recovery because they form the set of incremental backups. If you make sure to flush the logs when you make your full backup, the binary log files created afterward contain all the data changes made since the backup. Let's modify the previous `mysqldump` command a bit so that it flushes the MySQL binary logs at the moment of the full backup, and so that the dump file contains the name of the new current binary log:

```
shell> mysqldump --single-transaction --flush-logs --master-data=2 \
         --all-databases > backup_sunday_1_PM.sql
```

After executing this command, the data directory contains a new binary log file, `gbichot2-bin.000007`, because the `--flush-logs` option causes the server to flush its logs. The `--master-data` option causes `mysqldump` to write binary log information to its output, so the resulting `.sql` dump file includes these lines:

```
-- Position to start replication or point-in-time recovery from
-- CHANGE MASTER TO MASTER_LOG_FILE='gbichot2-bin.000007',MASTER_LOG_POS=4;
```

Because the `mysqldump` command made a full backup, those lines mean two things:

- The dump file contains all changes made before any changes written to the `gbichot2-bin.000007` binary log file or higher.
- All data changes logged after the backup are not present in the dump file, but are present in the `gbichot2-bin.000007` binary log file or higher.

On Monday at 1 p.m., we can create an incremental backup by flushing the logs to begin a new binary log file. For example, executing a `mysqladmin flush-logs` command creates `gbichot2-bin.000008`. All changes between the Sunday 1 p.m. full backup and Monday 1 p.m. are in the `gbichot2-bin.000007` file. This incremental backup is important, so it is a good idea to copy it to a safe place. (For example, back it up on tape or DVD, or copy it to another machine.) On Tuesday at 1 p.m., execute another `mysqladmin flush-logs` command. All changes between Monday 1 p.m. and Tuesday 1 p.m. are in the `gbichot2-bin.000008` file (which also should be copied somewhere safe).

The MySQL binary logs take up disk space. To free up space, purge them from time to time. One way to do this is by deleting the binary logs that are no longer needed, such as when we make a full backup:

```
shell> mysqldump --single-transaction --flush-logs --master-data=2 \
         --all-databases --delete-master-logs > backup_sunday_1_PM.sql
```

Note

Deleting the MySQL binary logs with `mysqldump --delete-master-logs` can be dangerous if your server is a replication source server, because replica servers might not yet fully have processed the contents of the binary log. The description for the `PURGE BINARY LOGS` statement explains what should be verified before deleting the MySQL binary logs. See Section 13.4.1.1, “PURGE BINARY LOGS Statement”.

7.3.2 Using Backups for Recovery

Now, suppose that we have a catastrophic unexpected exit on Wednesday at 8 a.m. that requires recovery from backups. To recover, first we restore the last full backup we have (the one from Sunday 1 p.m.). The full backup file is just a set of SQL statements, so restoring it is very easy:

```
shell> mysql < backup_sunday_1_PM.sql
```

At this point, the data is restored to its state as of Sunday 1 p.m.. To restore the changes made since then, we must use the incremental backups; that is, the `gbichot2-bin.000007` and `gbichot2-bin.000008` binary log files. Fetch the files if necessary from where they were backed up, and then process their contents like this:

```
shell> mysqlbinlog gbichot2-bin.000007 gbichot2-bin.000008 | mysql
```

We now have recovered the data to its state as of Tuesday 1 p.m., but still are missing the changes from that date to the date of the crash. To not lose them, we would have needed to have the MySQL server store its MySQL binary logs into a safe location (RAID disks, SAN, ...) different from the place where it stores its data files, so that these logs were not on the destroyed disk. (That is, we can start the server with a `--log-bin` option that specifies a location on a different physical device from the one on which the data directory resides. That way, the logs are safe even if the device containing

the directory is lost.) If we had done this, we would have the `gbichot2-bin.000009` file (and any subsequent files) at hand, and we could apply them using `mysqlbinlog` and `mysql` to restore the most recent data changes with no loss up to the moment of the crash:

```
shell> mysqlbinlog gbichot2-bin.000009 ... | mysql
```

For more information about using `mysqlbinlog` to process binary log files, see Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”.

7.3.3 Backup Strategy Summary

In case of an operating system crash or power failure, `InnoDB` itself does all the job of recovering data. But to make sure that you can sleep well, observe the following guidelines:

- Always run the MySQL server with the `--log-bin` option, or even `--log-bin=log_name`, where the log file name is located on some safe media different from the drive on which the data directory is located. If you have such safe media, this technique can also be good for disk load balancing (which results in a performance improvement).
- Make periodic full backups, using the `mysqldump` command shown earlier in Section 7.3.1, “Establishing a Backup Policy”, that makes an online, nonblocking backup.
- Make periodic incremental backups by flushing the logs with `FLUSH LOGS` or `mysqladmin flush-logs`.

7.4 Using mysqldump for Backups

This section describes how to use `mysqldump` to produce dump files, and how to reload dump files. A dump file can be used in several ways:

- As a backup to enable data recovery in case of data loss.
- As a source of data for setting up replicas.
- As a source of data for experimentation:
  - To make a copy of a database that you can use without changing the original data.
  - To test potential upgrade incompatibilities.

`mysqldump` produces two types of output, depending on whether the `--tab` option is given:

- Without `--tab`, `mysqldump` writes SQL statements to the standard output. This output consists of `CREATE` statements to create dumped objects (databases, tables, stored routines, and so forth), and `INSERT` statements to load data into tables. The output can be saved in a file and reloaded later using `mysql` to recreate the dumped objects. Options are available to modify the format of the SQL statements, and to control which objects are dumped.
- With `--tab`, `mysqldump` produces two output files for each dumped table. The server writes one file as tab-delimited text, one line per table row. This file is named *`tbl_name`*`.txt` in the output directory. The server also sends a `CREATE TABLE` statement for the table to `mysqldump`, which writes it as a file named *`tbl_name`*`.sql` in the output directory.

7.4.1 Dumping Data in SQL Format with mysqldump

This section describes how to use `mysqldump` to create SQL-format dump files. For information about reloading such dump files, see Section 7.4.2, “Reloading SQL-Format Backups”.

By default, `mysqldump` writes information as SQL statements to the standard output. You can save the output in a file:

```
shell> mysqldump [arguments] > file_name
```

To dump all databases, invoke `mysqldump` with the `--all-databases` option:

```
shell> mysqldump --all-databases > dump.sql
```

To dump only specific databases, name them on the command line and use the `--databases` option:

```
shell> mysqldump --databases db1 db2 db3 > dump.sql
```

The `--databases` option causes all names on the command line to be treated as database names. Without this option, `mysqldump` treats the first name as a database name and those following as table names.

With `--all-databases` or `--databases`, `mysqldump` writes `CREATE DATABASE` and `USE` statements prior to the dump output for each database. This ensures that when the dump file is reloaded, it creates each database if it does not exist and makes it the default database so database contents are loaded into the same database from which they came. If you want to cause the dump file to force a drop of each database before recreating it, use the `--add-drop-database` option as well. In this case, `mysqldump` writes a `DROP DATABASE` statement preceding each `CREATE DATABASE` statement.

To dump a single database, name it on the command line:

```
shell> mysqldump --databases test > dump.sql
```

In the single-database case, it is permissible to omit the `--databases` option:

```
shell> mysqldump test > dump.sql
```

The difference between the two preceding commands is that without `--databases`, the dump output contains no `CREATE DATABASE` or `USE` statements. This has several implications:

- When you reload the dump file, you must specify a default database name so that the server knows which database to reload.
- For reloading, you can specify a database name different from the original name, which enables you to reload the data into a different database.
- If the database to be reloaded does not exist, you must create it first.
- Because the output contains no `CREATE DATABASE` statement, the `--add-drop-database` option has no effect. If you use it, it produces no `DROP DATABASE` statement.

To dump only specific tables from a database, name them on the command line following the database name:

```
shell> mysqldump test t1 t3 t7 > dump.sql
```

7.4.2 Reloading SQL-Format Backups

To reload a dump file written by `mysqldump` that consists of SQL statements, use it as input to the `mysql` client. If the dump file was created by `mysqldump` with the `--all-databases` or `--databases` option, it contains `CREATE DATABASE` and `USE` statements and it is not necessary to specify a default database into which to load the data:

```
shell> mysql < dump.sql
```

Alternatively, from within `mysql`, use a `source` command:

```
mysql> source dump.sql
```

If the file is a single-database dump not containing `CREATE DATABASE` and `USE` statements, create the database first (if necessary):

```
shell> mysqladmin create db1
```

Then specify the database name when you load the dump file:

```
shell> mysql db1 < dump.sql
```

Alternatively, from within `mysql`, create the database, select it as the default database, and load the dump file:

```
mysql> CREATE DATABASE IF NOT EXISTS db1;
mysql> USE db1;
mysql> source dump.sql
```

Note

For Windows PowerShell users: Because the "<" character is reserved for future use in PowerShell, an alternative approach is required, such as using quotes `cmd.exe /c "mysql < dump.sql"`.

7.4.3 Dumping Data in Delimited-Text Format with mysqldump

This section describes how to use `mysqldump` to create delimited-text dump files. For information about reloading such dump files, see Section 7.4.4, “Reloading Delimited-Text Format Backups”.

If you invoke `mysqldump` with the `--tab=dir_name` option, it uses *`dir_name`* as the output directory and dumps tables individually in that directory using two files for each table. The table name is the base name for these files. For a table named `t1`, the files are named `t1.sql` and `t1.txt`. The `.sql` file contains a `CREATE TABLE` statement for the table. The `.txt` file contains the table data, one line per table row.

The following command dumps the contents of the `db1` database to files in the `/tmp` database:

```
shell> mysqldump --tab=/tmp db1
```

The `.txt` files containing table data are written by the server, so they are owned by the system account used for running the server. The server uses `SELECT ... INTO OUTFILE` to write the files, so you must have the `FILE` privilege to perform this operation, and an error occurs if a given `.txt` file already exists.

The server sends the `CREATE` definitions for dumped tables to `mysqldump`, which writes them to `.sql` files. These files therefore are owned by the user who executes `mysqldump`.

It is best that `--tab` be used only for dumping a local server. If you use it with a remote server, the `--tab` directory must exist on both the local and remote hosts, and the `.txt` files is written by the server in the remote directory (on the server host), whereas the `.sql` files are written by `mysqldump` in the local directory (on the client host).

For `mysqldump --tab`, the server by default writes table data to `.txt` files one line per row with tabs between column values, no quotation marks around column values, and newline as the line terminator. (These are the same defaults as for `SELECT ... INTO OUTFILE`.)

To enable data files to be written using a different format, `mysqldump` supports these options:

- `--fields-terminated-by=str`

  The string for separating column values (default: tab).

- `--fields-enclosed-by=char`

  The character within which to enclose column values (default: no character).

- `--fields-optionally-enclosed-by=char`

The character within which to enclose non-numeric column values (default: no character).

- `--fields-escaped-by=char`

  The character for escaping special characters (default: no escaping).

- `--lines-terminated-by=str`

  The line-termination string (default: newline).

Depending on the value you specify for any of these options, it might be necessary on the command line to quote or escape the value appropriately for your command interpreter. Alternatively, specify the value using hex notation. Suppose that you want `mysqldump` to quote column values within double quotation marks. To do so, specify double quote as the value for the `--fields-enclosed-by` option. But this character is often special to command interpreters and must be treated specially. For example, on Unix, you can quote the double quote like this:

```
--fields-enclosed-by='"'
```

On any platform, you can specify the value in hex:

```
--fields-enclosed-by=0x22
```

It is common to use several of the data-formatting options together. For example, to dump tables in comma-separated values format with lines terminated by carriage-return/newline pairs (`\r\n`), use this command (enter it on a single line):

```
shell> mysqldump --tab=/tmp --fields-terminated-by=,
         --fields-enclosed-by='"' --lines-terminated-by=0x0d0a db1
```

Should you use any of the data-formatting options to dump table data, you must specify the same format when you reload data files later, to ensure proper interpretation of the file contents.

7.4.4 Reloading Delimited-Text Format Backups

For backups produced with `mysqldump --tab`, each table is represented in the output directory by an `.sql` file containing the `CREATE TABLE` statement for the table, and a `.txt` file containing the table data. To reload a table, first change location into the output directory. Then process the `.sql` file with `mysql` to create an empty table and process the `.txt` file to load the data into the table:

```
shell> mysql db1 < t1.sql
shell> mysqlimport db1 t1.txt
```

An alternative to using `mysqlimport` to load the data file is to use the `LOAD DATA` statement from within the `mysql` client:

```
mysql> USE db1;
mysql> LOAD DATA INFILE 't1.txt' INTO TABLE t1;
```

If you used any data-formatting options with `mysqldump` when you initially dumped the table, you must use the same options with `mysqlimport` or `LOAD DATA` to ensure proper interpretation of the data file contents:

```
shell> mysqlimport --fields-terminated-by=,
         --fields-enclosed-by='"' --lines-terminated-by=0x0d0a db1 t1.txt
```

Or:

```
mysql> USE db1;
mysql> LOAD DATA INFILE 't1.txt' INTO TABLE t1
       FIELDS TERMINATED BY ',' FIELDS ENCLOSED BY '"'
       LINES TERMINATED BY '\r\n';
```

7.4.5 mysqldump Tips

This section surveys techniques that enable you to use `mysqldump` to solve specific problems:

- How to make a copy a database
- How to copy a database from one server to another
- How to dump stored programs (stored procedures and functions, triggers, and events)
- How to dump definitions and data separately

7.4.5.1 Making a Copy of a Database

```
shell> mysqldump db1 > dump.sql
shell> mysqladmin create db2
shell> mysql db2 < dump.sql
```

Do not use `--databases` on the `mysqldump` command line because that causes `USE db1` to be included in the dump file, which overrides the effect of naming `db2` on the `mysql` command line.

7.4.5.2 Copy a Database from one Server to Another

On Server 1:

```
shell> mysqldump --databases db1 > dump.sql
```

Copy the dump file from Server 1 to Server 2.

On Server 2:

```
shell> mysql < dump.sql
```

Use of `--databases` with the `mysqldump` command line causes the dump file to include `CREATE DATABASE` and `USE` statements that create the database if it does exist and make it the default database for the reloaded data.

Alternatively, you can omit `--databases` from the `mysqldump` command. Then you must create the database on Server 2 (if necessary) and specify it as the default database when you reload the dump file.

On Server 1:

```
shell> mysqldump db1 > dump.sql
```

On Server 2:

```
shell> mysqladmin create db1
shell> mysql db1 < dump.sql
```

You can specify a different database name in this case, so omitting `--databases` from the `mysqldump` command enables you to dump data from one database and load it into another.

7.4.5.3 Dumping Stored Programs

Several options control how `mysqldump` handles stored programs (stored procedures and functions, triggers, and events):

- `--events`: Dump Event Scheduler events
- `--routines`: Dump stored procedures and functions
- `--triggers`: Dump triggers for tables

The `--triggers` option is enabled by default so that when tables are dumped, they are accompanied by any triggers they have. The other options are disabled by default and must be specified explicitly to dump the corresponding objects. To disable any of these options explicitly, use its skip form: `--skip-events`, `--skip-routines`, or `--skip-triggers`.

7.4.5.4 Dumping Table Definitions and Content Separately

The `--no-data` option tells `mysqldump` not to dump table data, resulting in the dump file containing only statements to create the tables. Conversely, the `--no-create-info` option tells `mysqldump` to suppress `CREATE` statements from the output, so that the dump file contains only table data.

For example, to dump table definitions and data separately for the `test` database, use these commands:

```
shell> mysqldump --no-data test > dump-defs.sql
shell> mysqldump --no-create-info test > dump-data.sql
```

For a definition-only dump, add the `--routines` and `--events` options to also include stored routine and event definitions:

```
shell> mysqldump --no-data --routines --events test > dump-defs.sql
```

7.4.5.5 Using mysqldump to Test for Upgrade Incompatibilities

When contemplating a MySQL upgrade, it is prudent to install the newer version separately from your current production version. Then you can dump the database and database object definitions from the production server and load them into the new server to verify that they are handled properly. (This is also useful for testing downgrades.)

On the production server:

```
shell> mysqldump --all-databases --no-data --routines --events > dump-defs.sql
```

On the upgraded server:

```
shell> mysql < dump-defs.sql
```

Because the dump file does not contain table data, it can be processed quickly. This enables you to spot potential incompatibilities without waiting for lengthy data-loading operations. Look for warnings or errors while the dump file is being processed.

After you have verified that the definitions are handled properly, dump the data and try to load it into the upgraded server.

On the production server:

```
shell> mysqldump --all-databases --no-create-info > dump-data.sql
```

On the upgraded server:

```
shell> mysql < dump-data.sql
```

Now check the table contents and run some test queries.

7.5 Point-in-Time (Incremental) Recovery Using the Binary Log

Point-in-time recovery refers to recovery of data changes made since a given point in time. Typically, this type of recovery is performed after restoring a full backup that brings the server to its state as of the time the backup was made. (The full backup can be made in several ways, such as those listed in Section 7.2, “Database Backup Methods”.) Point-in-time recovery then brings the server up to date incrementally from the time of the full backup to a more recent time.

Note

Many of the examples here use the `mysql` client to process binary log output produced by `mysqlbinlog`. If your binary log contains `\0` (null) characters, that output cannot be parsed by `mysql` unless you invoke it with the `--binary-mode` option.

Point-in-time recovery is based on these principles:

- The source of information for point-in-time recovery is the set of incremental backups represented by the binary log files generated subsequent to the full backup operation. Therefore, the server must be started with the `--log-bin` option to enable binary logging (see Section 5.4.4, “The Binary Log”).

  To restore data from the binary log, you must know the name and location of the current binary log files. By default, the server creates binary log files in the data directory, but a path name can be specified with the `--log-bin` option to place the files in a different location. Section 5.4.4, “The Binary Log”.

  To see a listing of all binary log files, use this statement:

  ```
  mysql> SHOW BINARY LOGS;
  ```

  To determine the name of the current binary log file, issue the following statement:

  ```
  mysql> SHOW MASTER STATUS;
  ```

- The `mysqlbinlog` utility converts the events in the binary log files from binary format to text so that they can be executed or viewed. `mysqlbinlog` has options for selecting sections of the binary log based on event times or position of events within the log. See Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”.

- Executing events from the binary log causes the data modifications they represent to be redone. This enables recovery of data changes for a given span of time. To execute events from the binary log, process `mysqlbinlog` output using the `mysql` client:

  ```
  shell> mysqlbinlog binlog_files | mysql -u root -p
  ```

- Viewing log contents can be useful when you need to determine event times or positions to select partial log contents prior to executing events. To view events from the log, send `mysqlbinlog` output into a paging program:

  ```
  shell> mysqlbinlog binlog_files | more
  ```

  Alternatively, save the output in a file and view the file in a text editor:

  ```
  shell> mysqlbinlog binlog_files > tmpfile
  shell> ... edit tmpfile ...
  ```

- Saving the output in a file is useful as a preliminary to executing the log contents with certain events removed, such as an accidental `DROP DATABASE`. You can delete from the file any statements not to be executed before executing its contents. After editing the file, execute the contents as follows:

  ```
  shell> mysql -u root -p < tmpfile
  ```

If you have more than one binary log to execute on the MySQL server, the safe method is to process them all using a single connection to the server. Here is an example that demonstrates what may be *unsafe*:

```
shell> mysqlbinlog binlog.000001 | mysql -u root -p # DANGER!!
shell> mysqlbinlog binlog.000002 | mysql -u root -p # DANGER!!
```

Processing binary logs this way using different connections to the server causes problems if the first log file contains a `CREATE TEMPORARY TABLE` statement and the second log contains a

statement that uses the temporary table. When the first `mysql` process terminates, the server drops the temporary table. When the second `mysql` process attempts to use the table, the server reports "unknown table."

To avoid problems like this, use a *single* connection to execute the contents of all binary logs that you want to process. Here is one way to do so:

```
shell> mysqlbinlog binlog.000001 binlog.000002 | mysql -u root -p
```

Another approach is to write all the logs to a single file and then process the file:

```
shell> mysqlbinlog binlog.000001 >  /tmp/statements.sql
shell> mysqlbinlog binlog.000002 >> /tmp/statements.sql
shell> mysql -u root -p -e "source /tmp/statements.sql"
```

When writing to a dump file while reading back from a binary log containing GTIDs (see Section 17.1.3, "Replication with Global Transaction Identifiers"), use the `--skip-gtids` option with `mysqlbinlog`, like this:

```
shell> mysqlbinlog --skip-gtids binlog.000001 >  /tmp/dump.sql
shell> mysqlbinlog --skip-gtids binlog.000002 >> /tmp/dump.sql
shell> mysql -u root -p -e "source /tmp/dump.sql"
```

7.5.1 Point-in-Time Recovery Using Event Times

To indicate the start and end times for recovery, specify the `--start-datetime` and `--stop-datetime` options for `mysqlbinlog`, in `DATETIME` format. As an example, suppose that exactly at 10:00 a.m. on April 20, 2005 an SQL statement was executed that deleted a large table. To restore the table and data, you could restore the previous night's backup, and then execute the following command:

```
shell> mysqlbinlog --stop-datetime="2005-04-20 9:59:59" \
        /var/log/mysql/bin.123456 | mysql -u root -p
```

This command recovers all of the data up until the date and time given by the `--stop-datetime` option. If you did not detect the erroneous SQL statement that was entered until hours later, you probably also want to recover the activity that occurred afterward. Based on this, you could run `mysqlbinlog` again with a start date and time, like so:

```
shell> mysqlbinlog --start-datetime="2005-04-20 10:01:00" \
        /var/log/mysql/bin.123456 | mysql -u root -p
```

In this command, the SQL statements logged from 10:01 a.m. on is re-executed. The combination of restoring of the previous night's dump file and the two `mysqlbinlog` commands restores everything up until one second before 10:00 a.m. and everything from 10:01 a.m. on.

To use this method of point-in-time recovery, you should examine the log to be sure of the exact times to specify for the commands. To display the log file contents without executing them, use this command:

```
shell> mysqlbinlog /var/log/mysql/bin.123456 > /tmp/mysql_restore.sql
```

Then open the `/tmp/mysql_restore.sql` file with a text editor to examine it.

Excluding specific changes by specifying times for `mysqlbinlog` does not work well if multiple statements executed at the same time as the one to be excluded.

7.5.2 Point-in-Time Recovery Using Event Positions

Instead of specifying dates and times, the `--start-position` and `--stop-position` options for `mysqlbinlog` can be used for specifying log positions. They work the same as the start and stop date options, except that you specify log position numbers rather than dates. Using positions may enable you to be more precise about which part of the log to recover, especially if many transactions

occurred around the same time as a damaging SQL statement. To determine the position numbers, run `mysqlbinlog` for a range of times near the time when the unwanted transaction was executed, but redirect the results to a text file for examination. This can be done like so:

```
shell> mysqlbinlog --start-datetime="2005-04-20 9:55:00" \
         --stop-datetime="2005-04-20 10:05:00" \
         /var/log/mysql/bin.123456 > /tmp/mysql_restore.sql
```

This command creates a small text file in the `/tmp` directory that contains the SQL statements around the time that the deleterious SQL statement was executed. Open this file with a text editor and look for the statement that you do not want to repeat. Determine the positions in the binary log for stopping and resuming the recovery and make note of them. Positions are labeled as `log_pos` followed by a number. After restoring the previous backup file, use the position numbers to process the binary log file. For example, you would use commands something like these:

```
shell> mysqlbinlog --stop-position=368312 /var/log/mysql/bin.123456 \
         | mysql -u root -p

shell> mysqlbinlog --start-position=368315 /var/log/mysql/bin.123456 \
         | mysql -u root -p
```

The first command recovers all the transactions up until the stop position given. The second command recovers all transactions from the starting position given until the end of the binary log. Because the output of `mysqlbinlog` includes `SET TIMESTAMP` statements before each SQL statement recorded, the recovered data and related MySQL logs reflect the original times at which the transactions were executed.

7.6 MyISAM Table Maintenance and Crash Recovery

This section discusses how to use `myisamchk` to check or repair `MyISAM` tables (tables that have `.MYD` and `.MYI` files for storing data and indexes). For general `myisamchk` background, see Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”. Other table-repair information can be found at Section 2.11.10, “Rebuilding or Repairing Tables or Indexes”.

You can use `myisamchk` to check, repair, or optimize database tables. The following sections describe how to perform these operations and how to set up a table maintenance schedule. For information about using `myisamchk` to get information about your tables, see Section 4.6.3.5, “Obtaining Table Information with myisamchk”.

Even though table repair with `myisamchk` is quite secure, it is always a good idea to make a backup *before* doing a repair or any maintenance operation that could make a lot of changes to a table.

`myisamchk` operations that affect indexes can cause `MyISAM FULLTEXT` indexes to be rebuilt with full-text parameters that are incompatible with the values used by the MySQL server. To avoid this problem, follow the guidelines in Section 4.6.3.1, “myisamchk General Options”.

`MyISAM` table maintenance can also be done using the SQL statements that perform operations similar to what `myisamchk` can do:

- To check `MyISAM` tables, use `CHECK TABLE`.
- To repair `MyISAM` tables, use `REPAIR TABLE`.
- To optimize `MyISAM` tables, use `OPTIMIZE TABLE`.
- To analyze `MyISAM` tables, use `ANALYZE TABLE`.

For additional information about these statements, see Section 13.7.2, “Table Maintenance Statements”.

These statements can be used directly or by means of the `mysqlcheck` client program. One advantage of these statements over `myisamchk` is that the server does all the work. With `myisamchk`,

you must make sure that the server does not use the tables at the same time so that there is no unwanted interaction between `myisamchk` and the server.

7.6.1 Using myisamchk for Crash Recovery

This section describes how to check for and deal with data corruption in MySQL databases. If your tables become corrupted frequently, you should try to find the reason why. See Section B.3.3.3, “What to Do If MySQL Keeps Crashing”.

For an explanation of how `MyISAM` tables can become corrupted, see Section 15.2.4, “MyISAM Table Problems”.

If you run `mysqld` with external locking disabled (which is the default), you cannot reliably use `myisamchk` to check a table when `mysqld` is using the same table. If you can be certain that no one can access the tables through `mysqld` while you run `myisamchk`, you only have to execute `mysqladmin flush-tables` before you start checking the tables. If you cannot guarantee this, you must stop `mysqld` while you check the tables. If you run `myisamchk` to check tables that `mysqld` is updating at the same time, you may get a warning that a table is corrupt even when it is not.

If the server is run with external locking enabled, you can use `myisamchk` to check tables at any time. In this case, if the server tries to update a table that `myisamchk` is using, the server waits for `myisamchk` to finish before it continues.

If you use `myisamchk` to repair or optimize tables, you *must* always ensure that the `mysqld` server is not using the table (this also applies if external locking is disabled). If you do not stop `mysqld`, you should at least do a `mysqladmin flush-tables` before you run `myisamchk`. Your tables *may become corrupted* if the server and `myisamchk` access the tables simultaneously.

When performing crash recovery, it is important to understand that each `MyISAM` table *`tbl_name`* in a database corresponds to the three files in the database directory shown in the following table.

File	Purpose
`tbl_name``.frm`	Definition (format) file
`tbl_name``.MYD`	Data file
`tbl_name``.MYI`	Index file

Each of these three file types is subject to corruption in various ways, but problems occur most often in data files and index files.

`myisamchk` works by creating a copy of the `.MYD` data file row by row. It ends the repair stage by removing the old `.MYD` file and renaming the new file to the original file name. If you use `--quick`, `myisamchk` does not create a temporary `.MYD` file, but instead assumes that the `.MYD` file is correct and generates only a new index file without touching the `.MYD` file. This is safe, because `myisamchk` automatically detects whether the `.MYD` file is corrupt and aborts the repair if it is. You can also specify the `--quick` option twice to `myisamchk`. In this case, `myisamchk` does not abort on some errors (such as duplicate-key errors) but instead tries to resolve them by modifying the `.MYD` file. Normally the use of two `--quick` options is useful only if you have too little free disk space to perform a normal repair. In this case, you should at least make a backup of the table before running `myisamchk`.

7.6.2 How to Check MyISAM Tables for Errors

To check a `MyISAM` table, use the following commands:

- `myisamchk` *`tbl_name`*

  This finds 99.99% of all errors. What it cannot find is corruption that involves *only* the data file (which is very unusual). If you want to check a table, you should normally run `myisamchk` without options or with the `-s` (silent) option.

- `myisamchk -m tbl_name`

  This finds 99.999% of all errors. It first checks all index entries for errors and then reads through all rows. It calculates a checksum for all key values in the rows and verifies that the checksum matches the checksum for the keys in the index tree.

- `myisamchk -e tbl_name`

  This does a complete and thorough check of all data (`-e` means “extended check”). It does a check-read of every key for each row to verify that they indeed point to the correct row. This may take a long time for a large table that has many indexes. Normally, `myisamchk` stops after the first error it finds. If you want to obtain more information, you can add the `-v` (verbose) option. This causes `myisamchk` to keep going, up through a maximum of 20 errors.

- `myisamchk -e -i tbl_name`

  This is like the previous command, but the `-i` option tells `myisamchk` to print additional statistical information.

In most cases, a simple `myisamchk` command with no arguments other than the table name is sufficient to check a table.

7.6.3 How to Repair MyISAM Tables

The discussion in this section describes how to use `myisamchk` on `MyISAM` tables (extensions `.MYI` and `.MYD`).

You can also use the `CHECK TABLE` and `REPAIR TABLE` statements to check and repair `MyISAM` tables. See Section 13.7.2.2, “CHECK TABLE Statement”, and Section 13.7.2.5, “REPAIR TABLE Statement”.

Symptoms of corrupted tables include queries that abort unexpectedly and observable errors such as these:

- *`tbl_name`*`.frm` is locked against change
- Can't find file *`tbl_name`*`.MYI` (Errcode: *`nnn`*)
- Unexpected end of file
- Record file is crashed
- Got error *`nnn`* from table handler

To get more information about the error, run `perror` *`nnn`*, where *`nnn`* is the error number. The following example shows how to use `perror` to find the meanings for the most common error numbers that indicate a problem with a table:

```
shell> perror 126 127 132 134 135 136 141 144 145
MySQL error code 126 = Index file is crashed
MySQL error code 127 = Record-file is crashed
MySQL error code 132 = Old database file
MySQL error code 134 = Record was already deleted (or record file crashed)
MySQL error code 135 = No more room in record file
MySQL error code 136 = No more room in index file
MySQL error code 141 = Duplicate unique key or constraint on write or update
MySQL error code 144 = Table is crashed and last repair failed
MySQL error code 145 = Table was marked as crashed and should be repaired
```

Error 135 (no more room in record file) and error 136 (no more room in index file) are not errors that can be fixed by a simple repair. In this case, you must use `ALTER TABLE` to increase the `MAX_ROWS` and `AVG_ROW_LENGTH` table option values:

```
ALTER TABLE tbl_name MAX_ROWS=xxx AVG_ROW_LENGTH=yyy;
```

If you do not know the current table option values, use `SHOW CREATE TABLE`.

For the other errors, you must repair your tables. `myisamchk` can usually detect and fix most problems that occur.

The repair process involves up to four stages, described here. Before you begin, you should change location to the database directory and check the permissions of the table files. On Unix, make sure that they are readable by the user that `mysqld` runs as (and to you, because you need to access the files you are checking). If it turns out you need to modify files, they must also be writable by you.

This section is for the cases where a table check fails (such as those described in Section 7.6.2, “How to Check MyISAM Tables for Errors”), or you want to use the extended features that `myisamchk` provides.

The `myisamchk` options used for table maintenance with are described in Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”. `myisamchk` also has variables that you can set to control memory allocation that may improve performance. See Section 4.6.3.6, “myisamchk Memory Usage”.

If you are going to repair a table from the command line, you must first stop the `mysqld` server. Note that when you do `mysqladmin shutdown` on a remote server, the `mysqld` server is still available for a while after `mysqladmin` returns, until all statement-processing has stopped and all index changes have been flushed to disk.

Stage 1: Checking your tables

Run `myisamchk *.MYI` or `myisamchk -e *.MYI` if you have more time. Use the `-s` (silent) option to suppress unnecessary information.

If the `mysqld` server is stopped, you should use the `--update-state` option to tell `myisamchk` to mark the table as “checked.”

You have to repair only those tables for which `myisamchk` announces an error. For such tables, proceed to Stage 2.

If you get unexpected errors when checking (such as `out of memory` errors), or if `myisamchk` crashes, go to Stage 3.

Stage 2: Easy safe repair

First, try `myisamchk -r -q tbl_name` (`-r -q` means “quick recovery mode”). This attempts to repair the index file without touching the data file. If the data file contains everything that it should and the delete links point at the correct locations within the data file, this should work, and the table is fixed. Start repairing the next table. Otherwise, use the following procedure:

1. Make a backup of the data file before continuing.
2. Use `myisamchk -r tbl_name` (`-r` means “recovery mode”). This removes incorrect rows and deleted rows from the data file and reconstructs the index file.
3. If the preceding step fails, use `myisamchk --safe-recover tbl_name`. Safe recovery mode uses an old recovery method that handles a few cases that regular recovery mode does not (but is slower).

Note

If you want a repair operation to go much faster, you should set the values of the `sort_buffer_size` and `key_buffer_size` variables each to about 25% of your available memory when running `myisamchk`.

If you get unexpected errors when repairing (such as `out of memory` errors), or if `myisamchk` crashes, go to Stage 3.

Stage 3: Difficult repair

You should reach this stage only if the first 16KB block in the index file is destroyed or contains incorrect information, or if the index file is missing. In this case, it is necessary to create a new index file. Do so as follows:

1. Move the data file to a safe place.
2. Use the table description file to create new (empty) data and index files:

```
shell> mysql db_name
```

```
mysql> SET autocommit=1;
mysql> TRUNCATE TABLE tbl_name;
mysql> quit
```

3. Copy the old data file back onto the newly created data file. (Do not just move the old file back onto the new file. You want to retain a copy in case something goes wrong.)

Important

If you are using replication, you should stop it prior to performing the above procedure, since it involves file system operations, and these are not logged by MySQL.

Go back to Stage 2. `myisamchk -r -q` should work. (This should not be an endless loop.)

You can also use the `REPAIR TABLE tbl_name USE_FRM` SQL statement, which performs the whole procedure automatically. There is also no possibility of unwanted interaction between a utility and the server, because the server does all the work when you use `REPAIR TABLE`. See Section 13.7.2.5, “REPAIR TABLE Statement”.

Stage 4: Very difficult repair

You should reach this stage only if the `.frm` description file has also crashed. That should never happen, because the description file is not changed after the table is created:

1. Restore the description file from a backup and go back to Stage 3. You can also restore the index file and go back to Stage 2. In the latter case, you should start with `myisamchk -r`.
2. If you do not have a backup but know exactly how the table was created, create a copy of the table in another database. Remove the new data file, and then move the `.frm` description and `.MYI` index files from the other database to your crashed database. This gives you new description and index files, but leaves the `.MYD` data file alone. Go back to Stage 2 and attempt to reconstruct the index file.

7.6.4 MyISAM Table Optimization

To coalesce fragmented rows and eliminate wasted space that results from deleting or updating rows, run `myisamchk` in recovery mode:

```
shell> myisamchk -r tbl_name
```

You can optimize a table in the same way by using the `OPTIMIZE TABLE` SQL statement. `OPTIMIZE TABLE` does a table repair and a key analysis, and also sorts the index tree so that key lookups are faster. There is also no possibility of unwanted interaction between a utility and the server, because the server does all the work when you use `OPTIMIZE TABLE`. See Section 13.7.2.4, “OPTIMIZE TABLE Statement”.

`myisamchk` has a number of other options that you can use to improve the performance of a table:

- `--analyze` or `-a`: Perform key distribution analysis. This improves join performance by enabling the join optimizer to better choose the order in which to join the tables and which indexes it should use.

- `--sort-index` or `-S`: Sort the index blocks. This optimizes seeks and makes table scans that use indexes faster.
- `--sort-records=index_num` or `-R index_num`: Sort data rows according to a given index. This makes your data much more localized and may speed up range-based `SELECT` and `ORDER BY` operations that use this index.

For a full description of all available options, see Section 4.6.3, "`myisamchk` — MyISAM Table-Maintenance Utility".

7.6.5 Setting Up a MyISAM Table Maintenance Schedule

It is a good idea to perform table checks on a regular basis rather than waiting for problems to occur. One way to check and repair `MyISAM` tables is with the `CHECK TABLE` and `REPAIR TABLE` statements. See Section 13.7.2, "Table Maintenance Statements".

Another way to check tables is to use `myisamchk`. For maintenance purposes, you can use `myisamchk -s`. The `-s` option (short for `--silent`) causes `myisamchk` to run in silent mode, printing messages only when errors occur.

It is also a good idea to enable automatic `MyISAM` table checking. For example, whenever the machine has done a restart in the middle of an update, you usually need to check each table that could have been affected before it is used further. (These are "expected crashed tables.") To cause the server to check `MyISAM` tables automatically, start it with the `myisam_recover_options` system variable set. See Section 5.1.7, "Server System Variables".

You should also check your tables regularly during normal system operation. For example, you can run a `cron` job to check important tables once a week, using a line like this in a `crontab` file:

```
35 0 * * 0 /path/to/myisamchk --fast --silent /path/to/datadir/*/*.MYI
```

This prints out information about crashed tables so that you can examine and repair them as necessary.

To start with, execute `myisamchk -s` each night on all tables that have been updated during the last 24 hours. As you see that problems occur infrequently, you can back off the checking frequency to once a week or so.

Normally, MySQL tables need little maintenance. If you are performing many updates to `MyISAM` tables with dynamic-sized rows (tables with `VARCHAR`, `BLOB`, or `TEXT` columns) or have tables with many deleted rows you may want to defragment/reclaim space from the tables from time to time. You can do this by using `OPTIMIZE TABLE` on the tables in question. Alternatively, if you can stop the `mysqld` server for a while, change location into the data directory and use this command while the server is stopped:

```
shell> myisamchk -r -s --sort-index --myisam_sort_buffer_size=16M */*.MYI
```

Chapter 8 Optimization

Table of Contents

8.1 Optimization Overview 962
8.2 Optimizing SQL Statements 964
8.2.1 Optimizing SELECT Statements 964
8.2.2 Optimizing Subqueries and Derived Tables 1001
8.2.3 Optimizing INFORMATION_SCHEMA Queries 1010
8.2.4 Optimizing Data Change Statements 1015
8.2.5 Optimizing Database Privileges 1016
8.2.6 Other Optimization Tips 1016
8.3 Optimization and Indexes 1017
8.3.1 How MySQL Uses Indexes 1017
8.3.2 Primary Key Optimization 1018
8.3.3 Foreign Key Optimization 1019
8.3.4 Column Indexes 1019
8.3.5 Multiple-Column Indexes 1020
8.3.6 Verifying Index Usage 1022
8.3.7 InnoDB and MyISAM Index Statistics Collection 1022
8.3.8 Comparison of B-Tree and Hash Indexes 1023
8.3.9 Use of Index Extensions 1024
8.3.10 Indexed Lookups from TIMESTAMP Columns 1027
8.4 Optimizing Database Structure 1029
8.4.1 Optimizing Data Size 1029
8.4.2 Optimizing MySQL Data Types 1031
8.4.3 Optimizing for Many Tables 1032
8.4.4 Internal Temporary Table Use in MySQL 1034
8.4.5 Limits on Number of Databases and Tables 1035
8.4.6 Limits on Table Size 1035
8.4.7 Limits on Table Column Count and Row Size 1036
8.5 Optimizing for InnoDB Tables 1039
8.5.1 Optimizing Storage Layout for InnoDB Tables 1039
8.5.2 Optimizing InnoDB Transaction Management 1040
8.5.3 Optimizing InnoDB Read-Only Transactions 1041
8.5.4 Optimizing InnoDB Redo Logging 1042
8.5.5 Bulk Data Loading for InnoDB Tables 1042
8.5.6 Optimizing InnoDB Queries 1043
8.5.7 Optimizing InnoDB DDL Operations 1044
8.5.8 Optimizing InnoDB Disk I/O 1044
8.5.9 Optimizing InnoDB Configuration Variables 1047
8.5.10 Optimizing InnoDB for Systems with Many Tables 1048
8.6 Optimizing for MyISAM Tables 1048
8.6.1 Optimizing MyISAM Queries 1049
8.6.2 Bulk Data Loading for MyISAM Tables 1050
8.6.3 Optimizing REPAIR TABLE Statements 1051
8.7 Optimizing for MEMORY Tables 1053
8.8 Understanding the Query Execution Plan 1053
8.8.1 Optimizing Queries with EXPLAIN 1053
8.8.2 EXPLAIN Output Format 1054
8.8.3 Extended EXPLAIN Output Format 1066
8.8.4 Estimating Query Performance 1068
8.9 Controlling the Query Optimizer 1069
8.9.1 Controlling Query Plan Evaluation 1069
8.9.2 Switchable Optimizations 1069
8.9.3 Index Hints 1072

8.10 Buffering and Caching 1074
8.10.1 InnoDB Buffer Pool Optimization 1075
8.10.2 The MyISAM Key Cache 1075
8.10.3 The MySQL Query Cache 1079
8.10.4 Caching of Prepared Statements and Stored Programs 1086
8.11 Optimizing Locking Operations 1087
8.11.1 Internal Locking Methods 1087
8.11.2 Table Locking Issues 1090
8.11.3 Concurrent Inserts 1091
8.11.4 Metadata Locking 1092
8.11.5 External Locking 1095
8.12 Optimizing the MySQL Server 1096
8.12.1 System Factors 1096
8.12.2 Optimizing Disk I/O 1096
8.12.3 Using Symbolic Links 1098
8.12.4 Optimizing Memory Use 1101
8.13 Measuring Performance (Benchmarking) 1104
8.13.1 Measuring the Speed of Expressions and Functions 1105
8.13.2 The MySQL Benchmark Suite 1105
8.13.3 Using Your Own Benchmarks 1106
8.13.4 Measuring Performance with performance_schema 1106
8.14 Examining Server Thread (Process) Information 1106
8.14.1 Accessing the Process List 1107
8.14.2 Thread Command Values 1108
8.14.3 General Thread States 1110
8.14.4 Delayed-Insert Thread States 1116
8.14.5 Query Cache Thread States 1117
8.14.6 Replication Source Thread States 1118
8.14.7 Replication Replica I/O Thread States 1118
8.14.8 Replication Replica SQL Thread States 1119
8.14.9 Replication Replica Connection Thread States 1120
8.14.10 NDB Cluster Thread States 1120
8.14.11 Event Scheduler Thread States 1121

This chapter explains how to optimize MySQL performance and provides examples. Optimization involves configuring, tuning, and measuring performance, at several levels. Depending on your job role (developer, DBA, or a combination of both), you might optimize at the level of individual SQL statements, entire applications, a single database server, or multiple networked database servers. Sometimes you can be proactive and plan in advance for performance, while other times you might troubleshoot a configuration or code issue after a problem occurs. Optimizing CPU and memory usage can also improve scalability, allowing the database to handle more load without slowing down.

8.1 Optimization Overview

Database performance depends on several factors at the database level, such as tables, queries, and configuration settings. These software constructs result in CPU and I/O operations at the hardware level, which you must minimize and make as efficient as possible. As you work on database performance, you start by learning the high-level rules and guidelines for the software side, and measuring performance using wall-clock time. As you become an expert, you learn more about what happens internally, and start measuring things such as CPU cycles and I/O operations.

Typical users aim to get the best database performance out of their existing software and hardware configurations. Advanced users look for opportunities to improve the MySQL software itself, or develop their own storage engines and hardware appliances to expand the MySQL ecosystem.

- Optimizing at the Database Level
- Optimizing at the Hardware Level

- Balancing Portability and Performance

Optimizing at the Database Level

The most important factor in making a database application fast is its basic design:

- Are the tables structured properly? In particular, do the columns have the right data types, and does each table have the appropriate columns for the type of work? For example, applications that perform frequent updates often have many tables with few columns, while applications that analyze large amounts of data often have few tables with many columns.
- Are the right indexes in place to make queries efficient?
- Are you using the appropriate storage engine for each table, and taking advantage of the strengths and features of each storage engine you use? In particular, the choice of a transactional storage engine such as `InnoDB` or a nontransactional one such as `MyISAM` can be very important for performance and scalability.

Note

`InnoDB` is the default storage engine for new tables. In practice, the advanced `InnoDB` performance features mean that `InnoDB` tables often outperform the simpler `MyISAM` tables, especially for a busy database.

- Does each table use an appropriate row format? This choice also depends on the storage engine used for the table. In particular, compressed tables use less disk space and so require less disk I/O to read and write the data. Compression is available for all kinds of workloads with `InnoDB` tables, and for read-only `MyISAM` tables.
- Does the application use an appropriate locking strategy? For example, by allowing shared access when possible so that database operations can run concurrently, and requesting exclusive access when appropriate so that critical operations get top priority. Again, the choice of storage engine is significant. The `InnoDB` storage engine handles most locking issues without involvement from you, allowing for better concurrency in the database and reducing the amount of experimentation and tuning for your code.
- Are all memory areas used for caching sized correctly? That is, large enough to hold frequently accessed data, but not so large that they overload physical memory and cause paging. The main memory areas to configure are the `InnoDB` buffer pool, the `MyISAM` key cache, and the MySQL query cache.

Optimizing at the Hardware Level

Any database application eventually hits hardware limits as the database becomes more and more busy. A DBA must evaluate whether it is possible to tune the application or reconfigure the server to avoid these bottlenecks, or whether more hardware resources are required. System bottlenecks typically arise from these sources:

- Disk seeks. It takes time for the disk to find a piece of data. With modern disks, the mean time for this is usually lower than 10ms, so we can in theory do about 100 seeks a second. This time improves slowly with new disks and is very hard to optimize for a single table. The way to optimize seek time is to distribute the data onto more than one disk.
- Disk reading and writing. When the disk is at the correct position, we need to read or write the data. With modern disks, one disk delivers at least 10–20MB/s throughput. This is easier to optimize than seeks because you can read in parallel from multiple disks.
- CPU cycles. When the data is in main memory, we must process it to get our result. Having large tables compared to the amount of memory is the most common limiting factor. But with small tables, speed is usually not the problem.

- Memory bandwidth. When the CPU needs more data than can fit in the CPU cache, main memory bandwidth becomes a bottleneck. This is an uncommon bottleneck for most systems, but one to be aware of.

Balancing Portability and Performance

To use performance-oriented SQL extensions in a portable MySQL program, you can wrap MySQL-specific keywords in a statement within `/*! */` comment delimiters. Other SQL servers ignore the commented keywords. For information about writing comments, see Section 9.6, “Comments”.

8.2 Optimizing SQL Statements

The core logic of a database application is performed through SQL statements, whether issued directly through an interpreter or submitted behind the scenes through an API. The tuning guidelines in this section help to speed up all kinds of MySQL applications. The guidelines cover SQL operations that read and write data, the behind-the-scenes overhead for SQL operations in general, and operations used in specific scenarios such as database monitoring.

8.2.1 Optimizing SELECT Statements

Queries, in the form of `SELECT` statements, perform all the lookup operations in the database. Tuning these statements is a top priority, whether to achieve sub-second response times for dynamic web pages, or to chop hours off the time to generate huge overnight reports.

Besides `SELECT` statements, the tuning techniques for queries also apply to constructs such as `CREATE TABLE...AS SELECT`, `INSERT INTO...SELECT`, and `WHERE` clauses in `DELETE` statements. Those statements have additional performance considerations because they combine write operations with the read-oriented query operations.

NDB Cluster supports a join pushdown optimization whereby a qualifying join is sent in its entirety to NDB Cluster data nodes, where it can be distributed among them and executed in parallel. For more information about this optimization, see Conditions for NDB pushdown joins.

The main considerations for optimizing queries are:

- To make a slow `SELECT ... WHERE` query faster, the first thing to check is whether you can add an index. Set up indexes on columns used in the `WHERE` clause, to speed up evaluation, filtering, and the final retrieval of results. To avoid wasted disk space, construct a small set of indexes that speed up many related queries used in your application.

  Indexes are especially important for queries that reference different tables, using features such as joins and foreign keys. You can use the `EXPLAIN` statement to determine which indexes are used for a `SELECT`. See Section 8.3.1, “How MySQL Uses Indexes” and Section 8.8.1, “Optimizing Queries with EXPLAIN”.

- Isolate and tune any part of the query, such as a function call, that takes excessive time. Depending on how the query is structured, a function could be called once for every row in the result set, or even once for every row in the table, greatly magnifying any inefficiency.

- Minimize the number of full table scans in your queries, particularly for big tables.

- Keep table statistics up to date by using the `ANALYZE TABLE` statement periodically, so the optimizer has the information needed to construct an efficient execution plan.

- Learn the tuning techniques, indexing techniques, and configuration parameters that are specific to the storage engine for each table. Both `InnoDB` and `MyISAM` have sets of guidelines for enabling and sustaining high performance in queries. For details, see Section 8.5.6, “Optimizing InnoDB Queries” and Section 8.6.1, “Optimizing MyISAM Queries”.

- You can optimize single-query transactions for `InnoDB` tables, using the technique in Section 8.5.3, “Optimizing InnoDB Read-Only Transactions”.

- Avoid transforming the query in ways that make it hard to understand, especially if the optimizer does some of the same transformations automatically.
- If a performance issue is not easily solved by one of the basic guidelines, investigate the internal details of the specific query by reading the `EXPLAIN` plan and adjusting your indexes, `WHERE` clauses, join clauses, and so on. (When you reach a certain level of expertise, reading the `EXPLAIN` plan might be your first step for every query.)
- Adjust the size and properties of the memory areas that MySQL uses for caching. With efficient use of the `InnoDB` buffer pool, `MyISAM` key cache, and the MySQL query cache, repeated queries run faster because the results are retrieved from memory the second and subsequent times.
- Even for a query that runs fast using the cache memory areas, you might still optimize further so that they require less cache memory, making your application more scalable. Scalability means that your application can handle more simultaneous users, larger requests, and so on without experiencing a big drop in performance.
- Deal with locking issues, where the speed of your query might be affected by other sessions accessing the tables at the same time.

8.2.1.1 WHERE Clause Optimization

This section discusses optimizations that can be made for processing `WHERE` clauses. The examples use `SELECT` statements, but the same optimizations apply for `WHERE` clauses in `DELETE` and `UPDATE` statements.

Note

Because work on the MySQL optimizer is ongoing, not all of the optimizations that MySQL performs are documented here.

You might be tempted to rewrite your queries to make arithmetic operations faster, while sacrificing readability. Because MySQL does similar optimizations automatically, you can often avoid this work, and leave the query in a more understandable and maintainable form. Some of the optimizations performed by MySQL follow:

- Removal of unnecessary parentheses:

```
   ((a AND b) AND c OR (((a AND b) AND (c AND d))))
-> (a AND b AND c) OR (a AND b AND c AND d)
```

- Constant folding:

```
   (a<b AND b=c) AND a=5
-> b>5 AND b=c AND a=5
```

- Constant condition removal:

```
   (b>=5 AND b=5) OR (b=6 AND 5=5) OR (b=7 AND 5=6)
-> b=5 OR b=6
```

- Constant expressions used by indexes are evaluated only once.
- `COUNT(*)` on a single table without a `WHERE` is retrieved directly from the table information for `MyISAM` and `MEMORY` tables. This is also done for any `NOT NULL` expression when used with only one table.
- Early detection of invalid constant expressions. MySQL quickly detects that some `SELECT` statements are impossible and returns no rows.
- `HAVING` is merged with `WHERE` if you do not use `GROUP BY` or aggregate functions (`COUNT()`, `MIN()`, and so on).

- For each table in a join, a simpler `WHERE` is constructed to get a fast `WHERE` evaluation for the table and also to skip rows as soon as possible.
- All constant tables are read first before any other tables in the query. A constant table is any of the following:
  - An empty table or a table with one row.
  - A table that is used with a `WHERE` clause on a `PRIMARY KEY` or a `UNIQUE` index, where all index parts are compared to constant expressions and are defined as `NOT NULL`.

  All of the following tables are used as constant tables:

```
SELECT * FROM t WHERE primary_key=1;
SELECT * FROM t1,t2
  WHERE t1.primary_key=1 AND t2.primary_key=t1.id;
```

- The best join combination for joining the tables is found by trying all possibilities. If all columns in `ORDER BY` and `GROUP BY` clauses come from the same table, that table is preferred first when joining.
- If there is an `ORDER BY` clause and a different `GROUP BY` clause, or if the `ORDER BY` or `GROUP BY` contains columns from tables other than the first table in the join queue, a temporary table is created.
- If you use the `SQL_SMALL_RESULT` modifier, MySQL uses an in-memory temporary table.
- Each table index is queried, and the best index is used unless the optimizer believes that it is more efficient to use a table scan. At one time, a scan was used based on whether the best index spanned more than 30% of the table, but a fixed percentage no longer determines the choice between using an index or a scan. The optimizer now is more complex and bases its estimate on additional factors such as table size, number of rows, and I/O block size.
- In some cases, MySQL can read rows from the index without even consulting the data file. If all columns used from the index are numeric, only the index tree is used to resolve the query.
- Before each row is output, those that do not match the `HAVING` clause are skipped.

Some examples of queries that are very fast:

```
SELECT COUNT(*) FROM tbl_name;

SELECT MIN(key_part1),MAX(key_part1) FROM tbl_name;

SELECT MAX(key_part2) FROM tbl_name
  WHERE key_part1=constant;

SELECT ... FROM tbl_name
  ORDER BY key_part1,key_part2,... LIMIT 10;

SELECT ... FROM tbl_name
  ORDER BY key_part1 DESC, key_part2 DESC, ... LIMIT 10;
```

MySQL resolves the following queries using only the index tree, assuming that the indexed columns are numeric:

```
SELECT key_part1,key_part2 FROM tbl_name WHERE key_part1=val;

SELECT COUNT(*) FROM tbl_name
  WHERE key_part1=val1 AND key_part2=val2;

SELECT key_part2 FROM tbl_name GROUP BY key_part1;
```

The following queries use indexing to retrieve the rows in sorted order without a separate sorting pass:

```
SELECT ... FROM tbl_name
  ORDER BY key_part1,key_part2,... ;
```

```
SELECT ... FROM tbl_name
  ORDER BY key_part1 DESC, key_part2 DESC, ... ;
```

8.2.1.2 Range Optimization

The `range` access method uses a single index to retrieve a subset of table rows that are contained within one or several index value intervals. It can be used for a single-part or multiple-part index. The following sections give a detailed description of how intervals are extracted from the `WHERE` clause.

- Range Access Method for Single-Part Indexes
- Range Access Method for Multiple-Part Indexes
- Equality Range Optimization of Many-Valued Comparisons

Range Access Method for Single-Part Indexes

For a single-part index, index value intervals can be conveniently represented by corresponding conditions in the `WHERE` clause, denoted as *range conditions* rather than “intervals.”

The definition of a range condition for a single-part index is as follows:

- For both `BTREE` and `HASH` indexes, comparison of a key part with a constant value is a range condition when using the `=`, `<=>`, `IN()`, `IS NULL`, or `IS NOT NULL` operators.
- Additionally, for `BTREE` indexes, comparison of a key part with a constant value is a range condition when using the `>`, `<`, `>=`, `<=`, `BETWEEN`, `!=`, or `<>` operators, or `LIKE` comparisons if the argument to `LIKE` is a constant string that does not start with a wildcard character.
- For all index types, multiple range conditions combined with `OR` or `AND` form a range condition.

“Constant value” in the preceding descriptions means one of the following:

- A constant from the query string
- A column of a `const` or `system` table from the same join
- The result of an uncorrelated subquery
- Any expression composed entirely from subexpressions of the preceding types

Here are some examples of queries with range conditions in the `WHERE` clause:

```
SELECT * FROM t1
  WHERE key_col > 1
  AND key_col < 10;

SELECT * FROM t1
  WHERE key_col = 1
  OR key_col IN (15,18,20);

SELECT * FROM t1
  WHERE key_col LIKE 'ab%'
  OR key_col BETWEEN 'bar' AND 'foo';
```

Some nonconstant values may be converted to constants during the optimizer constant propagation phase.

MySQL tries to extract range conditions from the `WHERE` clause for each of the possible indexes. During the extraction process, conditions that cannot be used for constructing the range condition are dropped, conditions that produce overlapping ranges are combined, and conditions that produce empty ranges are removed.

Consider the following statement, where `key1` is an indexed column and `nonkey` is not indexed:

```
SELECT * FROM t1 WHERE
  (key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
  (key1 < 'bar' AND nonkey = 4) OR
  (key1 < 'uux' AND key1 > 'z');
```

The extraction process for key `key1` is as follows:

1. Start with original `WHERE` clause:

   ```
   (key1 < 'abc' AND (key1 LIKE 'abcde%' OR key1 LIKE '%b')) OR
   (key1 < 'bar' AND nonkey = 4) OR
   (key1 < 'uux' AND key1 > 'z')
   ```

2. Remove `nonkey = 4` and `key1 LIKE '%b'` because they cannot be used for a range scan. The correct way to remove them is to replace them with `TRUE`, so that we do not miss any matching rows when doing the range scan. Replacing them with `TRUE` yields:

   ```
   (key1 < 'abc' AND (key1 LIKE 'abcde%' OR TRUE)) OR
   (key1 < 'bar' AND TRUE) OR
   (key1 < 'uux' AND key1 > 'z')
   ```

3. Collapse conditions that are always true or false:

   - `(key1 LIKE 'abcde%' OR TRUE)` is always true
   - `(key1 < 'uux' AND key1 > 'z')` is always false

   Replacing these conditions with constants yields:

   ```
   (key1 < 'abc' AND TRUE) OR (key1 < 'bar' AND TRUE) OR (FALSE)
   ```

   Removing unnecessary `TRUE` and `FALSE` constants yields:

   ```
   (key1 < 'abc') OR (key1 < 'bar')
   ```

4. Combining overlapping intervals into one yields the final condition to be used for the range scan:

   ```
   (key1 < 'bar')
   ```

In general (and as demonstrated by the preceding example), the condition used for a range scan is less restrictive than the `WHERE` clause. MySQL performs an additional check to filter out rows that satisfy the range condition but not the full `WHERE` clause.

The range condition extraction algorithm can handle nested `AND`/`OR` constructs of arbitrary depth, and its output does not depend on the order in which conditions appear in `WHERE` clause.

MySQL does not support merging multiple ranges for the `range` access method for spatial indexes. To work around this limitation, you can use a `UNION` with identical `SELECT` statements, except that you put each spatial predicate in a different `SELECT`.

Range Access Method for Multiple-Part Indexes

Range conditions on a multiple-part index are an extension of range conditions for a single-part index. A range condition on a multiple-part index restricts index rows to lie within one or several key tuple intervals. Key tuple intervals are defined over a set of key tuples, using ordering from the index.

For example, consider a multiple-part index defined as `key1(key_part1, key_part2, key_part3)`, and the following set of key tuples listed in key order:

```
key_part1  key_part2  key_part3
  NULL       1          'abc'
  NULL       1          'xyz'
  NULL       2          'foo'
   1         1          'abc'
   1         1          'xyz'
   1         2          'abc'
```

```
  2         1          'aaa'
```

The condition `key_part1 = 1` defines this interval:

```
(1,-inf,-inf) <= (key_part1,key_part2,key_part3) < (1,+inf,+inf)
```

The interval covers the 4th, 5th, and 6th tuples in the preceding data set and can be used by the range access method.

By contrast, the condition `key_part3 = 'abc'` does not define a single interval and cannot be used by the range access method.

The following descriptions indicate how range conditions work for multiple-part indexes in greater detail.

- For `HASH` indexes, each interval containing identical values can be used. This means that the interval can be produced only for conditions in the following form:

  ```
     key_part1 cmp const1
  AND key_part2 cmp const2
  AND ...
  AND key_partN cmp constN;
  ```

  Here, `const1`, `const2`, ... are constants, `cmp` is one of the `=`, `<=>`, or `IS NULL` comparison operators, and the conditions cover all index parts. (That is, there are `N` conditions, one for each part of an `N`-part index.) For example, the following is a range condition for a three-part `HASH` index:

  ```
  key_part1 = 1 AND key_part2 IS NULL AND key_part3 = 'foo'
  ```

  For the definition of what is considered to be a constant, see Range Access Method for Single-Part Indexes.

- For a `BTREE` index, an interval might be usable for conditions combined with `AND`, where each condition compares a key part with a constant value using `=`, `<=>`, `IS NULL`, `>`, `<`, `>=`, `<=`, `!=`, `<>`, `BETWEEN`, or `LIKE 'pattern'` (where `'pattern'` does not start with a wildcard). An interval can be used as long as it is possible to determine a single key tuple containing all rows that match the condition (or two intervals if `<>` or `!=` is used).

  The optimizer attempts to use additional key parts to determine the interval as long as the comparison operator is `=`, `<=>`, or `IS NULL`. If the operator is `>`, `<`, `>=`, `<=`, `!=`, `<>`, `BETWEEN`, or `LIKE`, the optimizer uses it but considers no more key parts. For the following expression, the optimizer uses `=` from the first comparison. It also uses `>=` from the second comparison but considers no further key parts and does not use the third comparison for interval construction:

  ```
  key_part1 = 'foo' AND key_part2 >= 10 AND key_part3 > 10
  ```

  The single interval is:

  ```
  ('foo',10,-inf) < (key_part1,key_part2,key_part3) < ('foo',+inf,+inf)
  ```

  It is possible that the created interval contains more rows than the initial condition. For example, the preceding interval includes the value `('foo', 11, 0)`, which does not satisfy the original condition.

- If conditions that cover sets of rows contained within intervals are combined with `OR`, they form a condition that covers a set of rows contained within the union of their intervals. If the conditions are combined with `AND`, they form a condition that covers a set of rows contained within the intersection of their intervals. For example, for this condition on a two-part index:

  ```
  (key_part1 = 1 AND key_part2 < 2) OR (key_part1 > 5)
  ```

  The intervals are:

  ```
  (1,-inf) < (key_part1,key_part2) < (1,2)
  ```

```
(5,-inf) < (key_part1,key_part2)
```

In this example, the interval on the first line uses one key part for the left bound and two key parts for the right bound. The interval on the second line uses only one key part. The `key_len` column in the `EXPLAIN` output indicates the maximum length of the key prefix used.

In some cases, `key_len` may indicate that a key part was used, but that might be not what you would expect. Suppose that *`key_part1`* and *`key_part2`* can be `NULL`. Then the `key_len` column displays two key part lengths for the following condition:

```
key_part1 >= 1 AND key_part2 < 2
```

But, in fact, the condition is converted to this:

```
key_part1 >= 1 AND key_part2 IS NOT NULL
```

For a description of how optimizations are performed to combine or eliminate intervals for range conditions on a single-part index, see Range Access Method for Single-Part Indexes. Analogous steps are performed for range conditions on multiple-part indexes.

Equality Range Optimization of Many-Valued Comparisons

Consider these expressions, where *`col_name`* is an indexed column:

```
col_name IN(val1, ..., valN)
col_name = val1 OR ... OR col_name = valN
```

Each expression is true if *`col_name`* is equal to any of several values. These comparisons are equality range comparisons (where the “range” is a single value). The optimizer estimates the cost of reading qualifying rows for equality range comparisons as follows:

- If there is a unique index on *`col_name`*, the row estimate for each range is 1 because at most one row can have the given value.
- Otherwise, any index on *`col_name`* is nonunique and the optimizer can estimate the row count for each range using dives into the index or index statistics.

With index dives, the optimizer makes a dive at each end of a range and uses the number of rows in the range as the estimate. For example, the expression *`col_name`* `IN (10, 20, 30)` has three equality ranges and the optimizer makes two dives per range to generate a row estimate. Each pair of dives yields an estimate of the number of rows that have the given value.

Index dives provide accurate row estimates, but as the number of comparison values in the expression increases, the optimizer takes longer to generate a row estimate. Use of index statistics is less accurate than index dives but permits faster row estimation for large value lists.

The `eq_range_index_dive_limit` system variable enables you to configure the number of values at which the optimizer switches from one row estimation strategy to the other. To permit use of index dives for comparisons of up to *`N`* equality ranges, set `eq_range_index_dive_limit` to *`N`* + 1. To disable use of statistics and always use index dives regardless of *`N`*, set `eq_range_index_dive_limit` to 0.

To update table index statistics for best estimates, use `ANALYZE TABLE`.

Even under conditions when index dives would otherwise be used, they are skipped for queries that satisfy all these conditions:

- A single-index `FORCE INDEX` index hint is present. The idea is that if index use is forced, there is nothing to be gained from the additional overhead of performing dives into the index.
- The index is nonunique and not a `FULLTEXT` index.
- No subquery is present.

- No `DISTINCT`, `GROUP BY`, or `ORDER BY` clause is present.

Those dive-skipping conditions apply only for single-table queries. Index dives are not skipped for multiple-table queries (joins).

8.2.1.3 Index Merge Optimization

The *Index Merge* access method retrieves rows with multiple `range` scans and merges their results into one. This access method merges index scans from a single table only, not scans across multiple tables. The merge can produce unions, intersections, or unions-of-intersections of its underlying scans.

Example queries for which Index Merge may be used:

```
SELECT * FROM tbl_name WHERE key1 = 10 OR key2 = 20;

SELECT * FROM tbl_name
  WHERE (key1 = 10 OR key2 = 20) AND non_key = 30;

SELECT * FROM t1, t2
  WHERE (t1.key1 IN (1,2) OR t1.key2 LIKE 'value%')
  AND t2.key1 = t1.some_col;

SELECT * FROM t1, t2
  WHERE t1.key1 = 1
  AND (t2.key1 = t1.some_col OR t2.key2 = t1.some_col2);
```

Note

The Index Merge optimization algorithm has the following known limitations:

- If your query has a complex `WHERE` clause with deep `AND`/`OR` nesting and MySQL does not choose the optimal plan, try distributing terms using the following identity transformations:

  ```
  (x AND y) OR z => (x OR z) AND (y OR z)
  (x OR y) AND z => (x AND z) OR (y AND z)
  ```

- Index Merge is not applicable to full-text indexes.

In `EXPLAIN` output, the Index Merge method appears as `index_merge` in the `type` column. In this case, the `key` column contains a list of indexes used, and `key_len` contains a list of the longest key parts for those indexes.

The Index Merge access method has several algorithms, which are displayed in the `Extra` field of `EXPLAIN` output:

- `Using intersect(...)`
- `Using union(...)`
- `Using sort_union(...)`

The following sections describe these algorithms in greater detail. The optimizer chooses between different possible Index Merge algorithms and other access methods based on cost estimates of the various available options.

Use of Index Merge is subject to the value of the `index_merge`, `index_merge_intersection`, `index_merge_union`, and `index_merge_sort_union` flags of the `optimizer_switch` system variable. See Section 8.9.2, “Switchable Optimizations”. By default, all those flags are `on`. To enable only certain algorithms, set `index_merge` to `off`, and enable only such of the others as should be permitted.

- Index Merge Intersection Access Algorithm
- Index Merge Union Access Algorithm

- Index Merge Sort-Union Access Algorithm

Index Merge Intersection Access Algorithm

This access algorithm is applicable when a `WHERE` clause is converted to several range conditions on different keys combined with `AND`, and each condition is one of the following:

- An *`N`*-part expression of this form, where the index has exactly *`N`* parts (that is, all index parts are covered):

  ```
  key_part1 = const1 AND key_part2 = const2 ... AND key_partN = constN
  ```

- Any range condition over the primary key of an `InnoDB` table.

Examples:

```
SELECT * FROM innodb_table
  WHERE primary_key < 10 AND key_col1 = 20;

SELECT * FROM tbl_name
  WHERE key1_part1 = 1 AND key1_part2 = 2 AND key2 = 2;
```

The Index Merge intersection algorithm performs simultaneous scans on all used indexes and produces the intersection of row sequences that it receives from the merged index scans.

If all columns used in the query are covered by the used indexes, full table rows are not retrieved (`EXPLAIN` output contains `Using index` in `Extra` field in this case). Here is an example of such a query:

```
SELECT COUNT(*) FROM t1 WHERE key1 = 1 AND key2 = 1;
```

If the used indexes do not cover all columns used in the query, full rows are retrieved only when the range conditions for all used keys are satisfied.

If one of the merged conditions is a condition over the primary key of an `InnoDB` table, it is not used for row retrieval, but is used to filter out rows retrieved using other conditions.

Index Merge Union Access Algorithm

The criteria for this algorithm are similar to those for the Index Merge intersection algorithm. The algorithm is applicable when the table's `WHERE` clause is converted to several range conditions on different keys combined with `OR`, and each condition is one of the following:

- An *`N`*-part expression of this form, where the index has exactly *`N`* parts (that is, all index parts are covered):

  ```
  key_part1 = const1 AND key_part2 = const2 ... AND key_partN = constN
  ```

- Any range condition over a primary key of an `InnoDB` table.

- A condition for which the Index Merge intersection algorithm is applicable.

Examples:

```
SELECT * FROM t1
  WHERE key1 = 1 OR key2 = 2 OR key3 = 3;

SELECT * FROM innodb_table
  WHERE (key1 = 1 AND key2 = 2)
     OR (key3 = 'foo' AND key4 = 'bar') AND key5 = 5;
```

Index Merge Sort-Union Access Algorithm

This access algorithm is applicable when the `WHERE` clause is converted to several range conditions combined by `OR`, but the Index Merge union algorithm is not applicable.

Examples:

```
SELECT * FROM tbl_name
  WHERE key_col1 < 10 OR key_col2 < 20;

SELECT * FROM tbl_name
  WHERE (key_col1 > 10 OR key_col2 = 20) AND nonkey_col = 30;
```

The difference between the sort-union algorithm and the union algorithm is that the sort-union algorithm must first fetch row IDs for all rows and sort them before returning any rows.

8.2.1.4 Engine Condition Pushdown Optimization

This optimization improves the efficiency of direct comparisons between a nonindexed column and a constant. In such cases, the condition is “pushed down” to the storage engine for evaluation. This optimization can be used only by the `NDB` storage engine.

For NDB Cluster, this optimization can eliminate the need to send nonmatching rows over the network between the cluster's data nodes and the MySQL server that issued the query, and can speed up queries where it is used by a factor of 5 to 10 times over cases where condition pushdown could be but is not used.

Suppose that an NDB Cluster table is defined as follows:

```
CREATE TABLE t1 (
    a INT,
    b INT,
    KEY(a)
) ENGINE=NDB;
```

Engine condition pushdown can be used with queries such as the one shown here, which includes a comparison between a nonindexed column and a constant:

```
SELECT a, b FROM t1 WHERE b = 10;
```

The use of engine condition pushdown can be seen in the output of `EXPLAIN`:

```
mysql> EXPLAIN SELECT a,b FROM t1 WHERE b = 10\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 10
        Extra: Using where with pushed condition
```

However, engine condition pushdown *cannot* be used with either of these two queries:

```
SELECT a,b FROM t1 WHERE a = 10;
SELECT a,b FROM t1 WHERE b + 1 = 10;
```

Engine condition pushdown is not applicable to the first query because an index exists on column `a`. (An index access method would be more efficient and so would be chosen in preference to condition pushdown.) Engine condition pushdown cannot be employed for the second query because the comparison involving the nonindexed column `b` is indirect. (However, engine condition pushdown could be applied if you were to reduce `b + 1 = 10` to `b = 9` in the `WHERE` clause.)

Engine condition pushdown may also be employed when an indexed column is compared with a constant using a `>` or `<` operator:

```
mysql> EXPLAIN SELECT a, b FROM t1 WHERE a < 2\G
*************************** 1. row ***************************
```

```
           id: 1
  select_type: SIMPLE
        table: t1
         type: range
possible_keys: a
          key: a
      key_len: 5
          ref: NULL
         rows: 2
        Extra: Using where with pushed condition
```

Other supported comparisons for engine condition pushdown include the following:

- *`column`* `[NOT] LIKE` *`pattern`*

  `pattern` must be a string literal containing the pattern to be matched; for syntax, see Section 12.8.1, “String Comparison Functions and Operators”.

- *`column`* `IS [NOT] NULL`

- *`column`* `IN (`*`value_list`*`)`

  Each item in the *`value_list`* must be a constant, literal value.

- *`column`* `BETWEEN` *`constant1`* `AND` *`constant2`*

  `constant1` and *`constant2`* must each be a constant, literal value.

In all of the cases in the preceding list, it is possible for the condition to be converted into the form of one or more direct comparisons between a column and a constant.

Engine condition pushdown is enabled by default. To disable it at server startup, set the `optimizer_switch` system variable's `engine_condition_pushdown` flag to `off`. For example, in a `my.cnf` file, use these lines:

```
[mysqld]
optimizer_switch=engine_condition_pushdown=off
```

At runtime, disable condition pushdown like this:

```
SET optimizer_switch='engine_condition_pushdown=off';
```

Limitations. Engine condition pushdown is subject to the following limitations:

- Engine condition pushdown is supported only by the `NDB` storage engine.
- Columns may be compared with constants only; however, this includes expressions which evaluate to constant values.
- Columns used in comparisons cannot be of any of the `BLOB` or `TEXT` types. This exclusion extends to `BIT` and `ENUM` columns as well.
- A string value to be compared with a column must use the same collation as the column.
- Joins are not directly supported; conditions involving multiple tables are pushed separately where possible. Use extended `EXPLAIN` output to determine which conditions are actually pushed down. See Section 8.8.3, “Extended EXPLAIN Output Format”.

8.2.1.5 Index Condition Pushdown Optimization

Index Condition Pushdown (ICP) is an optimization for the case where MySQL retrieves rows from a table using an index. Without ICP, the storage engine traverses the index to locate rows in the base table and returns them to the MySQL server which evaluates the `WHERE` condition for the rows. With ICP enabled, and if parts of the `WHERE` condition can be evaluated by using only columns from the

index, the MySQL server pushes this part of the `WHERE` condition down to the storage engine. The storage engine then evaluates the pushed index condition by using the index entry and only if this is satisfied is the row read from the table. ICP can reduce the number of times the storage engine must access the base table and the number of times the MySQL server must access the storage engine.

Applicability of the Index Condition Pushdown optimization is subject to these conditions:

- ICP is used for the `range`, `ref`, `eq_ref`, and `ref_or_null` access methods when there is a need to access full table rows.
- ICP can be used for `InnoDB` and `MyISAM` tables. (Exception: ICP is not supported with partitioned tables in MySQL 5.6; this issue is resolved in MySQL 5.7.)
- For `InnoDB` tables, ICP is used only for secondary indexes. The goal of ICP is to reduce the number of full-row reads and thereby reduce I/O operations. For `InnoDB` clustered indexes, the complete record is already read into the `InnoDB` buffer. Using ICP in this case does not reduce I/O.
- Conditions that refer to subqueries cannot be pushed down.
- Conditions that refer to stored functions cannot be pushed down. Storage engines cannot invoke stored functions.
- Triggered conditions cannot be pushed down. (For information about triggered conditions, see Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”.)

To understand how this optimization works, first consider how an index scan proceeds when Index Condition Pushdown is not used:

1. Get the next row, first by reading the index tuple, and then by using the index tuple to locate and read the full table row.
2. Test the part of the `WHERE` condition that applies to this table. Accept or reject the row based on the test result.

Using Index Condition Pushdown, the scan proceeds like this instead:

1. Get the next row's index tuple (but not the full table row).
2. Test the part of the `WHERE` condition that applies to this table and can be checked using only index columns. If the condition is not satisfied, proceed to the index tuple for the next row.
3. If the condition is satisfied, use the index tuple to locate and read the full table row.
4. Test the remaining part of the `WHERE` condition that applies to this table. Accept or reject the row based on the test result.

`EXPLAIN` output shows `Using index condition` in the `Extra` column when Index Condition Pushdown is used. It does not show `Using index` because that does not apply when full table rows must be read.

Suppose that a table contains information about people and their addresses and that the table has an index defined as `INDEX (zipcode, lastname, firstname)`. If we know a person's `zipcode` value but are not sure about the last name, we can search like this:

```
SELECT * FROM people
  WHERE zipcode='95054'
  AND lastname LIKE '%etrunia%'
  AND address LIKE '%Main Street%';
```

MySQL can use the index to scan through people with `zipcode='95054'`. The second part (`lastname LIKE '%etrunia%'`) cannot be used to limit the number of rows that must be scanned, so without Index Condition Pushdown, this query must retrieve full table rows for all people who have `zipcode='95054'`.

With Index Condition Pushdown, MySQL checks the `lastname LIKE '%etrunia%'` part before reading the full table row. This avoids reading full rows corresponding to index tuples that match the `zipcode` condition but not the `lastname` condition.

Index Condition Pushdown is enabled by default. It can be controlled with the `optimizer_switch` system variable by setting the `index_condition_pushdown` flag:

```
SET optimizer_switch = 'index_condition_pushdown=off';
SET optimizer_switch = 'index_condition_pushdown=on';
```

See Section 8.9.2, “Switchable Optimizations”.

8.2.1.6 Nested-Loop Join Algorithms

MySQL executes joins between tables using a nested-loop algorithm or variations on it.

- Nested-Loop Join Algorithm
- Block Nested-Loop Join Algorithm

Nested-Loop Join Algorithm

A simple nested-loop join (NLJ) algorithm reads rows from the first table in a loop one at a time, passing each row to a nested loop that processes the next table in the join. This process is repeated as many times as there remain tables to be joined.

Assume that a join between three tables `t1`, `t2`, and `t3` is to be executed using the following join types:

```
Table   Join Type
t1      range
t2      ref
t3      ALL
```

If a simple NLJ algorithm is used, the join is processed like this:

```
for each row in t1 matching range {
  for each row in t2 matching reference key {
    for each row in t3 {
      if row satisfies join conditions, send to client
    }
  }
}
```

Because the NLJ algorithm passes rows one at a time from outer loops to inner loops, it typically reads tables processed in the inner loops many times.

Block Nested-Loop Join Algorithm

A Block Nested-Loop (BNL) join algorithm uses buffering of rows read in outer loops to reduce the number of times that tables in inner loops must be read. For example, if 10 rows are read into a buffer and the buffer is passed to the next inner loop, each row read in the inner loop can be compared against all 10 rows in the buffer. This reduces by an order of magnitude the number of times the inner table must be read.

MySQL join buffering has these characteristics:

- Join buffering can be used when the join is of type `ALL` or `index` (in other words, when no possible keys can be used, and a full scan is done, of either the data or index rows, respectively), or `range`. Use of buffering is also applicable to outer joins, as described in Section 8.2.1.11, “Block Nested-Loop and Batched Key Access Joins”.
- A join buffer is never allocated for the first nonconstant table, even if it would be of type `ALL` or `index`.

- Only columns of interest to a join are stored in its join buffer, not whole rows.
- The `join_buffer_size` system variable determines the size of each join buffer used to process a query.
- One buffer is allocated for each join that can be buffered, so a given query might be processed using multiple join buffers.
- A join buffer is allocated prior to executing the join and freed after the query is done.

For the example join described previously for the NLJ algorithm (without buffering), the join is done as follows using join buffering:

```
for each row in t1 matching range {
  for each row in t2 matching reference key {
    store used columns from t1, t2 in join buffer
    if buffer is full {
      for each row in t3 {
        for each t1, t2 combination in join buffer {
          if row satisfies join conditions, send to client
        }
      }
      empty join buffer
    }
  }
}

if buffer is not empty {
  for each row in t3 {
    for each t1, t2 combination in join buffer {
      if row satisfies join conditions, send to client
    }
  }
}
```

If *`S`* is the size of each stored `t1`, `t2` combination in the join buffer and *`C`* is the number of combinations in the buffer, the number of times table `t3` is scanned is:

```
(S * C)/join_buffer_size + 1
```

The number of `t3` scans decreases as the value of `join_buffer_size` increases, up to the point when `join_buffer_size` is large enough to hold all previous row combinations. At that point, no speed is gained by making it larger.

8.2.1.7 Nested Join Optimization

The syntax for expressing joins permits nested joins. The following discussion refers to the join syntax described in Section 13.2.9.2, “JOIN Clause”.

The syntax of *`table_factor`* is extended in comparison with the SQL Standard. The latter accepts only *`table_reference`*, not a list of them inside a pair of parentheses. This is a conservative extension if we consider each comma in a list of *`table_reference`* items as equivalent to an inner join. For example:

```
SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
                 ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

Is equivalent to:

```
SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)
                 ON (t2.a=t1.a AND t3.b=t1.b AND t4.c=t1.c)
```

In MySQL, `CROSS JOIN` is syntactically equivalent to `INNER JOIN`; they can replace each other. In standard SQL, they are not equivalent. `INNER JOIN` is used with an `ON` clause; `CROSS JOIN` is used otherwise.

In general, parentheses can be ignored in join expressions containing only inner join operations. Consider this join expression:

```
t1 LEFT JOIN (t2 LEFT JOIN t3 ON t2.b=t3.b OR t2.b IS NULL)
   ON t1.a=t2.a
```

After removing parentheses and grouping operations to the left, that join expression transforms into this expression:

```
(t1 LEFT JOIN t2 ON t1.a=t2.a) LEFT JOIN t3
    ON t2.b=t3.b OR t2.b IS NULL
```

Yet, the two expressions are not equivalent. To see this, suppose that the tables `t1`, `t2`, and `t3` have the following state:

- Table `t1` contains rows `(1)`, `(2)`
- Table `t2` contains row `(1,101)`
- Table `t3` contains row `(101)`

In this case, the first expression returns a result set including the rows `(1,1,101,101)`, `(2,NULL,NULL,NULL)`, whereas the second expression returns the rows `(1,1,101,101)`, `(2,NULL,NULL,101)`:

```
mysql> SELECT *
       FROM t1
            LEFT JOIN
            (t2 LEFT JOIN t3 ON t2.b=t3.b OR t2.b IS NULL)
            ON t1.a=t2.a;
+------+------+------+------+
| a    | a    | b    | b    |
+------+------+------+------+
|    1 |    1 |  101 |  101 |
|    2 | NULL | NULL | NULL |
+------+------+------+------+

mysql> SELECT *
       FROM (t1 LEFT JOIN t2 ON t1.a=t2.a)
            LEFT JOIN t3
            ON t2.b=t3.b OR t2.b IS NULL;
+------+------+------+------+
| a    | a    | b    | b    |
+------+------+------+------+
|    1 |    1 |  101 |  101 |
|    2 | NULL | NULL |  101 |
+------+------+------+------+
```

In the following example, an outer join operation is used together with an inner join operation:

```
t1 LEFT JOIN (t2, t3) ON t1.a=t2.a
```

That expression cannot be transformed into the following expression:

```
t1 LEFT JOIN t2 ON t1.a=t2.a, t3
```

For the given table states, the two expressions return different sets of rows:

```
mysql> SELECT *
       FROM t1 LEFT JOIN (t2, t3) ON t1.a=t2.a;
+------+------+------+------+
| a    | a    | b    | b    |
+------+------+------+------+
|    1 |    1 |  101 |  101 |
|    2 | NULL | NULL | NULL |
+------+------+------+------+

mysql> SELECT *
```

```
       FROM t1 LEFT JOIN t2 ON t1.a=t2.a, t3;
+------+------+------+------+
| a    | a    | b    | b    |
+------+------+------+------+
|    1 |    1 |  101 |  101 |
|    2 | NULL | NULL |  101 |
+------+------+------+------+
```

Therefore, if we omit parentheses in a join expression with outer join operators, we might change the result set for the original expression.

More exactly, we cannot ignore parentheses in the right operand of the left outer join operation and in the left operand of a right join operation. In other words, we cannot ignore parentheses for the inner table expressions of outer join operations. Parentheses for the other operand (operand for the outer table) can be ignored.

The following expression:

```
(t1,t2) LEFT JOIN t3 ON P(t2.b,t3.b)
```

Is equivalent to this expression for any tables `t1,t2,t3` and any condition `P` over attributes `t2.b` and `t3.b`:

```
t1, t2 LEFT JOIN t3 ON P(t2.b,t3.b)
```

Whenever the order of execution of join operations in a join expression (*`joined_table`*) is not from left to right, we talk about nested joins. Consider the following queries:

```
SELECT * FROM t1 LEFT JOIN (t2 LEFT JOIN t3 ON t2.b=t3.b) ON t1.a=t2.a
  WHERE t1.a > 1

SELECT * FROM t1 LEFT JOIN (t2, t3) ON t1.a=t2.a
  WHERE (t2.b=t3.b OR t2.b IS NULL) AND t1.a > 1
```

Those queries are considered to contain these nested joins:

```
t2 LEFT JOIN t3 ON t2.b=t3.b
t2, t3
```

In the first query, the nested join is formed with a left join operation. In the second query, it is formed with an inner join operation.

In the first query, the parentheses can be omitted: The grammatical structure of the join expression dictates the same order of execution for join operations. For the second query, the parentheses cannot be omitted, although the join expression here can be interpreted unambiguously without them. In our extended syntax, the parentheses in `(t2, t3)` of the second query are required, although theoretically the query could be parsed without them: We still would have unambiguous syntactical structure for the query because `LEFT JOIN` and `ON` play the role of the left and right delimiters for the expression `(t2,t3)`.

The preceding examples demonstrate these points:

- For join expressions involving only inner joins (and not outer joins), parentheses can be removed and joins evaluated left to right. In fact, tables can be evaluated in any order.
- The same is not true, in general, for outer joins or for outer joins mixed with inner joins. Removal of parentheses may change the result.

Queries with nested outer joins are executed in the same pipeline manner as queries with inner joins. More exactly, a variation of the nested-loop join algorithm is exploited. Recall the algorithm by which the nested-loop join executes a query (see Section 8.2.1.6, “Nested-Loop Join Algorithms”). Suppose that a join query over 3 tables `T1,T2,T3` has this form:

```
SELECT * FROM T1 INNER JOIN T2 ON P1(T1,T2)
```

```
                INNER JOIN T3 ON P2(T2,T3)
  WHERE P(T1,T2,T3)
```

Here, `P1(T1,T2)` and `P2(T3,T3)` are some join conditions (on expressions), whereas `P(T1,T2,T3)` is a condition over columns of tables `T1,T2,T3`.

The nested-loop join algorithm would execute this query in the following manner:

```
FOR each row t1 in T1 {
  FOR each row t2 in T2 such that P1(t1,t2) {
    FOR each row t3 in T3 such that P2(t2,t3) {
      IF P(t1,t2,t3) {
         t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

The notation `t1||t2||t3` indicates a row constructed by concatenating the columns of rows `t1`, `t2`, and `t3`. In some of the following examples, `NULL` where a table name appears means a row in which `NULL` is used for each column of that table. For example, `t1||t2||NULL` indicates a row constructed by concatenating the columns of rows `t1` and `t2`, and `NULL` for each column of `t3`. Such a row is said to be `NULL`-complemented.

Now consider a query with nested outer joins:

```
SELECT * FROM T1 LEFT JOIN
             (T2 LEFT JOIN T3 ON P2(T2,T3))
             ON P1(T1,T2)
  WHERE P(T1,T2,T3)
```

For this query, modify the nested-loop pattern to obtain:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t2 in T2 such that P1(t1,t2) {
    BOOL f2:=FALSE;
    FOR each row t3 in T3 such that P2(t2,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f2=TRUE;
      f1=TRUE;
    }
    IF (!f2) {
      IF P(t1,t2,NULL) {
        t:=t1||t2||NULL; OUTPUT t;
      }
      f1=TRUE;
    }
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

In general, for any nested loop for the first inner table in an outer join operation, a flag is introduced that is turned off before the loop and is checked after the loop. The flag is turned on when for the current row from the outer table a match from the table representing the inner operand is found. If at the end of the loop cycle the flag is still off, no match has been found for the current row of the outer table. In this case, the row is complemented by `NULL` values for the columns of the inner tables. The result row is passed to the final check for the output or into the next nested loop, but only if the row satisfies the join condition of all embedded outer joins.

In the example, the outer join table expressed by the following expression is embedded:

```
(T2 LEFT JOIN T3 ON P2(T2,T3))
```

For the query with inner joins, the optimizer could choose a different order of nested loops, such as this one:

```
FOR each row t3 in T3 {
  FOR each row t2 in T2 such that P2(t2,t3) {
    FOR each row t1 in T1 such that P1(t1,t2) {
      IF P(t1,t2,t3) {
         t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

For queries with outer joins, the optimizer can choose only such an order where loops for outer tables precede loops for inner tables. Thus, for our query with outer joins, only one nesting order is possible. For the following query, the optimizer evaluates two different nestings. In both nestings, `T1` must be processed in the outer loop because it is used in an outer join. `T2` and `T3` are used in an inner join, so that join must be processed in the inner loop. However, because the join is an inner join, `T2` and `T3` can be processed in either order.

```
SELECT * T1 LEFT JOIN (T2,T3) ON P1(T1,T2) AND P2(T1,T3)
  WHERE P(T1,T2,T3)
```

One nesting evaluates `T2`, then `T3`:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t2 in T2 such that P1(t1,t2) {
    FOR each row t3 in T3 such that P2(t1,t3) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f1:=TRUE
    }
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

The other nesting evaluates `T3`, then `T2`:

```
FOR each row t1 in T1 {
  BOOL f1:=FALSE;
  FOR each row t3 in T3 such that P2(t1,t3) {
    FOR each row t2 in T2 such that P1(t1,t2) {
      IF P(t1,t2,t3) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f1:=TRUE
    }
  }
  IF (!f1) {
    IF P(t1,NULL,NULL) {
      t:=t1||NULL||NULL; OUTPUT t;
    }
  }
}
```

When discussing the nested-loop algorithm for inner joins, we omitted some details whose impact on the performance of query execution may be huge. We did not mention so-called “pushed-down” conditions. Suppose that our `WHERE` condition `P(T1,T2,T3)` can be represented by a conjunctive formula:

```
P(T1,T2,T2) = C1(T1) AND C2(T2) AND C3(T3).
```

In this case, MySQL actually uses the following nested-loop algorithm for the execution of the query with inner joins:

```
FOR each row t1 in T1 such that C1(t1) {
  FOR each row t2 in T2 such that P1(t1,t2) AND C2(t2)  {
    FOR each row t3 in T3 such that P2(t2,t3) AND C3(t3) {
      IF P(t1,t2,t3) {
         t:=t1||t2||t3; OUTPUT t;
      }
    }
  }
}
```

You see that each of the conjuncts `C1(T1)`, `C2(T2)`, `C3(T3)` are pushed out of the most inner loop to the most outer loop where it can be evaluated. If `C1(T1)` is a very restrictive condition, this condition pushdown may greatly reduce the number of rows from table `T1` passed to the inner loops. As a result, the execution time for the query may improve immensely.

For a query with outer joins, the `WHERE` condition is to be checked only after it has been found that the current row from the outer table has a match in the inner tables. Thus, the optimization of pushing conditions out of the inner nested loops cannot be applied directly to queries with outer joins. Here we must introduce conditional pushed-down predicates guarded by the flags that are turned on when a match has been encountered.

Recall this example with outer joins:

```
P(T1,T2,T3)=C1(T1) AND C(T2) AND C3(T3)
```

For that example, the nested-loop algorithm using guarded pushed-down conditions looks like this:

```
FOR each row t1 in T1 such that C1(t1) {
  BOOL f1:=FALSE;
  FOR each row t2 in T2
      such that P1(t1,t2) AND (f1?C2(t2):TRUE) {
    BOOL f2:=FALSE;
    FOR each row t3 in T3
        such that P2(t2,t3) AND (f1&&f2?C3(t3):TRUE) {
      IF (f1&&f2?TRUE:(C2(t2) AND C3(t3))) {
        t:=t1||t2||t3; OUTPUT t;
      }
      f2=TRUE;
      f1=TRUE;
    }
    IF (!f2) {
      IF (f1?TRUE:C2(t2) && P(t1,t2,NULL)) {
        t:=t1||t2||NULL; OUTPUT t;
      }
      f1=TRUE;
    }
  }
  IF (!f1 && P(t1,NULL,NULL)) {
      t:=t1||NULL||NULL; OUTPUT t;
  }
}
```

In general, pushed-down predicates can be extracted from join conditions such as `P1(T1,T2)` and `P(T2,T3)`. In this case, a pushed-down predicate is guarded also by a flag that prevents checking the predicate for the `NULL`-complemented row generated by the corresponding outer join operation.

Access by key from one inner table to another in the same nested join is prohibited if it is induced by a predicate from the `WHERE` condition.

8.2.1.8 Outer Join Optimization

Outer joins include `LEFT JOIN` and `RIGHT JOIN`.

MySQL implements an *`A LEFT JOIN B join_specification`* as follows:

- Table *B* is set to depend on table *A* and all tables on which *A* depends.
- Table *A* is set to depend on all tables (except *B*) that are used in the `LEFT JOIN` condition.
- The `LEFT JOIN` condition is used to decide how to retrieve rows from table *B*. (In other words, any condition in the `WHERE` clause is not used.)
- All standard join optimizations are performed, with the exception that a table is always read after all tables on which it depends. If there is a circular dependency, an error occurs.
- All standard `WHERE` optimizations are performed.
- If there is a row in *A* that matches the `WHERE` clause, but there is no row in *B* that matches the `ON` condition, an extra *B* row is generated with all columns set to `NULL`.
- If you use `LEFT JOIN` to find rows that do not exist in some table and you have the following test: *`col_name`* `IS NULL` in the `WHERE` part, where *`col_name`* is a column that is declared as `NOT NULL`, MySQL stops searching for more rows (for a particular key combination) after it has found one row that matches the `LEFT JOIN` condition.

The `RIGHT JOIN` implementation is analogous to that of `LEFT JOIN` with the table roles reversed. Right joins are converted to equivalent left joins, as described in Section 8.2.1.9, “Outer Join Simplification”.

For a `LEFT JOIN`, if the `WHERE` condition is always false for the generated `NULL` row, the `LEFT JOIN` is changed to an inner join. For example, the `WHERE` clause would be false in the following query if `t2.column1` were `NULL`:

```
SELECT * FROM t1 LEFT JOIN t2 ON (column1) WHERE t2.column2=5;
```

Therefore, it is safe to convert the query to an inner join:

```
SELECT * FROM t1, t2 WHERE t2.column2=5 AND t1.column1=t2.column1;
```

Now the optimizer can use table `t2` before table `t1` if doing so would result in a better query plan. To provide a hint about the table join order, use `STRAIGHT_JOIN`; see Section 13.2.9, “SELECT Statement”. However, `STRAIGHT_JOIN` may prevent indexes from being used because it disables semijoin transformations; see Section 8.2.2.1, “Optimizing Subqueries with Semijoin Transformations”.

8.2.1.9 Outer Join Simplification

Table expressions in the `FROM` clause of a query are simplified in many cases.

At the parser stage, queries with right outer join operations are converted to equivalent queries containing only left join operations. In the general case, the conversion is performed such that this right join:

```
(T1, ...) RIGHT JOIN (T2, ...) ON P(T1, ..., T2, ...)
```

Becomes this equivalent left join:

```
(T2, ...) LEFT JOIN (T1, ...) ON P(T1, ..., T2, ...)
```

All inner join expressions of the form `T1 INNER JOIN T2 ON P(T1,T2)` are replaced by the list `T1,T2`, `P(T1,T2)` being joined as a conjunct to the `WHERE` condition (or to the join condition of the embedding join, if there is any).

When the optimizer evaluates plans for outer join operations, it takes into consideration only plans where, for each such operation, the outer tables are accessed before the inner tables. The optimizer choices are limited because only such plans enable outer joins to be executed using the nested-loop algorithm.

Consider a query of this form, where `R(T2)` greatly narrows the number of matching rows from table `T2`:

```
SELECT * T1 LEFT JOIN T2 ON P1(T1,T2)
  WHERE P(T1,T2) AND R(T2)
```

If the query is executed as written, the optimizer has no choice but to access the less-restricted table `T1` before the more-restricted table `T2`, which may produce a very inefficient execution plan.

Instead, MySQL converts the query to a query with no outer join operation if the `WHERE` condition is null-rejected. (That is, it converts the outer join to an inner join.) A condition is said to be null-rejected for an outer join operation if it evaluates to `FALSE` or `UNKNOWN` for any `NULL`-complemented row generated for the operation.

Thus, for this outer join:

```
T1 LEFT JOIN T2 ON T1.A=T2.A
```

Conditions such as these are null-rejected because they cannot be true for any `NULL`-complemented row (with `T2` columns set to `NULL`):

```
T2.B IS NOT NULL
T2.B > 3
T2.C <= T1.C
T2.B < 2 OR T2.C > 1
```

Conditions such as these are not null-rejected because they might be true for a `NULL`-complemented row:

```
T2.B IS NULL
T1.B < 3 OR T2.B IS NOT NULL
T1.B < 3 OR T2.B > 3
```

The general rules for checking whether a condition is null-rejected for an outer join operation are simple:

- It is of the form `A IS NOT NULL`, where `A` is an attribute of any of the inner tables
- It is a predicate containing a reference to an inner table that evaluates to `UNKNOWN` when one of its arguments is `NULL`
- It is a conjunction containing a null-rejected condition as a conjunct
- It is a disjunction of null-rejected conditions

A condition can be null-rejected for one outer join operation in a query and not null-rejected for another. In this query, the `WHERE` condition is null-rejected for the second outer join operation but is not null-rejected for the first one:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
                 LEFT JOIN T3 ON T3.B=T1.B
  WHERE T3.C > 0
```

If the `WHERE` condition is null-rejected for an outer join operation in a query, the outer join operation is replaced by an inner join operation.

For example, in the preceding query, the second outer join is null-rejected and can be replaced by an inner join:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
                 INNER JOIN T3 ON T3.B=T1.B
  WHERE T3.C > 0
```

For the original query, the optimizer evaluates only plans compatible with the single table-access order `T1,T2,T3`. For the rewritten query, it additionally considers the access order `T3,T1,T2`.

A conversion of one outer join operation may trigger a conversion of another. Thus, the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
                 LEFT JOIN T3 ON T3.B=T2.B
  WHERE T3.C > 0
```

Is first converted to the query:

```
SELECT * FROM T1 LEFT JOIN T2 ON T2.A=T1.A
                 INNER JOIN T3 ON T3.B=T2.B
  WHERE T3.C > 0
```

Which is equivalent to the query:

```
SELECT * FROM (T1 LEFT JOIN T2 ON T2.A=T1.A), T3
  WHERE T3.C > 0 AND T3.B=T2.B
```

The remaining outer join operation can also be replaced by an inner join because the condition `T3.B=T2.B` is null-rejected. This results in a query with no outer joins at all:

```
SELECT * FROM (T1 INNER JOIN T2 ON T2.A=T1.A), T3
  WHERE T3.C > 0 AND T3.B=T2.B
```

Sometimes the optimizer succeeds in replacing an embedded outer join operation, but cannot convert the embedding outer join. The following query:

```
SELECT * FROM T1 LEFT JOIN
              (T2 LEFT JOIN T3 ON T3.B=T2.B)
              ON T2.A=T1.A
  WHERE T3.C > 0
```

Is converted to:

```
SELECT * FROM T1 LEFT JOIN
              (T2 INNER JOIN T3 ON T3.B=T2.B)
              ON T2.A=T1.A
  WHERE T3.C > 0
```

That can be rewritten only to the form still containing the embedding outer join operation:

```
SELECT * FROM T1 LEFT JOIN
              (T2,T3)
              ON (T2.A=T1.A AND T3.B=T2.B)
  WHERE T3.C > 0
```

Any attempt to convert an embedded outer join operation in a query must take into account the join condition for the embedding outer join together with the `WHERE` condition. In this query, the `WHERE` condition is not null-rejected for the embedded outer join, but the join condition of the embedding outer join `T2.A=T1.A AND T3.C=T1.C` is null-rejected:

```
SELECT * FROM T1 LEFT JOIN
              (T2 LEFT JOIN T3 ON T3.B=T2.B)
              ON T2.A=T1.A AND T3.C=T1.C
  WHERE T3.D > 0 OR T1.D > 0
```

Consequently, the query can be converted to:

```
SELECT * FROM T1 LEFT JOIN
              (T2, T3)
              ON T2.A=T1.A AND T3.C=T1.C AND T3.B=T2.B
  WHERE T3.D > 0 OR T1.D > 0
```

8.2.1.10 Multi-Range Read Optimization

Reading rows using a range scan on a secondary index can result in many random disk accesses to the base table when the table is large and not stored in the storage engine's cache. With the Disk-

Sweep Multi-Range Read (MRR) optimization, MySQL tries to reduce the number of random disk access for range scans by first scanning the index only and collecting the keys for the relevant rows. Then the keys are sorted and finally the rows are retrieved from the base table using the order of the primary key. The motivation for Disk-sweep MRR is to reduce the number of random disk accesses and instead achieve a more sequential scan of the base table data.

The Multi-Range Read optimization provides these benefits:

- MRR enables data rows to be accessed sequentially rather than in random order, based on index tuples. The server obtains a set of index tuples that satisfy the query conditions, sorts them according to data row ID order, and uses the sorted tuples to retrieve data rows in order. This makes data access more efficient and less expensive.
- MRR enables batch processing of requests for key access for operations that require access to data rows through index tuples, such as range index scans and equi-joins that use an index for the join attribute. MRR iterates over a sequence of index ranges to obtain qualifying index tuples. As these results accumulate, they are used to access the corresponding data rows. It is not necessary to acquire all index tuples before starting to read data rows.

The following scenarios illustrate when MRR optimization can be advantageous:

Scenario A: MRR can be used for `InnoDB` and `MyISAM` tables for index range scans and equi-join operations.

1. A portion of the index tuples are accumulated in a buffer.
2. The tuples in the buffer are sorted by their data row ID.
3. Data rows are accessed according to the sorted index tuple sequence.

Scenario B: MRR can be used for `NDB` tables for multiple-range index scans or when performing an equi-join by an attribute.

1. A portion of ranges, possibly single-key ranges, is accumulated in a buffer on the central node where the query is submitted.
2. The ranges are sent to the execution nodes that access data rows.
3. The accessed rows are packed into packages and sent back to the central node.
4. The received packages with data rows are placed in a buffer.
5. Data rows are read from the buffer.

When MRR is used, the `Extra` column in `EXPLAIN` output shows `Using MRR`.

`InnoDB` and `MyISAM` do not use MRR if full table rows need not be accessed to produce the query result. This is the case if results can be produced entirely on the basis on information in the index tuples (through a covering index); MRR provides no benefit.

Two `optimizer_switch` system variable flags provide an interface to the use of MRR optimization. The `mrr` flag controls whether MRR is enabled. If `mrr` is enabled (`on`), the `mrr_cost_based` flag controls whether the optimizer attempts to make a cost-based choice between using and not using MRR (`on`) or uses MRR whenever possible (`off`). By default, `mrr` is `on` and `mrr_cost_based` is `on`. See Section 8.9.2, “Switchable Optimizations”.

For MRR, a storage engine uses the value of the `read_rnd_buffer_size` system variable as a guideline for how much memory it can allocate for its buffer. The engine uses up to `read_rnd_buffer_size` bytes and determines the number of ranges to process in a single pass.

8.2.1.11 Block Nested-Loop and Batched Key Access Joins

In MySQL, a Batched Key Access (BKA) Join algorithm is available that uses both index access to the joined table and a join buffer. The BKA algorithm supports inner join, outer join, and semijoin operations, including nested outer joins. Benefits of BKA include improved join performance due to more efficient table scanning. Also, the Block Nested-Loop (BNL) Join algorithm previously used only for inner joins is extended and can be employed for outer join and semijoin operations, including nested outer joins.

The following sections discuss the join buffer management that underlies the extension of the original BNL algorithm, the extended BNL algorithm, and the BKA algorithm. For information about semijoin strategies, see Section 8.2.2.1, “Optimizing Subqueries with Semijoin Transformations”

- Join Buffer Management for Block Nested-Loop and Batched Key Access Algorithms
- Block Nested-Loop Algorithm for Outer Joins and Semijoins
- Batched Key Access Joins

Join Buffer Management for Block Nested-Loop and Batched Key Access Algorithms

MySQL can employ join buffers to execute not only inner joins without index access to the inner table, but also outer joins and semijoins that appear after subquery flattening. Moreover, a join buffer can be effectively used when there is an index access to the inner table.

The join buffer management code slightly more efficiently utilizes join buffer space when storing the values of the interesting row columns: No additional bytes are allocated in buffers for a row column if its value is `NULL`, and the minimum number of bytes is allocated for any value of the `VARCHAR` type.

The code supports two types of buffers, regular and incremental. Suppose that join buffer `B1` is employed to join tables `t1` and `t2` and the result of this operation is joined with table `t3` using join buffer `B2`:

- A regular join buffer contains columns from each join operand. If `B2` is a regular join buffer, each row *`r`* put into `B2` is composed of the columns of a row *`r1`* from `B1` and the interesting columns of a matching row *`r2`* from table `t3`.
- An incremental join buffer contains only columns from rows of the table produced by the second join operand. That is, it is incremental to a row from the first operand buffer. If `B2` is an incremental join buffer, it contains the interesting columns of the row *`r2`* together with a link to the row *`r1`* from `B1`.

Incremental join buffers are always incremental relative to a join buffer from an earlier join operation, so the buffer from the first join operation is always a regular buffer. In the example just given, the buffer `B1` used to join tables `t1` and `t2` must be a regular buffer.

Each row of the incremental buffer used for a join operation contains only the interesting columns of a row from the table to be joined. These columns are augmented with a reference to the interesting columns of the matched row from the table produced by the first join operand. Several rows in the incremental buffer can refer to the same row *`r`* whose columns are stored in the previous join buffers insofar as all these rows match row *`r`*.

Incremental buffers enable less frequent copying of columns from buffers used for previous join operations. This provides a savings in buffer space because in the general case a row produced by the first join operand can be matched by several rows produced by the second join operand. It is unnecessary to make several copies of a row from the first operand. Incremental buffers also provide a savings in processing time due to the reduction in copying time.

The `block_nested_loop` and `batched_key_access` flags of the `optimizer_switch` system variable control how the optimizer uses the Block Nested-Loop and Batched Key Access join algorithms. By default, `block_nested_loop` is `on` and `batched_key_access` is `off`. See Section 8.9.2, “Switchable Optimizations”.

For information about semijoin strategies, see Section 8.2.2.1, “Optimizing Subqueries with Semijoin Transformations”

Block Nested-Loop Algorithm for Outer Joins and Semijoins

The original implementation of the MySQL BNL algorithm is extended to support outer join and semijoin operations.

When these operations are executed with a join buffer, each row put into the buffer is supplied with a match flag.

If an outer join operation is executed using a join buffer, each row of the table produced by the second operand is checked for a match against each row in the join buffer. When a match is found, a new extended row is formed (the original row plus columns from the second operand) and sent for further extensions by the remaining join operations. In addition, the match flag of the matched row in the buffer is enabled. After all rows of the table to be joined have been examined, the join buffer is scanned. Each row from the buffer that does not have its match flag enabled is extended by `NULL` complements (`NULL` values for each column in the second operand) and sent for further extensions by the remaining join operations.

The `block_nested_loop` flag of the `optimizer_switch` system variable controls how the optimizer uses the Block Nested-Loop algorithm. By default, `block_nested_loop` is `on`. See Section 8.9.2, “Switchable Optimizations”.

In `EXPLAIN` output, use of BNL for a table is signified when the `Extra` value contains `Using join buffer (Block Nested Loop)` and the `type` value is `ALL`, `index`, or `range`.

Some cases involving the combination of one or more subqueries with one or more left joins, particularly those returning many rows, may use BNL even though it is not ideal in such instances. This is a known issue which is fixed in MySQL 8.0. If upgrading MySQL is not immediately feasible for you, you may wish to disable BNL in the meantime by setting `optimizer_switch='block_nested_loop=off'`, to let the optimizer choose a better plan, using one or more index hints (see Section 8.9.3, “Index Hints”), or both, to improve the performance of such queries.

For information about semijoin strategies, see Section 8.2.2.1, “Optimizing Subqueries with Semijoin Transformations”

Batched Key Access Joins

MySQL implements a method of joining tables called the Batched Key Access (BKA) join algorithm. BKA can be applied when there is an index access to the table produced by the second join operand. Like the BNL join algorithm, the BKA join algorithm employs a join buffer to accumulate the interesting columns of the rows produced by the first operand of the join operation. Then the BKA algorithm builds keys to access the table to be joined for all rows in the buffer and submits these keys in a batch to the database engine for index lookups. The keys are submitted to the engine through the Multi-Range Read (MRR) interface (see Section 8.2.1.10, “Multi-Range Read Optimization”). After submission of the keys, the MRR engine functions perform lookups in the index in an optimal way, fetching the rows of the joined table found by these keys, and starts feeding the BKA join algorithm with matching rows. Each matching row is coupled with a reference to a row in the join buffer.

When BKA is used, the value of `join_buffer_size` defines how large the batch of keys is in each request to the storage engine. The larger the buffer, the more sequential access is made to the right hand table of a join operation, which can significantly improve performance.

For BKA to be used, the `batched_key_access` flag of the `optimizer_switch` system variable must be set to `on`. BKA uses MRR, so the `mrr` flag must also be `on`. Currently, the cost estimation for MRR is too pessimistic. Hence, it is also necessary for `mrr_cost_based` to be `off` for BKA to be used. The following setting enables BKA:

```
mysql> SET optimizer_switch='mrr=on,mrr_cost_based=off,batched_key_access=on';
```

There are two scenarios by which MRR functions execute:

- The first scenario is used for conventional disk-based storage engines such as `InnoDB` and `MyISAM`. For these engines, usually the keys for all rows from the join buffer are submitted to the MRR interface at once. Engine-specific MRR functions perform index lookups for the submitted keys, get row IDs (or primary keys) from them, and then fetch rows for all these selected row IDs one by one by request from BKA algorithm. Every row is returned with an association reference that enables access to the matched row in the join buffer. The rows are fetched by the MRR functions in an optimal way: They are fetched in the row ID (primary key) order. This improves performance because reads are in disk order rather than random order.

- The second scenario is used for remote storage engines such as `NDB`. A package of keys for a portion of rows from the join buffer, together with their associations, is sent by a MySQL Server (SQL node) to NDB Cluster data nodes. In return, the SQL node receives a package (or several packages) of matching rows coupled with corresponding associations. The BKA join algorithm takes these rows and builds new joined rows. Then a new set of keys is sent to the data nodes and the rows from the returned packages are used to build new joined rows. The process continues until the last keys from the join buffer are sent to the data nodes, and the SQL node has received and joined all rows matching these keys. This improves performance because fewer key-bearing packages sent by the SQL node to the data nodes means fewer round trips between it and the data nodes to perform the join operation.

With the first scenario, a portion of the join buffer is reserved to store row IDs (primary keys) selected by index lookups and passed as a parameter to the MRR functions.

There is no special buffer to store keys built for rows from the join buffer. Instead, a function that builds the key for the next row in the buffer is passed as a parameter to the MRR functions.

In `EXPLAIN` output, use of BKA for a table is signified when the `Extra` value contains `Using join buffer (Batched Key Access)` and the `type` value is `ref` or `eq_ref`.

8.2.1.12 IS NULL Optimization

MySQL can perform the same optimization on *`col_name`* `IS NULL` that it can use for *`col_name`* `=` *`constant_value`*. For example, MySQL can use indexes and ranges to search for `NULL` with `IS NULL`.

Examples:

```
SELECT * FROM tbl_name WHERE key_col IS NULL;

SELECT * FROM tbl_name WHERE key_col <=> NULL;

SELECT * FROM tbl_name
  WHERE key_col=const1 OR key_col=const2 OR key_col IS NULL;
```

If a `WHERE` clause includes a *`col_name`* `IS NULL` condition for a column that is declared as `NOT NULL`, that expression is optimized away. This optimization does not occur in cases when the column might produce `NULL` anyway (for example, if it comes from a table on the right side of a `LEFT JOIN`).

MySQL can also optimize the combination *`col_name = expr`* `OR` *`col_name`* `IS NULL`, a form that is common in resolved subqueries. `EXPLAIN` shows `ref_or_null` when this optimization is used.

This optimization can handle one `IS NULL` for any key part.

Some examples of queries that are optimized, assuming that there is an index on columns `a` and `b` of table `t2`:

```
SELECT * FROM t1 WHERE t1.a=expr OR t1.a IS NULL;

SELECT * FROM t1, t2 WHERE t1.a=t2.a OR t2.a IS NULL;
```

```
SELECT * FROM t1, t2
  WHERE (t1.a=t2.a OR t2.a IS NULL) AND t2.b=t1.b;

SELECT * FROM t1, t2
  WHERE t1.a=t2.a AND (t2.b=t1.b OR t2.b IS NULL);

SELECT * FROM t1, t2
  WHERE (t1.a=t2.a AND t2.a IS NULL AND ...)
  OR (t1.a=t2.a AND t2.a IS NULL AND ...);
```

`ref_or_null` works by first doing a read on the reference key, and then a separate search for rows with a `NULL` key value.

The optimization can handle only one `IS NULL` level. In the following query, MySQL uses key lookups only on the expression `(t1.a=t2.a AND t2.a IS NULL)` and is not able to use the key part on `b`:

```
SELECT * FROM t1, t2
  WHERE (t1.a=t2.a AND t2.a IS NULL)
  OR (t1.b=t2.b AND t2.b IS NULL);
```

8.2.1.13 ORDER BY Optimization

This section describes when MySQL can use an index to satisfy an `ORDER BY` clause, the `filesort` operation used when an index cannot be used, and execution plan information available from the optimizer about `ORDER BY`.

An `ORDER BY` with and without `LIMIT` may return rows in different orders, as discussed in Section 8.2.1.16, “LIMIT Query Optimization”.

- Use of Indexes to Satisfy ORDER BY
- Use of filesort to Satisfy ORDER BY
- Influencing ORDER BY Optimization
- ORDER BY Execution Plan Information Available

Use of Indexes to Satisfy ORDER BY

In some cases, MySQL may use an index to satisfy an `ORDER BY` clause and avoid the extra sorting involved in performing a `filesort` operation.

The index may also be used even if the `ORDER BY` does not match the index exactly, as long as all unused portions of the index and all extra `ORDER BY` columns are constants in the `WHERE` clause. If the index does not contain all columns accessed by the query, the index is used only if index access is cheaper than other access methods.

Assuming that there is an index on `(key_part1, key_part2)`, the following queries may use the index to resolve the `ORDER BY` part. Whether the optimizer actually does so depends on whether reading the index is more efficient than a table scan if columns not in the index must also be read.

- In this query, the index on `(key_part1, key_part2)` enables the optimizer to avoid sorting:

  ```
  SELECT * FROM t1
    ORDER BY key_part1, key_part2;
  ```

  However, the query uses `SELECT *`, which may select more columns than `key_part1` and `key_part2`. In that case, scanning an entire index and looking up table rows to find columns not in the index may be more expensive than scanning the table and sorting the results. If so, the optimizer is not likelt to use the index. If `SELECT *` selects only the index columns, the index is used and sorting avoided.

  If `t1` is an `InnoDB` table, the table primary key is implicitly part of the index, and the index can be used to resolve the `ORDER BY` for this query:

```
SELECT pk, key_part1, key_part2 FROM t1
  ORDER BY key_part1, key_part2;
```

- In this query, *key_part1* is constant, so all rows accessed through the index are in *key_part2* order, and an index on (*key_part1*, *key_part2*) avoids sorting if the WHERE clause is selective enough to make an index range scan cheaper than a table scan:

```
SELECT * FROM t1
  WHERE key_part1 = constant
  ORDER BY key_part2;
```

- In the next two queries, whether the index is used is similar to the same queries without DESC shown previously:

```
SELECT * FROM t1
  ORDER BY key_part1 DESC, key_part2 DESC;

SELECT * FROM t1
  WHERE key_part1 = constant
  ORDER BY key_part2 DESC;
```

- In the next two queries, *key_part1* is compared to a constant. The index is used if the WHERE clause is selective enough to make an index range scan cheaper than a table scan:

```
SELECT * FROM t1
  WHERE key_part1 > constant
  ORDER BY key_part1 ASC;

SELECT * FROM t1
  WHERE key_part1 < constant
  ORDER BY key_part1 DESC;
```

- In the next query, the ORDER BY does not name *key_part1*, but all rows selected have a constant *key_part1* value, so the index can still be used:

```
SELECT * FROM t1
  WHERE key_part1 = constant1 AND key_part2 > constant2
  ORDER BY key_part2;
```

In some cases, MySQL *cannot* use indexes to resolve the ORDER BY, although it may still use indexes to find the rows that match the WHERE clause. Examples:

- The query uses ORDER BY on different indexes:

```
SELECT * FROM t1 ORDER BY key1, key2;
```

- The query uses ORDER BY on nonconsecutive parts of an index:

```
SELECT * FROM t1 WHERE key2=constant ORDER BY key1_part1, key1_part3;
```

- The query mixes ASC and DESC:

```
SELECT * FROM t1 ORDER BY key_part1 DESC, key_part2 ASC;
```

- The index used to fetch the rows differs from the one used in the ORDER BY:

```
SELECT * FROM t1 WHERE key2=constant ORDER BY key1;
```

- The query uses ORDER BY with an expression that includes terms other than the index column name:

```
SELECT * FROM t1 ORDER BY ABS(key);
SELECT * FROM t1 ORDER BY -key;
```

- The query joins many tables, and the columns in the ORDER BY are not all from the first nonconstant table that is used to retrieve rows. (This is the first table in the EXPLAIN output that does not have a const join type.)

- The query has different `ORDER BY` and `GROUP BY` expressions.
- There is an index on only a prefix of a column named in the `ORDER BY` clause. In this case, the index cannot be used to fully resolve the sort order. For example, if only the first 10 bytes of a `CHAR(20)` column are indexed, the index cannot distinguish values past the 10th byte and a `filesort` is needed.
- The index does not store rows in order. For example, this is true for a `HASH` index in a `MEMORY` table.

Availability of an index for sorting may be affected by the use of column aliases. Suppose that the column `t1.a` is indexed. In this statement, the name of the column in the select list is `a`. It refers to `t1.a`, as does the reference to `a` in the `ORDER BY`, so the index on `t1.a` can be used:

```
SELECT a FROM t1 ORDER BY a;
```

In this statement, the name of the column in the select list is also `a`, but it is the alias name. It refers to `ABS(a)`, as does the reference to `a` in the `ORDER BY`, so the index on `t1.a` cannot be used:

```
SELECT ABS(a) AS a FROM t1 ORDER BY a;
```

In the following statement, the `ORDER BY` refers to a name that is not the name of a column in the select list. But there is a column in `t1` named `a`, so the `ORDER BY` refers to `t1.a` and the index on `t1.a` can be used. (The resulting sort order may be completely different from the order for `ABS(a)`, of course.)

```
SELECT ABS(a) AS b FROM t1 ORDER BY a;
```

By default, MySQL sorts `GROUP BY col1, col2, ...` queries as if you also included `ORDER BY col1, col2, ...` in the query. If you include an explicit `ORDER BY` clause that contains the same column list, MySQL optimizes it away without any speed penalty, although the sorting still occurs.

If a query includes `GROUP BY` but you want to avoid the overhead of sorting the result, you can suppress sorting by specifying `ORDER BY NULL`. For example:

```
INSERT INTO foo
SELECT a, COUNT(*) FROM bar GROUP BY a ORDER BY NULL;
```

The optimizer may still choose to use sorting to implement grouping operations. `ORDER BY NULL` suppresses sorting of the result, not prior sorting done by grouping operations to determine the result.

Note

`GROUP BY` implicitly sorts by default (that is, in the absence of `ASC` or `DESC` designators for `GROUP BY` columns), but relying on implicit `GROUP BY` sorting is deprecated. To produce a given sort order, use explicit `ASC` or `DESC` designators for `GROUP BY` columns or provide an `ORDER BY` clause. `GROUP BY` sorting is a MySQL extension that may change in a future release; for example, to make it possible for the optimizer to order groupings in whatever manner it deems most efficient and to avoid the sorting overhead.

Use of filesort to Satisfy ORDER BY

If an index cannot be used to satisfy an `ORDER BY` clause, MySQL performs a `filesort` operation that reads table rows and sorts them. A `filesort` constitutes an extra sorting phase in query execution.

To obtain memory for `filesort` operations, the optimizer allocates a fixed amount of `sort_buffer_size` bytes up front. Individual sessions can change the session value of this variable as desired to avoid excessive memory use, or to allocate more memory as necessary.

A `filesort` operation uses temporary disk files as necessary if the result set is too large to fit in memory. Some types of queries are particularly suited to completely in-memory `filesort` operations.

For example, the optimizer can use `filesort` to efficiently handle in memory, without temporary files, the `ORDER BY` operation for queries (and subqueries) of the following form:

```
SELECT ... FROM single_table ... ORDER BY non_index_column [DESC] LIMIT [M,]N;
```

Such queries are common in web applications that display only a few rows from a larger result set. Examples:

```
SELECT col1, ... FROM t1 ... ORDER BY name LIMIT 10;
SELECT col1, ... FROM t1 ... ORDER BY RAND() LIMIT 15;
```

Influencing ORDER BY Optimization

For slow `ORDER BY` queries for which `filesort` is not used, try lowering the `max_length_for_sort_data` system variable to a value that is appropriate to trigger a `filesort`. (A symptom of setting the value of this variable too high is a combination of high disk activity and low CPU activity.)

To increase `ORDER BY` speed, check whether you can get MySQL to use indexes rather than an extra sorting phase. If this is not possible, try the following strategies:

- Increase the `sort_buffer_size` variable value. Ideally, the value should be large enough for the entire result set to fit in the sort buffer (to avoid writes to disk and merge passes), but at minimum the value must be large enough to accommodate 15 tuples. (Up to 15 temporary disk files are merged and there must be room in memory for at least one tuple per file.)

  Take into account that the size of column values stored in the sort buffer is affected by the `max_sort_length` system variable value. For example, if tuples store values of long string columns and you increase the value of `max_sort_length`, the size of sort buffer tuples increases as well and may require you to increase `sort_buffer_size`. For column values calculated as a result of string expressions (such as those that invoke a string-valued function), the `filesort` algorithm cannot tell the maximum length of expression values, so it must allocate `max_sort_length` bytes for each tuple.

  To monitor the number of merge passes (to merge temporary files), check the `Sort_merge_passes` status variable.

- Increase the `read_rnd_buffer_size` variable value so that more rows are read at a time.

- Use less RAM per row by declaring columns only as large as they need to be to hold the values stored in them. For example, `CHAR(16)` is better than `CHAR(200)` if values never exceed 16 characters.

- Change the `tmpdir` system variable to point to a dedicated file system with large amounts of free space. The variable value can list several paths that are used in round-robin fashion; you can use this feature to spread the load across several directories. Separate the paths by colon characters (`:`) on Unix and semicolon characters (`;`) on Windows. The paths should name directories in file systems located on different *physical* disks, not different partitions on the same disk.

ORDER BY Execution Plan Information Available

With `EXPLAIN` (see Section 8.8.1, “Optimizing Queries with EXPLAIN”), you can check whether MySQL can use indexes to resolve an `ORDER BY` clause:

- If the `Extra` column of `EXPLAIN` output does not contain `Using filesort`, the index is used and a `filesort` is not performed.

- If the `Extra` column of `EXPLAIN` output contains `Using filesort`, the index is not used and a `filesort` is performed.

In addition, if a `filesort` is performed, optimizer trace output includes a `filesort_summary` block. For example:

```
"filesort_summary": {
  "rows": 100,
  "examined_rows": 100,
  "number_of_tmp_files": 0,
  "sort_buffer_size": 25192,
  "sort_mode": "<sort_key, additional_fields>"
}
```

The `sort_mode` value provides information about the contents of tuples in the sort buffer:

- `<sort_key, rowid>`: This indicates that sort buffer tuples are pairs that contain the sort key value and row ID of the original table row. Tuples are sorted by sort key value and the row ID is used to read the row from the table.
- `<sort_key, additional_fields>`: This indicates that sort buffer tuples contain the sort key value and columns referenced by the query. Tuples are sorted by sort key value and column values are read directly from the tuple.

`EXPLAIN` does not distinguish whether the optimizer does or does not perform a `filesort` in memory. Use of an in-memory `filesort` can be seen in optimizer trace output. Look for `filesort_priority_queue_optimization`. For information about the optimizer trace, see MySQL Internals: Tracing the Optimizer.

8.2.1.14 GROUP BY Optimization

The most general way to satisfy a `GROUP BY` clause is to scan the whole table and create a new temporary table where all rows from each group are consecutive, and then use this temporary table to discover groups and apply aggregate functions (if any). In some cases, MySQL is able to do much better than that and avoid creation of temporary tables by using index access.

The most important preconditions for using indexes for `GROUP BY` are that all `GROUP BY` columns reference attributes from the same index, and that the index stores its keys in order (as is true, for example, for a `BTREE` index, but not for a `HASH` index). Whether use of temporary tables can be replaced by index access also depends on which parts of an index are used in a query, the conditions specified for these parts, and the selected aggregate functions.

There are two ways to execute a `GROUP BY` query through index access, as detailed in the following sections. The first method applies the grouping operation together with all range predicates (if any). The second method first performs a range scan, and then groups the resulting tuples.

In MySQL, `GROUP BY` is used for sorting, so the server may also apply `ORDER BY` optimizations to grouping. However, relying on implicit `GROUP BY` sorting is deprecated. See Section 8.2.1.13, “ORDER BY Optimization”.

- Loose Index Scan
- Tight Index Scan

Loose Index Scan

The most efficient way to process `GROUP BY` is when an index is used to directly retrieve the grouping columns. With this access method, MySQL uses the property of some index types that the keys are ordered (for example, `BTREE`). This property enables use of lookup groups in an index without having to consider all keys in the index that satisfy all `WHERE` conditions. This access method considers only a fraction of the keys in an index, so it is called a *Loose Index Scan*. When there is no `WHERE` clause, a Loose Index Scan reads as many keys as the number of groups, which may be a much smaller number than that of all keys. If the `WHERE` clause contains range predicates (see the discussion of the `range` join type in Section 8.8.1, “Optimizing Queries with EXPLAIN”), a Loose Index Scan looks up the first key of each group that satisfies the range conditions, and again reads the smallest possible number of keys. This is possible under the following conditions:

- The query is over a single table.

- The `GROUP BY` names only columns that form a leftmost prefix of the index and no other columns. (If, instead of `GROUP BY`, the query has a `DISTINCT` clause, all distinct attributes refer to columns that form a leftmost prefix of the index.) For example, if a table `t1` has an index on `(c1,c2,c3)`, Loose Index Scan is applicable if the query has `GROUP BY c1, c2`. It is not applicable if the query has `GROUP BY c2, c3` (the columns are not a leftmost prefix) or `GROUP BY c1, c2, c4` (`c4` is not in the index).
- The only aggregate functions used in the select list (if any) are `MIN()` and `MAX()`, and all of them refer to the same column. The column must be in the index and must immediately follow the columns in the `GROUP BY`.
- Any other parts of the index than those from the `GROUP BY` referenced in the query must be constants (that is, they must be referenced in equalities with constants), except for the argument of `MIN()` or `MAX()` functions.
- For columns in the index, full column values must be indexed, not just a prefix. For example, with `c1 VARCHAR(20), INDEX (c1(10))`, the index uses only a prefix of `c1` values and cannot be used for Loose Index Scan.

If Loose Index Scan is applicable to a query, the `EXPLAIN` output shows `Using index for group-by` in the `Extra` column.

Assume that there is an index `idx(c1,c2,c3)` on table `t1(c1,c2,c3,c4)`. The Loose Index Scan access method can be used for the following queries:

```
SELECT c1, c2 FROM t1 GROUP BY c1, c2;
SELECT DISTINCT c1, c2 FROM t1;
SELECT c1, MIN(c2) FROM t1 GROUP BY c1;
SELECT c1, c2 FROM t1 WHERE c1 < const GROUP BY c1, c2;
SELECT MAX(c3), MIN(c3), c1, c2 FROM t1 WHERE c2 > const GROUP BY c1, c2;
SELECT c2 FROM t1 WHERE c1 < const GROUP BY c1, c2;
SELECT c1, c2 FROM t1 WHERE c3 = const GROUP BY c1, c2;
```

The following queries cannot be executed with this quick select method, for the reasons given:

- There are aggregate functions other than `MIN()` or `MAX()`:

  ```
  SELECT c1, SUM(c2) FROM t1 GROUP BY c1;
  ```

- The columns in the `GROUP BY` clause do not form a leftmost prefix of the index:

  ```
  SELECT c1, c2 FROM t1 GROUP BY c2, c3;
  ```

- The query refers to a part of a key that comes after the `GROUP BY` part, and for which there is no equality with a constant:

  ```
  SELECT c1, c3 FROM t1 GROUP BY c1, c2;
  ```

  Were the query to include `WHERE c3 = const`, Loose Index Scan could be used.

The Loose Index Scan access method can be applied to other forms of aggregate function references in the select list, in addition to the `MIN()` and `MAX()` references already supported:

- `AVG(DISTINCT)`, `SUM(DISTINCT)`, and `COUNT(DISTINCT)` are supported. `AVG(DISTINCT)` and `SUM(DISTINCT)` take a single argument. `COUNT(DISTINCT)` can have more than one column argument.
- There must be no `GROUP BY` or `DISTINCT` clause in the query.
- The Loose Index Scan limitations described previously still apply.

Assume that there is an index `idx(c1,c2,c3)` on table `t1(c1,c2,c3,c4)`. The Loose Index Scan access method can be used for the following queries:

```
SELECT COUNT(DISTINCT c1), SUM(DISTINCT c1) FROM t1;

SELECT COUNT(DISTINCT c1, c2), COUNT(DISTINCT c2, c1) FROM t1;
```

Tight Index Scan

A Tight Index Scan may be either a full index scan or a range index scan, depending on the query conditions.

When the conditions for a Loose Index Scan are not met, it still may be possible to avoid creation of temporary tables for `GROUP BY` queries. If there are range conditions in the `WHERE` clause, this method reads only the keys that satisfy these conditions. Otherwise, it performs an index scan. Because this method reads all keys in each range defined by the `WHERE` clause, or scans the whole index if there are no range conditions, it is called a *Tight Index Scan*. With a Tight Index Scan, the grouping operation is performed only after all keys that satisfy the range conditions have been found.

For this method to work, it is sufficient that there be a constant equality condition for all columns in a query referring to parts of the key coming before or in between parts of the `GROUP BY` key. The constants from the equality conditions fill in any “gaps” in the search keys so that it is possible to form complete prefixes of the index. These index prefixes then can be used for index lookups. If the `GROUP BY` result requires sorting, and it is possible to form search keys that are prefixes of the index, MySQL also avoids extra sorting operations because searching with prefixes in an ordered index already retrieves all the keys in order.

Assume that there is an index `idx(c1,c2,c3)` on table `t1(c1,c2,c3,c4)`. The following queries do not work with the Loose Index Scan access method described previously, but still work with the Tight Index Scan access method.

- There is a gap in the `GROUP BY`, but it is covered by the condition `c2 = 'a'`:

  ```
  SELECT c1, c2, c3 FROM t1 WHERE c2 = 'a' GROUP BY c1, c3;
  ```

- The `GROUP BY` does not begin with the first part of the key, but there is a condition that provides a constant for that part:

  ```
  SELECT c1, c2, c3 FROM t1 WHERE c1 = 'a' GROUP BY c2, c3;
  ```

8.2.1.15 DISTINCT Optimization

`DISTINCT` combined with `ORDER BY` needs a temporary table in many cases.

Because `DISTINCT` may use `GROUP BY`, learn how MySQL works with columns in `ORDER BY` or `HAVING` clauses that are not part of the selected columns. See Section 12.19.3, “MySQL Handling of GROUP BY”.

In most cases, a `DISTINCT` clause can be considered as a special case of `GROUP BY`. For example, the following two queries are equivalent:

```
SELECT DISTINCT c1, c2, c3 FROM t1
WHERE c1 > const;

SELECT c1, c2, c3 FROM t1
WHERE c1 > const GROUP BY c1, c2, c3;
```

Due to this equivalence, the optimizations applicable to `GROUP BY` queries can be also applied to queries with a `DISTINCT` clause. Thus, for more details on the optimization possibilities for `DISTINCT` queries, see Section 8.2.1.14, “GROUP BY Optimization”.

When combining `LIMIT row_count` with `DISTINCT`, MySQL stops as soon as it finds *`row_count`* unique rows.

If you do not use columns from all tables named in a query, MySQL stops scanning any unused tables as soon as it finds the first match. In the following case, assuming that `t1` is used before `t2` (which you

can check with EXPLAIN), MySQL stops reading from t2 (for any particular row in t1) when it finds the first row in t2:

```
SELECT DISTINCT t1.a FROM t1, t2 where t1.a=t2.a;
```

8.2.1.16 LIMIT Query Optimization

If you need only a specified number of rows from a result set, use a LIMIT clause in the query, rather than fetching the whole result set and throwing away the extra data.

MySQL sometimes optimizes a query that has a LIMIT *row_count* clause and no HAVING clause:

- If you select only a few rows with LIMIT, MySQL uses indexes in some cases when normally it would prefer to do a full table scan.
- If you combine LIMIT *row_count* with ORDER BY, MySQL stops sorting as soon as it has found the first *row_count* rows of the sorted result, rather than sorting the entire result. If ordering is done by using an index, this is very fast. If a filesort must be done, all rows that match the query without the LIMIT clause are selected, and most or all of them are sorted, before the first *row_count* are found. After the initial rows have been found, MySQL does not sort any remainder of the result set.

  One manifestation of this behavior is that an ORDER BY query with and without LIMIT may return rows in different order, as described later in this section.
- If you combine LIMIT *row_count* with DISTINCT, MySQL stops as soon as it finds *row_count* unique rows.
- In some cases, a GROUP BY can be resolved by reading the index in order (or doing a sort on the index), then calculating summaries until the index value changes. In this case, LIMIT *row_count* does not calculate any unnecessary GROUP BY values.
- As soon as MySQL has sent the required number of rows to the client, it aborts the query unless you are using SQL_CALC_FOUND_ROWS. In that case, the number of rows can be retrieved with SELECT FOUND_ROWS(). See Section 12.16, “Information Functions”.
- LIMIT 0 quickly returns an empty set. This can be useful for checking the validity of a query. It can also be employed to obtain the types of the result columns within applications that use a MySQL API that makes result set metadata available. With the mysql client program, you can use the --column-type-info option to display result column types.
- If the server uses temporary tables to resolve a query, it uses the LIMIT *row_count* clause to calculate how much space is required.
- If an index is not used for ORDER BY but a LIMIT clause is also present, the optimizer may be able to avoid using a merge file and sort the rows in memory using an in-memory filesort operation.

If multiple rows have identical values in the ORDER BY columns, the server is free to return those rows in any order, and may do so differently depending on the overall execution plan. In other words, the sort order of those rows is nondeterministic with respect to the nonordered columns.

One factor that affects the execution plan is LIMIT, so an ORDER BY query with and without LIMIT may return rows in different orders. Consider this query, which is sorted by the category column but nondeterministic with respect to the id and rating columns:

```
mysql> SELECT * FROM ratings ORDER BY category;
+----+----------+--------+
| id | category | rating |
+----+----------+--------+
|  1 |        1 |    4.5 |
|  5 |        1 |    3.2 |
|  3 |        2 |    3.7 |
|  4 |        2 |    3.5 |
|  6 |        2 |    3.5 |
```

```
|  2 |        3 |    5.0 |
|  7 |        3 |    2.7 |
+----+----------+--------+
```

Including `LIMIT` may affect order of rows within each `category` value. For example, this is a valid query result:

```
mysql> SELECT * FROM ratings ORDER BY category LIMIT 5;
+----+----------+--------+
| id | category | rating |
+----+----------+--------+
|  1 |        1 |    4.5 |
|  5 |        1 |    3.2 |
|  4 |        2 |    3.5 |
|  3 |        2 |    3.7 |
|  6 |        2 |    3.5 |
+----+----------+--------+
```

In each case, the rows are sorted by the `ORDER BY` column, which is all that is required by the SQL standard.

If it is important to ensure the same row order with and without `LIMIT`, include additional columns in the `ORDER BY` clause to make the order deterministic. For example, if `id` values are unique, you can make rows for a given `category` value appear in `id` order by sorting like this:

```
mysql> SELECT * FROM ratings ORDER BY category, id;
+----+----------+--------+
| id | category | rating |
+----+----------+--------+
|  1 |        1 |    4.5 |
|  5 |        1 |    3.2 |
|  3 |        2 |    3.7 |
|  4 |        2 |    3.5 |
|  6 |        2 |    3.5 |
|  2 |        3 |    5.0 |
|  7 |        3 |    2.7 |
+----+----------+--------+

mysql> SELECT * FROM ratings ORDER BY category, id LIMIT 5;
+----+----------+--------+
| id | category | rating |
+----+----------+--------+
|  1 |        1 |    4.5 |
|  5 |        1 |    3.2 |
|  3 |        2 |    3.7 |
|  4 |        2 |    3.5 |
|  6 |        2 |    3.5 |
+----+----------+--------+
```

8.2.1.17 Function Call Optimization

MySQL functions are tagged internally as deterministic or nondeterministic. A function is nondeterministic if, given fixed values for its arguments, it can return different results for different invocations. Examples of nondeterministic functions: `RAND()`, `UUID()`.

If a function is tagged nondeterministic, a reference to it in a `WHERE` clause is evaluated for every row (when selecting from one table) or combination of rows (when selecting from a multiple-table join).

MySQL also determines when to evaluate functions based on types of arguments, whether the arguments are table columns or constant values. A deterministic function that takes a table column as argument must be evaluated whenever that column changes value.

Nondeterministic functions may affect query performance. For example, some optimizations may not be available, or more locking might be required. The following discussion uses `RAND()` but applies to other nondeterministic functions as well.

Suppose that a table `t` has this definition:

```
CREATE TABLE t (id INT NOT NULL PRIMARY KEY, col_a VARCHAR(100));
```

Consider these two queries:

```
SELECT * FROM t WHERE id = POW(1,2);
SELECT * FROM t WHERE id = FLOOR(1 + RAND() * 49);
```

Both queries appear to use a primary key lookup because of the equality comparison against the primary key, but that is true only for the first of them:

- The first query always produces a maximum of one row because `POW()` with constant arguments is a constant value and is used for index lookup.
- The second query contains an expression that uses the nondeterministic function `RAND()`, which is not constant in the query but in fact has a new value for every row of table `t`. Consequently, the query reads every row of the table, evaluates the predicate for each row, and outputs all rows for which the primary key matches the random value. This might be zero, one, or multiple rows, depending on the `id` column values and the values in the `RAND()` sequence.

The effects of nondeterminism are not limited to `SELECT` statements. This `UPDATE` statement uses a nondeterministic function to select rows to be modified:

```
UPDATE t SET col_a = some_expr WHERE id = FLOOR(1 + RAND() * 49);
```

Presumably the intent is to update at most a single row for which the primary key matches the expression. However, it might update zero, one, or multiple rows, depending on the `id` column values and the values in the `RAND()` sequence.

The behavior just described has implications for performance and replication:

- Because a nondeterministic function does not produce a constant value, the optimizer cannot use strategies that might otherwise be applicable, such as index lookups. The result may be a table scan.
- `InnoDB` might escalate to a range-key lock rather than taking a single row lock for one matching row.
- Updates that do not execute deterministically are unsafe for replication.

The difficulties stem from the fact that the `RAND()` function is evaluated once for every row of the table. To avoid multiple function evaluations, use one of these techniques:

- Move the expression containing the nondeterministic function to a separate statement, saving the value in a variable. In the original statement, replace the expression with a reference to the variable, which the optimizer can treat as a constant value:

  ```
  SET @keyval = FLOOR(1 + RAND() * 49);
  UPDATE t SET col_a = some_expr WHERE id = @keyval;
  ```

- Assign the random value to a variable in a derived table. This technique causes the variable to be assigned a value, once, prior to its use in the comparison in the `WHERE` clause:

  ```
  UPDATE t, (SELECT @keyval := FLOOR(1 + RAND() * 49)) AS dt
  SET col_a = some_expr WHERE id = @keyval;
  ```

As mentioned previously, a nondeterministic expression in the `WHERE` clause might prevent optimizations and result in a table scan. However, it may be possible to partially optimize the `WHERE` clause if other expressions are deterministic. For example:

```
SELECT * FROM t WHERE partial_key=5 AND some_column=RAND();
```

If the optimizer can use `partial_key` to reduce the set of rows selected, `RAND()` is executed fewer times, which diminishes the effect of nondeterminism on optimization.

8.2.1.18 Row Constructor Expression Optimization

Row constructors permit simultaneous comparisons of multiple values. For example, these two statements are semantically equivalent:

```
SELECT * FROM t1 WHERE (column1,column2) = (1,1);
SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

In addition, the optimizer handles both expressions the same way.

The optimizer is less likely to use available indexes if the row constructor columns do not cover the prefix of an index. Consider the following table, which has a primary key on `(c1, c2, c3)`:

```
CREATE TABLE t1 (
  c1 INT, c2 INT, c3 INT, c4 CHAR(100),
  PRIMARY KEY(c1,c2,c3)
);
```

In this query, the `WHERE` clause uses all columns in the index. However, the row constructor itself does not cover an index prefix, with the result that the optimizer uses only `c1` (`key_len=4`, the size of `c1`):

```
mysql> EXPLAIN SELECT * FROM t1
       WHERE c1=1 AND (c2,c3) > (1,1)\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
   partitions: NULL
         type: ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: const
         rows: 3
        Extra: Using where
```

In such cases, rewriting the row constructor expression using an equivalent nonconstructor expression may result in more complete index use. For the given query, the row constructor and equivalent nonconstructor expressions are:

```
(c2,c3) > (1,1)
c2 > 1 OR ((c2 = 1) AND (c3 > 1))
```

Rewriting the query to use the nonconstructor expression results in the optimizer using all three columns in the index (`key_len=12`):

```
mysql> EXPLAIN SELECT * FROM t1
       WHERE c1 = 1 AND (c2 > 1 OR ((c2 = 1) AND (c3 > 1)))\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
   partitions: NULL
         type: range
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 12
          ref: NULL
         rows: 3
        Extra: Using where
```

Thus, for better results, avoid mixing row constructors with `AND`/`OR` expressions. Use one or the other.

Under certain conditions, the optimizer can apply the range access method to `IN()` expressions that have row constructor arguments. See Range Optimization of Row Constructor Expressions.

8.2.1.19 Avoiding Full Table Scans

The output from `EXPLAIN` shows `ALL` in the `type` column when MySQL uses a full table scan to resolve a query. This usually happens under the following conditions:

- The table is so small that it is faster to perform a table scan than to bother with a key lookup. This is common for tables with fewer than 10 rows and a short row length.
- There are no usable restrictions in the `ON` or `WHERE` clause for indexed columns.
- You are comparing indexed columns with constant values and MySQL has calculated (based on the index tree) that the constants cover too large a part of the table and that a table scan would be faster. See Section 8.2.1.1, “WHERE Clause Optimization”.
- You are using a key with low cardinality (many rows match the key value) through another column. In this case, MySQL assumes that by using the key it is likely to perform many key lookups and that a table scan would be faster.

For small tables, a table scan often is appropriate and the performance impact is negligible. For large tables, try the following techniques to avoid having the optimizer incorrectly choose a table scan:

- Use `ANALYZE TABLE tbl_name` to update the key distributions for the scanned table. See Section 13.7.2.1, “ANALYZE TABLE Statement”.
- Use `FORCE INDEX` for the scanned table to tell MySQL that table scans are very expensive compared to using the given index:

```
SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
  WHERE t1.col_name=t2.col_name;
```

  See Section 8.9.3, “Index Hints”.
- Start `mysqld` with the `--max-seeks-for-key=1000` option or use `SET max_seeks_for_key=1000` to tell the optimizer to assume that no key scan causes more than 1,000 key seeks. See Section 5.1.7, “Server System Variables”.

8.2.2 Optimizing Subqueries and Derived Tables

The MySQL query optimizer has different strategies available to evaluate subqueries:

- For `IN` (or `=ANY`) subqueries, the optimizer has these choices:
  - Semijoin
  - Materialization
  - `EXISTS` strategy
- For `NOT IN` (or `<>ALL`) subqueries, the optimizer has these choices:
  - Materialization
  - `EXISTS` strategy

For derived tables, the optimizer has these choices:

- Merge the derived table into the outer query block
- Materialize the derived table to an internal temporary table

The following discussion provides more information about the preceding optimization strategies.

Note

A limitation on `UPDATE` and `DELETE` statements that use a subquery to modify a single table is that the optimizer does not use semijoin or materialization subquery optimizations. As a workaround, try rewriting them as multiple-table `UPDATE` and `DELETE` statements that use a join rather than a subquery.

8.2.2.1 Optimizing Subqueries with Semijoin Transformations

A semijoin is a preparation-time transformation that enables multiple execution strategies such as table pullout, duplicate weedout, first match, loose scan, and materialization. The optimizer uses semijoin strategies to improve subquery execution, as described in this section.

For an inner join between two tables, the join returns a row from one table as many times as there are matches in the other table. But for some questions, the only information that matters is whether there is a match, not the number of matches. Suppose that there are tables named `class` and `roster` that list classes in a course curriculum and class rosters (students enrolled in each class), respectively. To list the classes that actually have students enrolled, you could use this join:

```
SELECT class.class_num, class.class_name
FROM class INNER JOIN roster
WHERE class.class_num = roster.class_num;
```

However, the result lists each class once for each enrolled student. For the question being asked, this is unnecessary duplication of information.

Assuming that `class_num` is a primary key in the `class` table, duplicate suppression is possible by using `SELECT DISTINCT`, but it is inefficient to generate all matching rows first only to eliminate duplicates later.

The same duplicate-free result can be obtained by using a subquery:

```
SELECT class_num, class_name
FROM class
WHERE class_num IN (SELECT class_num FROM roster);
```

Here, the optimizer can recognize that the `IN` clause requires the subquery to return only one instance of each class number from the `roster` table. In this case, the query can use a *semijoin*; that is, an operation that returns only one instance of each row in `class` that is matched by rows in `roster`.

Outer join and inner join syntax is permitted in the outer query specification, and table references may be base tables, derived tables, or view references.

In MySQL, a subquery must satisfy these criteria to be handled as a semijoin:

- It must be an `IN` (or `=ANY`) subquery that appears at the top level of the `WHERE` or `ON` clause, possibly as a term in an `AND` expression. For example:

  ```
  SELECT ...
  FROM ot1, ...
  WHERE (oe1, ...) IN (SELECT ie1, ... FROM it1, ... WHERE ...);
  ```

  Here, *`ot_i`* and *`it_i`* represent tables in the outer and inner parts of the query, and *`oe_i`* and *`ie_i`* represent expressions that refer to columns in the outer and inner tables.

- It must be a single `SELECT` without `UNION` constructs.
- It must not contain a `GROUP BY` or `HAVING` clause.
- It must not be implicitly grouped (it must contain no aggregate functions).
- It must not have `ORDER BY` with `LIMIT`.
- The `STRAIGHT_JOIN` modifier must not be present.
- The number of outer and inner tables together must be less than the maximum number of tables permitted in a join.

The subquery may be correlated or uncorrelated. `DISTINCT` is permitted, as is `LIMIT` unless `ORDER BY` is also used.

If a subquery meets the preceding criteria, MySQL converts it to a semijoin and makes a cost-based choice from these strategies:

- Convert the subquery to a join, or use table pullout and run the query as an inner join between subquery tables and outer tables. Table pullout pulls a table out from the subquery to the outer query.
- Duplicate Weedout: Run the semijoin as if it was a join and remove duplicate records using a temporary table.
- FirstMatch: When scanning the inner tables for row combinations and there are multiple instances of a given value group, choose one rather than returning them all. This "shortcuts" scanning and eliminates production of unnecessary rows.
- LooseScan: Scan a subquery table using an index that enables a single value to be chosen from each subquery's value group.
- Materialize the subquery into an indexed temporary table that is used to perform a join, where the index is used to remove duplicates. The index might also be used later for lookups when joining the temporary table with the outer tables; if not, the table is scanned. For more information about materialization, see Section 8.2.2.2, “Optimizing Subqueries with Materialization”.

Each of these strategies except Duplicate Weedout can be enabled or disabled using the `optimizer_switch` system variable:

- The `semijoin` flag controls whether semijoins are used.
- If `semijoin` is enabled, the `firstmatch`, `loosescan`, and `materialization` flags enable finer control over the permitted semijoin strategies.

These flags are enabled by default. See Section 8.9.2, “Switchable Optimizations”.

`EXPLAIN` output indicates the use of semijoin strategies as follows:

- Semijoined tables show up in the outer select. For extended `EXPLAIN` output, the text displayed by a following `SHOW WARNINGS` shows the rewritten query, which displays the semijoin structure. (See Section 8.8.3, “Extended EXPLAIN Output Format”.) From this you can get an idea about which tables were pulled out of the semijoin. If a subquery was converted to a semijoin, you can see that the subquery predicate is gone and its tables and `WHERE` clause were merged into the outer query join list and `WHERE` clause.
- Temporary table use for Duplicate Weedout is indicated by `Start temporary` and `End temporary` in the `Extra` column. Tables that were not pulled out and are in the range of `EXPLAIN` output rows covered by `Start temporary` and `End temporary` have their `rowid` in the temporary table.
- `FirstMatch(`*`tbl_name`*`)` in the `Extra` column indicates join shortcutting.
- `LooseScan(`*`m`*`..`*`n`*`)` in the `Extra` column indicates use of the LooseScan strategy. *`m`* and *`n`* are key part numbers.
- Temporary table use for materialization is indicated by rows with a `select_type` value of `MATERIALIZED` and rows with a `table` value of `<subquery`*`N`*`>`.

8.2.2.2 Optimizing Subqueries with Materialization

The optimizer uses materialization to enable more efficient subquery processing. Materialization speeds up query execution by generating a subquery result as a temporary table, normally in memory. The first time MySQL needs the subquery result, it materializes that result into a temporary table. Any subsequent time the result is needed, MySQL refers again to the temporary table. The optimizer may index the table with a hash index to make lookups fast and inexpensive. The index contains unique values to eliminate duplicates and make the table smaller.

Subquery materialization uses an in-memory temporary table when possible, falling back to on-disk storage if the table becomes too large. See Section 8.4.4, “Internal Temporary Table Use in MySQL”.

If materialization is not used, the optimizer sometimes rewrites a noncorrelated subquery as a correlated subquery. For example, the following `IN` subquery is noncorrelated (*`where_condition`* involves only columns from `t2` and not `t1`):

```
SELECT * FROM t1
WHERE t1.a IN (SELECT t2.b FROM t2 WHERE where_condition);
```

The optimizer might rewrite this as an `EXISTS` correlated subquery:

```
SELECT * FROM t1
WHERE EXISTS (SELECT t2.b FROM t2 WHERE where_condition AND t1.a=t2.b);
```

Subquery materialization using a temporary table avoids such rewrites and makes it possible to execute the subquery only once rather than once per row of the outer query.

For subquery materialization to be used in MySQL, the `optimizer_switch` system variable `materialization` flag must be enabled. (See Section 8.9.2, “Switchable Optimizations”.) With the `materialization` flag enabled, materialization applies to subquery predicates that appear anywhere (in the select list, `WHERE`, `ON`, `GROUP BY`, `HAVING`, or `ORDER BY`), for predicates that fall into any of these use cases:

- The predicate has this form, when no outer expression *`oe_i`* or inner expression *`ie_i`* is nullable. *`N`* is 1 or larger.

  ```
  (oe_1, oe_2, ..., oe_N) [NOT] IN (SELECT ie_1, i_2, ..., ie_N ...)
  ```

- The predicate has this form, when there is a single outer expression *`oe`* and inner expression *`ie`*. The expressions can be nullable.

  ```
  oe [NOT] IN (SELECT ie ...)
  ```

- The predicate is `IN` or `NOT IN` and a result of `UNKNOWN` (`NULL`) has the same meaning as a result of `FALSE`.

The following examples illustrate how the requirement for equivalence of `UNKNOWN` and `FALSE` predicate evaluation affects whether subquery materialization can be used. Assume that *`where_condition`* involves columns only from `t2` and not `t1` so that the subquery is noncorrelated.

This query is subject to materialization:

```
SELECT * FROM t1
WHERE t1.a IN (SELECT t2.b FROM t2 WHERE where_condition);
```

Here, it does not matter whether the `IN` predicate returns `UNKNOWN` or `FALSE`. Either way, the row from `t1` is not included in the query result.

An example where subquery materialization is not used is the following query, where `t2.b` is a nullable column:

```
SELECT * FROM t1
WHERE (t1.a,t1.b) NOT IN (SELECT t2.a,t2.b FROM t2
                          WHERE where_condition);
```

The following restrictions apply to the use of subquery materialization:

- The types of the inner and outer expressions must match. For example, the optimizer might be able to use materialization if both expressions are integer or both are decimal, but cannot if one expression is integer and the other is decimal.

- The inner expression cannot be a `BLOB`.

Use of EXPLAIN with a query provides some indication of whether the optimizer uses subquery materialization:

- Compared to query execution that does not use materialization, select_type may change from DEPENDENT SUBQUERY to SUBQUERY. This indicates that, for a subquery that would be executed once per outer row, materialization enables the subquery to be executed just once.
- For extended EXPLAIN output, the text displayed by a following SHOW WARNINGS includes materialize and materialized-subquery.

8.2.2.3 Optimizing Subqueries with the EXISTS Strategy

Certain optimizations are applicable to comparisons that use the IN (or =ANY) operator to test subquery results. This section discusses these optimizations, particularly with regard to the challenges that NULL values present. The last part of the discussion suggests how you can help the optimizer.

Consider the following subquery comparison:

```
outer_expr IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

MySQL evaluates queries “from outside to inside.” That is, it first obtains the value of the outer expression *outer_expr*, and then runs the subquery and captures the rows that it produces.

A very useful optimization is to “inform” the subquery that the only rows of interest are those where the inner expression *inner_expr* is equal to *outer_expr*. This is done by pushing down an appropriate equality into the subquery's WHERE clause to make it more restrictive. The converted comparison looks like this:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where AND outer_expr=inner_expr)
```

After the conversion, MySQL can use the pushed-down equality to limit the number of rows it must examine to evaluate the subquery.

More generally, a comparison of *N* values to a subquery that returns *N*-value rows is subject to the same conversion. If *oe_i* and *ie_i* represent corresponding outer and inner expression values, this subquery comparison:

```
(oe_1, ..., oe_N) IN
  (SELECT ie_1, ..., ie_N FROM ... WHERE subquery_where)
```

Becomes:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where
                           AND oe_1 = ie_1
                           AND ...
                           AND oe_N = ie_N)
```

For simplicity, the following discussion assumes a single pair of outer and inner expression values.

The conversion just described has its limitations. It is valid only if we ignore possible NULL values. That is, the “pushdown” strategy works as long as both of these conditions are true:

- *outer_expr* and *inner_expr* cannot be NULL.
- You need not distinguish NULL from FALSE subquery results. If the subquery is a part of an OR or AND expression in the WHERE clause, MySQL assumes that you do not care. Another instance where the optimizer notices that NULL and FALSE subquery results need not be distinguished is this construct:

  ```
  ... WHERE outer_expr IN (subquery)
  ```

  In this case, the WHERE clause rejects the row whether IN (*subquery*) returns NULL or FALSE.

When either or both of those conditions do not hold, optimization is more complex.

Suppose that *`outer_expr`* is known to be a non-`NULL` value but the subquery does not produce a row such that *`outer_expr`* = *`inner_expr`*. Then *`outer_expr`* `IN (SELECT ...)` evaluates as follows:

- `NULL`, if the `SELECT` produces any row where *`inner_expr`* is `NULL`
- `FALSE`, if the `SELECT` produces only non-`NULL` values or produces nothing

In this situation, the approach of looking for rows with *`outer_expr = inner_expr`* is no longer valid. It is necessary to look for such rows, but if none are found, also look for rows where *`inner_expr`* is `NULL`. Roughly speaking, the subquery can be converted to something like this:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where AND
        (outer_expr=inner_expr OR inner_expr IS NULL))
```

The need to evaluate the extra `IS NULL` condition is why MySQL has the `ref_or_null` access method:

```
mysql> EXPLAIN
       SELECT outer_expr IN (SELECT t2.maybe_null_key
                             FROM t2, t3 WHERE ...)
       FROM t1;
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: t1
...
*************************** 2. row ***************************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: t2
         type: ref_or_null
possible_keys: maybe_null_key
          key: maybe_null_key
      key_len: 5
          ref: func
         rows: 2
        Extra: Using where; Using index
...
```

The `unique_subquery` and `index_subquery` subquery-specific access methods also have "or `NULL`" variants. However, they are not visible in `EXPLAIN` output, so you must use `EXPLAIN EXTENDED` followed by `SHOW WARNINGS` (note the `checking NULL` in the warning message):

```
mysql> EXPLAIN EXTENDED
       SELECT outer_expr IN (SELECT maybe_null_key FROM t2) FROM t1\G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: t1
...
*************************** 2. row ***************************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: t2
         type: index_subquery
possible_keys: maybe_null_key
          key: maybe_null_key
      key_len: 5
          ref: func
         rows: 2
        Extra: Using index

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: select (`test`.`t1`.`outer_expr`,
```

```
        (((`test`.`t1`.`outer_expr`) in t2 on
        maybe_null_key checking NULL))) AS `outer_expr IN (SELECT
        maybe_null_key FROM t2)` from `test`.`t1`
```

The additional `OR ... IS NULL` condition makes query execution slightly more complicated (and some optimizations within the subquery become inapplicable), but generally this is tolerable.

The situation is much worse when *`outer_expr`* can be `NULL`. According to the SQL interpretation of `NULL` as “unknown value,” `NULL IN (SELECT` *`inner_expr ...`*`)` should evaluate to:

- `NULL`, if the `SELECT` produces any rows
- `FALSE`, if the `SELECT` produces no rows

For proper evaluation, it is necessary to be able to check whether the `SELECT` has produced any rows at all, so *`outer_expr = inner_expr`* cannot be pushed down into the subquery. This is a problem because many real world subqueries become very slow unless the equality can be pushed down.

Essentially, there must be different ways to execute the subquery depending on the value of *`outer_expr`*.

The optimizer chooses SQL compliance over speed, so it accounts for the possibility that *`outer_expr`* might be `NULL`:

- If *`outer_expr`* is `NULL`, to evaluate the following expression, it is necessary to execute the `SELECT` to determine whether it produces any rows:

  ```
  NULL IN (SELECT inner_expr FROM ... WHERE subquery_where)
  ```

  It is necessary to execute the original `SELECT` here, without any pushed-down equalities of the kind mentioned previously.

- On the other hand, when *`outer_expr`* is not `NULL`, it is absolutely essential that this comparison:

  ```
  outer_expr IN (SELECT inner_expr FROM ... WHERE subquery_where)
  ```

  Be converted to this expression that uses a pushed-down condition:

  ```
  EXISTS (SELECT 1 FROM ... WHERE subquery_where AND outer_expr=inner_expr)
  ```

  Without this conversion, subqueries are slow.

To solve the dilemma of whether or not to push down conditions into the subquery, the conditions are wrapped within “trigger” functions. Thus, an expression of the following form:

```
outer_expr IN (SELECT inner_expr FROM ... WHERE subquery_where)
```

Is converted into:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where
                          AND trigcond(outer_expr=inner_expr))
```

More generally, if the subquery comparison is based on several pairs of outer and inner expressions, the conversion takes this comparison:

```
(oe_1, ..., oe_N) IN (SELECT ie_1, ..., ie_N FROM ... WHERE subquery_where)
```

And converts it to this expression:

```
EXISTS (SELECT 1 FROM ... WHERE subquery_where
                          AND trigcond(oe_1=ie_1)
                          AND ...
                          AND trigcond(oe_N=ie_N)
      )
```

Each `trigcond(X)` is a special function that evaluates to the following values:

- *X* when the “linked” outer expression *oe_i* is not `NULL`
- `TRUE` when the “linked” outer expression *oe_i* is `NULL`

Note

Trigger functions are *not* triggers of the kind that you create with `CREATE TRIGGER`.

Equalities that are wrapped within `trigcond()` functions are not first class predicates for the query optimizer. Most optimizations cannot deal with predicates that may be turned on and off at query execution time, so they assume any `trigcond(X)` to be an unknown function and ignore it. Triggered equalities can be used by those optimizations:

- Reference optimizations: `trigcond(X=Y [OR Y IS NULL])` can be used to construct `ref`, `eq_ref`, or `ref_or_null` table accesses.
- Index lookup-based subquery execution engines: `trigcond(X=Y)` can be used to construct `unique_subquery` or `index_subquery` accesses.
- Table-condition generator: If the subquery is a join of several tables, the triggered condition is checked as soon as possible.

When the optimizer uses a triggered condition to create some kind of index lookup-based access (as for the first two items of the preceding list), it must have a fallback strategy for the case when the condition is turned off. This fallback strategy is always the same: Do a full table scan. In `EXPLAIN` output, the fallback shows up as `Full scan on NULL key` in the `Extra` column:

```
mysql> EXPLAIN SELECT t1.col1,
       t1.col1 IN (SELECT t2.key1 FROM t2 WHERE t2.col2=t1.col2) FROM t1\G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: t1
        ...
*************************** 2. row ***************************
           id: 2
  select_type: DEPENDENT SUBQUERY
        table: t2
         type: index_subquery
possible_keys: key1
          key: key1
      key_len: 5
          ref: func
         rows: 2
        Extra: Using where; Full scan on NULL key
```

If you run `EXPLAIN EXTENDED` followed by `SHOW WARNINGS`, you can see the triggered condition:

```
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: select `test`.`t1`.`col1` AS `col1`,
         <in_optimizer>(`test`.`t1`.`col1`,
         <exists>(<index_lookup>(<cache>(`test`.`t1`.`col1`) in t2
         on key1 checking NULL
         where (`test`.`t2`.`col2` = `test`.`t1`.`col2`) having
         trigcond(<is_not_null_test>(`test`.`t2`.`key1`))))) AS
         `t1.col1 IN (select t2.key1 from t2 where t2.col2=t1.col2)`
         from `test`.`t1`
```

The use of triggered conditions has some performance implications. A `NULL IN (SELECT ...)` expression now may cause a full table scan (which is slow) when it previously did not. This is the price paid for correct results (the goal of the trigger-condition strategy is to improve compliance, not speed).

For multiple-table subqueries, execution of `NULL IN (SELECT ...)` is particularly slow because the join optimizer does not optimize for the case where the outer expression is `NULL`. It assumes that subquery evaluations with `NULL` on the left side are very rare, even if there are statistics that indicate otherwise. On the other hand, if the outer expression might be `NULL` but never actually is, there is no performance penalty.

To help the query optimizer better execute your queries, use these suggestions:

- Declare a column as `NOT NULL` if it really is. This also helps other aspects of the optimizer by simplifying condition testing for the column.
- If you need not distinguish a `NULL` from `FALSE` subquery result, you can easily avoid the slow execution path. Replace a comparison that looks like this:

  ```
  outer_expr IN (SELECT inner_expr FROM ...)
  ```

  with this expression:

  ```
  (outer_expr IS NOT NULL) AND (outer_expr IN (SELECT inner_expr FROM ...))
  ```

  Then `NULL IN (SELECT ...)` is never evaluated because MySQL stops evaluating `AND` parts as soon as the expression result is clear.

  Another possible rewrite:

  ```
  EXISTS (SELECT inner_expr FROM ...
          WHERE inner_expr=outer_expr)
  ```

  This would apply when you need not distinguish `NULL` from `FALSE` subquery results, in which case you may actually want `EXISTS`.

The `subquery_materialization_cost_based` flag of the `optimizer_switch` system variable enables control over the choice between subquery materialization and `IN`-to-`EXISTS` subquery transformation. See Section 8.9.2, “Switchable Optimizations”.

8.2.2.4 Optimizing Derived Tables

The optimizer handles derived tables as follows:

- The optimizer postpones materialization of derived tables until their contents are needed during query execution, which improves performance.
- For non-`EXPLAIN` queries, delay of materialization may result in not having to do it at all. Consider a query that joins the result of a derived table to another table: If the optimizer processes that other table first and finds that it returns no rows, the join need not be carried out further and the optimizer can completely skip materializing the derived table.
- During query execution, the optimizer may add an index to a derived table to speed up row retrieval from it.

Consider the following `EXPLAIN` statement, for a `SELECT` query that contains a derived table:

```
EXPLAIN SELECT * FROM (SELECT * FROM t1) AS derived_t1;
```

The optimizer avoids materializing the derived table by delaying it until the result is needed during `SELECT` execution. In this case, the query is not executed (because it occurs in an `EXPLAIN` statement), so the result is never needed.

Even for queries that are executed, delay of derived table materialization may enable the optimizer to avoid materialization entirely. Consider the following query, which joins the result of a derived table to another table:

```
SELECT *
```

```
  FROM t1 JOIN (SELECT t2.f1 FROM t2) AS derived_t2
          ON t1.f2=derived_t2.f1
  WHERE t1.f1 > 0;
```

If the optimization processes `t1` first and the `WHERE` clause produces an empty result, the join must necessarily be empty and the derived table need not be materialized.

For cases when a derived table requires materialization, the optimizer may add an index to the materialized table to speed up access to it. If such an index enables `ref` access to the table, it can greatly reduce amount of data read during query execution. Consider the following query:

```
SELECT *
 FROM t1 JOIN (SELECT DISTINCT f1 FROM t2) AS derived_t2
         ON t1.f1=derived_t2.f1;
```

The optimizer constructs an index over column `f1` from `derived_t2` if doing so would enable use of `ref` access for the lowest cost execution plan. After adding the index, the optimizer can treat the materialized derived table the same as a regular table with an index, and it benefits similarly from the generated index. The overhead of index creation is negligible compared to the cost of query execution without the index. If `ref` access would result in higher cost than some other access method, the optimizer creates no index and loses nothing.

For optimizer trace output, a merged derived table or view reference is not shown as a node. Only its underlying tables appear in the top query's plan.

8.2.3 Optimizing INFORMATION_SCHEMA Queries

Applications that monitor databases may make frequent use of `INFORMATION_SCHEMA` tables. Certain types of queries for `INFORMATION_SCHEMA` tables can be optimized to execute more quickly. The goal is to minimize file operations (for example, scanning a directory or opening a table file) to collect the information that makes up these dynamic tables.

Note

Comparison behavior for database and table names in `INFORMATION_SCHEMA` queries might differ from what you expect. For details, see Section 10.8.7, “Using Collation in INFORMATION_SCHEMA Searches”.

1) Try to use constant lookup values for database and table names in the `WHERE` clause

You can take advantage of this principle as follows:

- To look up databases or tables, use expressions that evaluate to a constant, such as literal values, functions that return a constant, or scalar subqueries.
- Avoid queries that use a nonconstant database name lookup value (or no lookup value) because they require a scan of the data directory to find matching database directory names.
- Within a database, avoid queries that use a nonconstant table name lookup value (or no lookup value) because they require a scan of the database directory to find matching table files.

This principle applies to the `INFORMATION_SCHEMA` tables shown in the following table, which shows the columns for which a constant lookup value enables the server to avoid a directory scan. For example, if you are selecting from `TABLES`, using a constant lookup value for `TABLE_SCHEMA` in the `WHERE` clause enables a data directory scan to be avoided.

Table	Column to specify to avoid data directory scan	Column to specify to avoid database directory scan
`COLUMNS`	`TABLE_SCHEMA`	`TABLE_NAME`
`KEY_COLUMN_USAGE`	`TABLE_SCHEMA`	`TABLE_NAME`

Table	Column to specify to avoid data directory scan	Column to specify to avoid database directory scan
PARTITIONS	TABLE_SCHEMA	TABLE_NAME
REFERENTIAL_CONSTRAINTS	CONSTRAINT_SCHEMA	TABLE_NAME
STATISTICS	TABLE_SCHEMA	TABLE_NAME
TABLES	TABLE_SCHEMA	TABLE_NAME
TABLE_CONSTRAINTS	TABLE_SCHEMA	TABLE_NAME
TRIGGERS	EVENT_OBJECT_SCHEMA	EVENT_OBJECT_TABLE
VIEWS	TABLE_SCHEMA	TABLE_NAME

The benefit of a query that is limited to a specific constant database name is that checks need be made only for the named database directory. Example:

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'test';
```

Use of the literal database name `test` enables the server to check only the `test` database directory, regardless of how many databases there might be. By contrast, the following query is less efficient because it requires a scan of the data directory to determine which database names match the pattern `'test%'`:

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA LIKE 'test%';
```

For a query that is limited to a specific constant table name, checks need be made only for the named table within the corresponding database directory. Example:

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'test' AND TABLE_NAME = 't1';
```

Use of the literal table name `t1` enables the server to check only the files for the `t1` table, regardless of how many tables there might be in the `test` database. By contrast, the following query requires a scan of the `test` database directory to determine which table names match the pattern `'t%'`:

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'test' AND TABLE_NAME LIKE 't%';
```

The following query requires a scan of the database directory to determine matching database names for the pattern `'test%'`, and for each matching database, it requires a scan of the database directory to determine matching table names for the pattern `'t%'`:

```
SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'test%' AND TABLE_NAME LIKE 't%';
```

2) Write queries that minimize the number of table files that must be opened

For queries that refer to certain `INFORMATION_SCHEMA` table columns, several optimizations are available that minimize the number of table files that must be opened. Example:

```
SELECT TABLE_NAME, ENGINE FROM INFORMATION_SCHEMA.TABLES
WHERE TABLE_SCHEMA = 'test';
```

In this case, after the server has scanned the database directory to determine the names of the tables in the database, those names become available with no further file system lookups. Thus, `TABLE_NAME` requires no files to be opened. The `ENGINE` (storage engine) value can be determined by opening the table's `.frm` file, without touching other table files such as the `.MYD` or `.MYI` file.

Some values, such as `INDEX_LENGTH` for `MyISAM` tables, require opening the `.MYD` or `.MYI` file as well.

The file-opening optimization types are denoted thus:

- SKIP_OPEN_TABLE: Table files do not need to be opened. The information has already become available within the query by scanning the database directory.
- OPEN_FRM_ONLY: Only the table's .frm file need be opened.
- OPEN_TRIGGER_ONLY: Only the table's .TRG file need be opened.
- OPEN_FULL_TABLE: The unoptimized information lookup. The .frm, .MYD, and .MYI files must be opened.

The following list indicates how the preceding optimization types apply to INFORMATION_SCHEMA table columns. For tables and columns not named, none of the optimizations apply.

- COLUMNS: OPEN_FRM_ONLY applies to all columns
- KEY_COLUMN_USAGE: OPEN_FULL_TABLE applies to all columns
- PARTITIONS: OPEN_FULL_TABLE applies to all columns
- REFERENTIAL_CONSTRAINTS: OPEN_FULL_TABLE applies to all columns
- STATISTICS:

Column	Optimization type
TABLE_CATALOG	OPEN_FRM_ONLY
TABLE_SCHEMA	OPEN_FRM_ONLY
TABLE_NAME	OPEN_FRM_ONLY
NON_UNIQUE	OPEN_FRM_ONLY
INDEX_SCHEMA	OPEN_FRM_ONLY
INDEX_NAME	OPEN_FRM_ONLY
SEQ_IN_INDEX	OPEN_FRM_ONLY
COLUMN_NAME	OPEN_FRM_ONLY
COLLATION	OPEN_FRM_ONLY
CARDINALITY	OPEN_FULL_TABLE
SUB_PART	OPEN_FRM_ONLY
PACKED	OPEN_FRM_ONLY
NULLABLE	OPEN_FRM_ONLY
INDEX_TYPE	OPEN_FULL_TABLE
COMMENT	OPEN_FRM_ONLY

- TABLES:

Column	Optimization type
TABLE_CATALOG	SKIP_OPEN_TABLE
TABLE_SCHEMA	SKIP_OPEN_TABLE
TABLE_NAME	SKIP_OPEN_TABLE
TABLE_TYPE	OPEN_FRM_ONLY
ENGINE	OPEN_FRM_ONLY
VERSION	OPEN_FRM_ONLY

Column	Optimization type
ROW_FORMAT	OPEN_FULL_TABLE
TABLE_ROWS	OPEN_FULL_TABLE
AVG_ROW_LENGTH	OPEN_FULL_TABLE
DATA_LENGTH	OPEN_FULL_TABLE
MAX_DATA_LENGTH	OPEN_FULL_TABLE
INDEX_LENGTH	OPEN_FULL_TABLE
DATA_FREE	OPEN_FULL_TABLE
AUTO_INCREMENT	OPEN_FULL_TABLE
CREATE_TIME	OPEN_FULL_TABLE
UPDATE_TIME	OPEN_FULL_TABLE
CHECK_TIME	OPEN_FULL_TABLE
TABLE_COLLATION	OPEN_FRM_ONLY
CHECKSUM	OPEN_FULL_TABLE
CREATE_OPTIONS	OPEN_FRM_ONLY
TABLE_COMMENT	OPEN_FRM_ONLY

- TABLE_CONSTRAINTS: OPEN_FULL_TABLE applies to all columns
- TRIGGERS: OPEN_TRIGGER_ONLY applies to all columns
- VIEWS:

Column	Optimization type
TABLE_CATALOG	OPEN_FRM_ONLY
TABLE_SCHEMA	OPEN_FRM_ONLY
TABLE_NAME	OPEN_FRM_ONLY
VIEW_DEFINITION	OPEN_FRM_ONLY
CHECK_OPTION	OPEN_FRM_ONLY
IS_UPDATABLE	OPEN_FULL_TABLE
DEFINER	OPEN_FRM_ONLY
SECURITY_TYPE	OPEN_FRM_ONLY
CHARACTER_SET_CLIENT	OPEN_FRM_ONLY
COLLATION_CONNECTION	OPEN_FRM_ONLY

3) Use EXPLAIN to determine whether the server can use INFORMATION_SCHEMA optimizations for a query

This applies particularly for INFORMATION_SCHEMA queries that search for information from more than one database, which might take a long time and impact performance. The Extra value in EXPLAIN output indicates which, if any, of the optimizations described earlier the server can use to evaluate INFORMATION_SCHEMA queries. The following examples demonstrate the kinds of information you can expect to see in the Extra value.

```
mysql> EXPLAIN SELECT TABLE_NAME FROM INFORMATION_SCHEMA.VIEWS WHERE
       TABLE_SCHEMA = 'test' AND TABLE_NAME = 'v1'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: VIEWS
         type: ALL
```

```
possible_keys: NULL
          key: TABLE_SCHEMA,TABLE_NAME
      key_len: NULL
          ref: NULL
         rows: NULL
        Extra: Using where; Open_frm_only; Scanned 0 databases
```

Use of constant database and table lookup values enables the server to avoid directory scans. For references to `VIEWS.TABLE_NAME`, only the `.frm` file need be opened.

```
mysql> EXPLAIN SELECT TABLE_NAME, ROW_FORMAT FROM INFORMATION_SCHEMA.TABLES\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: TABLES
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
        Extra: Open_full_table; Scanned all databases
```

No lookup values are provided (there is no `WHERE` clause), so the server must scan the data directory and each database directory. For each table thus identified, the table name and row format are selected. `TABLE_NAME` requires no further table files to be opened (the `SKIP_OPEN_TABLE` optimization applies). `ROW_FORMAT` requires all table files to be opened (`OPEN_FULL_TABLE` applies). `EXPLAIN` reports `OPEN_FULL_TABLE` because it is more expensive than `SKIP_OPEN_TABLE`.

```
mysql> EXPLAIN SELECT TABLE_NAME, TABLE_TYPE FROM INFORMATION_SCHEMA.TABLES
       WHERE TABLE_SCHEMA = 'test'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: TABLES
         type: ALL
possible_keys: NULL
          key: TABLE_SCHEMA
      key_len: NULL
          ref: NULL
         rows: NULL
        Extra: Using where; Open_frm_only; Scanned 1 database
```

No table name lookup value is provided, so the server must scan the `test` database directory. For the `TABLE_NAME` and `TABLE_TYPE` columns, the `SKIP_OPEN_TABLE` and `OPEN_FRM_ONLY` optimizations apply, respectively. `EXPLAIN` reports `OPEN_FRM_ONLY` because it is more expensive.

```
mysql> EXPLAIN SELECT B.TABLE_NAME
       FROM INFORMATION_SCHEMA.TABLES AS A, INFORMATION_SCHEMA.COLUMNS AS B
       WHERE A.TABLE_SCHEMA = 'test'
       AND A.TABLE_NAME = 't1'
       AND B.TABLE_NAME = A.TABLE_NAME\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: A
         type: ALL
possible_keys: NULL
          key: TABLE_SCHEMA,TABLE_NAME
      key_len: NULL
          ref: NULL
         rows: NULL
        Extra: Using where; Skip_open_table; Scanned 0 databases
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: B
         type: ALL
possible_keys: NULL
          key: NULL
```

```
      key_len: NULL
          ref: NULL
         rows: NULL
        Extra: Using where; Open_frm_only; Scanned all databases;
               Using join buffer
```

For the first `EXPLAIN` output row: Constant database and table lookup values enable the server to avoid directory scans for `TABLES` values. References to `TABLES.TABLE_NAME` require no further table files.

For the second `EXPLAIN` output row: All `COLUMNS` table values are `OPEN_FRM_ONLY` lookups, so `COLUMNS.TABLE_NAME` requires the `.frm` file to be opened.

```
mysql> EXPLAIN SELECT * FROM INFORMATION_SCHEMA.COLLATIONS\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: COLLATIONS
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
        Extra:
```

In this case, no optimizations apply because `COLLATIONS` is not one of the `INFORMATION_SCHEMA` tables for which optimizations are available.

8.2.4 Optimizing Data Change Statements

This section explains how to speed up data change statements: `INSERT`, `UPDATE`, and `DELETE`. Traditional OLTP applications and modern web applications typically do many small data change operations, where concurrency is vital. Data analysis and reporting applications typically run data change operations that affect many rows at once, where the main considerations is the I/O to write large amounts of data and keep indexes up-to-date. For inserting and updating large volumes of data (known in the industry as ETL, for “extract-transform-load”), sometimes you use other SQL statements or external commands, that mimic the effects of `INSERT`, `UPDATE`, and `DELETE` statements.

8.2.4.1 Optimizing INSERT Statements

To optimize insert speed, combine many small operations into a single large operation. Ideally, you make a single connection, send the data for many new rows at once, and delay all index updates and consistency checking until the very end.

The time required for inserting a row is determined by the following factors, where the numbers indicate approximate proportions:

- Connecting: (3)
- Sending query to server: (2)
- Parsing query: (2)
- Inserting row: (1 × size of row)
- Inserting indexes: (1 × number of indexes)
- Closing: (1)

This does not take into consideration the initial overhead to open tables, which is done once for each concurrently running query.

The size of the table slows down the insertion of indexes by log *N*, assuming B-tree indexes.

You can use the following methods to speed up inserts:

- If you are inserting many rows from the same client at the same time, use `INSERT` statements with multiple `VALUES` lists to insert several rows at a time. This is considerably faster (many times faster in some cases) than using separate single-row `INSERT` statements. If you are adding data to a nonempty table, you can tune the `bulk_insert_buffer_size` variable to make data insertion even faster. See Section 5.1.7, “Server System Variables”.
- When loading a table from a text file, use `LOAD DATA`. This is usually 20 times faster than using `INSERT` statements. See Section 13.2.6, “LOAD DATA Statement”.
- Take advantage of the fact that columns have default values. Insert values explicitly only when the value to be inserted differs from the default. This reduces the parsing that MySQL must do and improves the insert speed.
- See Section 8.5.5, “Bulk Data Loading for InnoDB Tables” for tips specific to `InnoDB` tables.
- See Section 8.6.2, “Bulk Data Loading for MyISAM Tables” for tips specific to `MyISAM` tables.

8.2.4.2 Optimizing UPDATE Statements

An update statement is optimized like a `SELECT` query with the additional overhead of a write. The speed of the write depends on the amount of data being updated and the number of indexes that are updated. Indexes that are not changed do not get updated.

Another way to get fast updates is to delay updates and then do many updates in a row later. Performing multiple updates together is much quicker than doing one at a time if you lock the table.

For a `MyISAM` table that uses dynamic row format, updating a row to a longer total length may split the row. If you do this often, it is very important to use `OPTIMIZE TABLE` occasionally. See Section 13.7.2.4, “OPTIMIZE TABLE Statement”.

8.2.4.3 Optimizing DELETE Statements

The time required to delete individual rows in a `MyISAM` table is exactly proportional to the number of indexes. To delete rows more quickly, you can increase the size of the key cache by increasing the `key_buffer_size` system variable. See Section 5.1.1, “Configuring the Server”.

To delete all rows from a `MyISAM` table, `TRUNCATE TABLE` *`tbl_name`* is faster than `DELETE FROM` *`tbl_name`*. Truncate operations are not transaction-safe; an error occurs when attempting one in the course of an active transaction or active table lock. See Section 13.1.33, “TRUNCATE TABLE Statement”.

8.2.5 Optimizing Database Privileges

The more complex your privilege setup, the more overhead applies to all SQL statements. Simplifying the privileges established by `GRANT` statements enables MySQL to reduce permission-checking overhead when clients execute statements. For example, if you do not grant any table-level or column-level privileges, the server need not ever check the contents of the `tables_priv` and `columns_priv` tables. Similarly, if you place no resource limits on any accounts, the server does not have to perform resource counting. If you have a very high statement-processing load, consider using a simplified grant structure to reduce permission-checking overhead.

8.2.6 Other Optimization Tips

This section lists a number of miscellaneous tips for improving query processing speed:

- If your application makes several database requests to perform related updates, combining the statements into a stored routine can help performance. Similarly, if your application computes a single result based on several column values or large volumes of data, combining the computation

into a UDF (user-defined function) can help performance. The resulting fast database operations are then available to be reused by other queries, applications, and even code written in different programming languages. See Section 20.2, “Using Stored Routines” and Adding Functions to MySQL for more information.

- To fix any compression issues that occur with `ARCHIVE` tables, use `OPTIMIZE TABLE`. See Section 15.5, “The ARCHIVE Storage Engine”.
- If possible, classify reports as “live” or as “statistical”, where data needed for statistical reports is created only from summary tables that are generated periodically from the live data.
- If you have data that does not conform well to a rows-and-columns table structure, you can pack and store data into a `BLOB` column. In this case, you must provide code in your application to pack and unpack information, but this might save I/O operations to read and write the sets of related values.
- With Web servers, store images and other binary assets as files, with the path name stored in the database rather than the file itself. Most Web servers are better at caching files than database contents, so using files is generally faster. (Although you must handle backups and storage issues yourself in this case.)
- If you need really high speed, look at the low-level MySQL interfaces. For example, by accessing the MySQL `InnoDB` or `MyISAM` storage engine directly, you could get a substantial speed increase compared to using the SQL interface.

  Similarly, for databases using the `NDBCLUSTER` storage engine, you may wish to investigate possible use of the NDB API (see MySQL NDB Cluster API Developer Guide).
- Replication can provide a performance benefit for some operations. You can distribute client retrievals among replicas to split up the load. To avoid slowing down the source while making backups, you can make backups using a replica. See Chapter 17, *Replication*.

8.3 Optimization and Indexes

The best way to improve the performance of `SELECT` operations is to create indexes on one or more of the columns that are tested in the query. The index entries act like pointers to the table rows, allowing the query to quickly determine which rows match a condition in the `WHERE` clause, and retrieve the other column values for those rows. All MySQL data types can be indexed.

Although it can be tempting to create an indexes for every possible column used in a query, unnecessary indexes waste space and waste time for MySQL to determine which indexes to use. Indexes also add to the cost of inserts, updates, and deletes because each index must be updated. You must find the right balance to achieve fast queries using the optimal set of indexes.

8.3.1 How MySQL Uses Indexes

Indexes are used to find rows with specific column values quickly. Without an index, MySQL must begin with the first row and then read through the entire table to find the relevant rows. The larger the table, the more this costs. If the table has an index for the columns in question, MySQL can quickly determine the position to seek to in the middle of the data file without having to look at all the data. This is much faster than reading every row sequentially.

Most MySQL indexes (`PRIMARY KEY`, `UNIQUE`, `INDEX`, and `FULLTEXT`) are stored in B-trees. Exceptions: Indexes on spatial data types use R-trees; `MEMORY` tables also support hash indexes; `InnoDB` uses inverted lists for `FULLTEXT` indexes.

In general, indexes are used as described in the following discussion. Characteristics specific to hash indexes (as used in `MEMORY` tables) are described in Section 8.3.8, “Comparison of B-Tree and Hash Indexes”.

MySQL uses indexes for these operations:

- To find the rows matching a `WHERE` clause quickly.
- To eliminate rows from consideration. If there is a choice between multiple indexes, MySQL normally uses the index that finds the smallest number of rows (the most selective index).
- If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to look up rows. For example, if you have a three-column index on `(col1, col2, col3)`, you have indexed search capabilities on `(col1)`, `(col1, col2)`, and `(col1, col2, col3)`. For more information, see Section 8.3.5, “Multiple-Column Indexes”.
- To retrieve rows from other tables when performing joins. MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, `VARCHAR` and `CHAR` are considered the same if they are declared as the same size. For example, `VARCHAR(10)` and `CHAR(10)` are the same size, but `VARCHAR(10)` and `CHAR(15)` are not.

  For comparisons between nonbinary string columns, both columns should use the same character set. For example, comparing a `utf8` column with a `latin1` column precludes use of an index.

  Comparison of dissimilar columns (comparing a string column to a temporal or numeric column, for example) may prevent use of indexes if values cannot be compared directly without conversion. For a given value such as `1` in the numeric column, it might compare equal to any number of values in the string column such as `'1'`, `' 1'`, `'00001'`, or `'01.e1'`. This rules out use of any indexes for the string column.
- To find the `MIN()` or `MAX()` value for a specific indexed column *`key_col`*. This is optimized by a preprocessor that checks whether you are using `WHERE key_part_N = constant` on all key parts that occur before *`key_col`* in the index. In this case, MySQL does a single key lookup for each `MIN()` or `MAX()` expression and replaces it with a constant. If all expressions are replaced with constants, the query returns at once. For example:

```
SELECT MIN(key_part2),MAX(key_part2)
  FROM tbl_name WHERE key_part1=10;
```

- To sort or group a table if the sorting or grouping is done on a leftmost prefix of a usable index (for example, `ORDER BY key_part1, key_part2`). If all key parts are followed by `DESC`, the key is read in reverse order. See Section 8.2.1.13, “ORDER BY Optimization”, and Section 8.2.1.14, “GROUP BY Optimization”.
- In some cases, a query can be optimized to retrieve values without consulting the data rows. (An index that provides all the necessary results for a query is called a covering index.) If a query uses from a table only columns that are included in some index, the selected values can be retrieved from the index tree for greater speed:

```
SELECT key_part3 FROM tbl_name
  WHERE key_part1=1
```

Indexes are less important for queries on small tables, or big tables where report queries process most or all of the rows. When a query needs to access most of the rows, reading sequentially is faster than working through an index. Sequential reads minimize disk seeks, even if not all the rows are needed for the query. See Section 8.2.1.19, “Avoiding Full Table Scans” for details.

8.3.2 Primary Key Optimization

The primary key for a table represents the column or set of columns that you use in your most vital queries. It has an associated index, for fast query performance. Query performance benefits from the `NOT NULL` optimization, because it cannot include any `NULL` values. With the `InnoDB` storage engine, the table data is physically organized to do ultra-fast lookups and sorts based on the primary key column or columns.

If your table is big and important, but does not have an obvious column or set of columns to use as a primary key, you might create a separate column with auto-increment values to use as the primary key.

These unique IDs can serve as pointers to corresponding rows in other tables when you join tables using foreign keys.

8.3.3 Foreign Key Optimization

If a table has many columns, and you query many different combinations of columns, it might be efficient to split the less-frequently used data into separate tables with a few columns each, and relate them back to the main table by duplicating the numeric ID column from the main table. That way, each small table can have a primary key for fast lookups of its data, and you can query just the set of columns that you need using a join operation. Depending on how the data is distributed, the queries might perform less I/O and take up less cache memory because the relevant columns are packed together on disk. (To maximize performance, queries try to read as few data blocks as possible from disk; tables with only a few columns can fit more rows in each data block.)

8.3.4 Column Indexes

The most common type of index involves a single column, storing copies of the values from that column in a data structure, allowing fast lookups for the rows with the corresponding column values. The B-tree data structure lets the index quickly find a specific value, a set of values, or a range of values, corresponding to operators such as `=`, `>`, `≤`, `BETWEEN`, `IN`, and so on, in a `WHERE` clause.

The maximum number of indexes per table and the maximum index length is defined per storage engine. See Chapter 14, *The InnoDB Storage Engine*, and Chapter 15, *Alternative Storage Engines*. All storage engines support at least 16 indexes per table and a total index length of at least 256 bytes. Most storage engines have higher limits.

For additional information about column indexes, see Section 13.1.13, “CREATE INDEX Statement”.

- Index Prefixes
- FULLTEXT Indexes
- Spatial Indexes
- Indexes in the MEMORY Storage Engine

Index Prefixes

With *`col_name(N)`* syntax in an index specification for a string column, you can create an index that uses only the first *`N`* characters of the column. Indexing only a prefix of column values in this way can make the index file much smaller. When you index a `BLOB` or `TEXT` column, you *must* specify a prefix length for the index. For example:

```
CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
```

Prefixes can be up to 1000 bytes long (767 bytes for `InnoDB` tables, unless you have `innodb_large_prefix` set).

Note

Prefix limits are measured in bytes, whereas the prefix length in `CREATE TABLE`, `ALTER TABLE`, and `CREATE INDEX` statements is interpreted as number of characters for nonbinary string types (`CHAR`, `VARCHAR`, `TEXT`) and number of bytes for binary string types (`BINARY`, `VARBINARY`, `BLOB`). Take this into account when specifying a prefix length for a nonbinary string column that uses a multibyte character set.

If a search term exceeds the index prefix length, the index is used to exclude non-matching rows, and the remaining rows are examined for possible matches.

For additional information about index prefixes, see Section 13.1.13, “CREATE INDEX Statement”.

FULLTEXT Indexes

`FULLTEXT` indexes are used for full-text searches. Only the `InnoDB` and `MyISAM` storage engines support `FULLTEXT` indexes and only for `CHAR`, `VARCHAR`, and `TEXT` columns. Indexing always takes place over the entire column and column prefix indexing is not supported. For details, see Section 12.10, “Full-Text Search Functions”.

Optimizations are applied to certain kinds of `FULLTEXT` queries against single `InnoDB` tables. Queries with these characteristics are particularly efficient:

- `FULLTEXT` queries that only return the document ID, or the document ID and the search rank.
- `FULLTEXT` queries that sort the matching rows in descending order of score and apply a `LIMIT` clause to take the top N matching rows. For this optimization to apply, there must be no `WHERE` clauses and only a single `ORDER BY` clause in descending order.
- `FULLTEXT` queries that retrieve only the `COUNT(*)` value of rows matching a search term, with no additional `WHERE` clauses. Code the `WHERE` clause as `WHERE MATCH(text) AGAINST ('other_text')`, without any `> 0` comparison operator.

For queries that contain full-text expressions, MySQL evaluates those expressions during the optimization phase of query execution. The optimizer does not just look at full-text expressions and make estimates, it actually evaluates them in the process of developing an execution plan.

An implication of this behavior is that `EXPLAIN` for full-text queries is typically slower than for non-full-text queries for which no expression evaluation occurs during the optimization phase.

`EXPLAIN` for full-text queries may show `Select tables optimized away` in the `Extra` column due to matching occurring during optimization; in this case, no table access need occur during later execution.

Spatial Indexes

You can create indexes on spatial data types. Only `MyISAM` supports R-tree indexes on spatial types. Other storage engines use B-trees for indexing spatial types (except for `ARCHIVE`, which does not support spatial type indexing).

Indexes in the MEMORY Storage Engine

The `MEMORY` storage engine uses `HASH` indexes by default, but also supports `BTREE` indexes.

8.3.5 Multiple-Column Indexes

MySQL can create composite indexes (that is, indexes on multiple columns). An index may consist of up to 16 columns. For certain data types, you can index a prefix of the column (see Section 8.3.4, “Column Indexes”).

MySQL can use multiple-column indexes for queries that test all the columns in the index, or queries that test just the first column, the first two columns, the first three columns, and so on. If you specify the columns in the right order in the index definition, a single composite index can speed up several kinds of queries on the same table.

A multiple-column index can be considered a sorted array, the rows of which contain values that are created by concatenating the values of the indexed columns.

Note

As an alternative to a composite index, you can introduce a column that is “hashed” based on information from other columns. If this column is short, reasonably unique, and indexed, it might be faster than a “wide” index on many columns. In MySQL, it is very easy to use this extra column:

```
SELECT * FROM tbl_name
  WHERE hash_col=MD5(CONCAT(val1,val2))
  AND col1=val1 AND col2=val2;
```

Suppose that a table has the following specification:

```
CREATE TABLE test (
    id         INT NOT NULL,
    last_name  CHAR(30) NOT NULL,
    first_name CHAR(30) NOT NULL,
    PRIMARY KEY (id),
    INDEX name (last_name,first_name)
);
```

The `name` index is an index over the `last_name` and `first_name` columns. The index can be used for lookups in queries that specify values in a known range for combinations of `last_name` and `first_name` values. It can also be used for queries that specify just a `last_name` value because that column is a leftmost prefix of the index (as described later in this section). Therefore, the `name` index is used for lookups in the following queries:

```
SELECT * FROM test WHERE last_name='Jones';

SELECT * FROM test
  WHERE last_name='Jones' AND first_name='John';

SELECT * FROM test
  WHERE last_name='Jones'
  AND (first_name='John' OR first_name='Jon');

SELECT * FROM test
  WHERE last_name='Jones'
  AND first_name >='M' AND first_name < 'N';
```

However, the `name` index is *not* used for lookups in the following queries:

```
SELECT * FROM test WHERE first_name='John';

SELECT * FROM test
  WHERE last_name='Jones' OR first_name='John';
```

Suppose that you issue the following `SELECT` statement:

```
SELECT * FROM tbl_name
  WHERE col1=val1 AND col2=val2;
```

If a multiple-column index exists on `col1` and `col2`, the appropriate rows can be fetched directly. If separate single-column indexes exist on `col1` and `col2`, the optimizer attempts to use the Index Merge optimization (see Section 8.2.1.3, “Index Merge Optimization”), or attempts to find the most restrictive index by deciding which index excludes more rows and using that index to fetch the rows.

If the table has a multiple-column index, any leftmost prefix of the index can be used by the optimizer to look up rows. For example, if you have a three-column index on `(col1, col2, col3)`, you have indexed search capabilities on `(col1)`, `(col1, col2)`, and `(col1, col2, col3)`.

MySQL cannot use the index to perform lookups if the columns do not form a leftmost prefix of the index. Suppose that you have the `SELECT` statements shown here:

```
SELECT * FROM tbl_name WHERE col1=val1;
SELECT * FROM tbl_name WHERE col1=val1 AND col2=val2;

SELECT * FROM tbl_name WHERE col2=val2;
SELECT * FROM tbl_name WHERE col2=val2 AND col3=val3;
```

If an index exists on `(col1, col2, col3)`, only the first two queries use the index. The third and fourth queries do involve indexed columns, but do not use an index to perform lookups because `(col2)` and `(col2, col3)` are not leftmost prefixes of `(col1, col2, col3)`.

8.3.6 Verifying Index Usage

Always check whether all your queries really use the indexes that you have created in the tables. Use the `EXPLAIN` statement, as described in Section 8.8.1, “Optimizing Queries with EXPLAIN”.

8.3.7 InnoDB and MyISAM Index Statistics Collection

Storage engines collect statistics about tables for use by the optimizer. Table statistics are based on value groups, where a value group is a set of rows with the same key prefix value. For optimizer purposes, an important statistic is the average value group size.

MySQL uses the average value group size in the following ways:

- To estimate how many rows must be read for each `ref` access
- To estimate how many rows a partial join produces; that is, the number of rows that an operation of this form produces:

```
(...) JOIN tbl_name ON tbl_name.key = expr
```

As the average value group size for an index increases, the index is less useful for those two purposes because the average number of rows per lookup increases: For the index to be good for optimization purposes, it is best that each index value target a small number of rows in the table. When a given index value yields a large number of rows, the index is less useful and MySQL is less likely to use it.

The average value group size is related to table cardinality, which is the number of value groups. The `SHOW INDEX` statement displays a cardinality value based on *`N/S`*, where *`N`* is the number of rows in the table and *`S`* is the average value group size. That ratio yields an approximate number of value groups in the table.

For a join based on the `<=>` comparison operator, `NULL` is not treated differently from any other value: `NULL <=> NULL`, just as *`N <=> N`* for any other *`N`*.

However, for a join based on the `=` operator, `NULL` is different from non-`NULL` values: *`expr1 = expr2`* is not true when *`expr1`* or *`expr2`* (or both) are `NULL`. This affects `ref` accesses for comparisons of the form *`tbl_name.key = expr`*: MySQL does not access the table if the current value of *`expr`* is `NULL`, because the comparison cannot be true.

For `=` comparisons, it does not matter how many `NULL` values are in the table. For optimization purposes, the relevant value is the average size of the non-`NULL` value groups. However, MySQL does not currently enable that average size to be collected or used.

For `InnoDB` and `MyISAM` tables, you have some control over collection of table statistics by means of the `innodb_stats_method` and `myisam_stats_method` system variables, respectively. These variables have three possible values, which differ as follows:

- When the variable is set to `nulls_equal`, all `NULL` values are treated as identical (that is, they all form a single value group).

  If the `NULL` value group size is much higher than the average non-`NULL` value group size, this method skews the average value group size upward. This makes index appear to the optimizer to be less useful than it really is for joins that look for non-`NULL` values. Consequently, the `nulls_equal` method may cause the optimizer not to use the index for `ref` accesses when it should.

- When the variable is set to `nulls_unequal`, `NULL` values are not considered the same. Instead, each `NULL` value forms a separate value group of size 1.

  If you have many `NULL` values, this method skews the average value group size downward. If the average non-`NULL` value group size is large, counting `NULL` values each as a group of size 1 causes the optimizer to overestimate the value of the index for joins that look for non-`NULL` values.

Consequently, the `nulls_unequal` method may cause the optimizer to use this index for `ref` lookups when other methods may be better.

- When the variable is set to `nulls_ignored`, `NULL` values are ignored.

If you tend to use many joins that use `<=>` rather than `=`, `NULL` values are not special in comparisons and one `NULL` is equal to another. In this case, `nulls_equal` is the appropriate statistics method.

The `innodb_stats_method` system variable has a global value; the `myisam_stats_method` system variable has both global and session values. Setting the global value affects statistics collection for tables from the corresponding storage engine. Setting the session value affects statistics collection only for the current client connection. This means that you can force a table's statistics to be regenerated with a given method without affecting other clients by setting the session value of `myisam_stats_method`.

To regenerate `MyISAM` table statistics, you can use any of the following methods:

- Execute `myisamchk --stats_method=method_name --analyze`
- Change the table to cause its statistics to go out of date (for example, insert a row and then delete it), and then set `myisam_stats_method` and issue an `ANALYZE TABLE` statement

Some caveats regarding the use of `innodb_stats_method` and `myisam_stats_method`:

- You can force table statistics to be collected explicitly, as just described. However, MySQL may also collect statistics automatically. For example, if during the course of executing statements for a table, some of those statements modify the table, MySQL may collect statistics. (This may occur for bulk inserts or deletes, or some `ALTER TABLE` statements, for example.) If this happens, the statistics are collected using whatever value `innodb_stats_method` or `myisam_stats_method` has at the time. Thus, if you collect statistics using one method, but the system variable is set to the other method when a table's statistics are collected automatically later, the other method is used.
- There is no way to tell which method was used to generate statistics for a given table.
- These variables apply only to `InnoDB` and `MyISAM` tables. Other storage engines have only one method for collecting table statistics. Usually it is closer to the `nulls_equal` method.

8.3.8 Comparison of B-Tree and Hash Indexes

Understanding the B-tree and hash data structures can help predict how different queries perform on different storage engines that use these data structures in their indexes, particularly for the `MEMORY` storage engine that lets you choose B-tree or hash indexes.

- B-Tree Index Characteristics
- Hash Index Characteristics

B-Tree Index Characteristics

A B-tree index can be used for column comparisons in expressions that use the `=`, `>`, `>=`, `<`, `<=`, or `BETWEEN` operators. The index also can be used for `LIKE` comparisons if the argument to `LIKE` is a constant string that does not start with a wildcard character. For example, the following `SELECT` statements use indexes:

```
SELECT * FROM tbl_name WHERE key_col LIKE 'Patrick%';
SELECT * FROM tbl_name WHERE key_col LIKE 'Pat%_ck%';
```

In the first statement, only rows with `'Patrick' <= key_col < 'Patricl'` are considered. In the second statement, only rows with `'Pat' <= key_col < 'Pau'` are considered.

The following `SELECT` statements do not use indexes:

```
SELECT * FROM tbl_name WHERE key_col LIKE '%Patrick%';
```

```
SELECT * FROM tbl_name WHERE key_col LIKE other_col;
```

In the first statement, the `LIKE` value begins with a wildcard character. In the second statement, the `LIKE` value is not a constant.

If you use `... LIKE '%string%'` and *`string`* is longer than three characters, MySQL uses the *Turbo Boyer-Moore algorithm* to initialize the pattern for the string and then uses this pattern to perform the search more quickly.

A search using *`col_name`* `IS NULL` employs indexes if *`col_name`* is indexed.

Any index that does not span all `AND` levels in the `WHERE` clause is not used to optimize the query. In other words, to be able to use an index, a prefix of the index must be used in every `AND` group.

The following `WHERE` clauses use indexes:

```
... WHERE index_part1=1 AND index_part2=2 AND other_column=3

    /* index = 1 OR index = 2 */
... WHERE index=1 OR A=10 AND index=2

    /* optimized like "index_part1='hello'" */
... WHERE index_part1='hello' AND index_part3=5

    /* Can use index on index1 but not on index2 or index3 */
... WHERE index1=1 AND index2=2 OR index1=3 AND index3=3;
```

These `WHERE` clauses do *not* use indexes:

```
    /* index_part1 is not used */
... WHERE index_part2=1 AND index_part3=2

    /*  Index is not used in both parts of the WHERE clause  */
... WHERE index=1 OR A=10

    /* No index spans all rows  */
... WHERE index_part1=1 OR index_part2=10
```

Sometimes MySQL does not use an index, even if one is available. One circumstance under which this occurs is when the optimizer estimates that using the index would require MySQL to access a very large percentage of the rows in the table. (In this case, a table scan is likely to be much faster because it requires fewer seeks.) However, if such a query uses `LIMIT` to retrieve only some of the rows, MySQL uses an index anyway, because it can much more quickly find the few rows to return in the result.

Hash Index Characteristics

Hash indexes have somewhat different characteristics from those just discussed:

- They are used only for equality comparisons that use the `=` or `<=>` operators (but are *very* fast). They are not used for comparison operators such as `<` that find a range of values. Systems that rely on this type of single-value lookup are known as “key-value stores”; to use MySQL for such applications, use hash indexes wherever possible.
- The optimizer cannot use a hash index to speed up `ORDER BY` operations. (This type of index cannot be used to search for the next entry in order.)
- MySQL cannot determine approximately how many rows there are between two values (this is used by the range optimizer to decide which index to use). This may affect some queries if you change a `MyISAM` or `InnoDB` table to a hash-indexed `MEMORY` table.
- Only whole keys can be used to search for a row. (With a B-tree index, any leftmost prefix of the key can be used to find rows.)

8.3.9 Use of Index Extensions

InnoDB automatically extends each secondary index by appending the primary key columns to it. Consider this table definition:

```
CREATE TABLE t1 (
  i1 INT NOT NULL DEFAULT 0,
  i2 INT NOT NULL DEFAULT 0,
  d DATE DEFAULT NULL,
  PRIMARY KEY (i1, i2),
  INDEX k_d (d)
) ENGINE = InnoDB;
```

This table defines the primary key on columns (i1, i2). It also defines a secondary index k_d on column (d), but internally InnoDB extends this index and treats it as columns (d, i1, i2).

The optimizer takes into account the primary key columns of the extended secondary index when determining how and whether to use that index. This can result in more efficient query execution plans and better performance.

The optimizer can use extended secondary indexes for ref, range, and index_merge index access, for Loose Index Scan access, for join and sorting optimization, and for MIN()/MAX() optimization.

The following example shows how execution plans are affected by whether the optimizer uses extended secondary indexes. Suppose that t1 is populated with these rows:

```
INSERT INTO t1 VALUES
(1, 1, '1998-01-01'), (1, 2, '1999-01-01'),
(1, 3, '2000-01-01'), (1, 4, '2001-01-01'),
(1, 5, '2002-01-01'), (2, 1, '1998-01-01'),
(2, 2, '1999-01-01'), (2, 3, '2000-01-01'),
(2, 4, '2001-01-01'), (2, 5, '2002-01-01'),
(3, 1, '1998-01-01'), (3, 2, '1999-01-01'),
(3, 3, '2000-01-01'), (3, 4, '2001-01-01'),
(3, 5, '2002-01-01'), (4, 1, '1998-01-01'),
(4, 2, '1999-01-01'), (4, 3, '2000-01-01'),
(4, 4, '2001-01-01'), (4, 5, '2002-01-01'),
(5, 1, '1998-01-01'), (5, 2, '1999-01-01'),
(5, 3, '2000-01-01'), (5, 4, '2001-01-01'),
(5, 5, '2002-01-01');
```

Now consider this query:

```
EXPLAIN SELECT COUNT(*) FROM t1 WHERE i1 = 3 AND d = '2000-01-01'
```

The execution plan depends on whether the extended index is used.

When the optimizer does not consider index extensions, it treats the index k_d as only (d). EXPLAIN for the query produces this result:

```
mysql> EXPLAIN SELECT COUNT(*) FROM t1 WHERE i1 = 3 AND d = '2000-01-01'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: t1
         type: ref
possible_keys: PRIMARY,k_d
          key: k_d
      key_len: 4
          ref: const
         rows: 5
        Extra: Using where; Using index
```

When the optimizer takes index extensions into account, it treats k_d as (d, i1, i2). In this case, it can use the leftmost index prefix (d, i1) to produce a better execution plan:

```
mysql> EXPLAIN SELECT COUNT(*) FROM t1 WHERE i1 = 3 AND d = '2000-01-01'\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
```

```
        table: t1
         type: ref
possible_keys: PRIMARY,k_d
          key: k_d
      key_len: 8
          ref: const,const
         rows: 1
        Extra: Using index
```

In both cases, `key` indicates that the optimizer uses secondary index `k_d` but the `EXPLAIN` output shows these improvements from using the extended index:

- `key_len` goes from 4 bytes to 8 bytes, indicating that key lookups use columns `d` and `i1`, not just `d`.
- The `ref` value changes from `const` to `const,const` because the key lookup uses two key parts, not one.
- The `rows` count decreases from 5 to 1, indicating that `InnoDB` should need to examine fewer rows to produce the result.
- The `Extra` value changes from `Using where; Using index` to `Using index`. This means that rows can be read using only the index, without consulting columns in the data row.

Differences in optimizer behavior for use of extended indexes can also be seen with `SHOW STATUS`:

```
FLUSH TABLE t1;
FLUSH STATUS;
SELECT COUNT(*) FROM t1 WHERE i1 = 3 AND d = '2000-01-01';
SHOW STATUS LIKE 'handler_read%'
```

The preceding statements include `FLUSH TABLES` and `FLUSH STATUS` to flush the table cache and clear the status counters.

Without index extensions, `SHOW STATUS` produces this result:

```
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| Handler_read_first    | 0     |
| Handler_read_key      | 1     |
| Handler_read_last     | 0     |
| Handler_read_next     | 5     |
| Handler_read_prev     | 0     |
| Handler_read_rnd      | 0     |
| Handler_read_rnd_next | 0     |
+-----------------------+-------+
```

With index extensions, `SHOW STATUS` produces this result. The `Handler_read_next` value decreases from 5 to 1, indicating more efficient use of the index:

```
+-----------------------+-------+
| Variable_name         | Value |
+-----------------------+-------+
| Handler_read_first    | 0     |
| Handler_read_key      | 1     |
| Handler_read_last     | 0     |
| Handler_read_next     | 1     |
| Handler_read_prev     | 0     |
| Handler_read_rnd      | 0     |
| Handler_read_rnd_next | 0     |
+-----------------------+-------+
```

The `use_index_extensions` flag of the `optimizer_switch` system variable permits control over whether the optimizer takes the primary key columns into account when determining how to use an `InnoDB` table's secondary indexes. By default, `use_index_extensions` is enabled. To check whether disabling use of index extensions can improve performance, use this statement:

```
SET optimizer_switch = 'use_index_extensions=off';
```

Use of index extensions by the optimizer is subject to the usual limits on the number of key parts in an index (16) and the maximum key length (3072 bytes).

8.3.10 Indexed Lookups from TIMESTAMP Columns

Temporal values are stored in `TIMESTAMP` columns as UTC values, and values inserted into and retrieved from `TIMESTAMP` columns are converted between the session time zone and UTC. (This is the same type of conversion performed by the `CONVERT_TZ()` function. If the session time zone is UTC, there is effectively no time zone conversion.)

Due to conventions for local time zone changes such as Daylight Saving Time (DST), conversions between UTC and non-UTC time zones are not one-to-one in both directions. UTC values that are distinct may not be distinct in another time zone. The following example shows distinct UTC values that become identical in a non-UTC time zone:

```
mysql> CREATE TABLE tstable (ts TIMESTAMP);
mysql> SET time_zone = 'UTC'; -- insert UTC values
mysql> INSERT INTO tstable VALUES
       ('2018-10-28 00:30:00'),
       ('2018-10-28 01:30:00');
mysql> SELECT ts FROM tstable;
+---------------------+
| ts                  |
+---------------------+
| 2018-10-28 00:30:00 |
| 2018-10-28 01:30:00 |
+---------------------+
mysql> SET time_zone = 'MET'; -- retrieve non-UTC values
mysql> SELECT ts FROM tstable;
+---------------------+
| ts                  |
+---------------------+
| 2018-10-28 02:30:00 |
| 2018-10-28 02:30:00 |
+---------------------+
```

Note

To use named time zones such as `'MET'` or `'Europe/Amsterdam'`, the time zone tables must be properly set up. For instructions, see Section 5.1.13, “MySQL Server Time Zone Support”.

You can see that the two distinct UTC values are the same when converted to the `'MET'` time zone. This phenomenon can lead to different results for a given `TIMESTAMP` column query, depending on whether the optimizer uses an index to execute the query.

Suppose that a query selects values from the table shown earlier using a `WHERE` clause to search the `ts` column for a single specific value such as a user-provided timestamp literal:

```
SELECT ts FROM tstable
WHERE ts = 'literal';
```

Suppose further that the query executes under these conditions:

- The session time zone is not UTC and has a DST shift. For example:

  ```
  SET time_zone = 'MET';
  ```

- Unique UTC values stored in the `TIMESTAMP` column are not unique in the session time zone due to DST shifts. (The example shown earlier illustrates how this can occur.)

- The query specifies a search value that is within the hour of entry into DST in the session time zone.

Under those conditions, the comparison in the `WHERE` clause occurs in different ways for nonindexed and indexed lookups and leads to different results:

- If there is no index or the optimizer cannot use it, comparisons occur in the session time zone. The optimizer performs a table scan in which it retrieves each `ts` column value, converts it from UTC to the session time zone, and compares it to the search value (also interpreted in the session time zone):

```
mysql> SELECT ts FROM tstable
       WHERE ts = '2018-10-28 02:30:00';
+---------------------+
| ts                  |
+---------------------+
| 2018-10-28 02:30:00 |
| 2018-10-28 02:30:00 |
+---------------------+
```

  Because the stored `ts` values are converted to the session time zone, it is possible for the query to return two timestamp values that are distinct as UTC values but equal in the session time zone: One value that occurs before the DST shift when clocks are changed, and one value that was occurs after the DST shift.

- If there is a usable index, comparisons occur in UTC. The optimizer performs an index scan, first converting the search value from the session time zone to UTC, then comparing the result to the UTC index entries:

```
mysql> ALTER TABLE tstable ADD INDEX (ts);
mysql> SELECT ts FROM tstable
       WHERE ts = '2018-10-28 02:30:00';
+---------------------+
| ts                  |
+---------------------+
| 2018-10-28 02:30:00 |
+---------------------+
```

  In this case, the (converted) search value is matched only to index entries, and because the index entries for the distinct stored UTC values are also distinct, the search value can match only one of them.

Due to different optimizer operation for nonindexed and indexed lookups, the query produces different results in each case. The result from the nonindexed lookup returns all values that match in the session time zone. The indexed lookup cannot do so:

- It is performed within the storage engine, which knows only about UTC values.
- For the two distinct session time zone values that map to the same UTC value, the indexed lookup matches only the corresponding UTC index entry and returns only a single row.

In the preceding discussion, the data set stored in `tstable` happens to consist of distinct UTC values. In such cases, all index-using queries of the form shown match at most one index entry.

If the index is not `UNIQUE`, it is possible for the table (and the index) to store multiple instances of a given UTC value. For example, the `ts` column might contain multiple instances of the UTC value `'2018-10-28 00:30:00'`. In this case, the index-using query would return each of them (converted to the MET value `'2018-10-28 02:30:00'` in the result set). It remains true that index-using queries match the converted search value to a single value in the UTC index entries, rather than matching multiple UTC values that convert to the search value in the session time zone.

If it is important to return all `ts` values that match in the session time zone, the workaround is to suppress use of the index with an `IGNORE INDEX` hint:

```
mysql> SELECT ts FROM tstable
       IGNORE INDEX (ts)
       WHERE ts = '2018-10-28 02:30:00';
+---------------------+
| ts                  |
+---------------------+
| 2018-10-28 02:30:00 |
```

```
| 2018-10-28 02:30:00 |
+---------------------+
```

The same lack of one-to-one mapping for time zone conversions in both directions occurs in other contexts as well, such as conversions performed with the `FROM_UNIXTIME()` and `UNIX_TIMESTAMP()` functions. See Section 12.7, “Date and Time Functions”.

8.4 Optimizing Database Structure

In your role as a database designer, look for the most efficient way to organize your schemas, tables, and columns. As when tuning application code, you minimize I/O, keep related items together, and plan ahead so that performance stays high as the data volume increases. Starting with an efficient database design makes it easier for team members to write high-performing application code, and makes the database likely to endure as applications evolve and are rewritten.

8.4.1 Optimizing Data Size

Design your tables to minimize their space on the disk. This can result in huge improvements by reducing the amount of data written to and read from disk. Smaller tables normally require less main memory while their contents are being actively processed during query execution. Any space reduction for table data also results in smaller indexes that can be processed faster.

MySQL supports many different storage engines (table types) and row formats. For each table, you can decide which storage and indexing method to use. Choosing the proper table format for your application can give you a big performance gain. See Chapter 14, *The InnoDB Storage Engine*, and Chapter 15, *Alternative Storage Engines*.

You can get better performance for a table and minimize storage space by using the techniques listed here:

- Table Columns
- Row Format
- Indexes
- Joins
- Normalization

Table Columns

- Use the most efficient (smallest) data types possible. MySQL has many specialized types that save disk space and memory. For example, use the smaller integer types if possible to get smaller tables. `MEDIUMINT` is often a better choice than `INT` because a `MEDIUMINT` column uses 25% less space.
- Declare columns to be `NOT NULL` if possible. It makes SQL operations faster, by enabling better use of indexes and eliminating overhead for testing whether each value is `NULL`. You also save some storage space, one bit per column. If you really need `NULL` values in your tables, use them. Just avoid the default setting that allows `NULL` values in every column.

Row Format

- In MySQL 5.6, `InnoDB` tables use the `COMPACT` row storage format (`ROW_FORMAT=COMPACT`) by default. The compact family of row formats, which includes `COMPACT`, `DYNAMIC`, and `COMPRESSED`, decreases row storage space at the cost of increasing CPU use for some operations. If your workload is a typical one that is limited by cache hit rates and disk speed it is likely to be faster. If it is a rare case that is limited by CPU speed, it might be slower.

  The compact family of row formats also optimizes `CHAR` column storage when using a variable-length character set such as `utf8mb3` or `utf8mb4`. With `ROW_FORMAT=REDUNDANT`, `CHAR(N)`

occupies *N* × the maximum byte length of the character set. Many languages can be written primarily using single-byte `utf8` characters, so a fixed storage length often wastes space. With the compact family of rows formats, `InnoDB` allocates a variable amount of storage in the range of *N* to *N* × the maximum byte length of the character set for these columns by stripping trailing spaces. The minimum storage length is *N* bytes to facilitate in-place updates in typical cases. For more information, see Section 14.11, “InnoDB Row Formats”.

- To minimize space even further by storing table data in compressed form, specify `ROW_FORMAT=COMPRESSED` when creating `InnoDB` tables, or run the `myisampack` command on an existing `MyISAM` table. (`InnoDB` compressed tables are readable and writable, while `MyISAM` compressed tables are read-only.)
- For `MyISAM` tables, if you do not have any variable-length columns (`VARCHAR`, `TEXT`, or `BLOB` columns), a fixed-size row format is used. This is faster but may waste some space. See Section 15.2.3, “MyISAM Table Storage Formats”. You can hint that you want to have fixed length rows even if you have `VARCHAR` columns with the `CREATE TABLE` option `ROW_FORMAT=FIXED`.

Indexes

- The primary index of a table should be as short as possible. This makes identification of each row easy and efficient. For `InnoDB` tables, the primary key columns are duplicated in each secondary index entry, so a short primary key saves considerable space if you have many secondary indexes.
- Create only the indexes that you need to improve query performance. Indexes are good for retrieval, but slow down insert and update operations. If you access a table mostly by searching on a combination of columns, create a single composite index on them rather than a separate index for each column. The first part of the index should be the column most used. If you *always* use many columns when selecting from the table, the first column in the index should be the one with the most duplicates, to obtain better compression of the index.
- If it is very likely that a long string column has a unique prefix on the first number of characters, it is better to index only this prefix, using MySQL's support for creating an index on the leftmost part of the column (see Section 13.1.13, “CREATE INDEX Statement”). Shorter indexes are faster, not only because they require less disk space, but because they also give you more hits in the index cache, and thus fewer disk seeks. See Section 5.1.1, “Configuring the Server”.

Joins

- In some circumstances, it can be beneficial to split into two a table that is scanned very often. This is especially true if it is a dynamic-format table and it is possible to use a smaller static format table that can be used to find the relevant rows when scanning the table.
- Declare columns with identical information in different tables with identical data types, to speed up joins based on the corresponding columns.
- Keep column names simple, so that you can use the same name across different tables and simplify join queries. For example, in a table named `customer`, use a column name of `name` instead of `customer_name`. To make your names portable to other SQL servers, consider keeping them shorter than 18 characters.

Normalization

- Normally, try to keep all data nonredundant (observing what is referred to in database theory as *third normal form*). Instead of repeating lengthy values such as names and addresses, assign them unique IDs, repeat these IDs as needed across multiple smaller tables, and join the tables in queries by referencing the IDs in the join clause.
- If speed is more important than disk space and the maintenance costs of keeping multiple copies of data, for example in a business intelligence scenario where you analyze all the data from large tables, you can relax the normalization rules, duplicating information or creating summary tables to gain more speed.

8.4.2 Optimizing MySQL Data Types

8.4.2.1 Optimizing for Numeric Data

- For unique IDs or other values that can be represented as either strings or numbers, prefer numeric columns to string columns. Since large numeric values can be stored in fewer bytes than the corresponding strings, it is faster and takes less memory to transfer and compare them.
- If you are using numeric data, it is faster in many cases to access information from a database (using a live connection) than to access a text file. Information in the database is likely to be stored in a more compact format than in the text file, so accessing it involves fewer disk accesses. You also save code in your application because you can avoid parsing the text file to find line and column boundaries.

8.4.2.2 Optimizing for Character and String Types

For character and string columns, follow these guidelines:

- Use binary collation order for fast comparison and sort operations, when you do not need language-specific collation features. You can use the `BINARY` operator to use binary collation within a particular query.
- When comparing values from different columns, declare those columns with the same character set and collation wherever possible, to avoid string conversions while running the query.
- For column values less than 8KB in size, use binary `VARCHAR` instead of `BLOB`. The `GROUP BY` and `ORDER BY` clauses can generate temporary tables, and these temporary tables can use the `MEMORY` storage engine if the original table does not contain any `BLOB` columns.
- If a table contains string columns such as name and address, but many queries do not retrieve those columns, consider splitting the string columns into a separate table and using join queries with a foreign key when necessary. When MySQL retrieves any value from a row, it reads a data block containing all the columns of that row (and possibly other adjacent rows). Keeping each row small, with only the most frequently used columns, allows more rows to fit in each data block. Such compact tables reduce disk I/O and memory usage for common queries.
- When you use a randomly generated value as a primary key in an `InnoDB` table, prefix it with an ascending value such as the current date and time if possible. When consecutive primary values are physically stored near each other, `InnoDB` can insert and retrieve them faster.
- See Section 8.4.2.1, “Optimizing for Numeric Data” for reasons why a numeric column is usually preferable to an equivalent string column.

8.4.2.3 Optimizing for BLOB Types

- When storing a large blob containing textual data, consider compressing it first. Do not use this technique when the entire table is compressed by `InnoDB` or `MyISAM`.
- For a table with several columns, to reduce memory requirements for queries that do not use the BLOB column, consider splitting the BLOB column into a separate table and referencing it with a join query when needed.
- Since the performance requirements to retrieve and display a BLOB value might be very different from other data types, you could put the BLOB-specific table on a different storage device or even a separate database instance. For example, to retrieve a BLOB might require a large sequential disk read that is better suited to a traditional hard drive than to an SSD device.
- See Section 8.4.2.2, “Optimizing for Character and String Types” for reasons why a binary `VARCHAR` column is sometimes preferable to an equivalent BLOB column.
- Rather than testing for equality against a very long text string, you can store a hash of the column value in a separate column, index that column, and test the hashed value in queries. (Use the `MD5()`

or `CRC32()` function to produce the hash value.) Since hash functions can produce duplicate results for different inputs, you still include a clause `AND blob_column = long_string_value` in the query to guard against false matches; the performance benefit comes from the smaller, easily scanned index for the hashed values.

8.4.2.4 Using PROCEDURE ANALYSE

```
ANALYSE([max_elements[,max_memory]])
```

`ANALYSE()` examines the result from a query and returns an analysis of the results that suggests optimal data types for each column that may help reduce table sizes. To obtain this analysis, append `PROCEDURE ANALYSE` to the end of a `SELECT` statement:

```
SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([max_elements,[max_memory]])
```

For example:

```
SELECT col1, col2 FROM table1 PROCEDURE ANALYSE(10, 2000);
```

The results show some statistics for the values returned by the query, and propose an optimal data type for the columns. This can be helpful for checking your existing tables, or after importing new data. You may need to try different settings for the arguments so that `PROCEDURE ANALYSE()` does not suggest the `ENUM` data type when it is not appropriate.

The arguments are optional and are used as follows:

- *`max_elements`* (default 256) is the maximum number of distinct values that `ANALYSE()` notices per column. This is used by `ANALYSE()` to check whether the optimal data type should be of type `ENUM`; if there are more than *`max_elements`* distinct values, then `ENUM` is not a suggested type.
- *`max_memory`* (default 8192) is the maximum amount of memory that `ANALYSE()` should allocate per column while trying to find all distinct values.

A `PROCEDURE` clause is not permitted in a `UNION` statement.

8.4.3 Optimizing for Many Tables

Some techniques for keeping individual queries fast involve splitting data across many tables. When the number of tables runs into the thousands or even millions, the overhead of dealing with all these tables becomes a new performance consideration.

8.4.3.1 How MySQL Opens and Closes Tables

When you execute a `mysqladmin status` command, you should see something like this:

```
Uptime: 426 Running threads: 1 Questions: 11082
Reloads: 1 Open tables: 12
```

The `Open tables` value of 12 can be somewhat puzzling if you have fewer than 12 tables.

MySQL is multithreaded, so there may be many clients issuing queries for a given table simultaneously. To minimize the problem with multiple client sessions having different states on the same table, the table is opened independently by each concurrent session. This uses additional memory but normally increases performance. With `MyISAM` tables, one extra file descriptor is required for the data file for each client that has the table open. (By contrast, the index file descriptor is shared between all sessions.)

The `table_open_cache` and `max_connections` system variables affect the maximum number of files the server keeps open. If you increase one or both of these values, you may run up against a limit imposed by your operating system on the per-process number of open file descriptors. Many operating systems permit you to increase the open-files limit, although the method varies widely from system to

system. Consult your operating system documentation to determine whether it is possible to increase the limit and how to do so.

`table_open_cache` is related to `max_connections`. For example, for 200 concurrent running connections, specify a table cache size of at least `200 * N`, where `N` is the maximum number of tables per join in any of the queries which you execute. You must also reserve some extra file descriptors for temporary tables and files.

Make sure that your operating system can handle the number of open file descriptors implied by the `table_open_cache` setting. If `table_open_cache` is set too high, MySQL may run out of file descriptors and exhibit symptoms such as refusing connections or failing to perform queries.

Also take into account that the `MyISAM` storage engine needs two file descriptors for each unique open table. For a partitioned `MyISAM` table, two file descriptors are required for each partition of the opened table. (When `MyISAM` opens a partitioned table, it opens every partition of this table, whether or not a given partition is actually used. See MyISAM and partition file descriptor usage.) To increase the number of file descriptors available to MySQL, set the `open_files_limit` system variable. See Section B.3.2.17, “File Not Found and Similar Errors”.

The cache of open tables is kept at a level of `table_open_cache` entries. The server autosizes the cache size at startup. To set the size explicitly, set the `table_open_cache` system variable at startup. MySQL may temporarily open more tables than this to execute queries, as described later in this section.

MySQL closes an unused table and removes it from the table cache under the following circumstances:

- When the cache is full and a thread tries to open a table that is not in the cache.
- When the cache contains more than `table_open_cache` entries and a table in the cache is no longer being used by any threads.
- When a table-flushing operation occurs. This happens when someone issues a `FLUSH TABLES` statement or executes a `mysqladmin flush-tables` or `mysqladmin refresh` command.

When the table cache fills up, the server uses the following procedure to locate a cache entry to use:

- Tables not currently in use are released, beginning with the table least recently used.
- If a new table must be opened, but the cache is full and no tables can be released, the cache is temporarily extended as necessary. When the cache is in a temporarily extended state and a table goes from a used to unused state, the table is closed and released from the cache.

A `MyISAM` table is opened for each concurrent access. This means the table needs to be opened twice if two threads access the same table or if a thread accesses the table twice in the same query (for example, by joining the table to itself). Each concurrent open requires an entry in the table cache. The first open of any `MyISAM` table takes two file descriptors: one for the data file and one for the index file. Each additional use of the table takes only one file descriptor for the data file. The index file descriptor is shared among all threads.

If you are opening a table with the `HANDLER tbl_name OPEN` statement, a dedicated table object is allocated for the thread. This table object is not shared by other threads and is not closed until the thread calls `HANDLER tbl_name CLOSE` or the thread terminates. When this happens, the table is put back in the table cache (if the cache is not full). See Section 13.2.4, “HANDLER Statement”.

To determine whether your table cache is too small, check the `Opened_tables` status variable, which indicates the number of table-opening operations since the server started:

```
mysql> SHOW GLOBAL STATUS LIKE 'Opened_tables';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Opened_tables | 2741  |
```

```
+---------------+-------+
```

If the value is very large or increases rapidly, even when you have not issued many `FLUSH TABLES` statements, increase the `table_open_cache` value at server startup.

8.4.3.2 Disadvantages of Creating Many Tables in the Same Database

If you have many `MyISAM` tables in the same database directory, open, close, and create operations are slow. If you execute `SELECT` statements on many different tables, there is a little overhead when the table cache is full, because for every table that has to be opened, another must be closed. You can reduce this overhead by increasing the number of entries permitted in the table cache.

8.4.4 Internal Temporary Table Use in MySQL

In some cases, the server creates internal temporary tables while processing statements. Users have no direct control over when this occurs.

The server creates temporary tables under conditions such as these:

- Evaluation of `UNION` statements.
- Evaluation of some views, such those that use the `TEMPTABLE` algorithm, `UNION`, or aggregation.
- Evaluation of derived tables (see Section 13.2.10.8, “Derived Tables”).
- Tables created for subquery or semijoin materialization (see Section 8.2.2, “Optimizing Subqueries and Derived Tables”).
- Evaluation of statements that contain an `ORDER BY` clause and a different `GROUP BY` clause, or for which the `ORDER BY` or `GROUP BY` contains columns from tables other than the first table in the join queue.
- Evaluation of `DISTINCT` combined with `ORDER BY` may require a temporary table.
- For queries that use the `SQL_SMALL_RESULT` modifier, MySQL uses an in-memory temporary table, unless the query also contains elements (described later) that require on-disk storage.
- To evaluate `INSERT ... SELECT` statements that select from and insert into the same table, MySQL creates an internal temporary table to hold the rows from the `SELECT`, then inserts those rows into the target table. See Section 13.2.5.1, “INSERT ... SELECT Statement”.
- Evaluation of multiple-table `UPDATE` statements.
- Evaluation of `GROUP_CONCAT()` or `COUNT(DISTINCT)` expressions.

To determine whether a statement requires a temporary table, use `EXPLAIN` and check the `Extra` column to see whether it says `Using temporary` (see Section 8.8.1, “Optimizing Queries with EXPLAIN”). `EXPLAIN` does not necessarily say `Using temporary` for derived or materialized temporary tables.

Some query conditions prevent the use of an in-memory temporary table, in which case the server uses an on-disk table instead:

- Presence of a `BLOB` or `TEXT` column in the table. This includes user-defined variables having a string value because they are treated as `BLOB` or `TEXT` columns, depending on whether their value is a binary or nonbinary string, respectively.
- Presence of any string column in a `GROUP BY` or `DISTINCT` clause larger than 512 bytes for binary strings or 512 characters for nonbinary strings.
- Presence of any string column with a maximum length larger than 512 (bytes for binary strings, characters for nonbinary strings) in the `SELECT` list, if `UNION` or `UNION ALL` is used.

- The `SHOW COLUMNS` and `DESCRIBE` statements use `BLOB` as the type for some columns, thus the temporary table used for the results is an on-disk table.
- Internal Temporary Table Storage Engine
- Internal Temporary Table Storage Format

Internal Temporary Table Storage Engine

An internal temporary table can be held in memory and processed by the `MEMORY` storage engine, or stored on disk and processed by the `MyISAM` storage engine.

If an internal temporary table is created as an in-memory table but becomes too large, MySQL automatically converts it to an on-disk table. The maximum size of an in-memory temporary table is defined by the `tmp_table_size` or `max_heap_table_size` value, whichever is smaller. This differs from `MEMORY` tables explicitly created with `CREATE TABLE`. For such tables, only the `max_heap_table_size` variable determines how large a table can grow, and there is no conversion to on-disk format.

When an internal temporary table is created in memory or on disk, the server increments the `Created_tmp_tables` value. When an internal temporary table is created on disk, the server increments the `Created_tmp_disk_tables` value. If too many internal temporary tables are created on disk, consider increasing the `tmp_table_size` and `max_heap_table_size` settings.

Internal Temporary Table Storage Format

In-memory temporary tables are managed by the `MEMORY` storage engine, which uses fixed-length row format. `VARCHAR` and `VARBINARY` column values are padded to the maximum column length, in effect storing them as `CHAR` and `BINARY` columns.

On-disk temporary tables are managed by the `MyISAM` storage engine using dynamic-width row format. Columns take only as much storage as needed, which reduces disk I/O and space requirements, and processing time compared to on-disk tables that use fixed-length rows.

For statements that initially create an internal temporary table in memory, then convert it to an on-disk table, better performance might be achieved by skipping the conversion step and creating the table on disk to begin with. The `big_tables` variable can be used to force disk storage of internal temporary tables.

8.4.5 Limits on Number of Databases and Tables

MySQL has no limit on the number of databases. The underlying file system may have a limit on the number of directories.

MySQL has no limit on the number of tables. The underlying file system may have a limit on the number of files that represent tables. Individual storage engines may impose engine-specific constraints. `InnoDB` permits up to 4 billion tables.

8.4.6 Limits on Table Size

The effective maximum table size for MySQL databases is usually determined by operating system constraints on file sizes, not by MySQL internal limits. For up-to-date information operating system file size limits, refer to the documentation specific to your operating system.

Windows users, please note that FAT and VFAT (FAT32) are *not* considered suitable for production use with MySQL. Use NTFS instead.

If you encounter a full-table error, there are several reasons why it might have occurred:

- The disk might be full.

- You are using `InnoDB` tables and have run out of room in an `InnoDB` tablespace file. The maximum tablespace size is also the maximum size for a table. For tablespace size limits, see Section 14.22, “InnoDB Limits”.

  Generally, partitioning of tables into multiple tablespace files is recommended for tables larger than 1TB in size.

- You have hit an operating system file size limit. For example, you are using `MyISAM` tables on an operating system that supports files only up to 2GB in size and you have hit this limit for the data file or index file.

- You are using a `MyISAM` table and the space required for the table exceeds what is permitted by the internal pointer size. `MyISAM` permits data and index files to grow up to 256TB by default, but this limit can be changed up to the maximum permissible size of 65,536TB ($256^7 - 1$ bytes).

  If you need a `MyISAM` table that is larger than the default limit and your operating system supports large files, the `CREATE TABLE` statement supports `AVG_ROW_LENGTH` and `MAX_ROWS` options. See Section 13.1.17, “CREATE TABLE Statement”. The server uses these options to determine how large a table to permit.

  If the pointer size is too small for an existing table, you can change the options with `ALTER TABLE` to increase a table's maximum permissible size. See Section 13.1.7, “ALTER TABLE Statement”.

  ```
  ALTER TABLE tbl_name MAX_ROWS=1000000000 AVG_ROW_LENGTH=nnn;
  ```

  You have to specify `AVG_ROW_LENGTH` only for tables with `BLOB` or `TEXT` columns; in this case, MySQL cannot optimize the space required based only on the number of rows.

  To change the default size limit for `MyISAM` tables, set the `myisam_data_pointer_size`, which sets the number of bytes used for internal row pointers. The value is used to set the pointer size for new tables if you do not specify the `MAX_ROWS` option. The value of `myisam_data_pointer_size` can be from 2 to 7. A value of 4 permits tables up to 4GB; a value of 6 permits tables up to 256TB.

  You can check the maximum data and index sizes by using this statement:

  ```
  SHOW TABLE STATUS FROM db_name LIKE 'tbl_name';
  ```

  You also can use `myisamchk -dv /path/to/table-index-file`. See Section 13.7.5, “SHOW Statements”, or Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”.

  Other ways to work around file-size limits for `MyISAM` tables are as follows:

  - If your large table is read only, you can use `myisampack` to compress it. `myisampack` usually compresses a table by at least 50%, so you can have, in effect, much bigger tables. `myisampack` also can merge multiple tables into a single table. See Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”.

  - MySQL includes a `MERGE` library that enables you to handle a collection of `MyISAM` tables that have identical structure as a single `MERGE` table. See Section 15.7, “The MERGE Storage Engine”.

- You are using the `MEMORY` (`HEAP`) storage engine; in this case you need to increase the value of the `max_heap_table_size` system variable. See Section 5.1.7, “Server System Variables”.

8.4.7 Limits on Table Column Count and Row Size

This section describes limits on the number of columns in tables and the size of individual rows.

- Column Count Limits
- Row Size Limits

Column Count Limits

MySQL has hard limit of 4096 columns per table, but the effective maximum may be less for a given table. The exact column limit depends on several factors:

- The maximum row size for a table constrains the number (and possibly size) of columns because the total length of all columns cannot exceed this size. See Row Size Limits.
- The storage requirements of individual columns constrain the number of columns that fit within a given maximum row size. Storage requirements for some data types depend on factors such as storage engine, storage format, and character set. See Section 11.6, “Data Type Storage Requirements”.
- Storage engines may impose additional restrictions that limit table column count. For example, `InnoDB` has a limit of 1017 columns per table. See Section 14.22, “InnoDB Limits”. For information about other storage engines, see Chapter 15, *Alternative Storage Engines*.
- Each table has an `.frm` file that contains the table definition. The definition affects the content of this file in ways that may affect the number of columns permitted in the table. See Limits Imposed by .frm File Structure.

Row Size Limits

The maximum row size for a given table is determined by several factors:

- The internal representation of a MySQL table has a maximum row size limit of 65,535 bytes, even if the storage engine is capable of supporting larger rows. `BLOB` and `TEXT` columns only contribute 9 to 12 bytes toward the row size limit because their contents are stored separately from the rest of the row.
- The maximum row size for an `InnoDB` table, which applies to data stored locally within a database page, is slightly less than half a page. For example, the maximum row size is slightly less than 8KB for the default 16KB `InnoDB` page size, which is defined by the `innodb_page_size` configuration option. See Section 14.22, “InnoDB Limits”.

  If a row containing variable-length columns exceeds the `InnoDB` maximum row size, `InnoDB` selects variable-length columns for external off-page storage until the row fits within the `InnoDB` row size limit. The amount of data stored locally for variable-length columns that are stored off-page differs by row format. For more information, see Section 14.11, “InnoDB Row Formats”.
- Different storage formats use different amounts of page header and trailer data, which affects the amount of storage available for rows.
  - For information about `InnoDB` row formats, see Section 14.11, “InnoDB Row Formats”.
  - For information about `MyISAM` storage formats, see Section 15.2.3, “MyISAM Table Storage Formats”.

Row Size Limit Examples

- The MySQL maximum row size limit of 65,535 bytes is demonstrated in the following `InnoDB` and `MyISAM` examples. The limit is enforced regardless of storage engine, even though the storage engine may be capable of supporting larger rows.

```
mysql> CREATE TABLE t (a VARCHAR(10000), b VARCHAR(10000),
       c VARCHAR(10000), d VARCHAR(10000), e VARCHAR(10000),
       f VARCHAR(10000), g VARCHAR(6000)) ENGINE=InnoDB CHARACTER SET latin1;
ERROR 1118 (42000): Row size too large. The maximum row size for the used
table type, not counting BLOBs, is 65535. This includes storage overhead,
check the manual. You have to change some columns to TEXT or BLOBs
```

```
mysql> CREATE TABLE t (a VARCHAR(10000), b VARCHAR(10000),
       c VARCHAR(10000), d VARCHAR(10000), e VARCHAR(10000),
       f VARCHAR(10000), g VARCHAR(6000)) ENGINE=MyISAM CHARACTER SET latin1;
```

```
ERROR 1118 (42000): Row size too large. The maximum row size for the used
table type, not counting BLOBs, is 65535. This includes storage overhead,
check the manual. You have to change some columns to TEXT or BLOBs
```

In the following `MyISAM` example, changing a column to `TEXT` avoids the 65,535-byte row size limit and permits the operation to succeed because `BLOB` and `TEXT` columns only contribute 9 to 12 bytes toward the row size.

```
mysql> CREATE TABLE t (a VARCHAR(10000), b VARCHAR(10000),
       c VARCHAR(10000), d VARCHAR(10000), e VARCHAR(10000),
       f VARCHAR(10000), g TEXT(6000)) ENGINE=MyISAM CHARACTER SET latin1;
Query OK, 0 rows affected (0.02 sec)
```

The operation succeeds for an `InnoDB` table because changing a column to `TEXT` avoids the MySQL 65,535-byte row size limit, and `InnoDB` off-page storage of variable-length columns avoids the `InnoDB` row size limit.

```
mysql> CREATE TABLE t (a VARCHAR(10000), b VARCHAR(10000),
       c VARCHAR(10000), d VARCHAR(10000), e VARCHAR(10000),
       f VARCHAR(10000), g TEXT(6000)) ENGINE=InnoDB CHARACTER SET latin1;
Query OK, 0 rows affected (0.02 sec)
```

- Storage for variable-length columns includes length bytes, which are counted toward the row size. For example, a `VARCHAR(255) CHARACTER SET utf8mb3` column takes two bytes to store the length of the value, so each value can take up to 767 bytes.

  The statement to create table `t1` succeeds because the columns require 32,765 + 2 bytes and 32,766 + 2 bytes, which falls within the maximum row size of 65,535 bytes:

  ```
  mysql> CREATE TABLE t1
         (c1 VARCHAR(32765) NOT NULL, c2 VARCHAR(32766) NOT NULL)
         ENGINE = InnoDB CHARACTER SET latin1;
  Query OK, 0 rows affected (0.02 sec)
  ```

  The statement to create table `t2` fails because, although the column length is within the maximum length of 65,535 bytes, two additional bytes are required to record the length, which causes the row size to exceed 65,535 bytes:

  ```
  mysql> CREATE TABLE t2
         (c1 VARCHAR(65535) NOT NULL)
         ENGINE = InnoDB CHARACTER SET latin1;
  ERROR 1118 (42000): Row size too large. The maximum row size for the used
  table type, not counting BLOBs, is 65535. This includes storage overhead,
  check the manual. You have to change some columns to TEXT or BLOBs
  ```

  Reducing the column length to 65,533 or less permits the statement to succeed.

  ```
  mysql> CREATE TABLE t2
         (c1 VARCHAR(65533) NOT NULL)
         ENGINE = InnoDB CHARACTER SET latin1;
  Query OK, 0 rows affected (0.01 sec)
  ```

- For `MyISAM` tables, `NULL` columns require additional space in the row to record whether their values are `NULL`. Each `NULL` column takes one bit extra, rounded up to the nearest byte.

  The statement to create table `t3` fails because `MyISAM` requires space for `NULL` columns in addition to the space required for variable-length column length bytes, causing the row size to exceed 65,535 bytes:

  ```
  mysql> CREATE TABLE t3
         (c1 VARCHAR(32765) NULL, c2 VARCHAR(32766) NULL)
         ENGINE = MyISAM CHARACTER SET latin1;
  ERROR 1118 (42000): Row size too large. The maximum row size for the used
  table type, not counting BLOBs, is 65535. This includes storage overhead,
  check the manual. You have to change some columns to TEXT or BLOBs
  ```

  For information about `InnoDB` `NULL` column storage, see Section 14.11, “InnoDB Row Formats”.

- InnoDB restricts row size (for data stored locally within the database page) to slightly less than half a database page. For example, the maximum row size is slightly less than 8KB for the default 16KB InnoDB page size, which is defined by the innodb_page_size configuration option.

  The statement to create table t4 fails because the defined columns exceed the row size limit for a 16KB InnoDB page.

Note

innodb_strict_mode is enabled in the following example to ensure that InnoDB returns an error if the defined columns exceed the InnoDB row size limit. When innodb_strict_mode is disabled (the default), creating a table that uses REDUNDANT or COMPACT row format succeeds with a warning if the InnoDB row size limit is exceeded.

DYNAMIC and COMPRESSED row formats are more restrictive in this regard. Creating a table that uses DYNAMIC or COMRESSED row format fails with an error if the InnoDB row size limit is exceeded, regardless of the innodb_strict_mode setting.

```
mysql> SET SESSION innodb_strict_mode=1;
mysql> CREATE TABLE t4 (
       c1 CHAR(255),c2 CHAR(255),c3 CHAR(255),
       c4 CHAR(255),c5 CHAR(255),c6 CHAR(255),
       c7 CHAR(255),c8 CHAR(255),c9 CHAR(255),
       c10 CHAR(255),c11 CHAR(255),c12 CHAR(255),
       c13 CHAR(255),c14 CHAR(255),c15 CHAR(255),
       c16 CHAR(255),c17 CHAR(255),c18 CHAR(255),
       c19 CHAR(255),c20 CHAR(255),c21 CHAR(255),
       c22 CHAR(255),c23 CHAR(255),c24 CHAR(255),
       c25 CHAR(255),c26 CHAR(255),c27 CHAR(255),
       c28 CHAR(255),c29 CHAR(255),c30 CHAR(255),
       c31 CHAR(255),c32 CHAR(255),c33 CHAR(255)
       ) ENGINE=InnoDB ROW_FORMAT=COMPACT DEFAULT CHARSET latin1;
ERROR 1118 (42000): Row size too large (> 8126). Changing some columns to TEXT or BLOB or using
ROW_FORMAT=DYNAMIC or ROW_FORMAT=COMPRESSED may help. In current row format, BLOB prefix of 768
bytes is stored inline.
```

8.5 Optimizing for InnoDB Tables

InnoDB is the storage engine that MySQL customers typically use in production databases where reliability and concurrency are important. InnoDB is the default storage engine in MySQL 5.6 This section explains how to tables more often than before. This section explains how to optimize database operations for InnoDB tables.

8.5.1 Optimizing Storage Layout for InnoDB Tables

- Once your data reaches a stable size, or a growing table has increased by tens or some hundreds of megabytes, consider using the OPTIMIZE TABLE statement to reorganize the table and compact any wasted space. The reorganized tables require less disk I/O to perform full table scans. This is a straightforward technique that can improve performance when other techniques such as improving index usage or tuning application code are not practical.

  OPTIMIZE TABLE copies the data part of the table and rebuilds the indexes. The benefits come from improved packing of data within indexes, and reduced fragmentation within the tablespaces and on disk. The benefits vary depending on the data in each table. You may find that there are significant gains for some and not for others, or that the gains decrease over time until you next optimize the table. This operation can be slow if the table is large or if the indexes being rebuilt do not fit into the buffer pool. The first run after adding a lot of data to a table is often much slower than later runs.

- In InnoDB, having a long PRIMARY KEY (either a single column with a lengthy value, or several columns that form a long composite value) wastes a lot of disk space. The primary key value

for a row is duplicated in all the secondary index records that point to the same row. (See Section 14.6.2.1, “Clustered and Secondary Indexes”.) Create an `AUTO_INCREMENT` column as the primary key if your primary key is long, or index a prefix of a long `VARCHAR` column instead of the entire column.

- Use the `VARCHAR` data type instead of `CHAR` to store variable-length strings or for columns with many `NULL` values. A `CHAR(N)` column always takes *`N`* characters to store data, even if the string is shorter or its value is `NULL`. Smaller tables fit better in the buffer pool and reduce disk I/O.

  When using `COMPACT` row format (the default `InnoDB` format) and variable-length character sets, such as `utf8` or `sjis`, `CHAR(N)` columns occupy a variable amount of space, but still at least *`N`* bytes.

- For tables that are big, or contain lots of repetitive text or numeric data, consider using `COMPRESSED` row format. Less disk I/O is required to bring data into the buffer pool, or to perform full table scans. Before making a permanent decision, measure the amount of compression you can achieve by using `COMPRESSED` versus `COMPACT` row format.

8.5.2 Optimizing InnoDB Transaction Management

To optimize `InnoDB` transaction processing, find the ideal balance between the performance overhead of transactional features and the workload of your server. For example, an application might encounter performance issues if it commits thousands of times per second, and different performance issues if it commits only every 2-3 hours.

- The default MySQL setting `AUTOCOMMIT=1` can impose performance limitations on a busy database server. Where practical, wrap several related data change operations into a single transaction, by issuing `SET AUTOCOMMIT=0` or a `START TRANSACTION` statement, followed by a `COMMIT` statement after making all the changes.

  `InnoDB` must flush the log to disk at each transaction commit if that transaction made modifications to the database. When each change is followed by a commit (as with the default autocommit setting), the I/O throughput of the storage device puts a cap on the number of potential operations per second.

- Alternatively, for transactions that consist only of a single `SELECT` statement, turning on `AUTOCOMMIT` helps `InnoDB` to recognize read-only transactions and optimize them. See Section 8.5.3, “Optimizing InnoDB Read-Only Transactions” for requirements.

- Avoid performing rollbacks after inserting, updating, or deleting huge numbers of rows. If a big transaction is slowing down server performance, rolling it back can make the problem worse, potentially taking several times as long to perform as the original data change operations. Killing the database process does not help, because the rollback starts again on server startup.

  To minimize the chance of this issue occurring:

  - Increase the size of the buffer pool so that all the data change changes can be cached rather than immediately written to disk.
  - Set `innodb_change_buffering=all` so that update and delete operations are buffered in addition to inserts.
  - Consider issuing `COMMIT` statements periodically during the big data change operation, possibly breaking a single delete or update into multiple statements that operate on smaller numbers of rows.

  To get rid of a runaway rollback once it occurs, increase the buffer pool so that the rollback becomes CPU-bound and runs fast, or kill the server and restart with `innodb_force_recovery=3`, as explained in Section 14.18.2, “InnoDB Recovery”.

This issue is expected to be infrequent with the default setting `innodb_change_buffering=all`, which allows update and delete operations to be cached in memory, making them faster to perform in the first place, and also faster to roll back if needed. Make sure to use this parameter setting on servers that process long-running transactions with many inserts, updates, or deletes.

- If you can afford the loss of some of the latest committed transactions if an unexpected exit occurs, you can set the `innodb_flush_log_at_trx_commit` parameter to 0. `InnoDB` tries to flush the log once per second anyway, although the flush is not guaranteed. Also, set the value of `innodb_support_xa` to 0, which reduces the number of disk flushes due to synchronizing on disk data and the binary log.

- When rows are modified or deleted, the rows and associated undo logs are not physically removed immediately, or even immediately after the transaction commits. The old data is preserved until transactions that started earlier or concurrently are finished, so that those transactions can access the previous state of modified or deleted rows. Thus, a long-running transaction can prevent `InnoDB` from purging data that was changed by a different transaction.

- When rows are modified or deleted within a long-running transaction, other transactions using the `READ COMMITTED` and `REPEATABLE READ` isolation levels have to do more work to reconstruct the older data if they read those same rows.

- When a long-running transaction modifies a table, queries against that table from other transactions do not make use of the covering index technique. Queries that normally could retrieve all the result columns from a secondary index, instead look up the appropriate values from the table data.

  If secondary index pages are found to have a `PAGE_MAX_TRX_ID` that is too new, or if records in the secondary index are delete-marked, `InnoDB` may need to look up records using a clustered index.

8.5.3 Optimizing InnoDB Read-Only Transactions

`InnoDB` can avoid the overhead associated with setting up the transaction ID (`TRX_ID` field) for transactions that are known to be read-only. A transaction ID is only needed for a transaction that might perform write operations or locking reads such as `SELECT ... FOR UPDATE`. Eliminating unnecessary transaction IDs reduces the size of internal data structures that are consulted each time a query or data change statement constructs a read view.

`InnoDB` detects read-only transactions when:

- The transaction is started with the `START TRANSACTION READ ONLY` statement. In this case, attempting to make changes to the database (for `InnoDB`, `MyISAM`, or other types of tables) causes an error, and the transaction continues in read-only state:

  ```
  ERROR 1792 (25006): Cannot execute statement in a READ ONLY transaction.
  ```

  You can still make changes to session-specific temporary tables in a read-only transaction, or issue locking queries for them, because those changes and locks are not visible to any other transaction.

- The `autocommit` setting is turned on, so that the transaction is guaranteed to be a single statement, and the single statement making up the transaction is a “non-locking” `SELECT` statement. That is, a `SELECT` that does not use a `FOR UPDATE` or `LOCK IN SHARED MODE` clause.

Thus, for a read-intensive application such as a report generator, you can tune a sequence of `InnoDB` queries by grouping them inside `START TRANSACTION READ ONLY` and `COMMIT`, or by turning on the `autocommit` setting before running the `SELECT` statements, or simply by avoiding any data change statements interspersed with the queries.

For information about `START TRANSACTION` and `autocommit`, see Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”.

Note

Transactions that qualify as auto-commit, non-locking, and read-only (AC-NL-RO) are kept out of certain internal `InnoDB` data structures and are therefore not listed in `SHOW ENGINE INNODB STATUS` output.

8.5.4 Optimizing InnoDB Redo Logging

Consider the following guidelines for optimizing redo logging:

- Make your redo log files big, even as big as the buffer pool. When `InnoDB` has written the redo log files full, it must write the modified contents of the buffer pool to disk in a checkpoint. Small redo log files cause many unnecessary disk writes. Although historically big redo log files caused lengthy recovery times, recovery is now much faster and you can confidently use large redo log files.

  The size and number of redo log files are configured using the `innodb_log_file_size` and `innodb_log_files_in_group` configuration options. For information about modifying an existing redo log file configuration, see Changing the Number or Size of InnoDB Redo Log Files.

- Consider increasing the size of the log buffer. A large log buffer enables large transactions to run without a need to write the log to disk before the transactions commit. Thus, if you have transactions that update, insert, or delete many rows, making the log buffer larger saves disk I/O. Log buffer size is configured using the `innodb_log_buffer_size` configuration option.

8.5.5 Bulk Data Loading for InnoDB Tables

These performance tips supplement the general guidelines for fast inserts in Section 8.2.4.1, “Optimizing INSERT Statements”.

- When importing data into `InnoDB`, turn off autocommit mode, because it performs a log flush to disk for every insert. To disable autocommit during your import operation, surround it with `SET autocommit` and `COMMIT` statements:

```
SET autocommit=0;
... SQL import statements ...
COMMIT;
```

  The `mysqldump` option `--opt` creates dump files that are fast to import into an `InnoDB` table, even without wrapping them with the `SET autocommit` and `COMMIT` statements.

- If you have `UNIQUE` constraints on secondary keys, you can speed up table imports by temporarily turning off the uniqueness checks during the import session:

```
SET unique_checks=0;
... SQL import statements ...
SET unique_checks=1;
```

  For big tables, this saves a lot of disk I/O because `InnoDB` can use its change buffer to write secondary index records in a batch. Be certain that the data contains no duplicate keys.

- If you have `FOREIGN KEY` constraints in your tables, you can speed up table imports by turning off the foreign key checks for the duration of the import session:

```
SET foreign_key_checks=0;
... SQL import statements ...
SET foreign_key_checks=1;
```

  For big tables, this can save a lot of disk I/O.

- Use the multiple-row `INSERT` syntax to reduce communication overhead between the client and the server if you need to insert many rows:

```
INSERT INTO yourtable VALUES (1,2), (5,5), ...;
```

This tip is valid for inserts into any table, not just `InnoDB` tables.

- When doing bulk inserts into tables with auto-increment columns, set `innodb_autoinc_lock_mode` to 2 instead of the default value 1. See Section 14.6.1.6, “AUTO_INCREMENT Handling in InnoDB” for details.
- When performing bulk inserts, it is faster to insert rows in `PRIMARY KEY` order. `InnoDB` tables use a clustered index, which makes it relatively fast to use data in the order of the `PRIMARY KEY`. Performing bulk inserts in `PRIMARY KEY` order is particularly important for tables that do not fit entirely within the buffer pool.
- For optimal performance when loading data into an `InnoDB FULLTEXT` index, follow this set of steps:

  1. Define a column `FTS_DOC_ID` at table creation time, of type `BIGINT UNSIGNED NOT NULL`, with a unique index named `FTS_DOC_ID_INDEX`. For example:

     ```
     CREATE TABLE t1 (
     FTS_DOC_ID BIGINT unsigned NOT NULL AUTO_INCREMENT,
     title varchar(255) NOT NULL DEFAULT '',
     text mediumtext NOT NULL,
     PRIMARY KEY (`FTS_DOC_ID`)
     ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
     CREATE UNIQUE INDEX FTS_DOC_ID_INDEX on t1(FTS_DOC_ID);
     ```

  2. Load the data into the table.
  3. Create the `FULLTEXT` index after the data is loaded.

Note

When adding `FTS_DOC_ID` column at table creation time, ensure that the `FTS_DOC_ID` column is updated when the `FULLTEXT` indexed column is updated, as the `FTS_DOC_ID` must increase monotonically with each `INSERT` or `UPDATE`. If you choose not to add the `FTS_DOC_ID` at table creation time and have `InnoDB` manage DOC IDs for you, `InnoDB` adds the `FTS_DOC_ID` as a hidden column with the next `CREATE FULLTEXT INDEX` call. This approach, however, requires a table rebuild which can impact performance.

8.5.6 Optimizing InnoDB Queries

To tune queries for `InnoDB` tables, create an appropriate set of indexes on each table. See Section 8.3.1, “How MySQL Uses Indexes” for details. Follow these guidelines for `InnoDB` indexes:

- Because each `InnoDB` table has a primary key (whether you request one or not), specify a set of primary key columns for each table, columns that are used in the most important and time-critical queries.
- Do not specify too many or too long columns in the primary key, because these column values are duplicated in each secondary index. When an index contains unnecessary data, the I/O to read this data and memory to cache it reduce the performance and scalability of the server.
- Do not create a separate secondary index for each column, because each query can only make use of one index. Indexes on rarely tested columns or columns with only a few different values might not be helpful for any queries. If you have many queries for the same table, testing different combinations of columns, try to create a small number of concatenated indexes rather than a large number of single-column indexes. If an index contains all the columns needed for the result set (known as a covering index), the query might be able to avoid reading the table data at all.
- If an indexed column cannot contain any `NULL` values, declare it as `NOT NULL` when you create the table. The optimizer can better determine which index is most effective to use for a query, when it knows whether each column contains `NULL` values.

- You can optimize single-query transactions for `InnoDB` tables, using the technique in Section 8.5.3, “Optimizing InnoDB Read-Only Transactions”.

8.5.7 Optimizing InnoDB DDL Operations

- Many DDL operations on tables and indexes (`CREATE`, `ALTER`, and `DROP` statements) can be performed online. See Section 14.13, “InnoDB and Online DDL” for details.
- Online DDL support for adding secondary indexes means that you can generally speed up the process of creating and loading a table and associated indexes by creating the table without secondary indexes, then adding secondary indexes after the data is loaded.
- Use `TRUNCATE TABLE` to empty a table, not `DELETE FROM tbl_name`. Foreign key constraints can make a `TRUNCATE` statement work like a regular `DELETE` statement, in which case a sequence of commands like `DROP TABLE` and `CREATE TABLE` might be fastest.
- Because the primary key is integral to the storage layout of each `InnoDB` table, and changing the definition of the primary key involves reorganizing the whole table, always set up the primary key as part of the `CREATE TABLE` statement, and plan ahead so that you do not need to `ALTER` or `DROP` the primary key afterward.

8.5.8 Optimizing InnoDB Disk I/O

If you follow best practices for database design and tuning techniques for SQL operations, but your database is still slow due to heavy disk I/O activity, consider these disk I/O optimizations. If the Unix `top` tool or the Windows Task Manager shows that the CPU usage percentage with your workload is less than 70%, your workload is probably disk-bound.

- Increase buffer pool size

  When table data is cached in the `InnoDB` buffer pool, it can be accessed repeatedly by queries without requiring any disk I/O. Specify the size of the buffer pool with the `innodb_buffer_pool_size` option. This memory area is important enough that it is typically recommended that `innodb_buffer_pool_size` is configured to 50 to 75 percent of system memory. For more information see, Section 8.12.4.1, “How MySQL Uses Memory”.

- Adjust the flush method

  In some versions of GNU/Linux and Unix, flushing files to disk with the Unix `fsync()` call (which `InnoDB` uses by default) and similar methods is surprisingly slow. If database write performance is an issue, conduct benchmarks with the `innodb_flush_method` parameter set to `O_DSYNC`.

- Use a noop or deadline I/O scheduler with native AIO on Linux

  `InnoDB` uses the asynchronous I/O subsystem (native AIO) on Linux to perform read-ahead and write requests for data file pages. This behavior is controlled by the `innodb_use_native_aio` configuration option, which is enabled by default. With native AIO, the type of I/O scheduler has greater influence on I/O performance. Generally, noop and deadline I/O schedulers are recommended. Conduct benchmarks to determine which I/O scheduler provides the best results for your workload and environment. For more information, see Section 14.8.7, “Using Asynchronous I/O on Linux”.

- Use direct I/O on Solaris 10 for x86_64 architecture

  When using the `InnoDB` storage engine on Solaris 10 for x86_64 architecture (AMD Opteron), use direct I/O for `InnoDB`-related files to avoid degradation of `InnoDB` performance. To use direct I/O for an entire UFS file system used for storing `InnoDB`-related files, mount it with the `forcedirectio` option; see `mount_ufs(1M)`. (The default on Solaris 10/x86_64 is *not* to use this option.) To apply direct I/O only to `InnoDB` file operations rather than the whole file system, set `innodb_flush_method = O_DIRECT`. With this setting, `InnoDB` calls `directio()` instead of `fcntl()` for I/O to data files (not for I/O to log files).

- Use raw storage for data and log files with Solaris 2.6 or later

  When using the `InnoDB` storage engine with a large `innodb_buffer_pool_size` value on any release of Solaris 2.6 and up and any platform (sparc/x86/x64/amd64), conduct benchmarks with `InnoDB` data files and log files on raw devices or on a separate direct I/O UFS file system, using the `forcedirectio` mount option as described previously. (It is necessary to use the mount option rather than setting `innodb_flush_method` if you want direct I/O for the log files.) Users of the Veritas file system VxFS should use the `convosync=direct` mount option.

  Do not place other MySQL data files, such as those for `MyISAM` tables, on a direct I/O file system. Executables or libraries *must not* be placed on a direct I/O file system.

- Use additional storage devices

  Additional storage devices could be used to set up a RAID configuration. For related information, see Section 8.12.2, “Optimizing Disk I/O”.

  Alternatively, `InnoDB` tablespace data files and log files can be placed on different physical disks. For more information, refer to the following sections:

  - Section 14.8.1, “InnoDB Startup Configuration”
  - Section 14.6.1.2, “Creating Tables Externally”
  - Section 14.6.1.4, “Moving or Copying InnoDB Tables”

- Consider non-rotational storage

  Non-rotational storage generally provides better performance for random I/O operations; and rotational storage for sequential I/O operations. When distributing data and log files across rotational

and non-rotational storage devices, consider the type of I/O operations that are predominantly performed on each file.

Random I/O-oriented files typically include file-per-table data files and undo tablespace files. Sequential I/O-oriented files include `InnoDB` system tablespace files (due to doublewrite buffering and change buffering) and log files such as binary log files and redo log files.

Review settings for the following configuration options when using non-rotational storage:

- `innodb_checksum_algorithm`

  The `crc32` option uses a faster checksum algorithm and is recommended for fast storage systems.

- `innodb_flush_neighbors`

  Optimizes I/O for rotational storage devices. Disable it for non-rotational storage or a mix of rotational and non-rotational storage.

- `innodb_io_capacity`

  The default setting of 200 is generally sufficient for a lower-end non-rotational storage device. For higher-end, bus-attached devices, consider a higher setting such as 1000.

- `innodb_io_capacity_max`

  The default value of 2000 is intended for workloads that use non-rotational storage. For a high-end, bus-attached non-rotational storage device, consider a higher setting such as 2500.

- `innodb_log_compressed_pages`

  If redo logs are on non-rotational storage, consider disabling this option to reduce logging. See Disable logging of compressed pages.

- `innodb_log_file_size`

  If redo logs are on non-rotational storage, configure this option to maximize caching and write combining.

- `innodb_page_size`

  Consider using a page size that matches the internal sector size of the disk. Early-generation SSD devices often have a 4KB sector size. Some newer devices have a 16KB sector size. The default `InnoDB` page size is 16KB. Keeping the page size close to the storage device block size minimizes the amount of unchanged data that is rewritten to disk.

- `binlog_row_image`

  If binary logs are on non-rotational storage and all tables have primary keys, consider setting this option to `minimal` to reduce logging.

Ensure that TRIM support is enabled for your operating system. It is typically enabled by default.

- Increase I/O capacity to avoid backlogs

  If throughput drops periodically because of `InnoDB` checkpoint operations, consider increasing the value of the `innodb_io_capacity` configuration option. Higher values cause more frequent flushing, avoiding the backlog of work that can cause dips in throughput.

- Lower I/O capacity if flushing does not fall behind

  If the system is not falling behind with InnoDB flushing operations, consider lowering the value of the innodb_io_capacity configuration option. Typically, you keep this option value as low as practical, but not so low that it causes periodic drops in throughput as mentioned in the preceding bullet. In a typical scenario where you could lower the option value, you might see a combination like this in the output from SHOW ENGINE INNODB STATUS:

  - History list length low, below a few thousand.
  - Insert buffer merges close to rows inserted.
  - Modified pages in buffer pool consistently well below innodb_max_dirty_pages_pct of the buffer pool. (Measure at a time when the server is not doing bulk inserts; it is normal during bulk inserts for the modified pages percentage to rise significantly.)
  - Log sequence number - Last checkpoint is at less than 7/8 or ideally less than 6/8 of the total size of the InnoDB log files.

- Disable logging of compressed pages

  When using the InnoDB table compression feature, images of re-compressed pages are written to the redo log when changes are made to compressed data. This behavior is controlled by innodb_log_compressed_pages, which is enabled by default to prevent corruption that can occur if a different version of the zlib compression algorithm is used during recovery. If you are certain that the zlib version is not subject to change, disable innodb_log_compressed_pages to reduce redo log generation for workloads that modify compressed data.

8.5.9 Optimizing InnoDB Configuration Variables

Different settings work best for servers with light, predictable loads, versus servers that are running near full capacity all the time, or that experience spikes of high activity.

Because the InnoDB storage engine performs many of its optimizations automatically, many performance-tuning tasks involve monitoring to ensure that the database is performing well, and changing configuration options when performance drops. See Section 14.16, “InnoDB Integration with MySQL Performance Schema” for information about detailed InnoDB performance monitoring.

The main configuration steps you can perform include:

- Enabling InnoDB to use high-performance memory allocators on systems that include them. See Section 14.8.4, “Configuring the Memory Allocator for InnoDB”.
- Controlling the types of data change operations for which InnoDB buffers the changed data, to avoid frequent small disk writes. See Configuring Change Buffering. Because the default is to buffer all types of data change operations, only change this setting if you need to reduce the amount of buffering.
- Turning the adaptive hash indexing feature on and off using the innodb_adaptive_hash_index option. See Section 14.5.3, “Adaptive Hash Index” for more information. You might change this setting during periods of unusual activity, then restore it to its original setting.
- Setting a limit on the number of concurrent threads that InnoDB processes, if context switching is a bottleneck. See Section 14.8.5, “Configuring Thread Concurrency for InnoDB”.
- Controlling the amount of prefetching that InnoDB does with its read-ahead operations. When the system has unused I/O capacity, more read-ahead can improve the performance of queries. Too much read-ahead can cause periodic drops in performance on a heavily loaded system. See Section 14.8.3.3, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”.

- Increasing the number of background threads for read or write operations, if you have a high-end I/O subsystem that is not fully utilized by the default values. See Section 14.8.6, “Configuring the Number of Background InnoDB I/O Threads”.
- Controlling how much I/O `InnoDB` performs in the background. See Section 14.8.8, “Configuring InnoDB I/O Capacity”. You might scale back this setting if you observe periodic drops in performance.
- Controlling the algorithm that determines when `InnoDB` performs certain types of background writes. See Section 14.8.3.4, “Configuring Buffer Pool Flushing”. The algorithm works for some types of workloads but not others, so you might disable this feature if you observe periodic drops in performance.
- Taking advantage of multicore processors and their cache memory configuration, to minimize delays in context switching. See Section 14.8.9, “Configuring Spin Lock Polling”.
- Preventing one-time operations such as table scans from interfering with the frequently accessed data stored in the `InnoDB` buffer cache. See Section 14.8.3.2, “Making the Buffer Pool Scan Resistant”.
- Adjusting log files to a size that makes sense for reliability and crash recovery. `InnoDB` log files have often been kept small to avoid long startup times after a crash. Optimizations introduced in MySQL 5.5 speed up certain steps of the crash recovery process. In particular, scanning the redo log and applying the redo log are faster due to improved algorithms for memory management. If you have kept your log files artificially small to avoid long startup times, you can now consider increasing log file size to reduce the I/O that occurs due recycling of redo log records.
- Configuring the size and number of instances for the `InnoDB` buffer pool, especially important for systems with multi-gigabyte buffer pools. See Section 14.8.3.1, “Configuring Multiple Buffer Pool Instances”.
- Increasing the maximum number of concurrent transactions, which dramatically improves scalability for the busiest databases. See Section 14.6.7, “Undo Logs”.
- Moving purge operations (a type of garbage collection) into a background thread. See Section 14.8.10, “Purge Configuration”. To effectively measure the results of this setting, tune the other I/O-related and thread-related configuration settings first.
- Reducing the amount of switching that `InnoDB` does between concurrent threads, so that SQL operations on a busy server do not queue up and form a “traffic jam”. Set a value for the `innodb_thread_concurrency` option, up to approximately 32 for a high-powered modern system. Increase the value for the `innodb_concurrency_tickets` option, typically to 5000 or so. This combination of options sets a cap on the number of threads that `InnoDB` processes at any one time, and allows each thread to do substantial work before being swapped out, so that the number of waiting threads stays low and operations can complete without excessive context switching.

8.5.10 Optimizing InnoDB for Systems with Many Tables

- If you have configured non-persistent optimizer statistics (a non-default configuration), `InnoDB` computes index cardinality values for a table the first time that table is accessed after startup, instead of storing such values in the table. This step can take significant time on systems that partition the data into many tables. Since this overhead only applies to the initial table open operation, to “warm up” a table for later use, access it immediately after startup by issuing a statement such as `SELECT 1 FROM tbl_name LIMIT 1`.

  Optimizer statistics are persisted to disk by default, enabled by the `innodb_stats_persistent` configuration option. For information about persistent optimizer statistics, see Section 14.8.11.1, “Configuring Persistent Optimizer Statistics Parameters”.

8.6 Optimizing for MyISAM Tables

The MyISAM storage engine performs best with read-mostly data or with low-concurrency operations, because table locks limit the ability to perform simultaneous updates. In MySQL, InnoDB is the default storage engine rather than MyISAM.

8.6.1 Optimizing MyISAM Queries

Some general tips for speeding up queries on MyISAM tables:

- To help MySQL better optimize queries, use ANALYZE TABLE or run myisamchk --analyze on a table after it has been loaded with data. This updates a value for each index part that indicates the average number of rows that have the same value. (For unique indexes, this is always 1.) MySQL uses this to decide which index to choose when you join two tables based on a nonconstant expression. You can check the result from the table analysis by using SHOW INDEX FROM *tbl_name* and examining the Cardinality value. myisamchk --description --verbose shows index distribution information.

- To sort an index and data according to an index, use myisamchk --sort-index --sort-records=1 (assuming that you want to sort on index 1). This is a good way to make queries faster if you have a unique index from which you want to read all rows in order according to the index. The first time you sort a large table this way, it may take a long time.

- Try to avoid complex SELECT queries on MyISAM tables that are updated frequently, to avoid problems with table locking that occur due to contention between readers and writers.

- MyISAM supports concurrent inserts: If a table has no free blocks in the middle of the data file, you can INSERT new rows into it at the same time that other threads are reading from the table. If it is important to be able to do this, consider using the table in ways that avoid deleting rows. Another possibility is to run OPTIMIZE TABLE to defragment the table after you have deleted a lot of rows from it. This behavior is altered by setting the concurrent_insert variable. You can force new rows to be appended (and therefore permit concurrent inserts), even in tables that have deleted rows. See Section 8.11.3, “Concurrent Inserts”.

- For MyISAM tables that change frequently, try to avoid all variable-length columns (VARCHAR, BLOB, and TEXT). The table uses dynamic row format if it includes even a single variable-length column. See Chapter 15, *Alternative Storage Engines*.

- It is normally not useful to split a table into different tables just because the rows become large. In accessing a row, the biggest performance hit is the disk seek needed to find the first byte of the row. After finding the data, most modern disks can read the entire row fast enough for most applications. The only cases where splitting up a table makes an appreciable difference is if it is a MyISAM table using dynamic row format that you can change to a fixed row size, or if you very often need to scan the table but do not need most of the columns. See Chapter 15, *Alternative Storage Engines*.

- Use ALTER TABLE ... ORDER BY *expr1, expr2, ...* if you usually retrieve rows in *expr1, expr2, ...* order. By using this option after extensive changes to the table, you may be able to get higher performance.

- If you often need to calculate results such as counts based on information from a lot of rows, it may be preferable to introduce a new table and update the counter in real time. An update of the following form is very fast:

```
UPDATE tbl_name SET count_col=count_col+1 WHERE key_col=constant;
```

  This is very important when you use MySQL storage engines such as MyISAM that has only table-level locking (multiple readers with single writers). This also gives better performance with most database systems, because the row locking manager in this case has less to do.

- Use INSERT DELAYED for MyISAM (or other supported nontransactional tables) when you do not need to know when your data is written. This reduces the overall insertion impact because many rows can be written with a single disk write.

Note

`INSERT DELAYED` is deprecated; expect it to be removed in a future release. Use `INSERT` (without `DELAYED`) instead.

- Use `OPTIMIZE TABLE` periodically to avoid fragmentation with dynamic-format `MyISAM` tables. See Section 15.2.3, "MyISAM Table Storage Formats".
- Declaring a `MyISAM` table with the `DELAY_KEY_WRITE=1` table option makes index updates faster because they are not flushed to disk until the table is closed. The downside is that if something kills the server while such a table is open, you must ensure that the table is okay by running the server with the `myisam_recover_options` system variable set, or by running `myisamchk` before restarting the server. (However, even in this case, you should not lose anything by using `DELAY_KEY_WRITE`, because the key information can always be generated from the data rows.)
- Strings are automatically prefix- and end-space compressed in `MyISAM` indexes. See Section 13.1.13, "CREATE INDEX Statement".
- You can increase performance by caching queries or answers in your application and then executing many inserts or updates together. Locking the table during this operation ensures that the index cache is only flushed once after all updates. You can also take advantage of MySQL's query cache to achieve similar results; see Section 8.10.3, "The MySQL Query Cache".

8.6.2 Bulk Data Loading for MyISAM Tables

These performance tips supplement the general guidelines for fast inserts in Section 8.2.4.1, "Optimizing INSERT Statements".

- To improve performance when multiple clients insert a lot of rows, use the `INSERT DELAYED` statement. See Section 13.2.5.3, "INSERT DELAYED Statement". This technique works for `MyISAM` and some other storage engines, but not `InnoDB`.

Note

`INSERT DELAYED` is deprecated; expect it to be removed in a future release. Use `INSERT` (without `DELAYED`) instead.

- For a `MyISAM` table, you can use concurrent inserts to add rows at the same time that `SELECT` statements are running, if there are no deleted rows in middle of the data file. See Section 8.11.3, "Concurrent Inserts".
- With some extra work, it is possible to make `LOAD DATA` run even faster for a `MyISAM` table when the table has many indexes. Use the following procedure:

  1. Execute a `FLUSH TABLES` statement or a `mysqladmin flush-tables` command.
  2. Use `myisamchk --keys-used=0 -rq /path/to/db/tbl_name` to remove all use of indexes for the table.
  3. Insert data into the table with `LOAD DATA`. This does not update any indexes and therefore is very fast.
  4. If you intend only to read from the table in the future, use `myisampack` to compress it. See Section 15.2.3.3, "Compressed Table Characteristics".
  5. Re-create the indexes with `myisamchk -rq /path/to/db/tbl_name`. This creates the index tree in memory before writing it to disk, which is much faster than updating the index during `LOAD DATA` because it avoids lots of disk seeks. The resulting index tree is also perfectly balanced.
  6. Execute a `FLUSH TABLES` statement or a `mysqladmin flush-tables` command.

`LOAD DATA` performs the preceding optimization automatically if the `MyISAM` table into which you insert data is empty. The main difference between automatic optimization and using the procedure explicitly is that you can let `myisamchk` allocate much more temporary memory for the index creation than you might want the server to allocate for index re-creation when it executes the `LOAD DATA` statement.

You can also disable or enable the nonunique indexes for a `MyISAM` table by using the following statements rather than `myisamchk`. If you use these statements, you can skip the `FLUSH TABLES` operations:

```
ALTER TABLE tbl_name DISABLE KEYS;
ALTER TABLE tbl_name ENABLE KEYS;
```

- To speed up `INSERT` operations that are performed with multiple statements for nontransactional tables, lock your tables:

```
LOCK TABLES a WRITE;
INSERT INTO a VALUES (1,23),(2,34),(4,33);
INSERT INTO a VALUES (8,26),(6,29);
...
UNLOCK TABLES;
```

This benefits performance because the index buffer is flushed to disk only once, after all `INSERT` statements have completed. Normally, there would be as many index buffer flushes as there are `INSERT` statements. Explicit locking statements are not needed if you can insert all rows with a single `INSERT`.

Locking also lowers the total time for multiple-connection tests, although the maximum wait time for individual connections might go up because they wait for locks. Suppose that five clients attempt to perform inserts simultaneously as follows:

- Connection 1 does 1000 inserts
- Connections 2, 3, and 4 do 1 insert
- Connection 5 does 1000 inserts

If you do not use locking, connections 2, 3, and 4 finish before 1 and 5. If you use locking, connections 2, 3, and 4 probably do not finish before 1 or 5, but the total time should be about 40% faster.

`INSERT`, `UPDATE`, and `DELETE` operations are very fast in MySQL, but you can obtain better overall performance by adding locks around everything that does more than about five successive inserts or updates. If you do very many successive inserts, you could do a `LOCK TABLES` followed by an `UNLOCK TABLES` once in a while (each 1,000 rows or so) to permit other threads to access table. This would still result in a nice performance gain.

`INSERT` is still much slower for loading data than `LOAD DATA`, even when using the strategies just outlined.

- To increase performance for `MyISAM` tables, for both `LOAD DATA` and `INSERT`, enlarge the key cache by increasing the `key_buffer_size` system variable. See Section 5.1.1, “Configuring the Server”.

8.6.3 Optimizing REPAIR TABLE Statements

`REPAIR TABLE` for `MyISAM` tables is similar to using `myisamchk` for repair operations, and some of the same performance optimizations apply:

- `myisamchk` has variables that control memory allocation. You may be able to its improve performance by setting these variables, as described in Section 4.6.3.6, “myisamchk Memory Usage”.
- For `REPAIR TABLE`, the same principle applies, but because the repair is done by the server, you set server system variables instead of `myisamchk` variables. Also, in addition to setting memory-allocation variables, increasing the `myisam_max_sort_file_size` system variable increases the likelihood that the repair uses the faster filesort method and avoid the slower repair by key cache method. Set the variable to the maximum file size for your system, after checking to be sure that there is enough free space to hold a copy of the table files. The free space must be available in the file system containing the original table files.

Suppose that a `myisamchk` table-repair operation is done using the following options to set its memory-allocation variables:

```
--key_buffer_size=128M --myisam_sort_buffer_size=256M
--read_buffer_size=64M --write_buffer_size=64M
```

Some of those `myisamchk` variables correspond to server system variables:

`myisamchk` Variable	**System Variable**
`key_buffer_size`	`key_buffer_size`
`myisam_sort_buffer_size`	`myisam_sort_buffer_size`
`read_buffer_size`	`read_buffer_size`
`write_buffer_size`	none

Each of the server system variables can be set at runtime, and some of them (`myisam_sort_buffer_size`, `read_buffer_size`) have a session value in addition to a global value. Setting a session value limits the effect of the change to your current session and does not affect other users. Changing a global-only variable (`key_buffer_size`, `myisam_max_sort_file_size`) affects other users as well. For `key_buffer_size`, you must take into account that the buffer is shared with those users. For example, if you set the `myisamchk key_buffer_size` variable to 128MB, you could set the corresponding `key_buffer_size` system variable larger than that (if it is not already set larger), to permit key buffer use by activity in other sessions. However, changing the global key buffer size invalidates the buffer, causing increased disk I/O and slowdown for other sessions. An alternative that avoids this problem is to use a separate key cache, assign to it the indexes from the table to be repaired, and deallocate it when the repair is complete. See Section 8.10.2.2, “Multiple Key Caches”.

Based on the preceding remarks, a `REPAIR TABLE` operation can be done as follows to use settings similar to the `myisamchk` command. Here a separate 128MB key buffer is allocated and the file system is assumed to permit a file size of at least 100GB.

```
SET SESSION myisam_sort_buffer_size = 256*1024*1024;
SET SESSION read_buffer_size = 64*1024*1024;
SET GLOBAL myisam_max_sort_file_size = 100*1024*1024*1024;
SET GLOBAL repair_cache.key_buffer_size = 128*1024*1024;
CACHE INDEX tbl_name IN repair_cache;
LOAD INDEX INTO CACHE tbl_name;
REPAIR TABLE tbl_name ;
SET GLOBAL repair_cache.key_buffer_size = 0;
```

If you intend to change a global variable but want to do so only for the duration of a `REPAIR TABLE` operation to minimally affect other users, save its value in a user variable and restore it afterward. For example:

```
SET @old_myisam_sort_buffer_size = @@GLOBAL.myisam_max_sort_file_size;
SET GLOBAL myisam_max_sort_file_size = 100*1024*1024*1024;
REPAIR TABLE tbl_name ;
SET GLOBAL myisam_max_sort_file_size = @old_myisam_max_sort_file_size;
```

The system variables that affect `REPAIR TABLE` can be set globally at server startup if you want the values to be in effect by default. For example, add these lines to the server `my.cnf` file:

```
[mysqld]
myisam_sort_buffer_size=256M
key_buffer_size=1G
myisam_max_sort_file_size=100G
```

These settings do not include `read_buffer_size`. Setting `read_buffer_size` globally to a large value does so for all sessions and can cause performance to suffer due to excessive memory allocation for a server with many simultaneous sessions.

8.7 Optimizing for MEMORY Tables

Consider using `MEMORY` tables for noncritical data that is accessed often, and is read-only or rarely updated. Benchmark your application against equivalent `InnoDB` or `MyISAM` tables under a realistic workload, to confirm that any additional performance is worth the risk of losing data, or the overhead of copying data from a disk-based table at application start.

For best performance with `MEMORY` tables, examine the kinds of queries against each table, and specify the type to use for each associated index, either a B-tree index or a hash index. On the `CREATE INDEX` statement, use the clause `USING BTREE` or `USING HASH`. B-tree indexes are fast for queries that do greater-than or less-than comparisons through operators such as `>` or `BETWEEN`. Hash indexes are only fast for queries that look up single values through the `=` operator, or a restricted set of values through the `IN` operator. For why `USING BTREE` is often a better choice than the default `USING HASH`, see Section 8.2.1.19, “Avoiding Full Table Scans”. For implementation details of the different types of `MEMORY` indexes, see Section 8.3.8, “Comparison of B-Tree and Hash Indexes”.

8.8 Understanding the Query Execution Plan

Depending on the details of your tables, columns, indexes, and the conditions in your `WHERE` clause, the MySQL optimizer considers many techniques to efficiently perform the lookups involved in an SQL query. A query on a huge table can be performed without reading all the rows; a join involving several tables can be performed without comparing every combination of rows. The set of operations that the optimizer chooses to perform the most efficient query is called the “query execution plan”, also known as the `EXPLAIN` plan. Your goals are to recognize the aspects of the `EXPLAIN` plan that indicate a query is optimized well, and to learn the SQL syntax and indexing techniques to improve the plan if you see some inefficient operations.

8.8.1 Optimizing Queries with EXPLAIN

The `EXPLAIN` statement provides information about how MySQL executes statements:

- The `EXPLAIN` statement provides information about how MySQL executes statements. `EXPLAIN` works with `SELECT`, `DELETE`, `INSERT`, `REPLACE`, and `UPDATE` statements.
- When `EXPLAIN` is used with an explainable statement, MySQL displays information from the optimizer about the statement execution plan. That is, MySQL explains how it would process the statement, including information about how tables are joined and in which order. For information about using `EXPLAIN` to obtain execution plan information, see Section 8.8.2, “EXPLAIN Output Format”.
- `EXPLAIN EXTENDED` produces additional execution plan information that can be displayed using `SHOW WARNINGS`. See Section 8.8.3, “Extended EXPLAIN Output Format”.
- `EXPLAIN PARTITIONS` is useful for examining queries involving partitioned tables. See Section 19.3.5, “Obtaining Information About Partitions”.
- The `FORMAT` option can be used to select the output format. `TRADITIONAL` presents the output in tabular format. This is the default if no `FORMAT` option is present. `JSON` format displays the

information in JSON format. With `FORMAT = JSON`, the output includes extended and partition information.

With the help of `EXPLAIN`, you can see where you should add indexes to tables so that the statement executes faster by using indexes to find rows. You can also use `EXPLAIN` to check whether the optimizer joins the tables in an optimal order. To give a hint to the optimizer to use a join order corresponding to the order in which the tables are named in a `SELECT` statement, begin the statement with `SELECT STRAIGHT_JOIN` rather than just `SELECT`. (See Section 13.2.9, "SELECT Statement".) However, `STRAIGHT_JOIN` may prevent indexes from being used because it disables semijoin transformations. See Section 8.2.2.1, "Optimizing Subqueries with Semijoin Transformations".

If you have a problem with indexes not being used when you believe that they should be, run `ANALYZE TABLE` to update table statistics, such as cardinality of keys, that can affect the choices the optimizer makes. See Section 13.7.2.1, "ANALYZE TABLE Statement".

Note

`EXPLAIN` can also be used to obtain information about the columns in a table. `EXPLAIN *tbl_name*` is synonymous with `DESCRIBE *tbl_name*` and `SHOW COLUMNS FROM *tbl_name*`. For more information, see Section 13.8.1, "DESCRIBE Statement", and Section 13.7.5.6, "SHOW COLUMNS Statement".

8.8.2 EXPLAIN Output Format

The `EXPLAIN` statement provides information about how MySQL executes statements. `EXPLAIN` works with `SELECT`, `DELETE`, `INSERT`, `REPLACE`, and `UPDATE` statements.

`EXPLAIN` returns a row of information for each table used in the `SELECT` statement. It lists the tables in the output in the order that MySQL would read them while processing the statement. MySQL resolves all joins using a nested-loop join method. This means that MySQL reads a row from the first table, and then finds a matching row in the second table, the third table, and so on. When all tables are processed, MySQL outputs the selected columns and backtracks through the table list until a table is found for which there are more matching rows. The next row is read from this table and the process continues with the next table.

When the `EXTENDED` keyword is used, `EXPLAIN` produces extra information that can be viewed by issuing a `SHOW WARNINGS` statement following the `EXPLAIN` statement. `EXPLAIN EXTENDED` also displays the `filtered` column. See Section 8.8.3, "Extended EXPLAIN Output Format".

Note

You cannot use the `EXTENDED` and `PARTITIONS` keywords together in the same `EXPLAIN` statement. Neither of these keywords can be used together with the `FORMAT` option. (`FORMAT=JSON` causes `EXPLAIN` to display extended and partition information automatically; using `FORMAT=TRADITIONAL` has no effect on `EXPLAIN` output.)

Note

MySQL Workbench has a Visual Explain capability that provides a visual representation of `EXPLAIN` output. See Tutorial: Using Explain to Improve Query Performance.

- EXPLAIN Output Columns
- EXPLAIN Join Types
- EXPLAIN Extra Information
- EXPLAIN Output Interpretation

EXPLAIN Output Columns

This section describes the output columns produced by `EXPLAIN`. Later sections provide additional information about the `type` and `Extra` columns.

Each output row from `EXPLAIN` provides information about one table. Each row contains the values summarized in Table 8.1, “EXPLAIN Output Columns”, and described in more detail following the table. Column names are shown in the table's first column; the second column provides the equivalent property name shown in the output when `FORMAT=JSON` is used.

Table 8.1 EXPLAIN Output Columns

Column	JSON Name	Meaning
`id`	`select_id`	The `SELECT` identifier
`select_type`	None	The `SELECT` type
`table`	`table_name`	The table for the output row
`partitions`	`partitions`	The matching partitions
`type`	`access_type`	The join type
`possible_keys`	`possible_keys`	The possible indexes to choose
`key`	`key`	The index actually chosen
`key_len`	`key_length`	The length of the chosen key
`ref`	`ref`	The columns compared to the index
`rows`	`rows`	Estimate of rows to be examined
`filtered`	`filtered`	Percentage of rows filtered by table condition
`Extra`	None	Additional information

Note

JSON properties which are `NULL` are not displayed in JSON-formatted `EXPLAIN` output.

- `id` (JSON name: `select_id`)

  The `SELECT` identifier. This is the sequential number of the `SELECT` within the query. The value can be `NULL` if the row refers to the union result of other rows. In this case, the `table` column shows a value like `<unionM,N>` to indicate that the row refers to the union of the rows with `id` values of *`M`* and *`N`*.

- `select_type` (JSON name: none)

  The type of `SELECT`, which can be any of those shown in the following table. A JSON-formatted `EXPLAIN` exposes the `SELECT` type as a property of a `query_block`, unless it is `SIMPLE` or `PRIMARY`. The JSON names (where applicable) are also shown in the table.

`select_type` Value	JSON Name	Meaning
`SIMPLE`	None	Simple `SELECT` (not using `UNION` or subqueries)
`PRIMARY`	None	Outermost `SELECT`
`UNION`	None	Second or later `SELECT` statement in a `UNION`
`DEPENDENT UNION`	`dependent` (`true`)	Second or later `SELECT` statement in a `UNION`, dependent on outer query
`UNION RESULT`	`union_result`	Result of a `UNION`.
`SUBQUERY`	None	First `SELECT` in subquery

`select_type` Value	JSON Name	Meaning
`DEPENDENT SUBQUERY`	`dependent` (`true`)	First `SELECT` in subquery, dependent on outer query
`DERIVED`	None	Derived table
`MATERIALIZED`	`materialized_from_subquery`	Materialized subquery
`UNCACHEABLE SUBQUERY`	`cacheable` (`false`)	A subquery for which the result cannot be cached and must be re-evaluated for each row of the outer query
`UNCACHEABLE UNION`	`cacheable` (`false`)	The second or later select in a `UNION` that belongs to an uncacheable subquery (see `UNCACHEABLE SUBQUERY`)

`DEPENDENT` typically signifies the use of a correlated subquery. See Section 13.2.10.7, “Correlated Subqueries”.

`DEPENDENT SUBQUERY` evaluation differs from `UNCACHEABLE SUBQUERY` evaluation. For `DEPENDENT SUBQUERY`, the subquery is re-evaluated only once for each set of different values of the variables from its outer context. For `UNCACHEABLE SUBQUERY`, the subquery is re-evaluated for each row of the outer context.

Cacheability of subqueries differs from caching of query results in the query cache (which is described in Section 8.10.3.1, “How the Query Cache Operates”). Subquery caching occurs during query execution, whereas the query cache is used to store results only after query execution finishes.

When you specify `FORMAT=JSON` with `EXPLAIN`, the output has no single property directly equivalent to `select_type`; the `query_block` property corresponds to a given `SELECT`. Properties equivalent to most of the `SELECT` subquery types just shown are available (an example being `materialized_from_subquery` for `MATERIALIZED`), and are displayed when appropriate. There are no JSON equivalents for `SIMPLE` or `PRIMARY`.

- `table` (JSON name: `table_name`)

  The name of the table to which the row of output refers. This can also be one of the following values:

  - `<union*M*,*N*>`: The row refers to the union of the rows with `id` values of *`M`* and *`N`*.
  - `<derived*N*>`: The row refers to the derived table result for the row with an `id` value of *`N`*. A derived table may result, for example, from a subquery in the `FROM` clause.
  - `<subquery*N*>`: The row refers to the result of a materialized subquery for the row with an `id` value of *`N`*. See Section 8.2.2.2, “Optimizing Subqueries with Materialization”.

- `partitions` (JSON name: `partitions`)

  The partitions from which records would be matched by the query. This column is displayed only if the `PARTITIONS` keyword is used. The value is `NULL` for nonpartitioned tables. See Section 19.3.5, “Obtaining Information About Partitions”.

- `type` (JSON name: `access_type`)

  The join type. For descriptions of the different types, see `EXPLAIN` Join Types.

- `possible_keys` (JSON name: `possible_keys`)

  The `possible_keys` column indicates the indexes from which MySQL can choose to find the rows in this table. Note that this column is totally independent of the order of the tables as displayed in the output from `EXPLAIN`. That means that some of the keys in `possible_keys` might not be usable in practice with the generated table order.

If this column is `NULL` (or undefined in JSON-formatted output), there are no relevant indexes. In this case, you may be able to improve the performance of your query by examining the `WHERE` clause to check whether it refers to some column or columns that would be suitable for indexing. If so, create an appropriate index and check the query with `EXPLAIN` again. See Section 13.1.7, “ALTER TABLE Statement”.

To see what indexes a table has, use `SHOW INDEX FROM tbl_name`.

- `key` (JSON name: `key`)

  The `key` column indicates the key (index) that MySQL actually decided to use. If MySQL decides to use one of the `possible_keys` indexes to look up rows, that index is listed as the key value.

  It is possible that `key` names an index that is not present in the `possible_keys` value. This can happen if none of the `possible_keys` indexes are suitable for looking up rows, but all the columns selected by the query are columns of some other index. That is, the named index covers the selected columns, so although it is not used to determine which rows to retrieve, an index scan is more efficient than a data row scan.

  For `InnoDB`, a secondary index might cover the selected columns even if the query also selects the primary key because `InnoDB` stores the primary key value with each secondary index. If `key` is `NULL`, MySQL found no index to use for executing the query more efficiently.

  To force MySQL to use or ignore an index listed in the `possible_keys` column, use `FORCE INDEX`, `USE INDEX`, or `IGNORE INDEX` in your query. See Section 8.9.3, “Index Hints”.

  For `MyISAM` and `NDB` tables, running `ANALYZE TABLE` helps the optimizer choose better indexes. For `NDB` tables, this also improves performance of distributed pushed-down joins. For `MyISAM` tables, `myisamchk --analyze` does the same as `ANALYZE TABLE`. See Section 7.6, “MyISAM Table Maintenance and Crash Recovery”.

- `key_len` (JSON name: `key_length`)

  The `key_len` column indicates the length of the key that MySQL decided to use. The value of `key_len` enables you to determine how many parts of a multiple-part key MySQL actually uses. If the `key` column says `NULL`, the `key_len` column also says `NULL`.

  Due to the key storage format, the key length is one greater for a column that can be `NULL` than for a `NOT NULL` column.

- `ref` (JSON name: `ref`)

  The `ref` column shows which columns or constants are compared to the index named in the `key` column to select rows from the table.

  If the value is `func`, the value used is the result of some function. To see which function, use `SHOW WARNINGS` following `EXPLAIN EXTENDED` to see the extended `EXPLAIN` output. The function might actually be an operator such as an arithmetic operator.

- `rows` (JSON name: `rows`)

  The `rows` column indicates the number of rows MySQL believes it must examine to execute the query.

  For `InnoDB` tables, this number is an estimate, and may not always be exact.

- `filtered` (JSON name: `filtered`)

  The `filtered` column indicates an estimated percentage of table rows that are filtered by the table condition. The maximum value is 100, which means no filtering of rows occurred. Values decreasing from 100 indicate increasing amounts of filtering. `rows` shows the estimated number of rows

examined and `rows` × `filtered` shows the number of rows that are joined with the following table. For example, if `rows` is 1000 and `filtered` is 50.00 (50%), the number of rows to be joined with the following table is 1000 × 50% = 500. This column is displayed if you use `EXPLAIN EXTENDED`.

- `Extra` (JSON name: none)

  This column contains additional information about how MySQL resolves the query. For descriptions of the different values, see `EXPLAIN` Extra Information.

  There is no single JSON property corresponding to the `Extra` column; however, values that can occur in this column are exposed as JSON properties, or as the text of the `message` property.

EXPLAIN Join Types

The `type` column of `EXPLAIN` output describes how tables are joined. In JSON-formatted output, these are found as values of the `access_type` property. The following list describes the join types, ordered from the best type to the worst:

- `system`

  The table has only one row (= system table). This is a special case of the `const` join type.

- `const`

  The table has at most one matching row, which is read at the start of the query. Because there is only one row, values from the column in this row can be regarded as constants by the rest of the optimizer. `const` tables are very fast because they are read only once.

  `const` is used when you compare all parts of a `PRIMARY KEY` or `UNIQUE` index to constant values. In the following queries, *`tbl_name`* can be used as a `const` table:

```
SELECT * FROM tbl_name WHERE primary_key=1;

SELECT * FROM tbl_name
  WHERE primary_key_part1=1 AND primary_key_part2=2;
```

- `eq_ref`

  One row is read from this table for each combination of rows from the previous tables. Other than the `system` and `const` types, this is the best possible join type. It is used when all parts of an index are used by the join and the index is a `PRIMARY KEY` or `UNIQUE NOT NULL` index.

  `eq_ref` can be used for indexed columns that are compared using the `=` operator. The comparison value can be a constant or an expression that uses columns from tables that are read before this table. In the following examples, MySQL can use an `eq_ref` join to process *`ref_table`*:

```
SELECT * FROM ref_table,other_table
  WHERE ref_table.key_column=other_table.column;

SELECT * FROM ref_table,other_table
  WHERE ref_table.key_column_part1=other_table.column
  AND ref_table.key_column_part2=1;
```

- `ref`

  All rows with matching index values are read from this table for each combination of rows from the previous tables. `ref` is used if the join uses only a leftmost prefix of the key or if the key is not a `PRIMARY KEY` or `UNIQUE` index (in other words, if the join cannot select a single row based on the key value). If the key that is used matches only a few rows, this is a good join type.

  `ref` can be used for indexed columns that are compared using the `=` or `<=>` operator. In the following examples, MySQL can use a `ref` join to process *`ref_table`*:

```
SELECT * FROM ref_table WHERE key_column=expr;
```

```
SELECT * FROM ref_table,other_table
  WHERE ref_table.key_column=other_table.column;

SELECT * FROM ref_table,other_table
  WHERE ref_table.key_column_part1=other_table.column
  AND ref_table.key_column_part2=1;
```

- `fulltext`

  The join is performed using a `FULLTEXT` index.

- `ref_or_null`

  This join type is like `ref`, but with the addition that MySQL does an extra search for rows that contain `NULL` values. This join type optimization is used most often in resolving subqueries. In the following examples, MySQL can use a `ref_or_null` join to process *`ref_table`*:

  ```
  SELECT * FROM ref_table
    WHERE key_column=expr OR key_column IS NULL;
  ```

  See Section 8.2.1.12, “IS NULL Optimization”.

- `index_merge`

  This join type indicates that the Index Merge optimization is used. In this case, the `key` column in the output row contains a list of indexes used, and `key_len` contains a list of the longest key parts for the indexes used. For more information, see Section 8.2.1.3, “Index Merge Optimization”.

- `unique_subquery`

  This type replaces `eq_ref` for some `IN` subqueries of the following form:

  ```
  value IN (SELECT primary_key FROM single_table WHERE some_expr)
  ```

  `unique_subquery` is just an index lookup function that replaces the subquery completely for better efficiency.

- `index_subquery`

  This join type is similar to `unique_subquery`. It replaces `IN` subqueries, but it works for nonunique indexes in subqueries of the following form:

  ```
  value IN (SELECT key_column FROM single_table WHERE some_expr)
  ```

- `range`

  Only rows that are in a given range are retrieved, using an index to select the rows. The `key` column in the output row indicates which index is used. The `key_len` contains the longest key part that was used. The `ref` column is `NULL` for this type.

  `range` can be used when a key column is compared to a constant using any of the `=`, `<>`, `>`, `>=`, `<`, `<=`, `IS NULL`, `<=>`, `BETWEEN`, `LIKE`, or `IN()` operators:

  ```
  SELECT * FROM tbl_name
    WHERE key_column = 10;

  SELECT * FROM tbl_name
    WHERE key_column BETWEEN 10 and 20;

  SELECT * FROM tbl_name
    WHERE key_column IN (10,20,30);

  SELECT * FROM tbl_name
    WHERE key_part1 = 10 AND key_part2 IN (10,20,30);
  ```

- `index`

  The `index` join type is the same as `ALL`, except that the index tree is scanned. This occurs two ways:

  - If the index is a covering index for the queries and can be used to satisfy all data required from the table, only the index tree is scanned. In this case, the `Extra` column says `Using index`. An index-only scan usually is faster than `ALL` because the size of the index usually is smaller than the table data.
  - A full table scan is performed using reads from the index to look up data rows in index order. `Uses index` does not appear in the `Extra` column.

  MySQL can use this join type when the query uses only columns that are part of a single index.

- `ALL`

  A full table scan is done for each combination of rows from the previous tables. This is normally not good if the table is the first table not marked `const`, and usually *very* bad in all other cases. Normally, you can avoid `ALL` by adding indexes that enable row retrieval from the table based on constant values or column values from earlier tables.

EXPLAIN Extra Information

The `Extra` column of `EXPLAIN` output contains additional information about how MySQL resolves the query. The following list explains the values that can appear in this column. Each item also indicates for JSON-formatted output which property displays the `Extra` value. For some of these, there is a specific property. The others display as the text of the `message` property.

If you want to make your queries as fast as possible, look out for `Extra` column values of `Using filesort` and `Using temporary`, or, in JSON-formatted `EXPLAIN` output, for `using_filesort` and `using_temporary_table` properties equal to `true`.

- `Child of 'table' pushed join@1` (JSON: `message` text)

  This table is referenced as the child of *`table`* in a join that can be pushed down to the NDB kernel. Applies only in NDB Cluster, when pushed-down joins are enabled. See the description of the `ndb_join_pushdown` server system variable for more information and examples.

- `const row not found` (JSON property: `const_row_not_found`)

  For a query such as `SELECT ... FROM tbl_name`, the table was empty.

- `Deleting all rows` (JSON property: `message`)

  For `DELETE`, some storage engines (such as `MyISAM`) support a handler method that removes all table rows in a simple and fast way. This `Extra` value is displayed if the engine uses this optimization.

- `Distinct` (JSON property: `distinct`)

  MySQL is looking for distinct values, so it stops searching for more rows for the current row combination after it has found the first matching row.

- `FirstMatch(tbl_name)` (JSON property: `first_match`)

  The semijoin FirstMatch join shortcutting strategy is used for *`tbl_name`*.

- `Full scan on NULL key` (JSON property: `message`)

  This occurs for subquery optimization as a fallback strategy when the optimizer cannot use an index-lookup access method.

- `Impossible HAVING` (JSON property: `message`)

  The `HAVING` clause is always false and cannot select any rows.

- `Impossible WHERE` (JSON property: `message`)

  The `WHERE` clause is always false and cannot select any rows.

- `Impossible WHERE noticed after reading const tables` (JSON property: `message`)

  MySQL has read all `const` (and `system`) tables and notice that the `WHERE` clause is always false.

- `LooseScan(`*`m`*`..`*`n`*`)` (JSON property: `message`)

  The semijoin LooseScan strategy is used. *`m`* and *`n`* are key part numbers.

- `No matching min/max row` (JSON property: `message`)

  No row satisfies the condition for a query such as `SELECT MIN(...) FROM ... WHERE` *`condition`*.

- `no matching row in const table` (JSON property: `message`)

  For a query with a join, there was an empty table or a table with no rows satisfying a unique index condition.

- `No matching rows after partition pruning` (JSON property: `message`)

  For `DELETE` or `UPDATE`, the optimizer found nothing to delete or update after partition pruning. It is similar in meaning to `Impossible WHERE` for `SELECT` statements.

- `No tables used` (JSON property: `message`)

  The query has no `FROM` clause, or has a `FROM DUAL` clause.

  For `INSERT` or `REPLACE` statements, `EXPLAIN` displays this value when there is no `SELECT` part. For example, it appears for `EXPLAIN INSERT INTO t VALUES(10)` because that is equivalent to `EXPLAIN INSERT INTO t SELECT 10 FROM DUAL`.

- `Not exists` (JSON property: `message`)

  MySQL was able to do a `LEFT JOIN` optimization on the query and does not examine more rows in this table for the previous row combination after it finds one row that matches the `LEFT JOIN` criteria. Here is an example of the type of query that can be optimized this way:

  ```
  SELECT * FROM t1 LEFT JOIN t2 ON t1.id=t2.id
    WHERE t2.id IS NULL;
  ```

  Assume that `t2.id` is defined as `NOT NULL`. In this case, MySQL scans `t1` and looks up the rows in `t2` using the values of `t1.id`. If MySQL finds a matching row in `t2`, it knows that `t2.id` can never be `NULL`, and does not scan through the rest of the rows in `t2` that have the same `id` value. In other words, for each row in `t1`, MySQL needs to do only a single lookup in `t2`, regardless of how many rows actually match in `t2`.

- `Range checked for each record (index map: `*`N`*`)` (JSON property: `message`)

  MySQL found no good index to use, but found that some of indexes might be used after column values from preceding tables are known. For each row combination in the preceding tables, MySQL checks whether it is possible to use a `range` or `index_merge` access method to retrieve rows. This is not very fast, but is faster than performing a join with no index at all. The applicability criteria are as described in Section 8.2.1.2, “Range Optimization”, and Section 8.2.1.3, “Index Merge Optimization”, with the exception that all column values for the preceding table are known and considered to be constants.

Indexes are numbered beginning with 1, in the same order as shown by `SHOW INDEX` for the table. The index map value *`N`* is a bitmask value that indicates which indexes are candidates. For example, a value of `0x19` (binary 11001) means that indexes 1, 4, and 5 are considered.

- `Scanned N databases` (JSON property: `message`)

  This indicates how many directory scans the server performs when processing a query for `INFORMATION_SCHEMA` tables, as described in Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”. The value of *`N`* can be 0, 1, or `all`.

- `Select tables optimized away` (JSON property: `message`)

  The optimizer determined 1) that at most one row should be returned, and 2) that to produce this row, a deterministic set of rows must be read. When the rows to be read can be read during the optimization phase (for example, by reading index rows), there is no need to read any tables during query execution.

  The first condition is fulfilled when the query is implicitly grouped (contains an aggregate function but no `GROUP BY` clause). The second condition is fulfilled when one row lookup is performed per index used. The number of indexes read determines the number of rows to read.

  Consider the following implicitly grouped query:

  ```
  SELECT MIN(c1), MIN(c2) FROM t1;
  ```

  Suppose that `MIN(c1)` can be retrieved by reading one index row and `MIN(c2)` can be retrieved by reading one row from a different index. That is, for each column `c1` and `c2`, there exists an index where the column is the first column of the index. In this case, one row is returned, produced by reading two deterministic rows.

  This `Extra` value does not occur if the rows to read are not deterministic. Consider this query:

  ```
  SELECT MIN(c2) FROM t1 WHERE c1 <= 10;
  ```

  Suppose that `(c1, c2)` is a covering index. Using this index, all rows with `c1 <= 10` must be scanned to find the minimum `c2` value. By contrast, consider this query:

  ```
  SELECT MIN(c2) FROM t1 WHERE c1 = 10;
  ```

  In this case, the first index row with `c1 = 10` contains the minimum `c2` value. Only one row must be read to produce the returned row.

  For storage engines that maintain an exact row count per table (such as `MyISAM`, but not `InnoDB`), this `Extra` value can occur for `COUNT(*)` queries for which the `WHERE` clause is missing or always true and there is no `GROUP BY` clause. (This is an instance of an implicitly grouped query where the storage engine influences whether a deterministic number of rows can be read.)

- `Skip_open_table`, `Open_frm_only`, `Open_full_table` (JSON property: `message`)

  These values indicate file-opening optimizations that apply to queries for `INFORMATION_SCHEMA` tables, as described in Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”.

  - `Skip_open_table`: Table files do not need to be opened. The information has already become available within the query by scanning the database directory.
  - `Open_frm_only`: Only the table's `.frm` file need be opened.
  - `Open_full_table`: The unoptimized information lookup. The `.frm`, `.MYD`, and `.MYI` files must be opened.

- `Start temporary`, `End temporary` (JSON property: `message`)

  This indicates temporary table use for the semijoin Duplicate Weedout strategy.

- `unique row not found` (JSON property: `message`)

  For a query such as `SELECT ... FROM tbl_name`, no rows satisfy the condition for a `UNIQUE` index or `PRIMARY KEY` on the table.

- `Using filesort` (JSON property: `using_filesort`)

  MySQL must do an extra pass to find out how to retrieve the rows in sorted order. The sort is done by going through all rows according to the join type and storing the sort key and pointer to the row for all rows that match the `WHERE` clause. The keys then are sorted and the rows are retrieved in sorted order. See Section 8.2.1.13, “ORDER BY Optimization”.

- `Using index` (JSON property: `using_index`)

  The column information is retrieved from the table using only information in the index tree without having to do an additional seek to read the actual row. This strategy can be used when the query uses only columns that are part of a single index.

  For `InnoDB` tables that have a user-defined clustered index, that index can be used even when `Using index` is absent from the `Extra` column. This is the case if `type` is `index` and `key` is `PRIMARY`.

- `Using index condition` (JSON property: `using_index_condition`)

  Tables are read by accessing index tuples and testing them first to determine whether to read full table rows. In this way, index information is used to defer (“push down”) reading full table rows unless it is necessary. See Section 8.2.1.5, “Index Condition Pushdown Optimization”.

- `Using index for group-by` (JSON property: `using_index_for_group_by`)

  Similar to the `Using index` table access method, `Using index for group-by` indicates that MySQL found an index that can be used to retrieve all columns of a `GROUP BY` or `DISTINCT` query without any extra disk access to the actual table. Additionally, the index is used in the most efficient way so that for each group, only a few index entries are read. For details, see Section 8.2.1.14, “GROUP BY Optimization”.

- `Using join buffer (Block Nested Loop)`, `Using join buffer (Batched Key Access)` (JSON property: `using_join_buffer`)

  Tables from earlier joins are read in portions into the join buffer, and then their rows are used from the buffer to perform the join with the current table. `(Block Nested Loop)` indicates use of the Block Nested-Loop algorithm and `(Batched Key Access)` indicates use of the Batched Key Access algorithm. That is, the keys from the table on the preceding line of the `EXPLAIN` output are buffered, and the matching rows are fetched in batches from the table represented by the line in which `Using join buffer` appears.

  In JSON-formatted output, the value of `using_join_buffer` is always either one of `Block Nested Loop` or `Batched Key Access`.

  For more information about these algorithms, see Block Nested-Loop Join Algorithm, and Batched Key Access Joins.

- `Using MRR` (JSON property: `message`)

  Tables are read using the Multi-Range Read optimization strategy. See Section 8.2.1.10, “Multi-Range Read Optimization”.

- `Using sort_union(...)`, `Using union(...)`, `Using intersect(...)` (JSON property: `message`)

  These indicate the particular algorithm showing how index scans are merged for the `index_merge` join type. See Section 8.2.1.3, “Index Merge Optimization”.

- `Using temporary` (JSON property: `using_temporary_table`)

  To resolve the query, MySQL needs to create a temporary table to hold the result. This typically happens if the query contains `GROUP BY` and `ORDER BY` clauses that list columns differently.

- `Using where` (JSON property: `attached_condition`)

  A `WHERE` clause is used to restrict which rows to match against the next table or send to the client. Unless you specifically intend to fetch or examine all rows from the table, you may have something wrong in your query if the `Extra` value is not `Using where` and the table join type is `ALL` or `index`.

  `Using where` has no direct counterpart in JSON-formatted output; the `attached_condition` property contains any `WHERE` condition used.

- `Using where with pushed condition` (JSON property: `message`)

  This item applies to `NDB` tables *only*. It means that NDB Cluster is using the Condition Pushdown optimization to improve the efficiency of a direct comparison between a nonindexed column and a constant. In such cases, the condition is “pushed down” to the cluster's data nodes and is evaluated on all data nodes simultaneously. This eliminates the need to send nonmatching rows over the network, and can speed up such queries by a factor of 5 to 10 times over cases where Condition Pushdown could be but is not used. For more information, see Section 8.2.1.4, “Engine Condition Pushdown Optimization”.

EXPLAIN Output Interpretation

You can get a good indication of how good a join is by taking the product of the values in the `rows` column of the `EXPLAIN` output. This should tell you roughly how many rows MySQL must examine to execute the query. If you restrict queries with the `max_join_size` system variable, this row product also is used to determine which multiple-table `SELECT` statements to execute and which to abort. See Section 5.1.1, “Configuring the Server”.

The following example shows how a multiple-table join can be optimized progressively based on the information provided by `EXPLAIN`.

Suppose that you have the `SELECT` statement shown here and that you plan to examine it using `EXPLAIN`:

```
EXPLAIN SELECT tt.TicketNumber, tt.TimeIn,
               tt.ProjectReference, tt.EstimatedShipDate,
               tt.ActualShipDate, tt.ClientID,
               tt.ServiceCodes, tt.RepetitiveID,
               tt.CurrentProcess, tt.CurrentDPPerson,
               tt.RecordVolume, tt.DPPrinted, et.COUNTRY,
               et_1.COUNTRY, do.CUSTNAME
        FROM tt, et, et AS et_1, do
        WHERE tt.SubmitTime IS NULL
          AND tt.ActualPC = et.EMPLOYID
          AND tt.AssignedPC = et_1.EMPLOYID
          AND tt.ClientID = do.CUSTNMBR;
```

For this example, make the following assumptions:

- The columns being compared have been declared as follows.

Table	Column	Data Type
`tt`	`ActualPC`	`CHAR(10)`

Table	Column	Data Type
tt	AssignedPC	CHAR(10)
tt	ClientID	CHAR(10)
et	EMPLOYID	CHAR(15)
do	CUSTNMBR	CHAR(15)

- The tables have the following indexes.

Table	Index
tt	ActualPC
tt	AssignedPC
tt	ClientID
et	EMPLOYID (primary key)
do	CUSTNMBR (primary key)

- The tt.ActualPC values are not evenly distributed.

Initially, before any optimizations have been performed, the EXPLAIN statement produces the following information:

```
table type possible_keys key  key_len ref  rows  Extra
et    ALL  PRIMARY       NULL NULL    NULL 74
do    ALL  PRIMARY       NULL NULL    NULL 2135
et_1  ALL  PRIMARY       NULL NULL    NULL 74
tt    ALL  AssignedPC,   NULL NULL    NULL 3872
           ClientID,
           ActualPC
      Range checked for each record (index map: 0x23)
```

Because type is ALL for each table, this output indicates that MySQL is generating a Cartesian product of all the tables; that is, every combination of rows. This takes quite a long time, because the product of the number of rows in each table must be examined. For the case at hand, this product is 74 × 2135 × 74 × 3872 = 45,268,558,720 rows. If the tables were bigger, you can only imagine how long it would take.

One problem here is that MySQL can use indexes on columns more efficiently if they are declared as the same type and size. In this context, VARCHAR and CHAR are considered the same if they are declared as the same size. tt.ActualPC is declared as CHAR(10) and et.EMPLOYID is CHAR(15), so there is a length mismatch.

To fix this disparity between column lengths, use ALTER TABLE to lengthen ActualPC from 10 characters to 15 characters:

```
mysql> ALTER TABLE tt MODIFY ActualPC VARCHAR(15);
```

Now tt.ActualPC and et.EMPLOYID are both VARCHAR(15). Executing the EXPLAIN statement again produces this result:

```
table type   possible_keys key     key_len ref         rows    Extra
tt    ALL    AssignedPC,   NULL    NULL    NULL        3872    Using
             ClientID,                                         where
             ActualPC
do    ALL    PRIMARY       NULL    NULL    NULL        2135
      Range checked for each record (index map: 0x1)
et_1  ALL    PRIMARY       NULL    NULL    NULL        74
      Range checked for each record (index map: 0x1)
et    eq_ref PRIMARY       PRIMARY 15      tt.ActualPC 1
```

This is not perfect, but is much better: The product of the rows values is less by a factor of 74. This version executes in a couple of seconds.

A second alteration can be made to eliminate the column length mismatches for the `tt.AssignedPC = et_1.EMPLOYID` and `tt.ClientID = do.CUSTNMBR` comparisons:

```
mysql> ALTER TABLE tt MODIFY AssignedPC VARCHAR(15),
                      MODIFY ClientID   VARCHAR(15);
```

After that modification, `EXPLAIN` produces the output shown here:

```
table type   possible_keys key      key_len ref           rows Extra
et    ALL    PRIMARY       NULL     NULL    NULL          74
tt    ref    AssignedPC,   ActualPC 15      et.EMPLOYID   52   Using
             ClientID,                                        where
             ActualPC
et_1  eq_ref PRIMARY       PRIMARY  15      tt.AssignedPC 1
do    eq_ref PRIMARY       PRIMARY  15      tt.ClientID   1
```

At this point, the query is optimized almost as well as possible. The remaining problem is that, by default, MySQL assumes that values in the `tt.ActualPC` column are evenly distributed, and that is not the case for the `tt` table. Fortunately, it is easy to tell MySQL to analyze the key distribution:

```
mysql> ANALYZE TABLE tt;
```

With the additional index information, the join is perfect and `EXPLAIN` produces this result:

```
table type   possible_keys key     key_len ref           rows Extra
tt    ALL    AssignedPC    NULL    NULL    NULL          3872 Using
             ClientID,                                        where
             ActualPC
et    eq_ref PRIMARY       PRIMARY 15      tt.ActualPC   1
et_1  eq_ref PRIMARY       PRIMARY 15      tt.AssignedPC 1
do    eq_ref PRIMARY       PRIMARY 15      tt.ClientID   1
```

The `rows` column in the output from `EXPLAIN` is an educated guess from the MySQL join optimizer. Check whether the numbers are even close to the truth by comparing the `rows` product with the actual number of rows that the query returns. If the numbers are quite different, you might get better performance by using `STRAIGHT_JOIN` in your `SELECT` statement and trying to list the tables in a different order in the `FROM` clause. (However, `STRAIGHT_JOIN` may prevent indexes from being used because it disables semijoin transformations. See Section 8.2.2.1, “Optimizing Subqueries with Semijoin Transformations”.)

It is possible in some cases to execute statements that modify data when `EXPLAIN SELECT` is used with a subquery; for more information, see Section 13.2.10.8, “Derived Tables”.

8.8.3 Extended EXPLAIN Output Format

When `EXPLAIN` is used with the `EXTENDED` keyword, the output includes a `filtered` column not otherwise displayed. This column indicates the estimated percentage of table rows that are filtered by the table condition.

In addition, for `SELECT` statements, the statement produces extra (“extended”) information that is not part of `EXPLAIN` output but can be viewed by issuing a `SHOW WARNINGS` statement following `EXPLAIN`. The `Message` value in `SHOW WARNINGS` output displays how the optimizer qualifies table and column names in the `SELECT` statement, what the `SELECT` looks like after the application of rewriting and optimization rules, and possibly other notes about the optimization process.

The extended information displayable with a `SHOW WARNINGS` statement following `EXPLAIN` is produced only for `SELECT` statements. `SHOW WARNINGS` displays an empty result for other explainable statements (`DELETE`, `INSERT`, `REPLACE`, and `UPDATE`).

Here is an example of extended `EXPLAIN` output:

```
mysql> EXPLAIN EXTENDED
       SELECT t1.a, t1.a IN (SELECT t2.a FROM t2) FROM t1\G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: t1
```

```
         type: index
possible_keys: NULL
          key: PRIMARY
      key_len: 4
          ref: NULL
         rows: 4
     filtered: 100.00
        Extra: Using index
*************************** 2. row ***************************
           id: 2
  select_type: SUBQUERY
        table: t2
         type: index
possible_keys: a
          key: a
      key_len: 5
          ref: NULL
         rows: 3
     filtered: 100.00
        Extra: Using index
2 rows in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Note
   Code: 1003
Message: /* select#1 */ select `test`.`t1`.`a` AS `a`,
         <in_optimizer>(`test`.`t1`.`a`,`test`.`t1`.`a` in
         ( <materialize> (/* select#2 */ select `test`.`t2`.`a`
         from `test`.`t2` where 1 having 1 ),
         <primary_index_lookup>(`test`.`t1`.`a` in
         <temporary table> on <auto_key>
         where ((`test`.`t1`.`a` = `materialized-subquery`.`a`))))) AS `t1.a
         IN (SELECT t2.a FROM t2)` from `test`.`t1`
1 row in set (0.00 sec)
```

Because the statement displayed by `SHOW WARNINGS` may contain special markers to provide information about query rewriting or optimizer actions, the statement is not necessarily valid SQL and is not intended to be executed. The output may also include rows with `Message` values that provide additional non-SQL explanatory notes about actions taken by the optimizer.

The following list describes special markers that can appear in the extended output displayed by `SHOW WARNINGS`:

- `<auto_key>`

  An automatically generated key for a temporary table.

- `<cache>(expr)`

  The expression (such as a scalar subquery) is executed once and the resulting value is saved in memory for later use. For results consisting of multiple values, a temporary table may be created and you may see `<temporary table>` instead.

- `<exists>(query fragment)`

  The subquery predicate is converted to an `EXISTS` predicate and the subquery is transformed so that it can be used together with the `EXISTS` predicate.

- `<in_optimizer>(query fragment)`

  This is an internal optimizer object with no user significance.

- `<index_lookup>(query fragment)`

  The query fragment is processed using an index lookup to find qualifying rows.

- `<if>(condition, expr1, expr2)`

If the condition is true, evaluate to *expr1*, otherwise *expr2*.

- <is_not_null_test>(*expr*)

  A test to verify that the expression does not evaluate to NULL.

- <materialize>(*query fragment*)

  Subquery materialization is used.

- `materialized-subquery`.*col_name*, `materialized subselect`.*col_name*

  A reference to the column *col_name* in an internal temporary table materialized to hold the result from evaluating a subquery.

- <primary_index_lookup>(*query fragment*)

  The query fragment is processed using a primary key lookup to find qualifying rows.

- <ref_null_helper>(*expr*)

  This is an internal optimizer object with no user significance.

- /* select#*N* */ *select_stmt*

  The SELECT is associated with the row in non-extended EXPLAIN output that has an id value of *N*.

- *outer_tables* semi join (*inner_tables*)

  A semijoin operation. *inner_tables* shows the tables that were not pulled out. See Section 8.2.2.1, “Optimizing Subqueries with Semijoin Transformations”.

- <temporary table>

  This represents an internal temporary table created to cache an intermediate result.

When some tables are of const or system type, expressions involving columns from these tables are evaluated early by the optimizer and are not part of the displayed statement. However, with FORMAT=JSON, some const table accesses are displayed as a ref access that uses a const value.

8.8.4 Estimating Query Performance

In most cases, you can estimate query performance by counting disk seeks. For small tables, you can usually find a row in one disk seek (because the index is probably cached). For bigger tables, you can estimate that, using B-tree indexes, you need this many seeks to find a row: log(*row_count*) / log(*index_block_length* / 3 * 2 / (*index_length* + *data_pointer_length*)) + 1.

In MySQL, an index block is usually 1,024 bytes and the data pointer is usually four bytes. For a 500,000-row table with a key value length of three bytes (the size of MEDIUMINT), the formula indicates log(500,000)/log(1024/3*2/(3+4)) + 1 = 4 seeks.

This index would require storage of about 500,000 * 7 * 3/2 = 5.2MB (assuming a typical index buffer fill ratio of 2/3), so you probably have much of the index in memory and so need only one or two calls to read data to find the row.

For writes, however, you need four seek requests to find where to place a new index value and normally two seeks to update the index and write the row.

The preceding discussion does not mean that your application performance slowly degenerates by log *N*. As long as everything is cached by the OS or the MySQL server, things become only marginally slower as the table gets bigger. After the data gets too big to be cached, things start to go much slower until your applications are bound only by disk seeks (which increase by log *N*). To avoid this, increase

the key cache size as the data grows. For `MyISAM` tables, the key cache size is controlled by the `key_buffer_size` system variable. See Section 5.1.1, “Configuring the Server”.

8.9 Controlling the Query Optimizer

MySQL provides optimizer control through system variables that affect how query plans are evaluated, switchable optimizations, and index hints.

8.9.1 Controlling Query Plan Evaluation

The task of the query optimizer is to find an optimal plan for executing an SQL query. Because the difference in performance between “good” and “bad” plans can be orders of magnitude (that is, seconds versus hours or even days), most query optimizers, including that of MySQL, perform a more or less exhaustive search for an optimal plan among all possible query evaluation plans. For join queries, the number of possible plans investigated by the MySQL optimizer grows exponentially with the number of tables referenced in a query. For small numbers of tables (typically less than 7 to 10) this is not a problem. However, when larger queries are submitted, the time spent in query optimization may easily become the major bottleneck in the server's performance.

A more flexible method for query optimization enables the user to control how exhaustive the optimizer is in its search for an optimal query evaluation plan. The general idea is that the fewer plans that are investigated by the optimizer, the less time it spends in compiling a query. On the other hand, because the optimizer skips some plans, it may miss finding an optimal plan.

The behavior of the optimizer with respect to the number of plans it evaluates can be controlled using two system variables:

- The `optimizer_prune_level` variable tells the optimizer to skip certain plans based on estimates of the number of rows accessed for each table. Our experience shows that this kind of “educated guess” rarely misses optimal plans, and may dramatically reduce query compilation times. That is why this option is on (`optimizer_prune_level=1`) by default. However, if you believe that the optimizer missed a better query plan, this option can be switched off (`optimizer_prune_level=0`) with the risk that query compilation may take much longer. Note that, even with the use of this heuristic, the optimizer still explores a roughly exponential number of plans.
- The `optimizer_search_depth` variable tells how far into the “future” of each incomplete plan the optimizer should look to evaluate whether it should be expanded further. Smaller values of `optimizer_search_depth` may result in orders of magnitude smaller query compilation times. For example, queries with 12, 13, or more tables may easily require hours and even days to compile if `optimizer_search_depth` is close to the number of tables in the query. At the same time, if compiled with `optimizer_search_depth` equal to 3 or 4, the optimizer may compile in less than a minute for the same query. If you are unsure of what a reasonable value is for `optimizer_search_depth`, this variable can be set to 0 to tell the optimizer to determine the value automatically.

8.9.2 Switchable Optimizations

The `optimizer_switch` system variable enables control over optimizer behavior. Its value is a set of flags, each of which has a value of `on` or `off` to indicate whether the corresponding optimizer behavior is enabled or disabled. This variable has global and session values and can be changed at runtime. The global default can be set at server startup.

To see the current set of optimizer flags, select the variable value:

```
mysql> SELECT @@optimizer_switch\G
*************************** 1. row ***************************
@@optimizer_switch: index_merge=on,index_merge_union=on,
                    index_merge_sort_union=on,
                    index_merge_intersection=on,
                    engine_condition_pushdown=on,
                    index_condition_pushdown=on,
```

```
                    mrr=on,mrr_cost_based=on,
                    block_nested_loop=on,batched_key_access=off,
                    materialization=on,semijoin=on,loosescan=on,
                    firstmatch=on,
                    subquery_materialization_cost_based=on,
                    use_index_extensions=on
```

To change the value of `optimizer_switch`, assign a value consisting of a comma-separated list of one or more commands:

```
SET [GLOBAL|SESSION] optimizer_switch='command[,command]...';
```

Each *`command`* value should have one of the forms shown in the following table.

Command Syntax	Meaning
`default`	Reset every optimization to its default value
`opt_name``=default`	Set the named optimization to its default value
`opt_name``=off`	Disable the named optimization
`opt_name``=on`	Enable the named optimization

The order of the commands in the value does not matter, although the `default` command is executed first if present. Setting an *`opt_name`* flag to `default` sets it to whichever of `on` or `off` is its default value. Specifying any given *`opt_name`* more than once in the value is not permitted and causes an error. Any errors in the value cause the assignment to fail with an error, leaving the value of `optimizer_switch` unchanged.

The following list describes the permissible *`opt_name`* flag names, grouped by optimization strategy:

- Batched Key Access Flags
  - `batched_key_access` (default `off`)

    Controls use of BKA join algorithm.

  For `batched_key_access` to have any effect when set to `on`, the `mrr` flag must also be `on`. Currently, the cost estimation for MRR is too pessimistic. Hence, it is also necessary for `mrr_cost_based` to be `off` for BKA to be used.

  For more information, see Section 8.2.1.11, “Block Nested-Loop and Batched Key Access Joins”.
- Block Nested-Loop Flags
  - `block_nested_loop` (default `on`)

    Controls use of BNL join algorithm.

  For more information, see Section 8.2.1.11, “Block Nested-Loop and Batched Key Access Joins”.
- Engine Condition Pushdown Flags
  - `engine_condition_pushdown` (default `on`)

    Controls engine condition pushdown.

  For more information, see Section 8.2.1.4, “Engine Condition Pushdown Optimization”.
- Index Condition Pushdown Flags
  - `index_condition_pushdown` (default `on`)

    Controls index condition pushdown.

  For more information, see Section 8.2.1.5, “Index Condition Pushdown Optimization”.

- Index Extensions Flags
  - `use_index_extensions` (default `on`)

    Controls use of index extensions.

  For more information, see Section 8.3.9, “Use of Index Extensions”.
- Index Merge Flags
  - `index_merge` (default `on`)

    Controls all Index Merge optimizations.
  - `index_merge_intersection` (default `on`)

    Controls the Index Merge Intersection Access optimization.
  - `index_merge_sort_union` (default `on`)

    Controls the Index Merge Sort-Union Access optimization.
  - `index_merge_union` (default `on`)

    Controls the Index Merge Union Access optimization.

  For more information, see Section 8.2.1.3, “Index Merge Optimization”.
- Multi-Range Read Flags
  - `mrr` (default `on`)

    Controls the Multi-Range Read strategy.
  - `mrr_cost_based` (default `on`)

    Controls use of cost-based MRR if `mrr=on`.

  For more information, see Section 8.2.1.10, “Multi-Range Read Optimization”.
- Semijoin Flags
  - `semijoin` (default `on`)

    Controls all semijoin strategies.
  - `firstmatch` (default `on`)

    Controls the semijoin FirstMatch strategy.
  - `loosescan` (default `on`)

    Controls the semijoin LooseScan strategy (not to be confused with Loose Index Scan for `GROUP BY`).

  The `semijoin` flag controls whether semijoins are used. If it is set to `on`, the `firstmatch` and `loosescan` flags enable finer control over the permitted semijoin strategies.

  If `semijoin` and `materialization` are both `on`, semijoins also use materialization where applicable. These flags are `on` by default.

  For more information, see Section 8.2.2.1, “Optimizing Subqueries with Semijoin Transformations”.

- Subquery Materialization Flags

  - `materialization` (default `on`)

    Controls materialization (including semijoin materialization).

  - `subquery_materialization_cost_based` (default `on`)

    Use cost-based materialization choice.

  The `materialization` flag controls whether subquery materialization is used. If `semijoin` and `materialization` are both `on`, semijoins also use materialization where applicable. These flags are `on` by default.

  The `subquery_materialization_cost_based` flag enables control over the choice between subquery materialization and `IN`-to-`EXISTS` subquery transformation. If the flag is `on` (the default), the optimizer performs a cost-based choice between subquery materialization and `IN`-to-`EXISTS` subquery transformation if either method could be used. If the flag is `off`, the optimizer chooses subquery materialization over `IN`-to-`EXISTS` subquery transformation.

  For more information, see Section 8.2.2, “Optimizing Subqueries and Derived Tables”.

When you assign a value to `optimizer_switch`, flags that are not mentioned keep their current values. This makes it possible to enable or disable specific optimizer behaviors in a single statement without affecting other behaviors. The statement does not depend on what other optimizer flags exist and what their values are. Suppose that all Index Merge optimizations are enabled:

```
mysql> SELECT @@optimizer_switch\G
*************************** 1. row ***************************
@@optimizer_switch: index_merge=on,index_merge_union=on,
                    index_merge_sort_union=on,
                    index_merge_intersection=on,
                    engine_condition_pushdown=on,
                    index_condition_pushdown=on,
                    mrr=on,mrr_cost_based=on,
                    block_nested_loop=on,batched_key_access=off,
                    materialization=on,semijoin=on,loosescan=on,
                    firstmatch=on,
                    subquery_materialization_cost_based=on,
                    use_index_extensions=on
```

If the server is using the Index Merge Union or Index Merge Sort-Union access methods for certain queries and you want to check whether the optimizer performs better without them, set the variable value like this:

```
mysql> SET optimizer_switch='index_merge_union=off,index_merge_sort_union=off';

mysql> SELECT @@optimizer_switch\G
*************************** 1. row ***************************
@@optimizer_switch: index_merge=on,index_merge_union=off,
                    index_merge_sort_union=off,
                    index_merge_intersection=on,
                    engine_condition_pushdown=on,
                    index_condition_pushdown=on,
                    mrr=on,mrr_cost_based=on,
                    block_nested_loop=on,batched_key_access=off,
                    materialization=on,semijoin=on,loosescan=on,
                    firstmatch=on,
                    subquery_materialization_cost_based=on,
                    use_index_extensions=on
```

8.9.3 Index Hints

Index hints give the optimizer information about how to choose indexes during query processing. Index hints apply only to `SELECT` and `UPDATE` statements.

Index hints are specified following a table name. (For the general syntax for specifying tables in a SELECT statement, see Section 13.2.9.2, “JOIN Clause”.) The syntax for referring to an individual table, including index hints, looks like this:

```
tbl_name [[AS] alias] [index_hint_list]

index_hint_list:
    index_hint [index_hint] ...

index_hint:
    USE {INDEX|KEY}
      [FOR {JOIN|ORDER BY|GROUP BY}] ([index_list])
  | {IGNORE|FORCE} {INDEX|KEY}
      [FOR {JOIN|ORDER BY|GROUP BY}] (index_list)

index_list:
    index_name [, index_name] ...
```

The `USE INDEX (index_list)` hint tells MySQL to use only one of the named indexes to find rows in the table. The alternative syntax `IGNORE INDEX (index_list)` tells MySQL to not use some particular index or indexes. These hints are useful if `EXPLAIN` shows that MySQL is using the wrong index from the list of possible indexes.

The `FORCE INDEX` hint acts like `USE INDEX (index_list)`, with the addition that a table scan is assumed to be *very* expensive. In other words, a table scan is used only if there is no way to use one of the named indexes to find rows in the table.

Each hint requires index names, not column names. To refer to a primary key, use the name `PRIMARY`. To see the index names for a table, use the `SHOW INDEX` statement or the `INFORMATION_SCHEMA.STATISTICS` table.

An *`index_name`* value need not be a full index name. It can be an unambiguous prefix of an index name. If a prefix is ambiguous, an error occurs.

Examples:

```
SELECT * FROM table1 USE INDEX (col1_index,col2_index)
  WHERE col1=1 AND col2=2 AND col3=3;

SELECT * FROM table1 IGNORE INDEX (col3_index)
  WHERE col1=1 AND col2=2 AND col3=3;
```

The syntax for index hints has the following characteristics:

- It is syntactically valid to omit *`index_list`* for `USE INDEX`, which means “use no indexes.” Omitting *`index_list`* for `FORCE INDEX` or `IGNORE INDEX` is a syntax error.
- You can specify the scope of an index hint by adding a `FOR` clause to the hint. This provides more fine-grained control over optimizer selection of an execution plan for various phases of query processing. To affect only the indexes used when MySQL decides how to find rows in the table and how to process joins, use `FOR JOIN`. To influence index usage for sorting or grouping rows, use `FOR ORDER BY` or `FOR GROUP BY`.
- You can specify multiple index hints:

  ```
  SELECT * FROM t1 USE INDEX (i1) IGNORE INDEX FOR ORDER BY (i2) ORDER BY a;
  ```

  It is not an error to name the same index in several hints (even within the same hint):

  ```
  SELECT * FROM t1 USE INDEX (i1) USE INDEX (i1,i1);
  ```

  However, it is an error to mix `USE INDEX` and `FORCE INDEX` for the same table:

  ```
  SELECT * FROM t1 USE INDEX FOR JOIN (i1) FORCE INDEX FOR JOIN (i2);
  ```

If an index hint includes no FOR clause, the scope of the hint is to apply to all parts of the statement. For example, this hint:

```
IGNORE INDEX (i1)
```

is equivalent to this combination of hints:

```
IGNORE INDEX FOR JOIN (i1)
IGNORE INDEX FOR ORDER BY (i1)
IGNORE INDEX FOR GROUP BY (i1)
```

In MySQL 5.0, hint scope with no FOR clause was to apply only to row retrieval. To cause the server to use this older behavior when no FOR clause is present, enable the old system variable at server startup. Take care about enabling this variable in a replication setup. With statement-based binary logging, having different modes for the source and replicas might lead to replication errors.

When index hints are processed, they are collected in a single list by type (USE, FORCE, IGNORE) and by scope (FOR JOIN, FOR ORDER BY, FOR GROUP BY). For example:

```
SELECT * FROM t1
  USE INDEX () IGNORE INDEX (i2) USE INDEX (i1) USE INDEX (i2);
```

is equivalent to:

```
SELECT * FROM t1
   USE INDEX (i1,i2) IGNORE INDEX (i2);
```

The index hints then are applied for each scope in the following order:

1. {USE|FORCE} INDEX is applied if present. (If not, the optimizer-determined set of indexes is used.)
2. IGNORE INDEX is applied over the result of the previous step. For example, the following two queries are equivalent:

   ```
   SELECT * FROM t1 USE INDEX (i1) IGNORE INDEX (i2) USE INDEX (i2);

   SELECT * FROM t1 USE INDEX (i1);
   ```

For FULLTEXT searches, index hints work as follows:

- For natural language mode searches, index hints are silently ignored. For example, IGNORE INDEX(i1) is ignored with no warning and the index is still used.
- For boolean mode searches, index hints with FOR ORDER BY or FOR GROUP BY are silently ignored. Index hints with FOR JOIN or no FOR modifier are honored. In contrast to how hints apply for non-FULLTEXT searches, the hint is used for all phases of query execution (finding rows and retrieval, grouping, and ordering). This is true even if the hint is given for a non-FULLTEXT index.

  For example, the following two queries are equivalent:

  ```
  SELECT * FROM t
    USE INDEX (index1)
    IGNORE INDEX (index1) FOR ORDER BY
    IGNORE INDEX (index1) FOR GROUP BY
    WHERE ... IN BOOLEAN MODE ... ;

  SELECT * FROM t
    USE INDEX (index1)
    WHERE ... IN BOOLEAN MODE ... ;
  ```

8.10 Buffering and Caching

MySQL uses several strategies that cache information in memory buffers to increase performance.

8.10.1 InnoDB Buffer Pool Optimization

`InnoDB` maintains a storage area called the buffer pool for caching data and indexes in memory. Knowing how the `InnoDB` buffer pool works, and taking advantage of it to keep frequently accessed data in memory, is an important aspect of MySQL tuning.

For an explanation of the inner workings of the `InnoDB` buffer pool, an overview of its LRU replacement algorithm, and general configuration information, see Section 14.5.1, “Buffer Pool”.

For additional `InnoDB` buffer pool configuration and tuning information, see these sections:

- Section 14.8.3.3, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”
- Section 14.8.3.4, “Configuring Buffer Pool Flushing”
- Section 14.8.3.2, “Making the Buffer Pool Scan Resistant”
- Section 14.8.3.1, “Configuring Multiple Buffer Pool Instances”
- Section 14.8.3.5, “Saving and Restoring the Buffer Pool State”

8.10.2 The MyISAM Key Cache

To minimize disk I/O, the `MyISAM` storage engine exploits a strategy that is used by many database management systems. It employs a cache mechanism to keep the most frequently accessed table blocks in memory:

- For index blocks, a special structure called the *key cache* (or *key buffer*) is maintained. The structure contains a number of block buffers where the most-used index blocks are placed.
- For data blocks, MySQL uses no special cache. Instead it relies on the native operating system file system cache.

This section first describes the basic operation of the `MyISAM` key cache. Then it discusses features that improve key cache performance and that enable you to better control cache operation:

- Multiple sessions can access the cache concurrently.
- You can set up multiple key caches and assign table indexes to specific caches.

To control the size of the key cache, use the `key_buffer_size` system variable. If this variable is set equal to zero, no key cache is used. The key cache also is not used if the `key_buffer_size` value is too small to allocate the minimal number of block buffers (8).

When the key cache is not operational, index files are accessed using only the native file system buffering provided by the operating system. (In other words, table index blocks are accessed using the same strategy as that employed for table data blocks.)

An index block is a contiguous unit of access to the `MyISAM` index files. Usually the size of an index block is equal to the size of nodes of the index B-tree. (Indexes are represented on disk using a B-tree data structure. Nodes at the bottom of the tree are leaf nodes. Nodes above the leaf nodes are nonleaf nodes.)

All block buffers in a key cache structure are the same size. This size can be equal to, greater than, or less than the size of a table index block. Usually one these two values is a multiple of the other.

When data from any table index block must be accessed, the server first checks whether it is available in some block buffer of the key cache. If it is, the server accesses data in the key cache rather than on disk. That is, it reads from the cache or writes into it rather than reading from or writing to disk. Otherwise, the server chooses a cache block buffer containing a different table index block (or blocks)

and replaces the data there by a copy of required table index block. As soon as the new index block is in the cache, the index data can be accessed.

If it happens that a block selected for replacement has been modified, the block is considered “dirty.” In this case, prior to being replaced, its contents are flushed to the table index from which it came.

Usually the server follows an *LRU (Least Recently Used)* strategy: When choosing a block for replacement, it selects the least recently used index block. To make this choice easier, the key cache module maintains all used blocks in a special list (*LRU chain*) ordered by time of use. When a block is accessed, it is the most recently used and is placed at the end of the list. When blocks need to be replaced, blocks at the beginning of the list are the least recently used and become the first candidates for eviction.

The `InnoDB` storage engine also uses an LRU algorithm, to manage its buffer pool. See Section 14.5.1, “Buffer Pool”.

8.10.2.1 Shared Key Cache Access

Threads can access key cache buffers simultaneously, subject to the following conditions:

- A buffer that is not being updated can be accessed by multiple sessions.
- A buffer that is being updated causes sessions that need to use it to wait until the update is complete.
- Multiple sessions can initiate requests that result in cache block replacements, as long as they do not interfere with each other (that is, as long as they need different index blocks, and thus cause different cache blocks to be replaced).

Shared access to the key cache enables the server to improve throughput significantly.

8.10.2.2 Multiple Key Caches

Shared access to the key cache improves performance but does not eliminate contention among sessions entirely. They still compete for control structures that manage access to the key cache buffers. To reduce key cache access contention further, MySQL also provides multiple key caches. This feature enables you to assign different table indexes to different key caches.

Where there are multiple key caches, the server must know which cache to use when processing queries for a given `MyISAM` table. By default, all `MyISAM` table indexes are cached in the default key cache. To assign table indexes to a specific key cache, use the `CACHE INDEX` statement (see Section 13.7.6.2, “CACHE INDEX Statement”). For example, the following statement assigns indexes from the tables `t1`, `t2`, and `t3` to the key cache named `hot_cache`:

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
+---------+--------------------+----------+----------+
| Table   | Op                 | Msg_type | Msg_text |
+---------+--------------------+----------+----------+
| test.t1 | assign_to_keycache | status   | OK       |
| test.t2 | assign_to_keycache | status   | OK       |
| test.t3 | assign_to_keycache | status   | OK       |
+---------+--------------------+----------+----------+
```

The key cache referred to in a `CACHE INDEX` statement can be created by setting its size with a `SET GLOBAL` parameter setting statement or by using server startup options. For example:

```
mysql> SET GLOBAL keycache1.key_buffer_size=128*1024;
```

To destroy a key cache, set its size to zero:

```
mysql> SET GLOBAL keycache1.key_buffer_size=0;
```

You cannot destroy the default key cache. Any attempt to do this is ignored:

```
mysql> SET GLOBAL key_buffer_size = 0;

mysql> SHOW VARIABLES LIKE 'key_buffer_size';
+-----------------+---------+
| Variable_name   | Value   |
+-----------------+---------+
| key_buffer_size | 8384512 |
+-----------------+---------+
```

Key cache variables are structured system variables that have a name and components. For `keycache1.key_buffer_size`, `keycache1` is the cache variable name and `key_buffer_size` is the cache component. See Section 5.1.8.3, “Structured System Variables”, for a description of the syntax used for referring to structured key cache system variables.

By default, table indexes are assigned to the main (default) key cache created at the server startup. When a key cache is destroyed, all indexes assigned to it are reassigned to the default key cache.

For a busy server, you can use a strategy that involves three key caches:

- A “hot” key cache that takes up 20% of the space allocated for all key caches. Use this for tables that are heavily used for searches but that are not updated.
- A “cold” key cache that takes up 20% of the space allocated for all key caches. Use this cache for medium-sized, intensively modified tables, such as temporary tables.
- A “warm” key cache that takes up 60% of the key cache space. Employ this as the default key cache, to be used by default for all other tables.

One reason the use of three key caches is beneficial is that access to one key cache structure does not block access to the others. Statements that access tables assigned to one cache do not compete with statements that access tables assigned to another cache. Performance gains occur for other reasons as well:

- The hot cache is used only for retrieval queries, so its contents are never modified. Consequently, whenever an index block needs to be pulled in from disk, the contents of the cache block chosen for replacement need not be flushed first.
- For an index assigned to the hot cache, if there are no queries requiring an index scan, there is a high probability that the index blocks corresponding to nonleaf nodes of the index B-tree remain in the cache.
- An update operation most frequently executed for temporary tables is performed much faster when the updated node is in the cache and need not be read from disk first. If the size of the indexes of the temporary tables are comparable with the size of cold key cache, the probability is very high that the updated node is in the cache.

The `CACHE INDEX` statement sets up an association between a table and a key cache, but the association is lost each time the server restarts. If you want the association to take effect each time the server starts, one way to accomplish this is to use an option file: Include variable settings that configure your key caches, and an `init_file` system variable that names a file containing `CACHE INDEX` statements to be executed. For example:

```
key_buffer_size = 4G
hot_cache.key_buffer_size = 2G
cold_cache.key_buffer_size = 2G
init_file=/path/to/data-directory/mysqld_init.sql
```

The statements in `mysqld_init.sql` are executed each time the server starts. The file should contain one SQL statement per line. The following example assigns several tables each to `hot_cache` and `cold_cache`:

```
CACHE INDEX db1.t1, db1.t2, db2.t3 IN hot_cache
CACHE INDEX db1.t4, db2.t5, db2.t6 IN cold_cache
```

8.10.2.3 Midpoint Insertion Strategy

By default, the key cache management system uses a simple LRU strategy for choosing key cache blocks to be evicted, but it also supports a more sophisticated method called the *midpoint insertion strategy*.

When using the midpoint insertion strategy, the LRU chain is divided into two parts: a hot sublist and a warm sublist. The division point between two parts is not fixed, but the key cache management system takes care that the warm part is not "too short," always containing at least `key_cache_division_limit` percent of the key cache blocks. `key_cache_division_limit` is a component of structured key cache variables, so its value is a parameter that can be set per cache.

When an index block is read from a table into the key cache, it is placed at the end of the warm sublist. After a certain number of hits (accesses of the block), it is promoted to the hot sublist. At present, the number of hits required to promote a block (3) is the same for all index blocks.

A block promoted into the hot sublist is placed at the end of the list. The block then circulates within this sublist. If the block stays at the beginning of the sublist for a long enough time, it is demoted to the warm sublist. This time is determined by the value of the `key_cache_age_threshold` component of the key cache.

The threshold value prescribes that, for a key cache containing *`N`* blocks, the block at the beginning of the hot sublist not accessed within the last *`N`* `* key_cache_age_threshold / 100` hits is to be moved to the beginning of the warm sublist. It then becomes the first candidate for eviction, because blocks for replacement always are taken from the beginning of the warm sublist.

The midpoint insertion strategy enables you to keep more-valued blocks always in the cache. If you prefer to use the plain LRU strategy, leave the `key_cache_division_limit` value set to its default of 100.

The midpoint insertion strategy helps to improve performance when execution of a query that requires an index scan effectively pushes out of the cache all the index blocks corresponding to valuable high-level B-tree nodes. To avoid this, you must use a midpoint insertion strategy with the `key_cache_division_limit` set to much less than 100. Then valuable frequently hit nodes are preserved in the hot sublist during an index scan operation as well.

8.10.2.4 Index Preloading

If there are enough blocks in a key cache to hold blocks of an entire index, or at least the blocks corresponding to its nonleaf nodes, it makes sense to preload the key cache with index blocks before starting to use it. Preloading enables you to put the table index blocks into a key cache buffer in the most efficient way: by reading the index blocks from disk sequentially.

Without preloading, the blocks are still placed into the key cache as needed by queries. Although the blocks stay in the cache, because there are enough buffers for all of them, they are fetched from disk in random order, and not sequentially.

To preload an index into a cache, use the `LOAD INDEX INTO CACHE` statement. For example, the following statement preloads nodes (index blocks) of indexes of the tables `t1` and `t2`:

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
+---------+--------------+----------+----------+
| Table   | Op           | Msg_type | Msg_text |
+---------+--------------+----------+----------+
| test.t1 | preload_keys | status   | OK       |
| test.t2 | preload_keys | status   | OK       |
+---------+--------------+----------+----------+
```

The `IGNORE LEAVES` modifier causes only blocks for the nonleaf nodes of the index to be preloaded. Thus, the statement shown preloads all index blocks from `t1`, but only blocks for the nonleaf nodes from `t2`.

If an index has been assigned to a key cache using a `CACHE INDEX` statement, preloading places index blocks into that cache. Otherwise, the index is loaded into the default key cache.

8.10.2.5 Key Cache Block Size

It is possible to specify the size of the block buffers for an individual key cache using the `key_cache_block_size` variable. This permits tuning of the performance of I/O operations for index files.

The best performance for I/O operations is achieved when the size of read buffers is equal to the size of the native operating system I/O buffers. But setting the size of key nodes equal to the size of the I/O buffer does not always ensure the best overall performance. When reading the big leaf nodes, the server pulls in a lot of unnecessary data, effectively preventing reading other leaf nodes.

To control the size of blocks in the `.MYI` index file of `MyISAM` tables, use the `--myisam-block-size` option at server startup.

8.10.2.6 Restructuring a Key Cache

A key cache can be restructured at any time by updating its parameter values. For example:

```
mysql> SET GLOBAL cold_cache.key_buffer_size=4*1024*1024;
```

If you assign to either the `key_buffer_size` or `key_cache_block_size` key cache component a value that differs from the component's current value, the server destroys the cache's old structure and creates a new one based on the new values. If the cache contains any dirty blocks, the server saves them to disk before destroying and re-creating the cache. Restructuring does not occur if you change other key cache parameters.

When restructuring a key cache, the server first flushes the contents of any dirty buffers to disk. After that, the cache contents become unavailable. However, restructuring does not block queries that need to use indexes assigned to the cache. Instead, the server directly accesses the table indexes using native file system caching. File system caching is not as efficient as using a key cache, so although queries execute, a slowdown can be anticipated. After the cache has been restructured, it becomes available again for caching indexes assigned to it, and the use of file system caching for the indexes ceases.

8.10.3 The MySQL Query Cache

The query cache stores the text of a `SELECT` statement together with the corresponding result that was sent to the client. If an identical statement is received later, the server retrieves the results from the query cache rather than parsing and executing the statement again. The query cache is shared among sessions, so a result set generated by one client can be sent in response to the same query issued by another client.

The query cache can be useful in an environment where you have tables that do not change very often and for which the server receives many identical queries. This is a typical situation for many Web servers that generate many dynamic pages based on database content.

The query cache does not return stale data. When tables are modified, any relevant entries in the query cache are flushed.

Note

The query cache does not work in an environment where you have multiple `mysqld` servers updating the same `MyISAM` tables.

The query cache is used for prepared statements under the conditions described in Section 8.10.3.1, “How the Query Cache Operates”.

Note

The query cache is not supported for partitioned tables, and is automatically disabled for queries involving partitioned tables. The query cache cannot be enabled for such queries.

Some performance data for the query cache follows. These results were generated by running the MySQL benchmark suite on a Linux Alpha 2×500MHz system with 2GB RAM and a 64MB query cache.

- If all the queries you are performing are simple (such as selecting a row from a table with one row), but still differ so that the queries cannot be cached, the overhead for having the query cache active is 13%. This could be regarded as the worst case scenario. In real life, queries tend to be much more complicated, so the overhead normally is significantly lower.
- Searches for a single row in a single-row table are 238% faster with the query cache than without it. This can be regarded as close to the minimum speedup to be expected for a query that is cached.

To disable the query cache at server startup, set the `query_cache_size` system variable to 0. By disabling the query cache code, there is no noticeable overhead.

The query cache offers the potential for substantial performance improvement, but do not assume that it does so under all circumstances. With some query cache configurations or server workloads, you might actually see a performance decrease:

- Be cautious about sizing the query cache excessively large, which increases the overhead required to maintain the cache, possibly beyond the benefit of enabling it. Sizes in tens of megabytes are usually beneficial. Sizes in the hundreds of megabytes might not be.
- Server workload has a significant effect on query cache efficiency. A query mix consisting almost entirely of a fixed set of `SELECT` statements is much more likely to benefit from enabling the cache than a mix in which frequent `INSERT` statements cause continual invalidation of results in the cache. In some cases, a workaround is to use the `SQL_NO_CACHE` option to prevent results from even entering the cache for `SELECT` statements that use frequently modified tables. (See Section 8.10.3.2, “Query Cache SELECT Options”.)

To verify that enabling the query cache is beneficial, test the operation of your MySQL server with the cache enabled and disabled. Then retest periodically because query cache efficiency may change as server workload changes.

8.10.3.1 How the Query Cache Operates

This section describes how the query cache works when it is operational. Section 8.10.3.3, “Query Cache Configuration”, describes how to control whether it is operational.

Incoming queries are compared to those in the query cache before parsing, so the following two queries are regarded as different by the query cache:

```
SELECT * FROM tbl_name
Select * from tbl_name
```

Queries must be *exactly* the same (byte for byte) to be seen as identical. In addition, query strings that are identical may be treated as different for other reasons. Queries that use different databases, different protocol versions, or different default character sets are considered different queries and are cached separately.

The cache is not used for queries of the following types:

- Queries that are a subquery of an outer query
- Queries executed within the body of a stored function, trigger, or event

Before a query result is fetched from the query cache, MySQL checks whether the user has `SELECT` privilege for all databases and tables involved. If this is not the case, the cached result is not used.

If a query result is returned from query cache, the server increments the `Qcache_hits` status variable, not `Com_select`. See Section 8.10.3.4, “Query Cache Status and Maintenance”.

If a table changes, all cached queries that use the table become invalid and are removed from the cache. This includes queries that use `MERGE` tables that map to the changed table. A table can be changed by many types of statements, such as `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE TABLE`, `ALTER TABLE`, `DROP TABLE`, or `DROP DATABASE`.

The query cache also works within transactions when using `InnoDB` tables.

The result from a `SELECT` query on a view is cached.

The query cache works for `SELECT SQL_CALC_FOUND_ROWS ...` queries and stores a value that is returned by a following `SELECT FOUND_ROWS()` query. `FOUND_ROWS()` returns the correct value even if the preceding query was fetched from the cache because the number of found rows is also stored in the cache. The `SELECT FOUND_ROWS()` query itself cannot be cached.

Prepared statements that are issued using the binary protocol using `mysql_stmt_prepare()` and `mysql_stmt_execute()` (see C API Prepared Statements), are subject to limitations on caching. Comparison with statements in the query cache is based on the text of the statement after expansion of `?` parameter markers. The statement is compared only with other cached statements that were executed using the binary protocol. That is, for query cache purposes, prepared statements issued using the binary protocol are distinct from prepared statements issued using the text protocol (see Section 13.5, “Prepared Statements”).

A query cannot be cached if it uses any of the following functions:

- `BENCHMARK()`
- `CONNECTION_ID()`
- `CONVERT_TZ()`
- `CURDATE()`
- `CURRENT_DATE()`
- `CURRENT_TIME()`
- `CURRENT_TIMESTAMP()`
- `CURRENT_USER()`
- `CURTIME()`
- `DATABASE()`
- `ENCRYPT()` with one parameter
- `FOUND_ROWS()`
- `GET_LOCK()`
- `IS_FREE_LOCK()`
- `IS_USED_LOCK()`
- `LAST_INSERT_ID()`

- `LOAD_FILE()`
- `MASTER_POS_WAIT()`
- `NOW()`
- `PASSWORD()`
- `RAND()`
- `RANDOM_BYTES()`
- `RELEASE_ALL_LOCKS()`
- `RELEASE_LOCK()`
- `SLEEP()`
- `SYSDATE()`
- `UNIX_TIMESTAMP()` with no parameters
- `USER()`
- `UUID()`
- `UUID_SHORT()`

A query also is not cached under these conditions:

- It refers to user-defined functions (UDFs) or stored functions.
- It refers to user variables or local stored program variables.
- It refers to tables in the `mysql`, `INFORMATION_SCHEMA`, or `performance_schema` database.
- It refers to any partitioned tables.
- It is of any of the following forms:

```
SELECT ... LOCK IN SHARE MODE
SELECT ... FOR UPDATE
SELECT ... INTO OUTFILE ...
SELECT ... INTO DUMPFILE ...
SELECT * FROM ... WHERE autoincrement_col IS NULL
```

  The last form is not cached because it is used as the ODBC workaround for obtaining the last insert ID value. See the Connector/ODBC section of Chapter 23, *Connectors and APIs*.

  Statements within transactions that use `SERIALIZABLE` isolation level also cannot be cached because they use `LOCK IN SHARE MODE` locking.

- It uses `TEMPORARY` tables.
- It does not use any tables.
- It generates warnings.
- The user has a column-level privilege for any of the involved tables.

8.10.3.2 Query Cache SELECT Options

Two query cache-related options may be specified in `SELECT` statements:

- `SQL_CACHE`

The query result is cached if it is cacheable and the value of the `query_cache_type` system variable is `ON` or `DEMAND`.

- `SQL_NO_CACHE`

  The server does not use the query cache. It neither checks the query cache to see whether the result is already cached, nor does it cache the query result. (Due to a limitation in the parser, a space character must precede and follow the `SQL_NO_CACHE` keyword; a nonspace such as a newline causes the server to check the query cache to see whether the result is already cached.)

Examples:

```
SELECT SQL_CACHE id, name FROM customer;
SELECT SQL_NO_CACHE id, name FROM customer;
```

8.10.3.3 Query Cache Configuration

The `have_query_cache` server system variable indicates whether the query cache is available:

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| have_query_cache | YES   |
+------------------+-------+
```

When using a standard MySQL binary, this value is always `YES`, even if query caching is disabled.

Several other system variables control query cache operation. These can be set in an option file or on the command line when starting `mysqld`. The query cache system variables all have names that begin with `query_cache_`. They are described briefly in Section 5.1.7, “Server System Variables”, with additional configuration information given here.

To set the size of the query cache, set the `query_cache_size` system variable. Setting it to 0 disables the query cache, as does setting `query_cache_type=0`. By default, the query cache is disabled. This is achieved using a default size of 1M, with a default for `query_cache_type` of 0.

To reduce overhead significantly, start the server with `query_cache_type=0` if you do not intend to use the query cache.

Note

When using the Windows Configuration Wizard to install or configure MySQL, the default value for `query_cache_size` is configured automatically for you based on the different configuration types available. When using the Windows Configuration Wizard, the query cache may be enabled (that is, set to a nonzero value) due to the selected configuration. The query cache is also controlled by the setting of the `query_cache_type` variable. Check the values of these variables as set in your `my.ini` file after configuration has taken place.

When you set `query_cache_size` to a nonzero value, keep in mind that the query cache needs a minimum size of about 40KB to allocate its structures. (The exact size depends on system architecture.) If you set the value too small, you'll get a warning, as in this example:

```
mysql> SET GLOBAL query_cache_size = 40000;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1282
Message: Query cache failed to set size 39936;
```

```
         new query cache size is 0

mysql> SET GLOBAL query_cache_size = 41984;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'query_cache_size';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| query_cache_size | 41984 |
+------------------+-------+
```

For the query cache to actually be able to hold any query results, its size must be set larger:

```
mysql> SET GLOBAL query_cache_size = 1000000;
Query OK, 0 rows affected (0.04 sec)

mysql> SHOW VARIABLES LIKE 'query_cache_size';
+------------------+--------+
| Variable_name    | Value  |
+------------------+--------+
| query_cache_size | 999424 |
+------------------+--------+
1 row in set (0.00 sec)
```

The `query_cache_size` value is aligned to the nearest 1024 byte block. The value reported may therefore be different from the value that you assign.

If the query cache size is greater than 0, the `query_cache_type` variable influences how it works. This variable can be set to the following values:

- A value of `0` or `OFF` prevents caching or retrieval of cached results.
- A value of `1` or `ON` enables caching except of those statements that begin with `SELECT SQL_NO_CACHE`.
- A value of `2` or `DEMAND` causes caching of only those statements that begin with `SELECT SQL_CACHE`.

If `query_cache_size` is 0, you should also set `query_cache_type` variable to 0. In this case, the server does not acquire the query cache mutex at all, which means that the query cache cannot be enabled at runtime and there is reduced overhead in query execution.

Setting the `GLOBAL query_cache_type` value determines query cache behavior for all clients that connect after the change is made. Individual clients can control cache behavior for their own connection by setting the `SESSION query_cache_type` value. For example, a client can disable use of the query cache for its own queries like this:

```
mysql> SET SESSION query_cache_type = OFF;
```

If you set `query_cache_type` at server startup (rather than at runtime with a `SET` statement), only the numeric values are permitted.

To control the maximum size of individual query results that can be cached, set the `query_cache_limit` system variable. The default value is 1MB.

Be careful not to set the size of the cache too large. Due to the need for threads to lock the cache during updates, you may see lock contention issues with a very large cache.

Note

You can set the maximum size that can be specified for the query cache at runtime with the `SET` statement by using the `--maximum-query_cache_size=32M` option on the command line or in the configuration file.

When a query is to be cached, its result (the data sent to the client) is stored in the query cache during result retrieval. Therefore the data usually is not handled in one big chunk. The query cache allocates blocks for storing this data on demand, so when one block is filled, a new block is allocated. Because memory allocation operation is costly (timewise), the query cache allocates blocks with a minimum size given by the `query_cache_min_res_unit` system variable. When a query is executed, the last result block is trimmed to the actual data size so that unused memory is freed. Depending on the types of queries your server executes, you might find it helpful to tune the value of `query_cache_min_res_unit`:

- The default value of `query_cache_min_res_unit` is 4KB. This should be adequate for most cases.
- If you have a lot of queries with small results, the default block size may lead to memory fragmentation, as indicated by a large number of free blocks. Fragmentation can force the query cache to prune (delete) queries from the cache due to lack of memory. In this case, decrease the value of `query_cache_min_res_unit`. The number of free blocks and queries removed due to pruning are given by the values of the `Qcache_free_blocks` and `Qcache_lowmem_prunes` status variables.
- If most of your queries have large results (check the `Qcache_total_blocks` and `Qcache_queries_in_cache` status variables), you can increase performance by increasing `query_cache_min_res_unit`. However, be careful to not make it too large (see the previous item).

8.10.3.4 Query Cache Status and Maintenance

To check whether the query cache is present in your MySQL server, use the following statement:

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| have_query_cache | YES   |
+------------------+-------+
```

You can defragment the query cache to better utilize its memory with the `FLUSH QUERY CACHE` statement. The statement does not remove any queries from the cache.

The `RESET QUERY CACHE` statement removes all query results from the query cache. The `FLUSH TABLES` statement also does this.

To monitor query cache performance, use `SHOW STATUS` to view the cache status variables:

```
mysql> SHOW STATUS LIKE 'Qcache%';
+-------------------------+--------+
| Variable_name           | Value  |
+-------------------------+--------+
| Qcache_free_blocks      | 36     |
| Qcache_free_memory      | 138488 |
| Qcache_hits             | 79570  |
| Qcache_inserts          | 27087  |
| Qcache_lowmem_prunes    | 3114   |
| Qcache_not_cached       | 22989  |
| Qcache_queries_in_cache | 415    |
| Qcache_total_blocks     | 912    |
+-------------------------+--------+
```

Descriptions of each of these variables are given in Section 5.1.9, “Server Status Variables”. Some uses for them are described here.

The total number of `SELECT` queries is given by this formula:

```
  Com_select
+ Qcache_hits
```

```
+ queries with errors found by parser
```

The `Com_select` value is given by this formula:

```
  Qcache_inserts
+ Qcache_not_cached
+ queries with errors found during the column-privileges check
```

The query cache uses variable-length blocks, so `Qcache_total_blocks` and `Qcache_free_blocks` may indicate query cache memory fragmentation. After `FLUSH QUERY CACHE`, only a single free block remains.

Every cached query requires a minimum of two blocks (one for the query text and one or more for the query results). Also, every table that is used by a query requires one block. However, if two or more queries use the same table, only one table block needs to be allocated.

The information provided by the `Qcache_lowmem_prunes` status variable can help you tune the query cache size. It counts the number of queries that have been removed from the cache to free up memory for caching new queries. The query cache uses a least recently used (LRU) strategy to decide which queries to remove from the cache. Tuning information is given in Section 8.10.3.3, “Query Cache Configuration”.

8.10.4 Caching of Prepared Statements and Stored Programs

For certain statements that a client might execute multiple times during a session, the server converts the statement to an internal structure and caches that structure to be used during execution. Caching enables the server to perform more efficiently because it avoids the overhead of reconverting the statement should it be needed again during the session. Conversion and caching occurs for these statements:

- Prepared statements, both those processed at the SQL level (using the `PREPARE` statement) and those processed using the binary client/server protocol (using the `mysql_stmt_prepare()` C API function). The `max_prepared_stmt_count` system variable controls the total number of statements the server caches. (The sum of the number of prepared statements across all sessions.)
- Stored programs (stored procedures and functions, triggers, and events). In this case, the server converts and caches the entire program body. The `stored_program_cache` system variable indicates the approximate number of stored programs the server caches per session.

The server maintains caches for prepared statements and stored programs on a per-session basis. Statements cached for one session are not accessible to other sessions. When a session ends, the server discards any statements cached for it.

When the server uses a cached internal statement structure, it must take care that the structure does not go out of date. Metadata changes can occur for an object used by the statement, causing a mismatch between the current object definition and the definition as represented in the internal statement structure. Metadata changes occur for DDL statements such as those that create, drop, alter, rename, or truncate tables, or that analyze, optimize, or repair tables. Table content changes (for example, with `INSERT` or `UPDATE`) do not change metadata, nor do `SELECT` statements.

Here is an illustration of the problem. Suppose that a client prepares this statement:

```
PREPARE s1 FROM 'SELECT * FROM t1';
```

The `SELECT *` expands in the internal structure to the list of columns in the table. If the set of columns in the table is modified with `ALTER TABLE`, the prepared statement goes out of date. If the server does not detect this change the next time the client executes `s1`, the prepared statement returns incorrect results.

To avoid problems caused by metadata changes to tables or views referred to by the prepared statement, the server detects these changes and automatically reprepares the statement when it is

next executed. That is, the server reparses the statement and rebuilds the internal structure. Reparsing also occurs after referenced tables or views are flushed from the table definition cache, either implicitly to make room for new entries in the cache, or explicitly due to `FLUSH TABLES`.

Similarly, if changes occur to objects used by a stored program, the server reparses affected statements within the program.

The server also detects metadata changes for objects in expressions. These might be used in statements specific to stored programs, such as `DECLARE CURSOR` or flow-control statements such as `IF`, `CASE`, and `RETURN`.

To avoid reparsing entire stored programs, the server reparses affected statements or expressions within a program only as needed. Examples:

- Suppose that metadata for a table or view is changed. Reparsing occurs for a `SELECT *` within the program that accesses the table or view, but not for a `SELECT *` that does not access the table or view.
- When a statement is affected, the server reparses it only partially if possible. Consider this `CASE` statement:

  ```
  CASE case_expr
    WHEN when_expr1 ...
    WHEN when_expr2 ...
    WHEN when_expr3 ...
    ...
  END CASE
  ```

  If a metadata change affects only `WHEN when_expr3`, that expression is reparsed. *`case_expr`* and the other `WHEN` expressions are not reparsed.

Reparsing uses the default database and SQL mode that were in effect for the original conversion to internal form.

The server attempts reparsing up to three times. An error occurs if all attempts fail.

Reparsing is automatic, but to the extent that it occurs, diminishes prepared statement and stored program performance.

For prepared statements, the `Com_stmt_reprepare` status variable tracks the number of repreparations.

8.11 Optimizing Locking Operations

MySQL manages contention for table contents using locking:

- Internal locking is performed within the MySQL server itself to manage contention for table contents by multiple threads. This type of locking is internal because it is performed entirely by the server and involves no other programs. See Section 8.11.1, “Internal Locking Methods”.
- External locking occurs when the server and other programs lock `MyISAM` table files to coordinate among themselves which program can access the tables at which time. See Section 8.11.5, “External Locking”.

8.11.1 Internal Locking Methods

This section discusses internal locking; that is, locking performed within the MySQL server itself to manage contention for table contents by multiple sessions. This type of locking is internal because it is performed entirely by the server and involves no other programs. For locking performed on MySQL files by other programs, see Section 8.11.5, “External Locking”.

- Row-Level Locking

- Table-Level Locking
- Choosing the Type of Locking

Row-Level Locking

MySQL uses row-level locking for `InnoDB` tables to support simultaneous write access by multiple sessions, making them suitable for multi-user, highly concurrent, and OLTP applications.

To avoid deadlocks when performing multiple concurrent write operations on a single `InnoDB` table, acquire necessary locks at the start of the transaction by issuing a `SELECT ... FOR UPDATE` statement for each group of rows expected to be modified, even if the data change statements come later in the transaction. If transactions modify or lock more than one table, issue the applicable statements in the same order within each transaction. Deadlocks affect performance rather than representing a serious error, because `InnoDB` automatically detects deadlock conditions and rolls back one of the affected transactions.

Advantages of row-level locking:

- Fewer lock conflicts when different sessions access different rows.
- Fewer changes for rollbacks.
- Possible to lock a single row for a long time.

Table-Level Locking

MySQL uses table-level locking for `MyISAM`, `MEMORY`, and `MERGE` tables, permitting only one session to update those tables at a time. This locking level makes these storage engines more suitable for read-only, read-mostly, or single-user applications.

These storage engines avoid deadlocks by always requesting all needed locks at once at the beginning of a query and always locking the tables in the same order. The tradeoff is that this strategy reduces concurrency; other sessions that want to modify the table must wait until the current data change statement finishes.

Advantages of table-level locking:

- Relatively little memory required (row locking requires memory per row or group of rows locked)
- Fast when used on a large part of the table because only a single lock is involved.
- Fast if you often do `GROUP BY` operations on a large part of the data or must scan the entire table frequently.

MySQL grants table write locks as follows:

1. If there are no locks on the table, put a write lock on it.
2. Otherwise, put the lock request in the write lock queue.

MySQL grants table read locks as follows:

1. If there are no write locks on the table, put a read lock on it.
2. Otherwise, put the lock request in the read lock queue.

Table updates are given higher priority than table retrievals. Therefore, when a lock is released, the lock is made available to the requests in the write lock queue and then to the requests in the read lock queue. This ensures that updates to a table are not “starved” even when there is heavy `SELECT` activity for the table. However, if there are many updates for a table, `SELECT` statements wait until there are no more updates.

For information on altering the priority of reads and writes, see Section 8.11.2, “Table Locking Issues”.

You can analyze the table lock contention on your system by checking the `Table_locks_immediate` and `Table_locks_waited` status variables, which indicate the number of times that requests for table locks could be granted immediately and the number that had to wait, respectively:

```
mysql> SHOW STATUS LIKE 'Table%';
+-----------------------+---------+
| Variable_name         | Value   |
+-----------------------+---------+
| Table_locks_immediate | 1151552 |
| Table_locks_waited    | 15324   |
+-----------------------+---------+
```

The `MyISAM` storage engine supports concurrent inserts to reduce contention between readers and writers for a given table: If a `MyISAM` table has no free blocks in the middle of the data file, rows are always inserted at the end of the data file. In this case, you can freely mix concurrent `INSERT` and `SELECT` statements for a `MyISAM` table without locks. That is, you can insert rows into a `MyISAM` table at the same time other clients are reading from it. Holes can result from rows having been deleted from or updated in the middle of the table. If there are holes, concurrent inserts are disabled but are enabled again automatically when all holes have been filled with new data. To control this behavior, use the `concurrent_insert` system variable. See Section 8.11.3, “Concurrent Inserts”.

If you acquire a table lock explicitly with `LOCK TABLES`, you can request a `READ LOCAL` lock rather than a `READ` lock to enable other sessions to perform concurrent inserts while you have the table locked.

To perform many `INSERT` and `SELECT` operations on a table `t1` when concurrent inserts are not possible, you can insert rows into a temporary table `temp_t1` and update the real table with the rows from the temporary table:

```
mysql> LOCK TABLES t1 WRITE, temp_t1 WRITE;
mysql> INSERT INTO t1 SELECT * FROM temp_t1;
mysql> DELETE FROM temp_t1;
mysql> UNLOCK TABLES;
```

Choosing the Type of Locking

Generally, table locks are superior to row-level locks in the following cases:

- Most statements for the table are reads.
- Statements for the table are a mix of reads and writes, where writes are updates or deletes for a single row that can be fetched with one key read:

  ```
  UPDATE tbl_name SET column=value WHERE unique_key_col=key_value;
  DELETE FROM tbl_name WHERE unique_key_col=key_value;
  ```

- `SELECT` combined with concurrent `INSERT` statements, and very few `UPDATE` or `DELETE` statements.
- Many scans or `GROUP BY` operations on the entire table without any writers.

With higher-level locks, you can more easily tune applications by supporting locks of different types, because the lock overhead is less than for row-level locks.

Options other than row-level locking:

- Versioning (such as that used in MySQL for concurrent inserts) where it is possible to have one writer at the same time as many readers. This means that the database or table supports different views for the data depending on when access begins. Other common terms for this are “time travel,” “copy on write,” or “copy on demand.”

- Copy on demand is in many cases superior to row-level locking. However, in the worst case, it can use much more memory than using normal locks.
- Instead of using row-level locks, you can employ application-level locks, such as those provided by `GET_LOCK()` and `RELEASE_LOCK()` in MySQL. These are advisory locks, so they work only with applications that cooperate with each other. See Section 12.15, “Locking Functions”.

8.11.2 Table Locking Issues

`InnoDB` tables use row-level locking so that multiple sessions and applications can read from and write to the same table simultaneously, without making each other wait or producing inconsistent results. For this storage engine, avoid using the `LOCK TABLES` statement, because it does not offer any extra protection, but instead reduces concurrency. The automatic row-level locking makes these tables suitable for your busiest databases with your most important data, while also simplifying application logic since you do not need to lock and unlock tables. Consequently, the `InnoDB` storage engine is the default in MySQL.

MySQL uses table locking (instead of page, row, or column locking) for all storage engines except `InnoDB`. The locking operations themselves do not have much overhead. But because only one session can write to a table at any one time, for best performance with these other storage engines, use them primarily for tables that are queried often and rarely inserted into or updated.

- Performance Considerations Favoring InnoDB
- Workarounds for Locking Performance Issues

Performance Considerations Favoring InnoDB

When choosing whether to create a table using `InnoDB` or a different storage engine, keep in mind the following disadvantages of table locking:

- Table locking enables many sessions to read from a table at the same time, but if a session wants to write to a table, it must first get exclusive access, meaning it might have to wait for other sessions to finish with the table first. During the update, all other sessions that want to access this particular table must wait until the update is done.
- Table locking causes problems when a session is waiting because the disk is full and free space needs to become available before the session can proceed. In this case, all sessions that want to access the problem table are also put in a waiting state until more disk space is made available.
- A `SELECT` statement that takes a long time to run prevents other sessions from updating the table in the meantime, making the other sessions appear slow or unresponsive. While a session is waiting to get exclusive access to the table for updates, other sessions that issue `SELECT` statements queue up behind it, reducing concurrency even for read-only sessions.

Workarounds for Locking Performance Issues

The following items describe some ways to avoid or reduce contention caused by table locking:

- Consider switching the table to the `InnoDB` storage engine, either using `CREATE TABLE ... ENGINE=INNODB` during setup, or using `ALTER TABLE ... ENGINE=INNODB` for an existing table. See Chapter 14, *The InnoDB Storage Engine* for more details about this storage engine.
- Optimize `SELECT` statements to run faster so that they lock tables for a shorter time. You might have to create some summary tables to do this.
- Start `mysqld` with `--low-priority-updates`. For storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`), this gives all statements that update (modify) a table lower priority than `SELECT` statements. In this case, the second `SELECT` statement in the preceding scenario would execute before the `UPDATE` statement, and would not wait for the first `SELECT` to finish.

- To specify that all updates issued in a specific connection should be done with low priority, set the `low_priority_updates` server system variable equal to 1.
- To give a specific `INSERT`, `UPDATE`, or `DELETE` statement lower priority, use the `LOW_PRIORITY` attribute.
- To give a specific `SELECT` statement higher priority, use the `HIGH_PRIORITY` attribute. See Section 13.2.9, “SELECT Statement”.
- Start `mysqld` with a low value for the `max_write_lock_count` system variable to force MySQL to temporarily elevate the priority of all `SELECT` statements that are waiting for a table after a specific number of write locks to the table occur (for example, for insert operations). This permits read locks after a certain number of write locks.
- If you mix inserts and deletes on the same nontransactional table, `INSERT DELAYED` may help. See Section 13.2.5.3, “INSERT DELAYED Statement”.

Note

`INSERT DELAYED` is deprecated; expect it to be removed in a future release. Use `INSERT` (without `DELAYED`) instead.

- If you have problems with mixed `SELECT` and `DELETE` statements, the `LIMIT` option to `DELETE` may help. See Section 13.2.2, “DELETE Statement”.
- Using `SQL_BUFFER_RESULT` with `SELECT` statements can help to make the duration of table locks shorter. See Section 13.2.9, “SELECT Statement”.
- Splitting table contents into separate tables may help, by allowing queries to run against columns in one table, while updates are confined to columns in a different table.
- You could change the locking code in `mysys/thr_lock.c` to use a single queue. In this case, write locks and read locks would have the same priority, which might help some applications.

8.11.3 Concurrent Inserts

The `MyISAM` storage engine supports concurrent inserts to reduce contention between readers and writers for a given table: If a `MyISAM` table has no holes in the data file (deleted rows in the middle), an `INSERT` statement can be executed to add rows to the end of the table at the same time that `SELECT` statements are reading rows from the table. If there are multiple `INSERT` statements, they are queued and performed in sequence, concurrently with the `SELECT` statements. The results of a concurrent `INSERT` may not be visible immediately.

The `concurrent_insert` system variable can be set to modify the concurrent-insert processing. By default, the variable is set to `AUTO` (or 1) and concurrent inserts are handled as just described. If `concurrent_insert` is set to `NEVER` (or 0), concurrent inserts are disabled. If the variable is set to `ALWAYS` (or 2), concurrent inserts at the end of the table are permitted even for tables that have deleted rows. See also the description of the `concurrent_insert` system variable.

Under circumstances where concurrent inserts can be used, there is seldom any need to use the `DELAYED` modifier for `INSERT` statements. See Section 13.2.5.3, “INSERT DELAYED Statement”.

If you are using the binary log, concurrent inserts are converted to normal inserts for `CREATE ... SELECT` or `INSERT ... SELECT` statements. This is done to ensure that you can re-create an exact copy of your tables by applying the log during a backup operation. See Section 5.4.4, “The Binary Log”. In addition, for those statements a read lock is placed on the selected-from table such that inserts into that table are blocked. The effect is that concurrent inserts for that table must wait as well.

With `LOAD DATA`, if you specify `CONCURRENT` with a `MyISAM` table that satisfies the condition for concurrent inserts (that is, it contains no free blocks in the middle), other sessions can retrieve data

from the table while `LOAD DATA` is executing. Use of the `CONCURRENT` option affects the performance of `LOAD DATA` a bit, even if no other session is using the table at the same time.

If you specify `HIGH_PRIORITY`, it overrides the effect of the `--low-priority-updates` option if the server was started with that option. It also causes concurrent inserts not to be used.

For `LOCK TABLE`, the difference between `READ LOCAL` and `READ` is that `READ LOCAL` permits nonconflicting `INSERT` statements (concurrent inserts) to execute while the lock is held. However, this cannot be used if you are going to manipulate the database using processes external to the server while you hold the lock.

8.11.4 Metadata Locking

MySQL uses metadata locking to manage concurrent access to database objects and to ensure data consistency. Metadata locking applies not just to tables, but also to schemas and stored programs (procedures, functions, triggers, and scheduled events).

Metadata locking does involve some overhead, which increases as query volume increases. Metadata contention increases the more that multiple queries attempt to access the same objects.

Metadata locking is not a replacement for the table definition cache, and its mutexes and locks differ from the `LOCK_open` mutex. The following discussion provides some information about how metadata locking works.

- Metadata Lock Acquisition
- Metadata Lock Release

Metadata Lock Acquisition

If there are multiple waiters for a given lock, the highest-priority lock request is satisfied first, with an exception related to the `max_write_lock_count` system variable. Write lock requests have higher priority than read lock requests. However, if `max_write_lock_count` is set to some low value (say, 10), read lock requests may be preferred over pending write lock requests if the read lock requests have already been passed over in favor of 10 write lock requests. Normally this behavior does not occur because `max_write_lock_count` by default has a very large value.

Statements acquire metadata locks one by one, not simultaneously, and perform deadlock detection in the process.

DML statements normally acquire locks in the order in which tables are mentioned in the statement.

DDL statements, `LOCK TABLES`, and other similar statements try to reduce the number of possible deadlocks between concurrent DDL statements by acquiring locks on explicitly named tables in name order. Locks might be acquired in a different order for implicitly used tables (such as tables in foreign key relationships that also must be locked).

For example, `RENAME TABLE` is a DDL statement that acquires locks in name order:

- This `RENAME TABLE` statement renames `tbla` to something else, and renames `tblc` to `tbla`:

  ```
  RENAME TABLE tbla TO tbld, tblc TO tbla;
  ```

  The statement acquires metadata locks, in order, on `tbla`, `tblc`, and `tbld` (because `tbld` follows `tblc` in name order):

- This slightly different statement also renames `tbla` to something else, and renames `tblc` to `tbla`:

  ```
  RENAME TABLE tbla TO tblb, tblc TO tbla;
  ```

  In this case, the statement acquires metadata locks, in order, on `tbla`, `tblb`, and `tblc` (because `tblb` precedes `tblc` in name order):

Both statements acquire locks on `tbla` and `tblc`, in that order, but differ in whether the lock on the remaining table name is acquired before or after `tblc`.

Metadata lock acquisition order can make a difference in operation outcome when multiple transactions execute concurrently, as the following example illustrates.

Begin with two tables `x` and `x_new` that have identical structure. Three clients issue statements that involve these tables:

Client 1:

```
LOCK TABLE x WRITE, x_new WRITE;
```

The statement requests and acquires write locks in name order on `x` and `x_new`.

Client 2:

```
INSERT INTO x VALUES(1);
```

The statement requests and blocks waiting for a write lock on `x`.

Client 3:

```
RENAME TABLE x TO x_old, x_new TO x;
```

The statement requests exclusive locks in name order on `x`, `x_new`, and `x_old`, but blocks waiting for the lock on `x`.

Client 1:

```
UNLOCK TABLES;
```

The statement releases the write locks on `x` and `x_new`. The exclusive lock request for `x` by Client 3 has higher priority than the write lock request by Client 2, so Client 3 acquires its lock on `x`, then also on `x_new` and `x_old`, performs the renaming, and releases its locks. Client 2 then acquires its lock on `x`, performs the insert, and releases its lock.

Lock acquisition order results in the `RENAME TABLE` executing before the `INSERT`. The `x` into which the insert occurs is the table that was named `x_new` when Client 2 issued the insert and was renamed to `x` by Client 3:

```
mysql> SELECT * FROM x;
+------+
| i    |
+------+
|    1 |
+------+

mysql> SELECT * FROM x_old;
Empty set (0.01 sec)
```

Now begin instead with tables named `x` and `new_x` that have identical structure. Again, three clients issue statements that involve these tables:

Client 1:

```
LOCK TABLE x WRITE, new_x WRITE;
```

The statement requests and acquires write locks in name order on `new_x` and `x`.

Client 2:

```
INSERT INTO x VALUES(1);
```

The statement requests and blocks waiting for a write lock on `x`.

Client 3:

```
RENAME TABLE x TO old_x, new_x TO x;
```

The statement requests exclusive locks in name order on `new_x`, `old_x`, and `x`, but blocks waiting for the lock on `new_x`.

Client 1:

```
UNLOCK TABLES;
```

The statement releases the write locks on `x` and `new_x`. For `x`, the only pending request is by Client 2, so Client 2 acquires its lock, performs the insert, and releases the lock. For `new_x`, the only pending request is by Client 3, which is permitted to acquire that lock (and also the lock on `old_x`). The rename operation still blocks for the lock on `x` until the Client 2 insert finishes and releases its lock. Then Client 3 acquires the lock on `x`, performs the rename, and releases its lock.

In this case, lock acquisition order results in the `INSERT` executing before the `RENAME TABLE`. The `x` into which the insert occurs is the original `x`, now renamed to `old_x` by the rename operation:

```
mysql> SELECT * FROM x;
Empty set (0.01 sec)

mysql> SELECT * FROM old_x;
+------+
| i    |
+------+
|    1 |
+------+
```

If order of lock acquisition in concurrent statements makes a difference to an application in operation outcome, as in the preceding example, you may be able to adjust the table names to affect the order of lock acquisition.

Metadata Lock Release

To ensure transaction serializability, the server must not permit one session to perform a data definition language (DDL) statement on a table that is used in an uncompleted explicitly or implicitly started transaction in another session. The server achieves this by acquiring metadata locks on tables used within a transaction and deferring release of those locks until the transaction ends. A metadata lock on a table prevents changes to the table's structure. This locking approach has the implication that a table that is being used by a transaction within one session cannot be used in DDL statements by other sessions until the transaction ends.

This principle applies not only to transactional tables, but also to nontransactional tables. Suppose that a session begins a transaction that uses transactional table `t` and nontransactional table `nt` as follows:

```
START TRANSACTION;
SELECT * FROM t;
SELECT * FROM nt;
```

The server holds metadata locks on both `t` and `nt` until the transaction ends. If another session attempts a DDL or write lock operation on either table, it blocks until metadata lock release at transaction end. For example, a second session blocks if it attempts any of these operations:

```
DROP TABLE t;
ALTER TABLE t ...;
DROP TABLE nt;
ALTER TABLE nt ...;
LOCK TABLE t ... WRITE;
```

If the server acquires metadata locks for a statement that is syntactically valid but fails during execution, it does not release the locks early. Lock release is still deferred to the end of the transaction because the failed statement is written to the binary log and the locks protect log consistency.

In autocommit mode, each statement is in effect a complete transaction, so metadata locks acquired for the statement are held only to the end of the statement.

Metadata locks acquired during a `PREPARE` statement are released once the statement has been prepared, even if preparation occurs within a multiple-statement transaction.

8.11.5 External Locking

External locking is the use of file system locking to manage contention for `MyISAM` database tables by multiple processes. External locking is used in situations where a single process such as the MySQL server cannot be assumed to be the only process that requires access to tables. Here are some examples:

- If you run multiple servers that use the same database directory (not recommended), each server must have external locking enabled.
- If you use `myisamchk` to perform table maintenance operations on `MyISAM` tables, you must either ensure that the server is not running, or that the server has external locking enabled so that it locks table files as necessary to coordinate with `myisamchk` for access to the tables. The same is true for use of `myisampack` to pack `MyISAM` tables.

  If the server is run with external locking enabled, you can use `myisamchk` at any time for read operations such a checking tables. In this case, if the server tries to update a table that `myisamchk` is using, the server waits for `myisamchk` to finish before it continues.

  If you use `myisamchk` for write operations such as repairing or optimizing tables, or if you use `myisampack` to pack tables, you *must* always ensure that the `mysqld` server is not using the table. If you do not stop `mysqld`, at least do a `mysqladmin flush-tables` before you run `myisamchk`. Your tables *may become corrupted* if the server and `myisamchk` access the tables simultaneously.

With external locking in effect, each process that requires access to a table acquires a file system lock for the table files before proceeding to access the table. If all necessary locks cannot be acquired, the process is blocked from accessing the table until the locks can be obtained (after the process that currently holds the locks releases them).

External locking affects server performance because the server must sometimes wait for other processes before it can access tables.

External locking is unnecessary if you run a single server to access a given data directory (which is the usual case) and if no other programs such as `myisamchk` need to modify tables while the server is running. If you only *read* tables with other programs, external locking is not required, although `myisamchk` might report warnings if the server changes tables while `myisamchk` is reading them.

With external locking disabled, to use `myisamchk`, you must either stop the server while `myisamchk` executes or else lock and flush the tables before running `myisamchk`. (See Section 8.12.1, “System Factors”.) To avoid this requirement, use the `CHECK TABLE` and `REPAIR TABLE` statements to check and repair `MyISAM` tables.

For `mysqld`, external locking is controlled by the value of the `skip_external_locking` system variable. When this variable is enabled, external locking is disabled, and vice versa. External locking is disabled by default.

Use of external locking can be controlled at server startup by using the `--external-locking` or `--skip-external-locking` option.

If you do use external locking option to enable updates to `MyISAM` tables from many MySQL processes, you must ensure that the following conditions are satisfied:

- Do not use the query cache for queries that use tables that are updated by another process.

- Do not start the server with the `delay_key_write` system variable set to `ALL` or use the `DELAY_KEY_WRITE=1` table option for any shared tables. Otherwise, index corruption can occur.

The easiest way to satisfy these conditions is to always use `--external-locking` together with `--delay-key-write=OFF` and `--query-cache-size=0`. (This is not done by default because in many setups it is useful to have a mixture of the preceding options.)

8.12 Optimizing the MySQL Server

This section discusses optimization techniques for the database server, primarily dealing with system configuration rather than tuning SQL statements. The information in this section is appropriate for DBAs who want to ensure performance and scalability across the servers they manage; for developers constructing installation scripts that include setting up the database; and people running MySQL themselves for development, testing, and so on who want to maximize their own productivity.

8.12.1 System Factors

Some system-level factors can affect performance in a major way:

- If you have enough RAM, you could remove all swap devices. Some operating systems use a swap device in some contexts even if you have free memory.
- Avoid external locking for `MyISAM` tables. The default is for external locking to be disabled. The `--external-locking` and `--skip-external-locking` options explicitly enable and disable external locking.

  Disabling external locking does not affect MySQL's functionality as long as you run only one server. Just remember to take down the server (or lock and flush the relevant tables) before you run `myisamchk`. On some systems it is mandatory to disable external locking because it does not work, anyway.

  The only case in which you cannot disable external locking is when you run multiple MySQL *servers* (not clients) on the same data, or if you run `myisamchk` to check (not repair) a table without telling the server to flush and lock the tables first. Note that using multiple MySQL servers to access the same data concurrently is generally *not* recommended, except when using NDB Cluster.

  The `LOCK TABLES` and `UNLOCK TABLES` statements use internal locking, so you can use them even if external locking is disabled.

8.12.2 Optimizing Disk I/O

This section describes ways to configure storage devices when you can devote more and faster storage hardware to the database server. For information about optimizing an `InnoDB` configuration to improve I/O performance, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- Disk seeks are a huge performance bottleneck. This problem becomes more apparent when the amount of data starts to grow so large that effective caching becomes impossible. For large databases where you access data more or less randomly, you can be sure that you need at least one disk seek to read and a couple of disk seeks to write things. To minimize this problem, use disks with low seek times.
- Increase the number of available disk spindles (and thereby reduce the seek overhead) by either symlinking files to different disks or striping the disks:
  - Using symbolic links

    This means that, for `MyISAM` tables, you symlink the index file and data files from their usual location in the data directory to another disk (that may also be striped). This makes both the seek and read times better, assuming that the disk is not used for other purposes as well. See Section 8.12.3, “Using Symbolic Links”.

Symbolic links are not supported for use with `InnoDB` tables. However, it is possible to place `InnoDB` data and log files on different physical disks. For more information, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- Striping

  Striping means that you have many disks and put the first block on the first disk, the second block on the second disk, and the *`N`*-th block on the (*`N MOD number_of_disks`*) disk, and so on. This means if your normal data size is less than the stripe size (or perfectly aligned), you get much better performance. Striping is very dependent on the operating system and the stripe size, so benchmark your application with different stripe sizes. See Section 8.13.3, “Using Your Own Benchmarks”.

  The speed difference for striping is *very* dependent on the parameters. Depending on how you set the striping parameters and number of disks, you may get differences measured in orders of magnitude. You have to choose to optimize for random or sequential access.

- For reliability, you may want to use RAID 0+1 (striping plus mirroring), but in this case, you need 2 × *`N`* drives to hold *`N`* drives of data. This is probably the best option if you have the money for it. However, you may also have to invest in some volume-management software to handle it efficiently.
- A good option is to vary the RAID level according to how critical a type of data is. For example, store semi-important data that can be regenerated on a RAID 0 disk, but store really important data such as host information and logs on a RAID 0+1 or RAID *`N`* disk. RAID *`N`* can be a problem if you have many writes, due to the time required to update the parity bits.
- You can also set the parameters for the file system that the database uses:

  If you do not need to know when files were last accessed (which is not really useful on a database server), you can mount your file systems with the `-o noatime` option. That skips updates to the last access time in inodes on the file system, which avoids some disk seeks.

  On many operating systems, you can set a file system to be updated asynchronously by mounting it with the `-o async` option. If your computer is reasonably stable, this should give you better performance without sacrificing too much reliability. (This flag is on by default on Linux.)

Using NFS with MySQL

You should be cautious when considering whether to use NFS with MySQL. Potential issues, which vary by operating system and NFS version, include the following:

- MySQL data and log files placed on NFS volumes becoming locked and unavailable for use. Locking issues may occur in cases where multiple instances of MySQL access the same data directory or where MySQL is shut down improperly, due to a power outage, for example. NFS version 4 addresses underlying locking issues with the introduction of advisory and lease-based locking. However, sharing a data directory among MySQL instances is not recommended.
- Data inconsistencies introduced due to messages received out of order or lost network traffic. To avoid this issue, use TCP with `hard` and `intr` mount options.
- Maximum file size limitations. NFS Version 2 clients can only access the lowest 2GB of a file (signed 32 bit offset). NFS Version 3 clients support larger files (up to 64 bit offsets). The maximum supported file size also depends on the local file system of the NFS server.

Using NFS within a professional SAN environment or other storage system tends to offer greater reliability than using NFS outside of such an environment. However, NFS within a SAN environment may be slower than directly attached or bus-attached non-rotational storage.

If you choose to use NFS, NFS Version 4 or later is recommended, as is testing your NFS setup thoroughly before deploying into a production environment.

8.12.3 Using Symbolic Links

You can move databases or tables from the database directory to other locations and replace them with symbolic links to the new locations. You might want to do this, for example, to move a database to a file system with more free space or increase the speed of your system by spreading your tables to different disks.

For `InnoDB` tables, use the `DATA DIRECTORY` clause of the `CREATE TABLE` statement instead of symbolic links, as explained in Section 14.6.1.2, “Creating Tables Externally”. This new feature is a supported, cross-platform technique.

The recommended way to do this is to symlink entire database directories to a different disk. Symlink `MyISAM` tables only as a last resort.

To determine the location of your data directory, use this statement:

```
SHOW VARIABLES LIKE 'datadir';
```

8.12.3.1 Using Symbolic Links for Databases on Unix

On Unix, the way to symlink a database is first to create a directory on some disk where you have free space and then to create a soft link to it from the MySQL data directory.

```
shell> mkdir /dr1/databases/test
shell> ln -s /dr1/databases/test /path/to/datadir
```

MySQL does not support linking one directory to multiple databases. Replacing a database directory with a symbolic link works as long as you do not make a symbolic link between databases. Suppose that you have a database `db1` under the MySQL data directory, and then make a symlink `db2` that points to `db1`:

```
shell> cd /path/to/datadir
shell> ln -s db1 db2
```

The result is that, for any table `tbl_a` in `db1`, there also appears to be a table `tbl_a` in `db2`. If one client updates `db1.tbl_a` and another client updates `db2.tbl_a`, problems are likely to occur.

8.12.3.2 Using Symbolic Links for MyISAM Tables on Unix

Symlinks are fully supported only for `MyISAM` tables. For files used by tables for other storage engines, you may get strange problems if you try to use symbolic links. For `InnoDB` tables, use the alternative technique explained in Section 14.6.1.2, “Creating Tables Externally” instead.

Do not symlink tables on systems that do not have a fully operational `realpath()` call. (Linux and Solaris support `realpath()`). To determine whether your system supports symbolic links, check the value of the `have_symlink` system variable using this statement:

```
SHOW VARIABLES LIKE 'have_symlink';
```

The handling of symbolic links for `MyISAM` tables works as follows:

- In the data directory, you always have the table format (`.frm`) file, the data (`.MYD`) file, and the index (`.MYI`) file. The data file and index file can be moved elsewhere and replaced in the data directory by symlinks. The format file cannot.
- You can symlink the data file and the index file independently to different directories.
- To instruct a running MySQL server to perform the symlinking, use the `DATA DIRECTORY` and `INDEX DIRECTORY` options to `CREATE TABLE`. See Section 13.1.17, “CREATE TABLE Statement”. Alternatively, if `mysqld` is not running, symlinking can be accomplished manually using `ln -s` from the command line.

Note

The path used with either or both of the `DATA DIRECTORY` and `INDEX DIRECTORY` options may not include the MySQL `data` directory. (Bug #32167)

- `myisamchk` does not replace a symlink with the data file or index file. It works directly on the file to which the symlink points. Any temporary files are created in the directory where the data file or index file is located. The same is true for the `ALTER TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements.

-

  Note

  When you drop a table that is using symlinks, *both the symlink and the file to which the symlink points are dropped*. This is an extremely good reason *not* to run `mysqld` as the `root` operating system user or permit operating system users to have write access to MySQL database directories.

- If you rename a table with `ALTER TABLE ... RENAME` or `RENAME TABLE` and you do not move the table to another database, the symlinks in the database directory are renamed to the new names and the data file and index file are renamed accordingly.

- If you use `ALTER TABLE ... RENAME` or `RENAME TABLE` to move a table to another database, the table is moved to the other database directory. If the table name changed, the symlinks in the new database directory are renamed to the new names and the data file and index file are renamed accordingly.

- If you are not using symlinks, start `mysqld` with the `--skip-symbolic-links` option to ensure that no one can use `mysqld` to drop or rename a file outside of the data directory.

These table symlink operations are not supported:

- `ALTER TABLE` ignores the `DATA DIRECTORY` and `INDEX DIRECTORY` table options.

- As indicated previously, only the data and index files can be symbolic links. The `.frm` file must *never* be a symbolic link. Attempting to do this (for example, to make one table name a synonym for another) produces incorrect results. Suppose that you have a database `db1` under the MySQL data directory, a table `tbl1` in this database, and in the `db1` directory you make a symlink `tbl2` that points to `tbl1`:

```
shell> cd /path/to/datadir/db1
shell> ln -s tbl1.frm tbl2.frm
shell> ln -s tbl1.MYD tbl2.MYD
shell> ln -s tbl1.MYI tbl2.MYI
```

  Problems result if one thread reads `db1.tbl1` and another thread updates `db1.tbl2`:

  - The query cache is “fooled” (it has no way of knowing that `tbl1` has not been updated, so it returns outdated results).

  - `ALTER` statements on `tbl2` fail.

8.12.3.3 Using Symbolic Links for Databases on Windows

On Windows, symbolic links can be used for database directories. This enables you to put a database directory at a different location (for example, on a different disk) by setting up a symbolic link to it. Use of database symlinks on Windows is similar to their use on Unix, although the procedure for setting up the link differs.

Suppose that you want to place the database directory for a database named `mydb` at `D:\data\mydb`. To do this, create a symbolic link in the MySQL data directory that points to `D:\data\mydb`. However,

before creating the symbolic link, make sure that the `D:\data\mydb` directory exists by creating it if necessary. If you already have a database directory named `mydb` in the data directory, move it to `D:\data`. Otherwise, the symbolic link is ineffective. To avoid problems, make sure that the server is not running when you move the database directory.

The procedure for creating the database symbolic link depends on your version of Windows.

On Windows, you can create a symlink using the `mklink` command. This command requires administrative privileges.

1. Change location into the data directory:

   ```
   C:\> cd \path\to\datadir
   ```

2. In the data directory, create a symlink named `mydb` that points to the location of the database directory:

   ```
   C:\> mklink /d mydb D:\data\mydb
   ```

After this, all tables created in the database `mydb` are created in `D:\data\mydb`.

Alternatively, on any version of Windows supported by MySQL, you can create a symbolic link to a MySQL database by creating a `.sym` file in the data directory that contains the path to the destination directory. The file should be named *`db_name`*`.sym`, where *`db_name`* is the database name.

Support for database symbolic links on Windows using `.sym` files is enabled by default. If you do not need `.sym` file symbolic links, you can disable support for them by starting `mysqld` with the `--skip-symbolic-links` option. To determine whether your system supports `.sym` file symbolic links, check the value of the `have_symlink` system variable using this statement:

```
SHOW VARIABLES LIKE 'have_symlink';
```

To create a `.sym` file symlink, use this procedure:

1. Change location into the data directory:

   ```
   C:\> cd \path\to\datadir
   ```

2. In the data directory, create a text file named `mydb.sym` that contains this path name: `D:\data\mydb\`

Note

The path name to the new database and tables should be absolute. If you specify a relative path, the location is relative to the `mydb.sym` file.

After this, all tables created in the database `mydb` are created in `D:\data\mydb`.

Note

Because support for `.sym` files is redundant with native symlink support available using `mklink`, use of `.sym` files is deprecated; expect support for them to be removed in a future MySQL release.

The following limitations apply to the use of `.sym` files for database symbolic linking on Windows. These limitations do not apply for symlinks created using `mklink`.

- The symbolic link is not used if a directory with the same name as the database exists in the MySQL data directory.
- The `--innodb_file_per_table` option cannot be used.
- If you run `mysqld` as a service, you cannot use a mapped drive to a remote server as the destination of the symbolic link. As a workaround, you can use the full path (`\\servername\path\`).

8.12.4 Optimizing Memory Use

8.12.4.1 How MySQL Uses Memory

MySQL allocates buffers and caches to improve performance of database operations. You can improve MySQL performance by increasing the values of certain cache and buffer-related system variables. You can also modify these variables to run MySQL on systems with limited memory.

The following list describes some of the ways that MySQL uses memory. Where applicable, relevant system variables are referenced. Some items are storage engine or feature specific.

- The `InnoDB` buffer pool is a memory area that holds cached `InnoDB` data for tables, indexes, and other auxiliary buffers. For efficiency of high-volume read operations, the buffer pool is divided into pages that can potentially hold multiple rows. For efficiency of cache management, the buffer pool is implemented as a linked list of pages; data that is rarely used is aged out of the cache, using a variation of the LRU algorithm. For more information, see Section 14.5.1, “Buffer Pool”.

  The size of the buffer pool is important for system performance:

  - `InnoDB` allocates memory for the entire buffer pool at server startup, using `malloc()` operations. The `innodb_buffer_pool_size` system variable defines the buffer pool size. Typically, a recommended `innodb_buffer_pool_size` value is 50 to 75 percent of system memory. For more information, see Configuring InnoDB Buffer Pool Size.

  - On systems with a large amount of memory, you can improve concurrency by dividing the buffer pool into multiple buffer pool instances. The `innodb_buffer_pool_instances` system variable defines the number of buffer pool instances.

  - A buffer pool that is too small may cause excessive churning as pages are flushed from the buffer pool only to be required again a short time later.

  - A buffer pool that is too large may cause swapping due to competition for memory.

- All threads share the `MyISAM` key buffer. The `key_buffer_size` system variable determines its size.

  For each `MyISAM` table the server opens, the index file is opened once; the data file is opened once for each concurrently running thread that accesses the table. For each concurrent thread, a table structure, column structures for each column, and a buffer of size `3 * N` are allocated (where *`N`* is the maximum row length, not counting `BLOB` columns). A `BLOB` column requires five to eight bytes plus the length of the `BLOB` data. The `MyISAM` storage engine maintains one extra row buffer for internal use.

- The `myisam_use_mmap` system variable can be set to 1 to enable memory-mapping for all `MyISAM` tables.

- If an internal in-memory temporary table becomes too large (as determined using the `tmp_table_size` and `max_heap_table_size` system variables), MySQL automatically converts the table from in-memory to on-disk format. On-disk temporary tables use the `MyISAM` storage engine. You can increase the permissible temporary table size as described in Section 8.4.4, “Internal Temporary Table Use in MySQL”.

  For `MEMORY` tables explicitly created with `CREATE TABLE`, only the `max_heap_table_size` system variable determines how large a table can grow, and there is no conversion to on-disk format.

- The MySQL Performance Schema is a feature for monitoring MySQL server execution at a low level. For performance reasons, fixed memory buffers for Performance Schema are allocated at server startup and do not change in size while the server is running.

- Each thread that the server uses to manage client connections requires some thread-specific space. The following list indicates these and which system variables control their size:

  - A stack (`thread_stack`)
  - A connection buffer (`net_buffer_length`)
  - A result buffer (`net_buffer_length`)

  The connection buffer and result buffer each begin with a size equal to `net_buffer_length` bytes, but are dynamically enlarged up to `max_allowed_packet` bytes as needed. The result buffer shrinks to `net_buffer_length` bytes after each SQL statement. While a statement is running, a copy of the current statement string is also allocated.

  Each connection thread uses memory for computing statement digests. In MySQL 5.6.24 and up, the server allocates `max_digest_length` bytes per session. Before MySQL 5.6.24, the server allocates 1024 bytes per session. See Section 22.10, “Performance Schema Statement Digests”.

- All threads share the same base memory.
- When a thread is no longer needed, the memory allocated to it is released and returned to the system unless the thread goes back into the thread cache. In that case, the memory remains allocated.
- Each request that performs a sequential scan of a table allocates a *read buffer*. The `read_buffer_size` system variable determines the buffer size.
- When reading rows in an arbitrary sequence (for example, following a sort), a *random-read buffer* may be allocated to avoid disk seeks. The `read_rnd_buffer_size` system variable determines the buffer size.
- All joins are executed in a single pass, and most joins can be done without even using a temporary table. Most temporary tables are memory-based hash tables. Temporary tables with a large row length (calculated as the sum of all column lengths) or that contain `BLOB` columns are stored on disk.
- Most requests that perform a sort allocate a sort buffer and zero to two temporary files depending on the result set size. See Section B.3.3.5, “Where MySQL Stores Temporary Files”.
- Almost all parsing and calculating is done in thread-local and reusable memory pools. No memory overhead is needed for small items, thus avoiding the normal slow memory allocation and freeing. Memory is allocated only for unexpectedly large strings.
- For each table having `BLOB` columns, a buffer is enlarged dynamically to read in larger `BLOB` values. If you scan a table, the buffer grows as large as the largest `BLOB` value.
- MySQL requires memory and descriptors for the table cache. Handler structures for all in-use tables are saved in the table cache and managed as “First In, First Out” (FIFO). The `table_open_cache` system variable defines the initial table cache size; see Section 8.4.3.1, “How MySQL Opens and Closes Tables”.

  MySQL also requires memory for the table definition cache. The `table_definition_cache` system variable defines the number of table definitions (from `.frm` files) that can be stored in the table definition cache. If you use a large number of tables, you can create a large table definition cache to speed up the opening of tables. The table definition cache takes less space and does not use file descriptors, unlike the table cache.

- A `FLUSH TABLES` statement or `mysqladmin flush-tables` command closes all tables that are not in use at once and marks all in-use tables to be closed when the currently executing thread finishes. This effectively frees most in-use memory. `FLUSH TABLES` does not return until all tables have been closed.
- The server caches information in memory as a result of `GRANT`, `CREATE USER`, `CREATE SERVER`, and `INSTALL PLUGIN` statements. This memory is not released by the corresponding `REVOKE`, `DROP USER`, `DROP SERVER`, and `UNINSTALL PLUGIN` statements, so for a server that executes

many instances of the statements that cause caching, there is likely to be an increase in cached memory use unless it is freed with `FLUSH PRIVILEGES`.

`ps` and other system status programs may report that `mysqld` uses a lot of memory. This may be caused by thread stacks on different memory addresses. For example, the Solaris version of `ps` counts the unused memory between stacks as used memory. To verify this, check available swap with `swap -s`. We test `mysqld` with several memory-leakage detectors (both commercial and Open Source), so there should be no memory leaks.

8.12.4.2 Enabling Large Page Support

Some hardware/operating system architectures support memory pages greater than the default (usually 4KB). The actual implementation of this support depends on the underlying hardware and operating system. Applications that perform a lot of memory accesses may obtain performance improvements by using large pages due to reduced Translation Lookaside Buffer (TLB) misses.

In MySQL, large pages can be used by InnoDB, to allocate memory for its buffer pool and additional memory pool.

Standard use of large pages in MySQL attempts to use the largest size supported, up to 4MB. Under Solaris, a “super large pages” feature enables uses of pages up to 256MB. This feature is available for recent SPARC platforms. It can be enabled or disabled by using the `--super-large-pages` or `--skip-super-large-pages` option.

MySQL also supports the Linux implementation of large page support (which is called HugeTLB in Linux).

Before large pages can be used on Linux, the kernel must be enabled to support them and it is necessary to configure the HugeTLB memory pool. For reference, the HugeTBL API is documented in the `Documentation/vm/hugetlbpage.txt` file of your Linux sources.

The kernel for some recent systems such as Red Hat Enterprise Linux appear to have the large pages feature enabled by default. To check whether this is true for your kernel, use the following command and look for output lines containing “huge”:

```
shell> cat /proc/meminfo | grep -i huge
HugePages_Total:       0
HugePages_Free:        0
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       4096 kB
```

The nonempty command output indicates that large page support is present, but the zero values indicate that no pages are configured for use.

If your kernel needs to be reconfigured to support large pages, consult the `hugetlbpage.txt` file for instructions.

Assuming that your Linux kernel has large page support enabled, configure it for use by MySQL using the following commands. Normally, you put these in an `rc` file or equivalent startup file that is executed during the system boot sequence, so that the commands execute each time the system starts. The commands should execute early in the boot sequence, before the MySQL server starts. Be sure to change the allocation numbers and the group number as appropriate for your system.

```
# Set the number of pages to be used.
# Each page is normally 2MB, so a value of 20 = 40MB.
# This command actually allocates memory, so this much
# memory must be available.
echo 20 > /proc/sys/vm/nr_hugepages

# Set the group number that is permitted to access this
# memory (102 in this case). The mysql user must be a
# member of this group.
echo 102 > /proc/sys/vm/hugetlb_shm_group
```

```
# Increase the amount of shmem permitted per segment
# (12G in this case).
echo 1560281088 > /proc/sys/kernel/shmmax

# Increase total amount of shared memory.  The value
# is the number of pages. At 4KB/page, 4194304 = 16GB.
echo 4194304 > /proc/sys/kernel/shmall
```

For MySQL usage, you normally want the value of `shmmax` to be close to the value of `shmall`.

To verify the large page configuration, check `/proc/meminfo` again as described previously. Now you should see some nonzero values:

```
shell> cat /proc/meminfo | grep -i huge
HugePages_Total:      20
HugePages_Free:       20
HugePages_Rsvd:        0
HugePages_Surp:        0
Hugepagesize:       4096 kB
```

The final step to make use of the `hugetlb_shm_group` is to give the `mysql` user an “unlimited” value for the memlock limit. This can be done either by editing `/etc/security/limits.conf` or by adding the following command to your `mysqld_safe` script:

```
ulimit -l unlimited
```

Adding the `ulimit` command to `mysqld_safe` causes the `root` user to set the memlock limit to `unlimited` before switching to the `mysql` user. (This assumes that `mysqld_safe` is started by `root`.)

Large page support in MySQL is disabled by default. To enable it, start the server with the `--large-pages` option. For example, you can use the following lines in the server `my.cnf` file:

```
[mysqld]
large-pages
```

With this option, `InnoDB` uses large pages automatically for its buffer pool and additional memory pool. If `InnoDB` cannot do this, it falls back to use of traditional memory and writes a warning to the error log: `Warning: Using conventional memory pool`

To verify that large pages are being used, check `/proc/meminfo` again:

```
shell> cat /proc/meminfo | grep -i huge
HugePages_Total:      20
HugePages_Free:       20
HugePages_Rsvd:        2
HugePages_Surp:        0
Hugepagesize:       4096 kB
```

8.13 Measuring Performance (Benchmarking)

To measure performance, consider the following factors:

- Whether you are measuring the speed of a single operation on a quiet system, or how a set of operations (a “workload”) works over a period of time. With simple tests, you usually test how changing one aspect (a configuration setting, the set of indexes on a table, the SQL clauses in a query) affects performance. Benchmarks are typically long-running and elaborate performance tests, where the results could dictate high-level choices such as hardware and storage configuration, or how soon to upgrade to a new MySQL version.
- For benchmarking, sometimes you must simulate a heavy database workload to get an accurate picture.
- Performance can vary depending on so many different factors that a difference of a few percentage points might not be a decisive victory. The results might shift the opposite way when you test in a different environment.

- Certain MySQL features help or do not help performance depending on the workload. For completeness, always test performance with those features turned on and turned off. The two most important features to try with each workload are the MySQL query cache, and the adaptive hash index for `InnoDB` tables.

This section progresses from simple and direct measurement techniques that a single developer can do, to more complicated ones that require additional expertise to perform and interpret the results.

8.13.1 Measuring the Speed of Expressions and Functions

To measure the speed of a specific MySQL expression or function, invoke the `BENCHMARK()` function using the `mysql` client program. Its syntax is `BENCHMARK(loop_count,expr)`. The return value is always zero, but `mysql` prints a line displaying approximately how long the statement took to execute. For example:

```
mysql> SELECT BENCHMARK(1000000,1+1);
+------------------------+
| BENCHMARK(1000000,1+1) |
+------------------------+
|                      0 |
+------------------------+
1 row in set (0.32 sec)
```

This result was obtained on a Pentium II 400MHz system. It shows that MySQL can execute 1,000,000 simple addition expressions in 0.32 seconds on that system.

The built-in MySQL functions are typically highly optimized, but there may be some exceptions. `BENCHMARK()` is an excellent tool for finding out if some function is a problem for your queries.

8.13.2 The MySQL Benchmark Suite

This benchmark suite is meant to tell any user what operations a given SQL implementation performs well or poorly. You can get a good idea for how the benchmarks work by looking at the code and results in the `sql-bench` directory in any MySQL source distribution.

To use the benchmark suite, the following requirements must be satisfied:

- The benchmark suite is provided with MySQL source distributions. You can either download a released distribution from https://dev.mysql.com/downloads/, or use the current development source tree. (See Section 2.9.5, “Installing MySQL Using a Development Source Tree”.)
- The benchmark scripts are written in Perl and use the Perl DBI module to access database servers, so DBI must be installed. You also need the server-specific DBD drivers for each of the servers you want to test. For example, to test MySQL, PostgreSQL, and DB2, you must have the `DBD::mysql`, `DBD::Pg`, and `DBD::DB2` modules installed. See Section 2.13, “Perl Installation Notes”.

After you obtain a MySQL source distribution, you can find the benchmark suite located in its `sql-bench` directory. To run the benchmark tests, build MySQL, and then change location into the `sql-bench` directory and execute the `run-all-tests` script:

```
shell> cd sql-bench
shell> perl run-all-tests --server=server_name
```

`server_name` should be the name of one of the supported servers. To get a list of all options and supported servers, invoke this command:

```
shell> perl run-all-tests --help
```

The `crash-me` script also is located in the `sql-bench` directory. `crash-me` tries to determine what features a database system supports and what its capabilities and limitations are by actually running queries. For example, it determines:

- What data types are supported

- How many indexes are supported
- What functions are supported
- How big a query can be
- How big a `VARCHAR` column can be

For more information about benchmark results, visit http://www.mysql.com/why-mysql/benchmarks/.

8.13.3 Using Your Own Benchmarks

Benchmark your application and database to find out where the bottlenecks are. After fixing one bottleneck (or by replacing it with a "dummy" module), you can proceed to identify the next bottleneck. Even if the overall performance for your application currently is acceptable, you should at least make a plan for each bottleneck and decide how to solve it if someday you really need the extra performance.

For examples of portable benchmark programs, look at those in the MySQL benchmark suite. See Section 8.13.2, "The MySQL Benchmark Suite". You can take any program from this suite and modify it for your own needs. By doing this, you can try different solutions to your problem and test which really is fastest for you.

Another free benchmark suite is the Open Source Database Benchmark, available at http://osdb.sourceforge.net/.

It is very common for a problem to occur only when the system is very heavily loaded. We have had many customers who contact us when they have a (tested) system in production and have encountered load problems. In most cases, performance problems turn out to be due to issues of basic database design (for example, table scans are not good under high load) or problems with the operating system or libraries. Most of the time, these problems would be much easier to fix if the systems were not already in production.

To avoid problems like this, benchmark your whole application under the worst possible load:

- The `mysqlslap` program can be helpful for simulating a high load produced by multiple clients issuing queries simultaneously. See Section 4.5.7, "`mysqlslap` — A Load Emulation Client".
- You can also try benchmarking packages such as SysBench and DBT2, available at https://launchpad.net/sysbench, and http://osdldbt.sourceforge.net/#dbt2.

These programs or packages can bring a system to its knees, so be sure to use them only on your development systems.

8.13.4 Measuring Performance with performance_schema

You can query the tables in the `performance_schema` database to see real-time information about the performance characteristics of your server and the applications it is running. See Chapter 22, *MySQL Performance Schema* for details.

8.14 Examining Server Thread (Process) Information

To ascertain what your MySQL server is doing, it can be helpful to examine the process list, which indicates the operations currently being performed by the set of threads executing within the server. For example:

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 1
   User: event_scheduler
   Host: localhost
     db: NULL
Command: Daemon
   Time: 2756681
```

```
  State: Waiting on empty queue
   Info: NULL
*************************** 2. row ***************************
     Id: 20
   User: me
   Host: localhost:52943
     db: test
Command: Query
   Time: 0
  State: init
   Info: SHOW PROCESSLIST
```

Threads can be killed with the `KILL` statement. See Section 13.7.6.4, “KILL Statement”.

8.14.1 Accessing the Process List

The following discussion enumerates the sources of process information, the privileges required to see process information, and describes the content of process list entries.

- Sources of Process Information
- Privileges Required to Access the Process List
- Content of Process List Entries

Sources of Process Information

Process information is available from these sources:

- The `SHOW PROCESSLIST` statement: Section 13.7.5.30, “SHOW PROCESSLIST Statement”
- The `mysqladmin processlist` command: Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
- The `INFORMATION_SCHEMA PROCESSLIST` table: Section 21.16, “The INFORMATION_SCHEMA PROCESSLIST Table”
- The Performance Schema `threads` table columns with names having a prefix of `PROCESSLIST_`: Section 22.12.10.3, “The threads Table”

The `threads` table compares to `SHOW PROCESSLIST`, `INFORMATION_SCHEMA PROCESSLIST`, and `mysqladmin processlist` as follows:

- Access to the `threads` table does not require a mutex and has minimal impact on server performance. The other sources have negative performance consequences because they require a mutex.
- The `threads` table displays background threads, which the other sources do not. It also provides additional information for each thread that the other sources do not, such as whether the thread is a foreground or background thread, and the location within the server associated with the thread. This means that the `threads` table can be used to monitor thread activity the other sources cannot.
- You can enable or disable Performance Schema thread monitoring, as described in Section 22.12.10.3, “The threads Table”.

For these reasons, DBAs who perform server monitoring using one of the other thread information sources may wish to monitor using the `threads` table instead.

Privileges Required to Access the Process List

For most sources of process information, if you have the `PROCESS` privilege, you can see all threads, even those belonging to other users. Otherwise (without the `PROCESS` privilege), nonanonymous users have access to information about their own threads but not threads for other users, and anonymous users have no access to thread information.

The Performance Schema `threads` table also provides thread information, but table access uses a different privilege model. See Section 22.12.10.3, “The threads Table”.

Content of Process List Entries

Each process list entry contains several pieces of information. The following list describes them using the labels from `SHOW PROCESSLIST` output. Other process information sources use similar labels.

- `Id` is the connection identifier for the client associated with the thread.
- `User` and `Host` indicate the account associated with the thread.
- `db` is the default database for the thread, or `NULL` if none has been selected.
- `Command` and `State` indicate what the thread is doing.

  Most states correspond to very quick operations. If a thread stays in a given state for many seconds, there might be a problem that needs to be investigated.

  The following sections list the possible `Command` values, and `State` values grouped by category. The meaning for some of these values is self-evident. For others, additional description is provided.

  Note

  Applications that examine process list information should be aware that the commands and states are subject to change.

- `Time` indicates how long the thread has been in its current state. The thread's notion of the current time may be altered in some cases: The thread can change the time with `SET TIMESTAMP = value`. For a replica SQL thread, the value is the number of seconds between the timestamp of the last replicated event and the real time of the replica host. See Section 17.2.1, “Replication Threads”.
- `Info` indicates the statement the thread is executing, or `NULL` if it is executing no statement. For `SHOW PROCESSLIST`, this value contains only the first 100 characters of the statement. To see complete statements, use `SHOW FULL PROCESSLIST` (or query a diffferent process information source).

8.14.2 Thread Command Values

A thread can have any of the following `Command` values:

- `Binlog Dump`

  This is a thread on a replication source for sending binary log contents to a replica.

- `Change user`

  The thread is executing a change user operation.

- `Close stmt`

  The thread is closing a prepared statement.

- `Connect`

  A replica is connected to its source.

- `Connect Out`

  A replica is connecting to its source.

- `Create DB`

The thread is executing a create database operation.

- `Daemon`

  This thread is internal to the server, not a thread that services a client connection.

- `Debug`

  The thread is generating debugging information.

- `Delayed insert`

  The thread is a delayed insert handler.

- `Drop DB`

  The thread is executing a drop database operation.

- `Error`

- `Execute`

  The thread is executing a prepared statement.

- `Fetch`

  The thread is fetching the results from executing a prepared statement.

- `Field List`

  The thread is retrieving information for table columns.

- `Init DB`

  The thread is selecting a default database.

- `Kill`

  The thread is killing another thread.

- `Long Data`

  The thread is retrieving long data in the result of executing a prepared statement.

- `Ping`

  The thread is handling a server ping request.

- `Prepare`

  The thread is preparing a prepared statement.

- `Processlist`

  The thread is producing information about server threads.

- `Query`

  The thread is executing a statement.

- `Quit`

  The thread is terminating.

- `Refresh`

The thread is flushing table, logs, or caches, or resetting status variable or replication server information.

- `Register Slave`

  The thread is registering a replica server.

- `Reset stmt`

  The thread is resetting a prepared statement.

- `Set option`

  The thread is setting or resetting a client statement execution option.

- `Shutdown`

  The thread is shutting down the server.

- `Sleep`

  The thread is waiting for the client to send a new statement to it.

- `Statistics`

  The thread is producing server status information.

- `Time`

  Unused.

8.14.3 General Thread States

The following list describes thread `State` values that are associated with general query processing and not more specialized activities such as replication. Many of these are useful only for finding bugs in the server.

- `After create`

  This occurs when the thread creates a table (including internal temporary tables), at the end of the function that creates the table. This state is used even if the table could not be created due to some error.

- `altering table`

  The server is in the process of executing an in-place `ALTER TABLE`.

- `Analyzing`

  The thread is calculating a `MyISAM` table key distributions (for example, for `ANALYZE TABLE`).

- `checking permissions`

  The thread is checking whether the server has the required privileges to execute the statement.

- `Checking table`

  The thread is performing a table check operation.

- `cleaning up`

  The thread has processed one command and is preparing to free memory and reset certain state variables.

- `closing tables`

  The thread is flushing the changed table data to disk and closing the used tables. This should be a fast operation. If not, verify that you do not have a full disk and that the disk is not in very heavy use.

- `committing alter table to storage engine`

  The server has finished an in-place `ALTER TABLE` and is committing the result.

- `converting HEAP to MyISAM`

  The thread is converting an internal temporary table from a `MEMORY` table to an on-disk `MyISAM` table.

- `copy to tmp table`

  The thread is processing an `ALTER TABLE` statement. This state occurs after the table with the new structure has been created but before rows are copied into it.

- `Copying to group table`

  If a statement has different `ORDER BY` and `GROUP BY` criteria, the rows are sorted by group and copied to a temporary table.

- `Copying to tmp table`

  The server is copying to a temporary table in memory.

- `Copying to tmp table on disk`

  The server is copying to a temporary table on disk. The temporary result set has become too large (see Section 8.4.4, “Internal Temporary Table Use in MySQL”). Consequently, the thread is changing the temporary table from in-memory to disk-based format to save memory.

- `Creating index`

  The thread is processing `ALTER TABLE ... ENABLE KEYS` for a `MyISAM` table.

- `Creating sort index`

  The thread is processing a `SELECT` that is resolved using an internal temporary table.

- `creating table`

  The thread is creating a table. This includes creation of temporary tables.

- `Creating tmp table`

  The thread is creating a temporary table in memory or on disk. If the table is created in memory but later is converted to an on-disk table, the state during that operation is `Copying to tmp table on disk`.

- `deleting from main table`

  The server is executing the first part of a multiple-table delete. It is deleting only from the first table, and saving columns and offsets to be used for deleting from the other (reference) tables.

- `deleting from reference tables`

  The server is executing the second part of a multiple-table delete and deleting the matched rows from the other tables.

- `discard_or_import_tablespace`

The thread is processing an `ALTER TABLE ... DISCARD TABLESPACE` or `ALTER TABLE ... IMPORT TABLESPACE` statement.

- `end`

  This occurs at the end but before the cleanup of `ALTER TABLE`, `CREATE VIEW`, `DELETE`, `INSERT`, `SELECT`, or `UPDATE` statements.

  For the `end` state, the following operations could be happening:

  - Removing query cache entries after data in a table is changed
  - Writing an event to the binary log
  - Freeing memory buffers, including for blobs

- `executing`

  The thread has begun executing a statement.

- `Execution of init_command`

  The thread is executing statements in the value of the `init_command` system variable.

- `freeing items`

  The thread has executed a command. Some freeing of items done during this state involves the query cache. This state is usually followed by `cleaning up`.

- `FULLTEXT initialization`

  The server is preparing to perform a natural-language full-text search.

- `init`

  This occurs before the initialization of `ALTER TABLE`, `DELETE`, `INSERT`, `SELECT`, or `UPDATE` statements. Actions taken by the server in this state include flushing the binary log, the `InnoDB` log, and some query cache cleanup operations.

- `Killed`

  Someone has sent a `KILL` statement to the thread and it should abort next time it checks the kill flag. The flag is checked in each major loop in MySQL, but in some cases it might still take a short time for the thread to die. If the thread is locked by some other thread, the kill takes effect as soon as the other thread releases its lock.

- `logging slow query`

  The thread is writing a statement to the slow-query log.

- `login`

  The initial state for a connection thread until the client has been authenticated successfully.

- `manage keys`

  The server is enabling or disabling a table index.

- `Opening tables`

  The thread is trying to open a table. This is should be very fast procedure, unless something prevents opening. For example, an `ALTER TABLE` or a `LOCK TABLE` statement can prevent opening

a table until the statement is finished. It is also worth checking that your `table_open_cache` value is large enough.

- `optimizing`

  The server is performing initial optimizations for a query.

- `preparing`

  This state occurs during query optimization.

- `preparing for alter table`

  The server is preparing to execute an in-place `ALTER TABLE`.

- `Purging old relay logs`

  The thread is removing unneeded relay log files.

- `query end`

  This state occurs after processing a query but before the `freeing items` state.

- `Reading from net`

  The server is reading a packet from the network.

- `Removing duplicates`

  The query was using `SELECT DISTINCT` in such a way that MySQL could not optimize away the distinct operation at an early stage. Because of this, MySQL requires an extra stage to remove all duplicated rows before sending the result to the client.

- `removing tmp table`

  The thread is removing an internal temporary table after processing a `SELECT` statement. This state is not used if no temporary table was created.

- `rename`

  The thread is renaming a table.

- `rename result table`

  The thread is processing an `ALTER TABLE` statement, has created the new table, and is renaming it to replace the original table.

- `Reopen tables`

  The thread got a lock for the table, but noticed after getting the lock that the underlying table structure changed. It has freed the lock, closed the table, and is trying to reopen it.

- `Repair by sorting`

  The repair code is using a sort to create indexes.

- `Repair done`

  The thread has completed a multithreaded repair for a `MyISAM` table.

- `Repair with keycache`

  The repair code is using creating keys one by one through the key cache. This is much slower than `Repair by sorting`.

- `Rolling back`

  The thread is rolling back a transaction.

- `Saving state`

  For `MyISAM` table operations such as repair or analysis, the thread is saving the new table state to the `.MYI` file header. State includes information such as number of rows, the `AUTO_INCREMENT` counter, and key distributions.

- `Searching rows for update`

  The thread is doing a first phase to find all matching rows before updating them. This has to be done if the `UPDATE` is changing the index that is used to find the involved rows.

- `Sending data`

  The thread is reading and processing rows for a `SELECT` statement, and sending data to the client. Because operations occurring during this state tend to perform large amounts of disk access (reads), it is often the longest-running state over the lifetime of a given query.

- `setup`

  The thread is beginning an `ALTER TABLE` operation.

- `Sorting for group`

  The thread is doing a sort to satisfy a `GROUP BY`.

- `Sorting for order`

  The thread is doing a sort to satisfy an `ORDER BY`.

- `Sorting index`

  The thread is sorting index pages for more efficient access during a `MyISAM` table optimization operation.

- `Sorting result`

  For a `SELECT` statement, this is similar to `Creating sort index`, but for nontemporary tables.

- `statistics`

  The server is calculating statistics to develop a query execution plan. If a thread is in this state for a long time, the server is probably disk-bound performing other work.

- `System lock`

  The thread has called `mysql_lock_tables()` and the thread state has not been updated since. This is a very general state that can occur for many reasons.

  For example, the thread is going to request or is waiting for an internal or external system lock for the table. This can occur when `InnoDB` waits for a table-level lock during execution of `LOCK TABLES`. If this state is being caused by requests for external locks and you are not using multiple `mysqld` servers that are accessing the same `MyISAM` tables, you can disable external system locks with the `--skip-external-locking` option. However, external locking is disabled by default, so it is likely that this option has no effect. For `SHOW PROFILE`, this state means the thread is requesting the lock (not waiting for it).

- `update`

  The thread is getting ready to start updating the table.

- `Updating`

  The thread is searching for rows to update and is updating them.

- `updating main table`

  The server is executing the first part of a multiple-table update. It is updating only the first table, and saving columns and offsets to be used for updating the other (reference) tables.

- `updating reference tables`

  The server is executing the second part of a multiple-table update and updating the matched rows from the other tables.

- `User lock`

  The thread is going to request or is waiting for an advisory lock requested with a `GET_LOCK()` call. For `SHOW PROFILE`, this state means the thread is requesting the lock (not waiting for it).

- `User sleep`

  The thread has invoked a `SLEEP()` call.

- `Waiting for commit lock`

  `FLUSH TABLES WITH READ LOCK` is waiting for a commit lock.

- `Waiting for global read lock`

  `FLUSH TABLES WITH READ LOCK` is waiting for a global read lock or the global `read_only` system variable is being set.

- `Waiting for tables`

  The thread got a notification that the underlying structure for a table has changed and it needs to reopen the table to get the new structure. However, to reopen the table, it must wait until all other threads have closed the table in question.

  This notification takes place if another thread has used `FLUSH TABLES` or one of the following statements on the table in question: `FLUSH TABLES` *`tbl_name`*, `ALTER TABLE`, `RENAME TABLE`, `REPAIR TABLE`, `ANALYZE TABLE`, or `OPTIMIZE TABLE`.

- `Waiting for table flush`

  The thread is executing `FLUSH TABLES` and is waiting for all threads to close their tables, or the thread got a notification that the underlying structure for a table has changed and it needs to reopen the table to get the new structure. However, to reopen the table, it must wait until all other threads have closed the table in question.

  This notification takes place if another thread has used `FLUSH TABLES` or one of the following statements on the table in question: `FLUSH TABLES` *`tbl_name`*, `ALTER TABLE`, `RENAME TABLE`, `REPAIR TABLE`, `ANALYZE TABLE`, or `OPTIMIZE TABLE`.

- `Waiting for` *`lock_type`* `lock`

  The server is waiting to acquire a `THR_LOCK` lock or a lock from the metadata locking subsystem, where *`lock_type`* indicates the type of lock.

  This state indicates a wait for a `THR_LOCK`:

  - `Waiting for table level lock`

  These states indicate a wait for a metadata lock:

- `Waiting for event metadata lock`
- `Waiting for global read lock`
- `Waiting for schema metadata lock`
- `Waiting for stored function metadata lock`
- `Waiting for stored procedure metadata lock`
- `Waiting for table metadata lock`
- `Waiting for trigger metadata lock`

For information about table lock indicators, see Section 8.11.1, “Internal Locking Methods”. For information about metadata locking, see Section 8.11.4, “Metadata Locking”.

- `Waiting on cond`

  A generic state in which the thread is waiting for a condition to become true. No specific state information is available.

- `Writing to net`

  The server is writing a packet to the network.

8.14.4 Delayed-Insert Thread States

These thread states are associated with processing for `DELAYED` inserts (see Section 13.2.5.3, “INSERT DELAYED Statement”). Some states are associated with connection threads that process `INSERT DELAYED` statements from clients. Other states are associated with delayed-insert handler threads that insert the rows. There is a delayed-insert handler thread for each table for which `INSERT DELAYED` statements are issued.

States associated with a connection thread that processes an `INSERT DELAYED` statement from the client:

- `allocating local table`

  The thread is preparing to feed rows to the delayed-insert handler thread.

- `Creating delayed handler`

  The thread is creating a handler for `DELAYED` inserts.

- `got handler lock`

  This occurs before the `allocating local table` state and after the `waiting for handler lock` state, when the connection thread gets access to the delayed-insert handler thread.

- `got old table`

  This occurs after the `waiting for handler open` state. The delayed-insert handler thread has signaled that it has ended its initialization phase, which includes opening the table for delayed inserts.

- `storing row into queue`

  The thread is adding a new row to the list of rows that the delayed-insert handler thread must insert.

- `waiting for delay_list`

This occurs during the initialization phase when the thread is trying to find the delayed-insert handler thread for the table, and before attempting to gain access to the list of delayed-insert threads.

- `waiting for handler insert`

  An `INSERT DELAYED` handler has processed all pending inserts and is waiting for new ones.

- `waiting for handler lock`

  This occurs before the `allocating local table` state when the connection thread waits for access to the delayed-insert handler thread.

- `waiting for handler open`

  This occurs after the `Creating delayed handler` state and before the `got old table` state. The delayed-insert handler thread has just been started, and the connection thread is waiting for it to initialize.

States associated with a delayed-insert handler thread that inserts the rows:

- `insert`

  The state that occurs just before inserting rows into the table.

- `reschedule`

  After inserting a number of rows, the delayed-insert thread sleeps to let other threads do work.

- `upgrading lock`

  A delayed-insert handler is trying to get a lock for the table to insert rows.

- `Waiting for INSERT`

  A delayed-insert handler is waiting for a connection thread to add rows to the queue (see `storing row into queue`).

8.14.5 Query Cache Thread States

These thread states are associated with the query cache (see Section 8.10.3, “The MySQL Query Cache”).

- `checking privileges on cached query`

  The server is checking whether the user has privileges to access a cached query result.

- `checking query cache for query`

  The server is checking whether the current query is present in the query cache.

- `invalidating query cache entries`

  Query cache entries are being marked invalid because the underlying tables have changed.

- `sending cached result to client`

  The server is taking the result of a query from the query cache and sending it to the client.

- `storing result in query cache`

  The server is storing the result of a query in the query cache.

- `Waiting for query cache lock`

This state occurs while a session is waiting to take the query cache lock. This can happen for any statement that needs to perform some query cache operation, such as an `INSERT` or `DELETE` that invalidates the query cache, a `SELECT` that looks for a cached entry, `RESET QUERY CACHE`, and so forth.

8.14.6 Replication Source Thread States

The following list shows the most common states you may see in the `State` column for the `Binlog Dump` thread of the replication source. If you see no `Binlog Dump` threads on a source, this means that replication is not running; that is, that no replicas are currently connected.

- `Finished reading one binlog; switching to next binlog`

  The thread has finished reading a binary log file and is opening the next one to send to the replica.

- `Master has sent all binlog to slave; waiting for binlog to be updated`

  The thread has read all remaining updates from the binary logs and sent them to the replica. The thread is now idle, waiting for new events to appear in the binary log resulting from new updates occurring on the source.

- `Sending binlog event to slave`

  Binary logs consist of *events*, where an event is usually an update plus some other information. The thread has read an event from the binary log and is now sending it to the replica.

- `Waiting to finalize termination`

  A very brief state that occurs as the thread is stopping.

8.14.7 Replication Replica I/O Thread States

The following list shows the most common states you see in the `State` column for a replica server I/O thread. This state also appears in the `Slave_IO_State` column displayed by `SHOW SLAVE STATUS`, so you can get a good view of what is happening by using that statement.

- `Checking master version`

  A state that occurs very briefly, after the connection to the source is established.

- `Connecting to master`

  The thread is attempting to connect to the source.

- `Queueing master event to the relay log`

  The thread has read an event and is copying it to the relay log so that the SQL thread can process it.

- `Reconnecting after a failed binlog dump request`

  The thread is trying to reconnect to the source.

- `Reconnecting after a failed master event read`

  The thread is trying to reconnect to the source. When connection is established again, the state becomes `Waiting for master to send event`.

- `Registering slave on master`

  A state that occurs very briefly after the connection to the source is established.

- `Requesting binlog dump`

A state that occurs very briefly, after the connection to the source is established. The thread sends to the source a request for the contents of its binary logs, starting from the requested binary log file name and position.

- `Waiting for master to send event`

  The thread has connected to the source and is waiting for binary log events to arrive. This can last for a long time if the source is idle. If the wait lasts for `slave_net_timeout` seconds, a timeout occurs. At that point, the thread considers the connection to be broken and makes an attempt to reconnect.

- `Waiting for master update`

  The initial state before `Connecting to master`.

- `Waiting for slave mutex on exit`

  A state that occurs briefly as the thread is stopping.

- `Waiting for the slave SQL thread to free enough relay log space`

  You are using a nonzero `relay_log_space_limit` value, and the relay logs have grown large enough that their combined size exceeds this value. The I/O thread is waiting until the SQL thread frees enough space by processing relay log contents so that it can delete some relay log files.

- `Waiting to reconnect after a failed binlog dump request`

  If the binary log dump request failed (due to disconnection), the thread goes into this state while it sleeps, then tries to reconnect periodically. The interval between retries can be specified using the `CHANGE MASTER TO` statement.

- `Waiting to reconnect after a failed master event read`

  An error occurred while reading (due to disconnection). The thread is sleeping for the number of seconds set by the `CHANGE MASTER TO` statement (default 60) before attempting to reconnect.

8.14.8 Replication Replica SQL Thread States

The following list shows the most common states you may see in the `State` column for a replica server SQL thread:

- `Making temporary file (append) before replaying LOAD DATA INFILE`

  The thread is executing a `LOAD DATA` statement and is appending the data to a temporary file containing the data from which the replica reads rows.

- `Making temporary file (create) before replaying LOAD DATA INFILE`

  The thread is executing a `LOAD DATA` statement and is creating a temporary file containing the data from which the replica reads rows. This state can only be encountered if the original `LOAD DATA` statement was logged by a source running a version of MySQL lower than MySQL 5.0.3.

- `Reading event from the relay log`

  The thread has read an event from the relay log so that the event can be processed.

- `Slave has read all relay log; waiting for more updates`

  The thread has processed all events in the relay log files, and is now waiting for the I/O thread to write new events to the relay log.

- `Waiting for an event from Coordinator`

Using the multithreaded replica (`slave_parallel_workers` is greater than 1), one of the replica worker threads is waiting for an event from the coordinator thread.

- `Waiting for slave mutex on exit`

  A very brief state that occurs as the thread is stopping.

- `Waiting for Slave Workers to free pending events`

  This waiting action occurs when the total size of events being processed by Workers exceeds the size of the `slave_pending_jobs_size_max` system variable. The Coordinator resumes scheduling when the size drops below this limit. This state occurs only when `slave_parallel_workers` is set greater than 0.

- `Waiting for the next event in relay log`

  The initial state before `Reading event from the relay log`.

- `Waiting until MASTER_DELAY seconds after master executed event`

  The SQL thread has read an event but is waiting for the replica delay to lapse. This delay is set with the `MASTER_DELAY` option of `CHANGE MASTER TO`.

The `Info` column for the SQL thread may also show the text of a statement. This indicates that the thread has read an event from the relay log, extracted the statement from it, and may be executing it.

8.14.9 Replication Replica Connection Thread States

These thread states occur on a replica server but are associated with connection threads, not with the I/O or SQL threads.

- `Changing master`

  The thread is processing a `CHANGE MASTER TO` statement.

- `Killing slave`

  The thread is processing a `STOP SLAVE` statement.

- `Opening master dump table`

  This state occurs after `Creating table from master dump`.

- `Reading master dump table data`

  This state occurs after `Opening master dump table`.

- `Rebuilding the index on master dump table`

  This state occurs after `Reading master dump table data`.

8.14.10 NDB Cluster Thread States

- `Committing events to binlog`
- `Opening mysql.ndb_apply_status`
- `Processing events`

  The thread is processing events for binary logging.

- `Processing events from schema table`

The thread is doing the work of schema replication.

- `Shutting down`
- `Syncing ndb table schema operation and binlog`

  This is used to have a correct binary log of schema operations for NDB.

- `Waiting for allowed to take ndbcluster global schema lock`

  The thread is waiting for permission to take a global schema lock.

- `Waiting for event from ndbcluster`

  The server is acting as an SQL node in an NDB Cluster, and is connected to a cluster management node.

- `Waiting for first event from ndbcluster`
- `Waiting for ndbcluster binlog update to reach current position`
- `Waiting for ndbcluster global schema lock`

  The thread is waiting for a global schema lock held by another thread to be released.

- `Waiting for ndbcluster to start`
- `Waiting for schema epoch`

  The thread is waiting for a schema epoch (that is, a global checkpoint).

8.14.11 Event Scheduler Thread States

These states occur for the Event Scheduler thread, threads that are created to execute scheduled events, or threads that terminate the scheduler.

- `Clearing`

  The scheduler thread or a thread that was executing an event is terminating and is about to end.

- `Initialized`

  The scheduler thread or a thread that executes an event has been initialized.

- `Waiting for next activation`

  The scheduler has a nonempty event queue but the next activation is in the future.

- `Waiting for scheduler to stop`

  The thread issued `SET GLOBAL event_scheduler=OFF` and is waiting for the scheduler to stop.

- `Waiting on empty queue`

  The scheduler's event queue is empty and it is sleeping.

Chapter 9 Language Structure

Table of Contents

9.1 Literal Values 1123
9.1.1 String Literals 1123
9.1.2 Numeric Literals 1126
9.1.3 Date and Time Literals 1126
9.1.4 Hexadecimal Literals 1128
9.1.5 Bit-Value Literals 1130
9.1.6 Boolean Literals 1131
9.1.7 NULL Values 1131
9.2 Schema Object Names 1132
9.2.1 Identifier Length Limits 1133
9.2.2 Identifier Qualifiers 1134
9.2.3 Identifier Case Sensitivity 1136
9.2.4 Mapping of Identifiers to File Names 1138
9.2.5 Function Name Parsing and Resolution 1140
9.3 Keywords and Reserved Words 1143
9.4 User-Defined Variables 1165
9.5 Expressions 1168
9.6 Comments 1172

This chapter discusses the rules for writing the following elements of SQL statements when using MySQL:

- Literal values such as strings and numbers
- Identifiers such as database, table, and column names
- Keywords and reserved words
- User-defined and system variables
- Expressions
- Comments

9.1 Literal Values

This section describes how to write literal values in MySQL. These include strings, numbers, hexadecimal and bit values, boolean values, and `NULL`. The section also covers various nuances that you may encounter when dealing with these basic types in MySQL.

9.1.1 String Literals

A string is a sequence of bytes or characters, enclosed within either single quote (`'`) or double quote (`"`) characters. Examples:

```
'a string'
"another string"
```

Quoted strings placed next to each other are concatenated to a single string. The following lines are equivalent:

```
'a string'
'a' ' ' 'string'
```

If the ANSI_QUOTES SQL mode is enabled, string literals can be quoted only within single quotation marks because a string quoted within double quotation marks is interpreted as an identifier.

A *binary string* is a string of bytes. Every binary string has a character set and collation named binary. A *nonbinary string* is a string of characters. It has a character set other than binary and a collation that is compatible with the character set.

For both types of strings, comparisons are based on the numeric values of the string unit. For binary strings, the unit is the byte; comparisons use numeric byte values. For nonbinary strings, the unit is the character and some character sets support multibyte characters; comparisons use numeric character code values. Character code ordering is a function of the string collation. (For more information, see Section 10.8.5, “The binary Collation Compared to _bin Collations”.)

A character string literal may have an optional character set introducer and COLLATE clause, to designate it as a string that uses a particular character set and collation:

```
[_charset_name]'string' [COLLATE collation_name]
```

Examples:

```
SELECT _latin1'string';
SELECT _binary'string';
SELECT _utf8'string' COLLATE utf8_danish_ci;
```

You can use N'literal' (or n'literal') to create a string in the national character set. These statements are equivalent:

```
SELECT N'some text';
SELECT n'some text';
SELECT _utf8'some text';
```

For information about these forms of string syntax, see Section 10.3.7, “The National Character Set”, and Section 10.3.8, “Character Set Introducers”.

Within a string, certain sequences have special meaning unless the NO_BACKSLASH_ESCAPES SQL mode is enabled. Each of these sequences begins with a backslash (\), known as the *escape character*. MySQL recognizes the escape sequences shown in Table 9.1, “Special Character Escape Sequences”. For all other escape sequences, backslash is ignored. That is, the escaped character is interpreted as if it was not escaped. For example, \x is just x. These sequences are case-sensitive. For example, \b is interpreted as a backspace, but \B is interpreted as B. Escape processing is done according to the character set indicated by the character_set_connection system variable. This is true even for strings that are preceded by an introducer that indicates a different character set, as discussed in Section 10.3.6, “Character String Literal Character Set and Collation”.

Table 9.1 Special Character Escape Sequences

Escape Sequence	Character Represented by Sequence
\0	An ASCII NUL (X'00') character
\'	A single quote (') character
\"	A double quote (") character
\b	A backspace character
\n	A newline (linefeed) character
\r	A carriage return character
\t	A tab character
\Z	ASCII 26 (Control+Z); see note following the table
\\	A backslash (\) character
\%	A % character; see note following the table

Escape Sequence	Character Represented by Sequence
`\_`	A _ character; see note following the table

The ASCII 26 character can be encoded as `\Z` to enable you to work around the problem that ASCII 26 stands for END-OF-FILE on Windows. ASCII 26 within a file causes problems if you try to use `mysql db_name < file_name`.

The `\%` and `\_` sequences are used to search for literal instances of `%` and `_` in pattern-matching contexts where they would otherwise be interpreted as wildcard characters. See the description of the `LIKE` operator in Section 12.8.1, “String Comparison Functions and Operators”. If you use `\%` or `\_` outside of pattern-matching contexts, they evaluate to the strings `\%` and `\_`, not to `%` and `_`.

There are several ways to include quote characters within a string:

- A `'` inside a string quoted with `'` may be written as `''`.
- A `"` inside a string quoted with `"` may be written as `""`.
- Precede the quote character by an escape character (`\`).
- A `'` inside a string quoted with `"` needs no special treatment and need not be doubled or escaped. In the same way, `"` inside a string quoted with `'` needs no special treatment.

The following `SELECT` statements demonstrate how quoting and escaping work:

```
mysql> SELECT 'hello', '"hello"', '""hello""', 'hel''lo', '\'hello';
+-------+---------+-----------+--------+--------+
| hello | "hello" | ""hello"" | hel'lo | 'hello |
+-------+---------+-----------+--------+--------+

mysql> SELECT "hello", "'hello'", "''hello''", "hel""lo", "\"hello";
+-------+---------+-----------+--------+--------+
| hello | 'hello' | ''hello'' | hel"lo | "hello |
+-------+---------+-----------+--------+--------+

mysql> SELECT 'This\nIs\nFour\nLines';
+--------------------+
| This
Is
Four
Lines |
+--------------------+

mysql> SELECT 'disappearing\ backslash';
+------------------------+
| disappearing backslash |
+------------------------+
```

To insert binary data into a string column (such as a `BLOB` column), you should represent certain characters by escape sequences. Backslash (`\`) and the quote character used to quote the string must be escaped. In certain client environments, it may also be necessary to escape `NUL` or Control +Z. The `mysql` client truncates quoted strings containing `NUL` characters if they are not escaped, and Control+Z may be taken for END-OF-FILE on Windows if not escaped. For the escape sequences that represent each of these characters, see Table 9.1, “Special Character Escape Sequences”.

When writing application programs, any string that might contain any of these special characters must be properly escaped before the string is used as a data value in an SQL statement that is sent to the MySQL server. You can do this in two ways:

- Process the string with a function that escapes the special characters. In a C program, you can use the `mysql_real_escape_string()` C API function to escape characters. See mysql_real_escape_string(). Within SQL statements that construct other SQL statements, you can

use the `QUOTE()` function. The Perl DBI interface provides a `quote` method to convert special characters to the proper escape sequences. See Section 23.9, “MySQL Perl API”. Other language interfaces may provide a similar capability.

- As an alternative to explicitly escaping special characters, many MySQL APIs provide a placeholder capability that enables you to insert special markers into a statement string, and then bind data values to them when you issue the statement. In this case, the API takes care of escaping special characters in the values for you.

9.1.2 Numeric Literals

Number literals include exact-value (integer and `DECIMAL`) literals and approximate-value (floating-point) literals.

Integers are represented as a sequence of digits. Numbers may include `.` as a decimal separator. Numbers may be preceded by `-` or `+` to indicate a negative or positive value, respectively. Numbers represented in scientific notation with a mantissa and exponent are approximate-value numbers.

Exact-value numeric literals have an integer part or fractional part, or both. They may be signed. Examples: `1`, `.2`, `3.4`, `-5`, `-6.78`, `+9.10`.

Approximate-value numeric literals are represented in scientific notation with a mantissa and exponent. Either or both parts may be signed. Examples: `1.2E3`, `1.2E-3`, `-1.2E3`, `-1.2E-3`.

Two numbers that look similar may be treated differently. For example, `2.34` is an exact-value (fixed-point) number, whereas `2.34E0` is an approximate-value (floating-point) number.

The `DECIMAL` data type is a fixed-point type and calculations are exact. In MySQL, the `DECIMAL` type has several synonyms: `NUMERIC`, `DEC`, `FIXED`. The integer types also are exact-value types. For more information about exact-value calculations, see Section 12.21, “Precision Math”.

The `FLOAT` and `DOUBLE` data types are floating-point types and calculations are approximate. In MySQL, types that are synonymous with `FLOAT` or `DOUBLE` are `DOUBLE PRECISION` and `REAL`.

An integer may be used in floating-point context; it is interpreted as the equivalent floating-point number.

9.1.3 Date and Time Literals

Date and time values can be represented in several formats, such as quoted strings or as numbers, depending on the exact type of the value and other factors. For example, in contexts where MySQL expects a date, it interprets any of `'2015-07-21'`, `'20150721'`, and `20150721` as a date.

This section describes the acceptable formats for date and time literals. For more information about the temporal data types, such as the range of permitted values, see Section 11.2, “Date and Time Data Types”.

Standard SQL and ODBC Date and Time Literals. Standard SQL requires temporal literals to be specified using a type keyword and a string. The space between the keyword and string is optional.

```
DATE 'str'
TIME 'str'
TIMESTAMP 'str'
```

MySQL recognizes but, unlike standard SQL, does not require the type keyword. Applications that are to be standard-compliant should include the type keyword for temporal literals.

MySQL also recognizes the ODBC syntax corresponding to the standard SQL syntax:

```
{ d 'str' }
{ t 'str' }
```

```
{ ts 'str' }
```

Before MySQL 5.6.4, MySQL ignores the type keyword and each of the preceding constructions produces the string value `'str'`, with a type of `VARCHAR`.

As of 5.6.4, MySQL uses the type keywords and the ODBC constructions to produce `DATE`, `TIME`, and `DATETIME` values, respectively, including a trailing fractional seconds part if specified. The `TIMESTAMP` syntax produces a `DATETIME` value in MySQL because `DATETIME` has a range that more closely corresponds to the standard SQL `TIMESTAMP` type, which has a year range from `0001` to `9999`. (The MySQL `TIMESTAMP` year range is `1970` to `2038`.)

String and Numeric Literals in Date and Time Context. MySQL recognizes `DATE` values in these formats:

- As a string in either `'YYYY-MM-DD'` or `'YY-MM-DD'` format. A “relaxed” syntax is permitted: Any punctuation character may be used as the delimiter between date parts. For example, `'2012-12-31'`, `'2012/12/31'`, `'2012^12^31'`, and `'2012@12@31'` are equivalent.
- As a string with no delimiters in either `'YYYYMMDD'` or `'YYMMDD'` format, provided that the string makes sense as a date. For example, `'20070523'` and `'070523'` are interpreted as `'2007-05-23'`, but `'071332'` is illegal (it has nonsensical month and day parts) and becomes `'0000-00-00'`.
- As a number in either *YYYYMMDD* or *YYMMDD* format, provided that the number makes sense as a date. For example, `19830905` and `830905` are interpreted as `'1983-09-05'`.

MySQL recognizes `DATETIME` and `TIMESTAMP` values in these formats:

- As a string in either `'YYYY-MM-DD hh:mm:ss'` or `'YY-MM-DD hh:mm:ss'` format. A “relaxed” syntax is permitted here, too: Any punctuation character may be used as the delimiter between date parts or time parts. For example, `'2012-12-31 11:30:45'`, `'2012^12^31 11+30+45'`, `'2012/12/31 11*30*45'`, and `'2012@12@31 11^30^45'` are equivalent.

  The only delimiter recognized between a date and time part and a fractional seconds part is the decimal point.

  The date and time parts can be separated by `T` rather than a space. For example, `'2012-12-31 11:30:45'` `'2012-12-31T11:30:45'` are equivalent.
- As a string with no delimiters in either `'YYYYMMDDhhmmss'` or `'YYMMDDhhmmss'` format, provided that the string makes sense as a date. For example, `'20070523091528'` and `'070523091528'` are interpreted as `'2007-05-23 09:15:28'`, but `'071122129015'` is illegal (it has a nonsensical minute part) and becomes `'0000-00-00 00:00:00'`.
- As a number in either *YYYYMMDDhhmmss* or *YYMMDDhhmmss* format, provided that the number makes sense as a date. For example, `19830905132800` and `830905132800` are interpreted as `'1983-09-05 13:28:00'`.

A `DATETIME` or `TIMESTAMP` value can include a trailing fractional seconds part in up to microseconds (6 digits) precision. The fractional part should always be separated from the rest of the time by a decimal point; no other fractional seconds delimiter is recognized. For information about fractional seconds support in MySQL, see Section 11.2.7, “Fractional Seconds in Time Values”.

Dates containing two-digit year values are ambiguous because the century is unknown. MySQL interprets two-digit year values using these rules:

- Year values in the range `70-99` become `1970-1999`.
- Year values in the range `00-69` become `2000-2069`.

See also Section 11.2.9, “2-Digit Years in Dates”.

For values specified as strings that include date part delimiters, it is unnecessary to specify two digits for month or day values that are less than `10`. `'2015-6-9'` is the same as `'2015-06-09'`. Similarly, for values specified as strings that include time part delimiters, it is unnecessary to specify two digits for hour, minute, or second values that are less than `10`. `'2015-10-30 1:2:3'` is the same as `'2015-10-30 01:02:03'`.

Values specified as numbers should be 6, 8, 12, or 14 digits long. If a number is 8 or 14 digits long, it is assumed to be in *`YYYYMMDD`* or *`YYYYMMDDhhmmss`* format and that the year is given by the first 4 digits. If the number is 6 or 12 digits long, it is assumed to be in *`YYMMDD`* or *`YYMMDDhhmmss`* format and that the year is given by the first 2 digits. Numbers that are not one of these lengths are interpreted as though padded with leading zeros to the closest length.

Values specified as nondelimited strings are interpreted according their length. For a string 8 or 14 characters long, the year is assumed to be given by the first 4 characters. Otherwise, the year is assumed to be given by the first 2 characters. The string is interpreted from left to right to find year, month, day, hour, minute, and second values, for as many parts as are present in the string. This means you should not use strings that have fewer than 6 characters. For example, if you specify `'9903'`, thinking that represents March, 1999, MySQL converts it to the “zero” date value. This occurs because the year and month values are `99` and `03`, but the day part is completely missing. However, you can explicitly specify a value of zero to represent missing month or day parts. For example, to insert the value `'1999-03-00'`, use `'990300'`.

MySQL recognizes `TIME` values in these formats:

- As a string in *`'D hh:mm:ss'`* format. You can also use one of the following “relaxed” syntaxes: *`'hh:mm:ss'`*, *`'hh:mm'`*, *`'D hh:mm'`*, *`'D hh'`*, or *`'ss'`*. Here *`D`* represents days and can have a value from 0 to 34.
- As a string with no delimiters in *`'hhmmss'`* format, provided that it makes sense as a time. For example, `'101112'` is understood as `'10:11:12'`, but `'109712'` is illegal (it has a nonsensical minute part) and becomes `'00:00:00'`.
- As a number in *`hhmmss`* format, provided that it makes sense as a time. For example, `101112` is understood as `'10:11:12'`. The following alternative formats are also understood: *`ss`*, *`mmss`*, or *`hhmmss`*.

A trailing fractional seconds part is recognized in the *`'D hh:mm:ss.fraction'`*, *`'hh:mm:ss.fraction'`*, *`'hhmmss.fraction'`*, and *`hhmmss.fraction`* time formats, where `fraction` is the fractional part in up to microseconds (6 digits) precision. The fractional part should always be separated from the rest of the time by a decimal point; no other fractional seconds delimiter is recognized. For information about fractional seconds support in MySQL, see Section 11.2.7, “Fractional Seconds in Time Values”.

For `TIME` values specified as strings that include a time part delimiter, it is unnecessary to specify two digits for hours, minutes, or seconds values that are less than `10`. `'8:3:2'` is the same as `'08:03:02'`.

9.1.4 Hexadecimal Literals

Hexadecimal literal values are written using `X'val'` or `0xval` notation, where *`val`* contains hexadecimal digits (`0..9`, `A..F`). Lettercase of the digits and of any leading `X` does not matter. A leading `0x` is case-sensitive and cannot be written as `0X`.

Legal hexadecimal literals:

```
X'01AF'
X'01af'
x'01AF'
x'01af'
0x01AF
0x01af
```

Illegal hexadecimal literals:

```
X'0G'   (G is not a hexadecimal digit)
0X01AF  (0X must be written as 0x)
```

Values written using `X'val'` notation must contain an even number of digits or a syntax error occurs. To correct the problem, pad the value with a leading zero:

```
mysql> SET @s = X'FFF';
ERROR 1064 (42000): You have an error in your SQL syntax;
check the manual that corresponds to your MySQL server
version for the right syntax to use near 'X'FFF''

mysql> SET @s = X'0FFF';
Query OK, 0 rows affected (0.00 sec)
```

Values written using `0xval` notation that contain an odd number of digits are treated as having an extra leading `0`. For example, `0xaaa` is interpreted as `0x0aaa`.

By default, a hexadecimal literal is a binary string, where each pair of hexadecimal digits represents a character:

```
mysql> SELECT X'4D7953514C', CHARSET(X'4D7953514C');
+---------------+-------------------------+
| X'4D7953514C' | CHARSET(X'4D7953514C') |
+---------------+-------------------------+
| MySQL         | binary                  |
+---------------+-------------------------+
mysql> SELECT 0x5461626c65, CHARSET(0x5461626c65);
+--------------+------------------------+
| 0x5461626c65 | CHARSET(0x5461626c65) |
+--------------+------------------------+
| Table        | binary                 |
+--------------+------------------------+
```

A hexadecimal literal may have an optional character set introducer and `COLLATE` clause, to designate it as a string that uses a particular character set and collation:

```
[_charset_name] X'val' [COLLATE collation_name]
```

Examples:

```
SELECT _latin1 X'4D7953514C';
SELECT _utf8 0x4D7953514C COLLATE utf8_danish_ci;
```

The examples use `X'val'` notation, but `0xval` notation permits introducers as well. For information about introducers, see Section 10.3.8, “Character Set Introducers”.

In numeric contexts, MySQL treats a hexadecimal literal like a `BIGINT` (64-bit integer). To ensure numeric treatment of a hexadecimal literal, use it in numeric context. Ways to do this include adding 0 or using `CAST(... AS UNSIGNED)`. For example, a hexadecimal literal assigned to a user-defined variable is a binary string by default. To assign the value as a number, use it in numeric context:

```
mysql> SET @v1 = X'41';
mysql> SET @v2 = X'41'+0;
mysql> SET @v3 = CAST(X'41' AS UNSIGNED);
mysql> SELECT @v1, @v2, @v3;
+------+------+------+
| @v1  | @v2  | @v3  |
+------+------+------+
| A    |   65 |   65 |
+------+------+------+
```

An empty hexadecimal value (`X''`) evaluates to a zero-length binary string. Converted to a number, it produces 0:

```
mysql> SELECT CHARSET(X''), LENGTH(X'');
```

```
+--------------+-------------+
| CHARSET(X'') | LENGTH(X'') |
+--------------+-------------+
| binary       |           0 |
+--------------+-------------+
mysql> SELECT X''+0;
+-------+
| X''+0 |
+-------+
|     0 |
+-------+
```

The `X'val'` notation is based on standard SQL. The `0x` notation is based on ODBC, for which hexadecimal strings are often used to supply values for `BLOB` columns.

To convert a string or a number to a string in hexadecimal format, use the `HEX()` function:

```
mysql> SELECT HEX('cat');
+------------+
| HEX('cat') |
+------------+
| 636174     |
+------------+
mysql> SELECT X'636174';
+-----------+
| X'636174' |
+-----------+
| cat       |
+-----------+
```

9.1.5 Bit-Value Literals

Bit-value literals are written using `b'val'` or `0bval` notation. *`val`* is a binary value written using zeros and ones. Lettercase of any leading `b` does not matter. A leading `0b` is case-sensitive and cannot be written as `0B`.

Legal bit-value literals:

```
b'01'
B'01'
0b01
```

Illegal bit-value literals:

```
b'2'    (2 is not a binary digit)
0B01    (0B must be written as 0b)
```

By default, a bit-value literal is a binary string:

```
mysql> SELECT b'1000001', CHARSET(b'1000001');
+------------+----------------------+
| b'1000001' | CHARSET(b'1000001') |
+------------+----------------------+
| A          | binary               |
+------------+----------------------+
mysql> SELECT 0b1100001, CHARSET(0b1100001);
+-----------+--------------------+
| 0b1100001 | CHARSET(0b1100001) |
+-----------+--------------------+
| a         | binary             |
+-----------+--------------------+
```

A bit-value literal may have an optional character set introducer and `COLLATE` clause, to designate it as a string that uses a particular character set and collation:

```
[_charset_name] b'val' [COLLATE collation_name]
```

Examples:

```
SELECT _latin1 b'1000001';
SELECT _utf8 0b1000001 COLLATE utf8_danish_ci;
```

The examples use `b'val'` notation, but `0bval` notation permits introducers as well. For information about introducers, see Section 10.3.8, “Character Set Introducers”.

In numeric contexts, MySQL treats a bit literal like an integer. To ensure numeric treatment of a bit literal, use it in numeric context. Ways to do this include adding 0 or using `CAST(... AS UNSIGNED)`. For example, a bit literal assigned to a user-defined variable is a binary string by default. To assign the value as a number, use it in numeric context:

```
mysql> SET @v1 = b'1100001';
mysql> SET @v2 = b'1100001'+0;
mysql> SET @v3 = CAST(b'1100001' AS UNSIGNED);
mysql> SELECT @v1, @v2, @v3;
+------+------+------+
| @v1  | @v2  | @v3  |
+------+------+------+
| a    |   97 |   97 |
+------+------+------+
```

An empty bit value (`b''`) evaluates to a zero-length binary string. Converted to a number, it produces 0:

```
mysql> SELECT CHARSET(b''), LENGTH(b'');
+--------------+-------------+
| CHARSET(b'') | LENGTH(b'') |
+--------------+-------------+
| binary       |           0 |
+--------------+-------------+
mysql> SELECT b''+0;
+-------+
| b''+0 |
+-------+
|     0 |
+-------+
```

Bit-value notation is convenient for specifying values to be assigned to `BIT` columns:

```
mysql> CREATE TABLE t (b BIT(8));
mysql> INSERT INTO t SET b = b'11111111';
mysql> INSERT INTO t SET b = b'1010';
mysql> INSERT INTO t SET b = b'0101';
```

Bit values in result sets are returned as binary values, which may not display well. To convert a bit value to printable form, use it in numeric context or use a conversion function such as `BIN()` or `HEX()`. High-order 0 digits are not displayed in the converted value.

```
mysql> SELECT b+0, BIN(b), OCT(b), HEX(b) FROM t;
+------+----------+--------+--------+
| b+0  | BIN(b)   | OCT(b) | HEX(b) |
+------+----------+--------+--------+
|  255 | 11111111 | 377    | FF     |
|   10 | 1010     | 12     | A      |
|    5 | 101      | 5      | 5      |
+------+----------+--------+--------+
```

9.1.6 Boolean Literals

The constants `TRUE` and `FALSE` evaluate to `1` and `0`, respectively. The constant names can be written in any lettercase.

```
mysql> SELECT TRUE, true, FALSE, false;
        -> 1, 1, 0, 0
```

9.1.7 NULL Values

The `NULL` value means “no data.” `NULL` can be written in any lettercase. A synonym is `\N` (case-sensitive).

Be aware that the `NULL` value is different from values such as `0` for numeric types or the empty string for string types. For more information, see Section B.3.4.3, “Problems with NULL Values”.

For text file import or export operations performed with `LOAD DATA` or `SELECT ... INTO OUTFILE`, `NULL` is represented by the `\N` sequence. See Section 13.2.6, “LOAD DATA Statement”.

For sorting with `ORDER BY`, `NULL` values sort before other values for ascending sorts, after other values for descending sorts.

9.2 Schema Object Names

Certain objects within MySQL, including database, table, index, column, alias, view, stored procedure, partition, tablespace, and other object names are known as identifiers. This section describes the permissible syntax for identifiers in MySQL. Section 9.2.1, “Identifier Length Limits”, indicates the maximum length of each type of identifier. Section 9.2.3, “Identifier Case Sensitivity”, describes which types of identifiers are case-sensitive and under what conditions.

An identifier may be quoted or unquoted. If an identifier contains special characters or is a reserved word, you *must* quote it whenever you refer to it. (Exception: A reserved word that follows a period in a qualified name must be an identifier, so it need not be quoted.) Reserved words are listed at Section 9.3, “Keywords and Reserved Words”.

Internally, identifiers are converted to and are stored as Unicode (UTF-8). The permissible Unicode characters in identifiers are those in the Basic Multilingual Plane (BMP). Supplementary characters are not permitted. Identifiers thus may contain these characters:

- Permitted characters in unquoted identifiers:
  - ASCII: [0-9,a-z,A-Z$_] (basic Latin letters, digits 0-9, dollar, underscore)
  - Extended: U+0080 .. U+FFFF
- Permitted characters in quoted identifiers include the full Unicode Basic Multilingual Plane (BMP), except U+0000:
  - ASCII: U+0001 .. U+007F
  - Extended: U+0080 .. U+FFFF
- ASCII NUL (U+0000) and supplementary characters (U+10000 and higher) are not permitted in quoted or unquoted identifiers.
- Identifiers may begin with a digit but unless quoted may not consist solely of digits.
- Database, table, and column names cannot end with space characters.

The identifier quote character is the backtick (`):

```
mysql> SELECT * FROM `select` WHERE `select`.id > 100;
```

If the `ANSI_QUOTES` SQL mode is enabled, it is also permissible to quote identifiers within double quotation marks:

```
mysql> CREATE TABLE "test" (col INT);
ERROR 1064: You have an error in your SQL syntax...
mysql> SET sql_mode='ANSI_QUOTES';
mysql> CREATE TABLE "test" (col INT);
```

```
Query OK, 0 rows affected (0.00 sec)
```

The `ANSI_QUOTES` mode causes the server to interpret double-quoted strings as identifiers. Consequently, when this mode is enabled, string literals must be enclosed within single quotation marks. They cannot be enclosed within double quotation marks. The server SQL mode is controlled as described in Section 5.1.10, “Server SQL Modes”.

Identifier quote characters can be included within an identifier if you quote the identifier. If the character to be included within the identifier is the same as that used to quote the identifier itself, then you need to double the character. The following statement creates a table named `a``b` that contains a column named `c"d`:

```
mysql> CREATE TABLE `a``b` (`c"d` INT);
```

In the select list of a query, a quoted column alias can be specified using identifier or string quoting characters:

```
mysql> SELECT 1 AS `one`, 2 AS 'two';
+-----+-----+
| one | two |
+-----+-----+
|   1 |   2 |
+-----+-----+
```

Elsewhere in the statement, quoted references to the alias must use identifier quoting or the reference is treated as a string literal.

It is recommended that you do not use names that begin with *`Me`* or *`MeN`*, where *`M`* and *`N`* are integers. For example, avoid using `1e` as an identifier, because an expression such as `1e+3` is ambiguous. Depending on context, it might be interpreted as the expression `1e + 3` or as the number `1e+3`.

Be careful when using `MD5()` to produce table names because it can produce names in illegal or ambiguous formats such as those just described.

A user variable cannot be used directly in an SQL statement as an identifier or as part of an identifier. See Section 9.4, “User-Defined Variables”, for more information and examples of workarounds.

Special characters in database and table names are encoded in the corresponding file system names as described in Section 9.2.4, “Mapping of Identifiers to File Names”. If you have databases or tables from an older version of MySQL that contain special characters and for which the underlying directory names or file names have not been updated to use the new encoding, the server displays their names with a prefix of `#mysql50#`. For information about referring to such names or converting them to the newer encoding, see that section.

9.2.1 Identifier Length Limits

The following table describes the maximum length for each type of identifier.

Identifier Type	Maximum Length (characters)
Database	64 (`NDB` storage engine: 63)
Table	64 (`NDB` storage engine: 63)
Column	64

Identifier Type	Maximum Length (characters)
Index	64
Constraint	64
Stored Program	64
View	64
Tablespace	64
Server	64
Log File Group	64
Alias	256 (see exception following table)
Compound Statement Label	16

Aliases for column names in `CREATE VIEW` statements are checked against the maximum column length of 64 characters (not the maximum alias length of 256 characters).

For constraint definitions that include no constraint name, the server internally generates a name derived from the associated table name. For example, internally generated foreign key constraint names consist of the table name plus `_ibfk_` and a number. If the table name is close to the length limit for constraint names, the additional characters required for the constraint name may cause that name to exceed the limit, resulting in an error.

Identifiers are stored using Unicode (UTF-8). This applies to identifiers in table definitions that are stored in `.frm` files and to identifiers stored in the grant tables in the `mysql` database. The sizes of the identifier string columns in the grant tables are measured in characters. You can use multibyte characters without reducing the number of characters permitted for values stored in these columns.

NDB Cluster imposes a maximum length of 63 characters for names of databases and tables. See Section 18.1.7.5, “Limits Associated with Database Objects in NDB Cluster”.

Values such as user name and host names in MySQL account names are strings rather than identifiers. For information about the maximum length of such values as stored in grant tables, see Grant Table Scope Column Properties.

9.2.2 Identifier Qualifiers

Object names may be unqualified or qualified. An unqualified name is permitted in contexts where interpretation of the name is unambiguous. A qualified name includes at least one qualifier to clarify the interpretive context by overriding a default context or providing missing context.

For example, this statement creates a table using the unqualified name `t1`:

```
CREATE TABLE t1 (i INT);
```

Because `t1` includes no qualifier to specify a database, the statement creates the table in the default database. If there is no default database, an error occurs.

This statement creates a table using the qualified name `db1.t1`:

```
CREATE TABLE db1.t1 (i INT);
```

Because `db1.t1` includes a database qualifier `db1`, the statement creates `t1` in the database named `db1`, regardless of the default database. The qualifier *must* be specified if there is no default database. The qualifier *may* be specified if there is a default database, to specify a database different from the default, or to make the database explicit if the default is the same as the one specified.

Qualifiers have these characteristics:

- An unqualified name consists of a single identifier. A qualified name consists of multiple identifiers.
- The components of a multiple-part name must be separated by period (`.`) characters. The initial parts of a multiple-part name act as qualifiers that affect the context within which to interpret the final identifier.
- The qualifier character is a separate token and need not be contiguous with the associated identifiers. For example, *`tbl_name.col_name`* and *`tbl_name . col_name`* are equivalent.
- If any components of a multiple-part name require quoting, quote them individually rather than quoting the name as a whole. For example, write `` `my-table`.`my-column` ``, not `` `my-table.my-column` ``.
- A reserved word that follows a period in a qualified name must be an identifier, so in that context it need not be quoted.
- The syntax *`.tbl_name`* means the table *`tbl_name`* in the default database. This syntax is accepted for ODBC compatibility because some ODBC programs prefix table names with a `.` character.

The permitted qualifiers for object names depend on the object type:

- A database name is fully qualified and takes no qualifier:

  ```
  CREATE DATABASE db1;
  ```

- A table, view, or stored program name may be given a database-name qualifier. Examples of unqualified and qualified names in `CREATE` statements:

  ```
  CREATE TABLE mytable ...;
  CREATE VIEW myview ...;
  CREATE PROCEDURE myproc ...;
  CREATE FUNCTION myfunc ...;
  CREATE EVENT myevent ...;

  CREATE TABLE mydb.mytable ...;
  CREATE VIEW mydb.myview ...;
  CREATE PROCEDURE mydb.myproc ...;
  CREATE FUNCTION mydb.myfunc ...;
  CREATE EVENT mydb.myevent ...;
  ```

- A trigger is associated with a table, so any qualifier applies to the table name:

  ```
  CREATE TRIGGER mytrigger ... ON mytable ...;

  CREATE TRIGGER mytrigger ... ON mydb.mytable ...;
  ```

- A column name may be given multiple qualifiers to indicate context in statements that reference it, as shown in the following table.

Column Reference	Meaning
`col_name`	Column *`col_name`* from whichever table used in the statement contains a column of that name
`tbl_name.col_name`	Column *`col_name`* from table *`tbl_name`* of the default database

Column Reference	Meaning
db_name.tbl_name.col_name	Column *col_name* from table *tbl_name* of the database *db_name*

In other words, a column name may be given a table-name qualifier, which itself may be given a database-name qualifier. Examples of unqualified and qualified column references in SELECT statements:

```
SELECT c1 FROM mytable
WHERE c2 > 100;

SELECT mytable.c1 FROM mytable
WHERE mytable.c2 > 100;

SELECT mydb.mytable.c1 FROM mydb.mytable
WHERE mydb.mytable.c2 > 100;
```

You need not specify a qualifier for an object reference in a statement unless the unqualified reference is ambiguous. Suppose that column c1 occurs only in table t1, c2 only in t2, and c in both t1 and t2. Any unqualified reference to c is ambiguous in a statement that refers to both tables and must be qualified as t1.c or t2.c to indicate which table you mean:

```
SELECT c1, c2, t1.c FROM t1 INNER JOIN t2
WHERE t2.c > 100;
```

Similarly, to retrieve from a table t in database db1 and from a table t in database db2 in the same statement, you must qualify the table references: For references to columns in those tables, qualifiers are required only for column names that appear in both tables. Suppose that column c1 occurs only in table db1.t, c2 only in db2.t, and c in both db1.t and db2.t. In this case, c is ambiguous and must be qualified but c1 and c2 need not be:

```
SELECT c1, c2, db1.t.c FROM db1.t INNER JOIN db2.t
WHERE db2.t.c > 100;
```

Table aliases enable qualified column references to be written more simply:

```
SELECT c1, c2, t1.c FROM db1.t AS t1 INNER JOIN db2.t AS t2
WHERE t2.c > 100;
```

9.2.3 Identifier Case Sensitivity

In MySQL, databases correspond to directories within the data directory. Each table within a database corresponds to at least one file within the database directory (and possibly more, depending on the storage engine). Triggers also correspond to files. Consequently, the case sensitivity of the underlying operating system plays a part in the case sensitivity of database, table, and trigger names. This means such names are not case-sensitive in Windows, but are case-sensitive in most varieties of Unix. One notable exception is macOS, which is Unix-based but uses a default file system type (HFS+) that is not case-sensitive. However, macOS also supports UFS volumes, which are case-sensitive just as on any Unix. See Section 1.7.1, “MySQL Extensions to Standard SQL”. The lower_case_table_names system variable also affects how the server handles identifier case sensitivity, as described later in this section.

Note

Although database, table, and trigger names are not case-sensitive on some platforms, you should not refer to one of these using different cases within the same statement. The following statement would not work because it refers to a table both as my_table and as MY_TABLE:

```
mysql> SELECT * FROM my_table WHERE MY_TABLE.col=1;
```

Column, index, stored routine, and event names are not case-sensitive on any platform, nor are column aliases.

However, names of logfile groups are case-sensitive. This differs from standard SQL.

By default, table aliases are case-sensitive on Unix, but not so on Windows or macOS. The following statement would not work on Unix, because it refers to the alias both as `a` and as `A`:

```
mysql> SELECT col_name FROM tbl_name AS a
       WHERE a.col_name = 1 OR A.col_name = 2;
```

However, this same statement is permitted on Windows. To avoid problems caused by such differences, it is best to adopt a consistent convention, such as always creating and referring to databases and tables using lowercase names. This convention is recommended for maximum portability and ease of use.

How table and database names are stored on disk and used in MySQL is affected by the `lower_case_table_names` system variable, which you can set when starting `mysqld`. `lower_case_table_names` can take the values shown in the following table. This variable does *not* affect case sensitivity of trigger identifiers. On Unix, the default value of `lower_case_table_names` is 0. On Windows, the default value is 1. On macOS, the default value is 2.

Value	Meaning
`0`	Table and database names are stored on disk using the lettercase specified in the `CREATE TABLE` or `CREATE DATABASE` statement. Name comparisons are case-sensitive. You should *not* set this variable to 0 if you are running MySQL on a system that has case-insensitive file names (such as Windows or macOS). If you force this variable to 0 with `--lower-case-table-names=0` on a case-insensitive file system and access `MyISAM` tablenames using different lettercases, index corruption may result.
`1`	Table names are stored in lowercase on disk and name comparisons are not case-sensitive. MySQL converts all table names to lowercase on storage and lookup. This behavior also applies to database names and table aliases.
`2`	Table and database names are stored on disk using the lettercase specified in the `CREATE TABLE` or `CREATE DATABASE` statement, but MySQL converts them to lowercase on lookup. Name comparisons are not case-sensitive. This works *only* on file systems that are not case-sensitive! `InnoDB` table names are stored in lowercase, as for `lower_case_table_names=1`.

If you are using MySQL on only one platform, you do not normally have to change the `lower_case_table_names` variable from its default value. However, you may encounter difficulties if you want to transfer tables between platforms that differ in file system case sensitivity. For example, on Unix, you can have two different tables named `my_table` and `MY_TABLE`, but on Windows these two names are considered identical. To avoid data transfer problems arising from lettercase of database or table names, you have two options:

- Use `lower_case_table_names=1` on all systems. The main disadvantage with this is that when you use `SHOW TABLES` or `SHOW DATABASES`, you do not see the names in their original lettercase.
- Use `lower_case_table_names=0` on Unix and `lower_case_table_names=2` on Windows. This preserves the lettercase of database and table names. The disadvantage of this is that you must ensure that your statements always refer to your database and table names with the correct lettercase on Windows. If you transfer your statements to Unix, where lettercase is significant, they do not work if the lettercase is incorrect.

  Exception: If you are using `InnoDB` tables and you are trying to avoid these data transfer problems, you should set `lower_case_table_names` to 1 on all platforms to force names to be converted to lowercase.

If you plan to set the `lower_case_table_names` system variable to 1 on Unix, you must first convert your old database and table names to lowercase before stopping `mysqld` and restarting it with the new variable setting. To do this for an individual table, use `RENAME TABLE`:

```
RENAME TABLE T1 TO t1;
```

To convert one or more entire databases, dump them before setting `lower_case_table_names`, then drop the databases, and reload them after setting `lower_case_table_names`:

1. Use `mysqldump` to dump each database:

   ```
   mysqldump --databases db1 > db1.sql
   mysqldump --databases db2 > db2.sql
   ...
   ```

   Do this for each database that must be recreated.

2. Use `DROP DATABASE` to drop each database.

3. Stop the server, set `lower_case_table_names`, and restart the server.

4. Reload the dump file for each database. Because `lower_case_table_names` is set, each database and table name is converted to lowercase as it is recreated:

   ```
   mysql < db1.sql
   mysql < db2.sql
   ...
   ```

Object names may be considered duplicates if their uppercase forms are equal according to a binary collation. That is true for names of cursors, conditions, procedures, functions, savepoints, stored routine parameters, stored program local variables, and plugins. It is not true for names of columns, constraints, databases, partitions, statements prepared with `PREPARE`, tables, triggers, users, and user-defined variables.

File system case sensitivity can affect searches in string columns of `INFORMATION_SCHEMA` tables. For more information, see Section 10.8.7, “Using Collation in INFORMATION_SCHEMA Searches”.

9.2.4 Mapping of Identifiers to File Names

There is a correspondence between database and table identifiers and names in the file system. For the basic structure, MySQL represents each database as a directory in the data directory, and each table by one or more files in the appropriate database directory. For the table format files (`.FRM`), the data is always stored in this structure and location.

For the data and index files, the exact representation on disk is storage engine specific. These files may be stored in the same location as the `FRM` files, or the information may be stored in a separate file. `InnoDB` data is stored in the InnoDB data files. If you are using tablespaces with `InnoDB`, then the specific tablespace files you create are used instead.

Any character is legal in database or table identifiers except ASCII NUL (`X'00'`). MySQL encodes any characters that are problematic in the corresponding file system objects when it creates database directories or table files:

- Basic Latin letters (`a..zA..Z`), digits (`0..9`) and underscore (_) are encoded as is. Consequently, their case sensitivity directly depends on file system features.
- All other national letters from alphabets that have uppercase/lowercase mapping are encoded as shown in the following table. Values in the Code Range column are UCS-2 values.

Code Range	Pattern	Number	Used	Unused	Blocks
00C0..017F	[@][0..4][g..z]	5*20= 100	97	3	Latin-1 Supplement + Latin Extended-A
0370..03FF	[@][5..9][g..z]	5*20= 100	88	12	Greek and Coptic
0400..052F	[@][g..z][0..6]	20*7= 140	137	3	Cyrillic + Cyrillic Supplement

Code Range	Pattern	Number	Used	Unused	Blocks
0530..058F	[@][g..z][7..8]	20*2= 40	38	2	Armenian
2160..217F	[@][g..z][9]	20*1= 20	16	4	Number Forms
0180..02AF	[@][g..z][a..k]	20*11=220	203	17	Latin Extended-B + IPA Extensions
1E00..1EFF	[@][g..z][l..r]	20*7= 140	136	4	Latin Extended Additional
1F00..1FFF	[@][g..z][s..z]	20*8= 160	144	16	Greek Extended
....	[@][a..f][g..z]	6*20= 120	0	120	RESERVED
24B6..24E9	[@][@][a..z]	26	26	0	Enclosed Alphanumerics
FF21..FF5A	[@][a..z][@]	26	26	0	Halfwidth and Fullwidth forms

One of the bytes in the sequence encodes lettercase. For example: `LATIN CAPITAL LETTER A WITH GRAVE` is encoded as `@0G`, whereas `LATIN SMALL LETTER A WITH GRAVE` is encoded as `@0g`. Here the third byte (`G` or `g`) indicates lettercase. (On a case-insensitive file system, both letters are treated as the same.)

For some blocks, such as Cyrillic, the second byte determines lettercase. For other blocks, such as Latin1 Supplement, the third byte determines lettercase. If two bytes in the sequence are letters (as in Greek Extended), the leftmost letter character stands for lettercase. All other letter bytes must be in lowercase.

- All nonletter characters except underscore (_), as well as letters from alphabets that do not have uppercase/lowercase mapping (such as Hebrew) are encoded using hexadecimal representation using lowercase letters for hexadecimal digits `a..f`:

```
0x003F -> @003f
0xFFFF -> @ffff
```

The hexadecimal values correspond to character values in the `ucs2` double-byte character set.

On Windows, some names such as `nul`, `prn`, and `aux` are encoded by appending `@@@` to the name when the server creates the corresponding file or directory. This occurs on all platforms for portability of the corresponding database object between platforms.

If you have databases or tables from a version of MySQL older than 5.1.6 that contain special characters and for which the underlying directory names or file names have not been updated to use the new encoding, the server displays their names with a prefix of `#mysql50#` in the output from `INFORMATION_SCHEMA` tables or `SHOW` statements. For example, if you have a table named `a@b` and its name encoding has not been updated, `SHOW TABLES` displays it like this:

```
mysql> SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| #mysql50#a@b   |
+----------------+
```

To refer to such a name for which the encoding has not been updated, you must supply the `#mysql50#` prefix:

```
mysql> SHOW COLUMNS FROM `a@b`;
ERROR 1146 (42S02): Table 'test.a@b' doesn't exist

mysql> SHOW COLUMNS FROM `#mysql50#a@b`;
+-------+---------+------+-----+---------+-------+
| Field | Type    | Null | Key | Default | Extra |
```

```
+-------+---------+------+-----+---------+-------+
| i     | int(11) | YES  |     | NULL    |       |
+-------+---------+------+-----+---------+-------+
```

To update old names to eliminate the need to use the special prefix to refer to them, re-encode them with `mysqlcheck`. The following commands update all names to the new encoding:

```
mysqlcheck --check-upgrade --all-databases
mysqlcheck --fix-db-names --fix-table-names --all-databases
```

To check only specific databases or tables, omit `--all-databases` and provide the appropriate database or table arguments. For information about `mysqlcheck` invocation syntax, see Section 4.5.3, “mysqlcheck — A Table Maintenance Program”.

Note

The `#mysql50#` prefix is intended only to be used internally by the server. You should not create databases or tables with names that use this prefix.

Also, `mysqlcheck` cannot fix names that contain literal instances of the `@` character that is used for encoding special characters. If you have databases or tables that contain this character, use `mysqldump` to dump them before upgrading to MySQL 5.1.6 or later, and then reload the dump file after upgrading.

9.2.5 Function Name Parsing and Resolution

MySQL supports built-in (native) functions, user-defined functions (UDFs), and stored functions. This section describes how the server recognizes whether the name of a built-in function is used as a function call or as an identifier, and how the server determines which function to use in cases when functions of different types exist with a given name.

- Built-In Function Name Parsing
- Function Name Resolution

Built-In Function Name Parsing

The parser uses default rules for parsing names of built-in functions. These rules can be changed by enabling the `IGNORE_SPACE` SQL mode.

When the parser encounters a word that is the name of a built-in function, it must determine whether the name signifies a function call or is instead a nonexpression reference to an identifier such as a table or column name. For example, in the following statements, the first reference to `count` is a function call, whereas the second reference is a table name:

```
SELECT COUNT(*) FROM mytable;
CREATE TABLE count (i INT);
```

The parser should recognize the name of a built-in function as indicating a function call only when parsing what is expected to be an expression. That is, in nonexpression context, function names are permitted as identifiers.

However, some built-in functions have special parsing or implementation considerations, so the parser uses the following rules by default to distinguish whether their names are being used as function calls or as identifiers in nonexpression context:

- To use the name as a function call in an expression, there must be no whitespace between the name and the following `(` parenthesis character.
- Conversely, to use the function name as an identifier, it must not be followed immediately by a parenthesis.

The requirement that function calls be written with no whitespace between the name and the parenthesis applies only to the built-in functions that have special considerations. `COUNT` is one such name. The `sql_functions[]` array in the `sql/lex.h` source file lists the names of these special functions for which following whitespace determines their interpretation.

The following list names the functions in MySQL 5.6 that are affected by the `IGNORE_SPACE` setting and listed as special in the `sql/lex.h` source file. You may find it easiest to treat the no-whitespace requirement as applying to all function calls.

- `ADDDATE`
- `BIT_AND`
- `BIT_OR`
- `BIT_XOR`
- `CAST`
- `COUNT`
- `CURDATE`
- `CURTIME`
- `DATE_ADD`
- `DATE_SUB`
- `EXTRACT`
- `GROUP_CONCAT`
- `MAX`
- `MID`
- `MIN`
- `NOW`
- `POSITION`
- `SESSION_USER`
- `STD`
- `STDDEV`
- `STDDEV_POP`
- `STDDEV_SAMP`
- `SUBDATE`
- `SUBSTR`
- `SUBSTRING`
- `SUM`
- `SYSDATE`
- `SYSTEM_USER`

- TRIM
- VARIANCE
- VAR_POP
- VAR_SAMP

For functions not listed as special in sql/lex.h, whitespace does not matter. They are interpreted as function calls only when used in expression context and may be used freely as identifiers otherwise. ASCII is one such name. However, for these nonaffected function names, interpretation may vary in expression context: *func_name* () is interpreted as a built-in function if there is one with the given name; if not, *func_name* () is interpreted as a user-defined function or stored function if one exists with that name.

The IGNORE_SPACE SQL mode can be used to modify how the parser treats function names that are whitespace-sensitive:

- With IGNORE_SPACE disabled, the parser interprets the name as a function call when there is no whitespace between the name and the following parenthesis. This occurs even when the function name is used in nonexpression context:

  ```
  mysql> CREATE TABLE count(i INT);
  ERROR 1064 (42000): You have an error in your SQL syntax ...
  near 'count(i INT)'
  ```

  To eliminate the error and cause the name to be treated as an identifier, either use whitespace following the name or write it as a quoted identifier (or both):

  ```
  CREATE TABLE count (i INT);
  CREATE TABLE `count`(i INT);
  CREATE TABLE `count` (i INT);
  ```

- With IGNORE_SPACE enabled, the parser loosens the requirement that there be no whitespace between the function name and the following parenthesis. This provides more flexibility in writing function calls. For example, either of the following function calls are legal:

  ```
  SELECT COUNT(*) FROM mytable;
  SELECT COUNT (*) FROM mytable;
  ```

  However, enabling IGNORE_SPACE also has the side effect that the parser treats the affected function names as reserved words (see Section 9.3, “Keywords and Reserved Words”). This means that a space following the name no longer signifies its use as an identifier. The name can be used in function calls with or without following whitespace, but causes a syntax error in nonexpression context unless it is quoted. For example, with IGNORE_SPACE enabled, both of the following statements fail with a syntax error because the parser interprets count as a reserved word:

  ```
  CREATE TABLE count(i INT);
  CREATE TABLE count (i INT);
  ```

  To use the function name in nonexpression context, write it as a quoted identifier:

  ```
  CREATE TABLE `count`(i INT);
  CREATE TABLE `count` (i INT);
  ```

To enable the IGNORE_SPACE SQL mode, use this statement:

```
SET sql_mode = 'IGNORE_SPACE';
```

IGNORE_SPACE is also enabled by certain other composite modes such as ANSI that include it in their value:

```
SET sql_mode = 'ANSI';
```

Check Section 5.1.10, “Server SQL Modes”, to see which composite modes enable IGNORE_SPACE.

To minimize the dependency of SQL code on the `IGNORE_SPACE` setting, use these guidelines:

- Avoid creating UDFs or stored functions that have the same name as a built-in function.
- Avoid using function names in nonexpression context. For example, these statements use `count` (one of the affected function names affected by `IGNORE_SPACE`), so they fail with or without whitespace following the name if `IGNORE_SPACE` is enabled:

```
CREATE TABLE count(i INT);
CREATE TABLE count (i INT);
```

If you must use a function name in nonexpression context, write it as a quoted identifier:

```
CREATE TABLE `count`(i INT);
CREATE TABLE `count` (i INT);
```

Function Name Resolution

The following rules describe how the server resolves references to function names for function creation and invocation:

- Built-in functions and user-defined functions

  An error occurs if you try to create a UDF with the same name as a built-in function.

- Built-in functions and stored functions

  It is possible to create a stored function with the same name as a built-in function, but to invoke the stored function it is necessary to qualify it with a schema name. For example, if you create a stored function named `PI` in the `test` schema, invoke it as `test.PI()` because the server resolves `PI()` without a qualifier as a reference to the built-in function. The server generates a warning if the stored function name collides with a built-in function name. The warning can be displayed with `SHOW WARNINGS`.

- User-defined functions and stored functions

  User-defined functions and stored functions share the same namespace, so you cannot create a UDF and a stored function with the same name.

The preceding function name resolution rules have implications for upgrading to versions of MySQL that implement new built-in functions:

- If you have already created a user-defined function with a given name and upgrade MySQL to a version that implements a new built-in function with the same name, the UDF becomes inaccessible. To correct this, use `DROP FUNCTION` to drop the UDF and `CREATE FUNCTION` to re-create the UDF with a different nonconflicting name. Then modify any affected code to use the new name.
- If a new version of MySQL implements a built-in function with the same name as an existing stored function, you have two choices: Rename the stored function to use a nonconflicting name, or change calls to the function so that they use a schema qualifier (that is, use *`schema_name.func_name()`* syntax). In either case, modify any affected code accordingly.

9.3 Keywords and Reserved Words

Keywords are words that have significance in SQL. Certain keywords, such as `SELECT`, `DELETE`, or `BIGINT`, are reserved and require special treatment for use as identifiers such as table and column names. This may also be true for the names of built-in functions.

Nonreserved keywords are permitted as identifiers without quoting. Reserved words are permitted as identifiers if you quote them as described in Section 9.2, “Schema Object Names”:

```
mysql> CREATE TABLE interval (begin INT, end INT);
```

```
ERROR 1064 (42000): You have an error in your SQL syntax ...
near 'interval (begin INT, end INT)'
```

BEGIN and END are keywords but not reserved, so their use as identifiers does not require quoting. INTERVAL is a reserved keyword and must be quoted to be used as an identifier:

```
mysql> CREATE TABLE `interval` (begin INT, end INT);
Query OK, 0 rows affected (0.01 sec)
```

Exception: A word that follows a period in a qualified name must be an identifier, so it need not be quoted even if it is reserved:

```
mysql> CREATE TABLE mydb.interval (begin INT, end INT);
Query OK, 0 rows affected (0.01 sec)
```

Names of built-in functions are permitted as identifiers but may require care to be used as such. For example, COUNT is acceptable as a column name. However, by default, no whitespace is permitted in function invocations between the function name and the following (character. This requirement enables the parser to distinguish whether the name is used in a function call or in nonfunction context. For further details on recognition of function names, see Section 9.2.5, “Function Name Parsing and Resolution”.

- MySQL 5.6 Keywords and Reserved Words
- MySQL 5.6 New Keywords and Reserved Words
- MySQL 5.6 Removed Keywords and Reserved Words

MySQL 5.6 Keywords and Reserved Words

The following list shows the keywords and reserved words in MySQL 5.6, along with changes to individual words from version to version. Reserved keywords are marked with (R). In addition, _FILENAME is reserved.

At some point, you might upgrade to a higher version, so it is a good idea to have a look at future reserved words, too. You can find these in the manuals that cover higher versions of MySQL. Most of the reserved words in the list are forbidden by standard SQL as column or table names (for example, GROUP). A few are reserved because MySQL needs them and uses a yacc parser.

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

A

- ACCESSIBLE (R)
- ACTION
- ADD (R)
- AFTER
- AGAINST
- AGGREGATE
- ALGORITHM
- ALL (R)
- ALTER (R)
- ANALYSE; added in 5.6.6 (nonreserved)
- ANALYZE (R)

- AND (R)
- ANY
- AS (R)
- ASC (R)
- ASCII
- ASENSITIVE (R)
- AT
- AUTHORS; removed in 5.6.8
- AUTOEXTEND_SIZE
- AUTO_INCREMENT
- AVG
- AVG_ROW_LENGTH

B

- BACKUP
- BEFORE (R)
- BEGIN
- BETWEEN (R)
- BIGINT (R)
- BINARY (R)
- BINLOG
- BIT
- BLOB (R)
- BLOCK
- BOOL
- BOOLEAN
- BOTH (R)
- BTREE
- BY (R)
- BYTE

C

- CACHE
- CALL (R)
- CASCADE (R)

- CASCADED
- CASE (R)
- CATALOG_NAME
- CHAIN
- CHANGE (R)
- CHANGED
- CHAR (R)
- CHARACTER (R)
- CHARSET
- CHECK (R)
- CHECKSUM
- CIPHER
- CLASS_ORIGIN
- CLIENT
- CLOSE
- COALESCE
- CODE
- COLLATE (R)
- COLLATION
- COLUMN (R)
- COLUMNS
- COLUMN_FORMAT; added in 5.6.6 (nonreserved)
- COLUMN_NAME
- COMMENT
- COMMIT
- COMMITTED
- COMPACT
- COMPLETION
- COMPRESSED
- CONCURRENT
- CONDITION (R)
- CONNECTION
- CONSISTENT

- CONSTRAINT (R)
- CONSTRAINT_CATALOG
- CONSTRAINT_NAME
- CONSTRAINT_SCHEMA
- CONTAINS
- CONTEXT
- CONTINUE (R)
- CONTRIBUTORS; removed in 5.6.8
- CONVERT (R)
- CPU
- CREATE (R)
- CROSS (R)
- CUBE
- CURRENT; added in 5.6.4 (nonreserved)
- CURRENT_DATE (R)
- CURRENT_TIME (R)
- CURRENT_TIMESTAMP (R)
- CURRENT_USER (R)
- CURSOR (R)
- CURSOR_NAME

D

- DATA
- DATABASE (R)
- DATABASES (R)
- DATAFILE
- DATE
- DATETIME
- DAY
- DAY_HOUR (R)
- DAY_MICROSECOND (R)
- DAY_MINUTE (R)
- DAY_SECOND (R)
- DEALLOCATE

- DEC (R)
- DECIMAL (R)
- DECLARE (R)
- DEFAULT (R)
- DEFAULT_AUTH; added in 5.6.4 (nonreserved)
- DEFINER
- DELAYED (R)
- DELAY_KEY_WRITE
- DELETE (R)
- DESC (R)
- DESCRIBE (R)
- DES_KEY_FILE
- DETERMINISTIC (R)
- DIAGNOSTICS; added in 5.6.4 (nonreserved)
- DIRECTORY
- DISABLE
- DISCARD
- DISK
- DISTINCT (R)
- DISTINCTROW (R)
- DIV (R)
- DO
- DOUBLE (R)
- DROP (R)
- DUAL (R)
- DUMPFILE
- DUPLICATE
- DYNAMIC

E

- EACH (R)
- ELSE (R)
- ELSEIF (R)
- ENABLE

- ENCLOSED (R)
- END
- ENDS
- ENGINE
- ENGINES
- ENUM
- ERROR
- ERRORS
- ESCAPE
- ESCAPED (R)
- EVENT
- EVENTS
- EVERY
- EXCHANGE
- EXECUTE
- EXISTS (R)
- EXIT (R)
- EXPANSION
- EXPIRE; added in 5.6.6 (nonreserved)
- EXPLAIN (R)
- EXPORT; added in 5.6.6 (nonreserved)
- EXTENDED
- EXTENT_SIZE

F

- FALSE (R)
- FAST
- FAULTS
- FETCH (R)
- FIELDS
- FILE
- FIRST
- FIXED
- FLOAT (R)

- FLOAT4 (R)
- FLOAT8 (R)
- FLUSH
- FOR (R)
- FORCE (R)
- FOREIGN (R)
- FORMAT; added in 5.6.5 (nonreserved)
- FOUND
- FROM (R)
- FULL
- FULLTEXT (R)
- FUNCTION

G

- GENERAL; became nonreserved in 5.6.1
- GEOMETRY
- GEOMETRYCOLLECTION
- GET (R); added in 5.6.4 (reserved)
- GET_FORMAT
- GLOBAL
- GRANT (R)
- GRANTS
- GROUP (R)

H

- HANDLER
- HASH
- HAVING (R)
- HELP
- HIGH_PRIORITY (R)
- HOST
- HOSTS
- HOUR
- HOUR_MICROSECOND (R)
- HOUR_MINUTE (R)

- HOUR_SECOND (R)

I

- IDENTIFIED
- IF (R)
- IGNORE (R)
- IGNORE_SERVER_IDS; became nonreserved in 5.6.1
- IMPORT
- IN (R)
- INDEX (R)
- INDEXES
- INFILE (R)
- INITIAL_SIZE
- INNER (R)
- INOUT (R)
- INSENSITIVE (R)
- INSERT (R)
- INSERT_METHOD
- INSTALL
- INT (R)
- INT1 (R)
- INT2 (R)
- INT3 (R)
- INT4 (R)
- INT8 (R)
- INTEGER (R)
- INTERVAL (R)
- INTO (R)
- INVOKER
- IO
- IO_AFTER_GTIDS (R); added in 5.6.5 (reserved)
- IO_BEFORE_GTIDS (R); added in 5.6.5 (reserved)
- IO_THREAD
- IPC

- IS (R)
- ISOLATION
- ISSUER
- ITERATE (R)

J

- JOIN (R)

K

- KEY (R)
- KEYS (R)
- KEY_BLOCK_SIZE
- KILL (R)

L

- LANGUAGE
- LAST
- LEADING (R)
- LEAVE (R)
- LEAVES
- LEFT (R)
- LESS
- LEVEL
- LIKE (R)
- LIMIT (R)
- LINEAR (R)
- LINES (R)
- LINESTRING
- LIST
- LOAD (R)
- LOCAL
- LOCALTIME (R)
- LOCALTIMESTAMP (R)
- LOCK (R)
- LOCKS
- LOGFILE

- LOGS
- LONG (R)
- LONGBLOB (R)
- LONGTEXT (R)
- LOOP (R)
- LOW_PRIORITY (R)

M

- MASTER
- MASTER_AUTO_POSITION; added in 5.6.5 (nonreserved)
- MASTER_BIND (R); added in 5.6.1 (reserved)
- MASTER_CONNECT_RETRY
- MASTER_DELAY
- MASTER_HEARTBEAT_PERIOD; became nonreserved in 5.6.1
- MASTER_HOST
- MASTER_LOG_FILE
- MASTER_LOG_POS
- MASTER_PASSWORD
- MASTER_PORT
- MASTER_RETRY_COUNT; added in 5.6.1 (nonreserved)
- MASTER_SERVER_ID
- MASTER_SSL
- MASTER_SSL_CA
- MASTER_SSL_CAPATH
- MASTER_SSL_CERT
- MASTER_SSL_CIPHER
- MASTER_SSL_CRL; added in 5.6.3 (nonreserved)
- MASTER_SSL_CRLPATH; added in 5.6.3 (nonreserved)
- MASTER_SSL_KEY
- MASTER_SSL_VERIFY_SERVER_CERT (R)
- MASTER_USER
- MATCH (R)
- MAXVALUE (R)
- MAX_CONNECTIONS_PER_HOUR

- MAX_QUERIES_PER_HOUR
- MAX_ROWS
- MAX_SIZE
- MAX_UPDATES_PER_HOUR
- MAX_USER_CONNECTIONS
- MEDIUM
- MEDIUMBLOB (R)
- MEDIUMINT (R)
- MEDIUMTEXT (R)
- MEMORY
- MERGE
- MESSAGE_TEXT
- MICROSECOND
- MIDDLEINT (R)
- MIGRATE
- MINUTE
- MINUTE_MICROSECOND (R)
- MINUTE_SECOND (R)
- MIN_ROWS
- MOD (R)
- MODE
- MODIFIES (R)
- MODIFY
- MONTH
- MULTILINESTRING
- MULTIPOINT
- MULTIPOLYGON
- MUTEX
- MYSQL_ERRNO

N

- NAME
- NAMES
- NATIONAL

- NATURAL (R)
- NCHAR
- NDB
- NDBCLUSTER
- NEW
- NEXT
- NO
- NODEGROUP
- NONE
- NOT (R)
- NO_WAIT
- NO_WRITE_TO_BINLOG (R)
- NULL (R)
- NUMBER; added in 5.6.4 (nonreserved)
- NUMERIC (R)
- NVARCHAR

O

- OFFSET
- OLD_PASSWORD
- ON (R)
- ONE
- ONE_SHOT; became reserved in 5.6.1; removed in 5.6.5
- ONLY; added in 5.6.5 (nonreserved)
- OPEN
- OPTIMIZE (R)
- OPTION (R)
- OPTIONALLY (R)
- OPTIONS
- OR (R)
- ORDER (R)
- OUT (R)
- OUTER (R)
- OUTFILE (R)

- OWNER

P

- PACK_KEYS
- PAGE
- PARSER
- PARTIAL
- PARTITION (R); became reserved in 5.6.2
- PARTITIONING
- PARTITIONS
- PASSWORD
- PHASE
- PLUGIN
- PLUGINS
- PLUGIN_DIR; added in 5.6.4 (nonreserved)
- POINT
- POLYGON
- PORT
- PRECISION (R)
- PREPARE
- PRESERVE
- PREV
- PRIMARY (R)
- PRIVILEGES
- PROCEDURE (R)
- PROCESSLIST
- PROFILE
- PROFILES
- PROXY; added in 5.6.1 (nonreserved)
- PURGE (R)

Q

- QUARTER
- QUERY
- QUICK

R

- RANGE (R)
- READ (R)
- READS (R)
- READ_ONLY
- READ_WRITE (R)
- REAL (R)
- REBUILD
- RECOVER
- REDOFILE
- REDO_BUFFER_SIZE
- REDUNDANT
- REFERENCES (R)
- REGEXP (R)
- RELAY
- RELAYLOG
- RELAY_LOG_FILE
- RELAY_LOG_POS
- RELAY_THREAD
- RELEASE (R)
- RELOAD
- REMOVE
- RENAME (R)
- REORGANIZE
- REPAIR
- REPEAT (R)
- REPEATABLE
- REPLACE (R)
- REPLICATION
- REQUIRE (R)
- RESET
- RESIGNAL (R)
- RESTORE

- RESTRICT (R)
- RESUME
- RETURN (R)
- RETURNED_SQLSTATE; added in 5.6.4 (nonreserved)
- RETURNS
- REVERSE
- REVOKE (R)
- RIGHT (R)
- RLIKE (R)
- ROLLBACK
- ROLLUP
- ROUTINE
- ROW
- ROWS
- ROW_COUNT; added in 5.6.4 (nonreserved)
- ROW_FORMAT
- RTREE

S

- SAVEPOINT
- SCHEDULE
- SCHEMA (R)
- SCHEMAS (R)
- SCHEMA_NAME
- SECOND
- SECOND_MICROSECOND (R)
- SECURITY
- SELECT (R)
- SENSITIVE (R)
- SEPARATOR (R)
- SERIAL
- SERIALIZABLE
- SERVER
- SESSION

- SET (R)
- SHARE
- SHOW (R)
- SHUTDOWN
- SIGNAL (R)
- SIGNED
- SIMPLE
- SLAVE
- SLOW; became nonreserved in 5.6.1
- SMALLINT (R)
- SNAPSHOT
- SOCKET
- SOME
- SONAME
- SOUNDS
- SOURCE
- SPATIAL (R)
- SPECIFIC (R)
- SQL (R)
- SQLEXCEPTION (R)
- SQLSTATE (R)
- SQLWARNING (R)
- SQL_AFTER_GTIDS; added in 5.6.5 (reserved); became nonreserved in 5.6.6
- SQL_AFTER_MTS_GAPS; added in 5.6.6 (nonreserved)
- SQL_BEFORE_GTIDS; added in 5.6.5 (reserved); became nonreserved in 5.6.6
- SQL_BIG_RESULT (R)
- SQL_BUFFER_RESULT
- SQL_CACHE
- SQL_CALC_FOUND_ROWS (R)
- SQL_NO_CACHE
- SQL_SMALL_RESULT (R)
- SQL_THREAD
- SQL_TSI_DAY

- SQL_TSI_HOUR
- SQL_TSI_MINUTE
- SQL_TSI_MONTH
- SQL_TSI_QUARTER
- SQL_TSI_SECOND
- SQL_TSI_WEEK
- SQL_TSI_YEAR
- SSL (R)
- START
- STARTING (R)
- STARTS
- STATS_AUTO_RECALC; added in 5.6.6 (nonreserved)
- STATS_PERSISTENT; added in 5.6.6 (nonreserved)
- STATS_SAMPLE_PAGES; added in 5.6.6 (nonreserved)
- STATUS
- STOP
- STORAGE
- STRAIGHT_JOIN (R)
- STRING
- SUBCLASS_ORIGIN
- SUBJECT
- SUBPARTITION
- SUBPARTITIONS
- SUPER
- SUSPEND
- SWAPS
- SWITCHES

T

- TABLE (R)
- TABLES
- TABLESPACE
- TABLE_CHECKSUM
- TABLE_NAME

- TEMPORARY
- TEMPTABLE
- TERMINATED (R)
- TEXT
- THAN
- THEN (R)
- TIME
- TIMESTAMP
- TIMESTAMPADD
- TIMESTAMPDIFF
- TINYBLOB (R)
- TINYINT (R)
- TINYTEXT (R)
- TO (R)
- TRAILING (R)
- TRANSACTION
- TRIGGER (R)
- TRIGGERS
- TRUE (R)
- TRUNCATE
- TYPE
- TYPES

U

- UNCOMMITTED
- UNDEFINED
- UNDO (R)
- UNDOFILE
- UNDO_BUFFER_SIZE
- UNICODE
- UNINSTALL
- UNION (R)
- UNIQUE (R)
- UNKNOWN

- UNLOCK (R)
- UNSIGNED (R)
- UNTIL
- UPDATE (R)
- UPGRADE
- USAGE (R)
- USE (R)
- USER
- USER_RESOURCES
- USE_FRM
- USING (R)
- UTC_DATE (R)
- UTC_TIME (R)
- UTC_TIMESTAMP (R)

V

- VALUE
- VALUES (R)
- VARBINARY (R)
- VARCHAR (R)
- VARCHARACTER (R)
- VARIABLES
- VARYING (R)
- VIEW

W

- WAIT
- WARNINGS
- WEEK
- WEIGHT_STRING
- WHEN (R)
- WHERE (R)
- WHILE (R)
- WITH (R)
- WORK

- WRAPPER
- WRITE (R)

X

- X509
- XA
- XML
- XOR (R)

Y

- YEAR
- YEAR_MONTH (R)

Z

- ZEROFILL (R)

MySQL 5.6 New Keywords and Reserved Words

The following list shows the keywords and reserved words that are added in MySQL 5.6, compared to MySQL 5.5. Reserved keywords are marked with (R).

A | C | D | E | F | G | I | M | N | O | P | R | S | W

A

- ANALYSE

C

- COLUMN_FORMAT
- CURRENT

D

- DEFAULT_AUTH
- DIAGNOSTICS

E

- EXCHANGE
- EXPIRE
- EXPORT

F

- FORMAT

G

- GET (R)

I

- IO_AFTER_GTIDS (R)
- IO_BEFORE_GTIDS (R)

M

- MASTER_AUTO_POSITION
- MASTER_BIND (R)
- MASTER_DELAY
- MASTER_RETRY_COUNT
- MASTER_SSL_CRL
- MASTER_SSL_CRLPATH

N

- NUMBER

O

- ONLY

P

- PLUGIN_DIR

R

- RETURNED_SQLSTATE
- REVERSE
- ROW_COUNT

S

- SQL_AFTER_GTIDS
- SQL_AFTER_MTS_GAPS
- SQL_BEFORE_GTIDS
- STATS_AUTO_RECALC
- STATS_PERSISTENT
- STATS_SAMPLE_PAGES

W

- WEIGHT_STRING

MySQL 5.6 Removed Keywords and Reserved Words

The following list shows the keywords and reserved words that are removed in MySQL 5.6, compared to MySQL 5.5. Reserved keywords are marked with (R).

- AUTHORS
- CONTRIBUTORS

- `ONE_SHOT`

9.4 User-Defined Variables

You can store a value in a user-defined variable in one statement and refer to it later in another statement. This enables you to pass values from one statement to another.

User variables are written as *`@var_name`*, where the variable name *`var_name`* consists of alphanumeric characters, `.`, `_`, and `$`. A user variable name can contain other characters if you quote it as a string or identifier (for example, `@'my-var'`, `@"my-var"`, or `@`my-var``).

User-defined variables are session specific. A user variable defined by one client cannot be seen or used by other clients. All variables for a given client session are automatically freed when that client exits.

User variable names are not case-sensitive.

One way to set a user-defined variable is by issuing a `SET` statement:

```
SET @var_name = expr [, @var_name = expr] ...
```

For `SET`, either `=` or `:=` can be used as the assignment operator.

User variables can be assigned a value from a limited set of data types: integer, decimal, floating-point, binary or nonbinary string, or `NULL` value. Assignment of decimal and real values does not preserve the precision or scale of the value. A value of a type other than one of the permissible types is converted to a permissible type. For example, a value having a temporal or spatial data type is converted to a binary string.

If a user variable is assigned a nonbinary (character) string value, it has the same character set and collation as the string. The coercibility of user variables is implicit. (This is the same coercibility as for table column values.)

Hexadecimal or bit values assigned to user variables are treated as binary strings. To assign a hexadecimal or bit value as a number to a user variable, use it in numeric context. For example, add 0 or use `CAST(... AS UNSIGNED)`:

```
mysql> SET @v1 = X'41';
mysql> SET @v2 = X'41'+0;
mysql> SET @v3 = CAST(X'41' AS UNSIGNED);
mysql> SELECT @v1, @v2, @v3;
+------+------+------+
| @v1  | @v2  | @v3  |
+------+------+------+
| A    |   65 |   65 |
+------+------+------+
mysql> SET @v1 = b'1000001';
mysql> SET @v2 = b'1000001'+0;
mysql> SET @v3 = CAST(b'1000001' AS UNSIGNED);
mysql> SELECT @v1, @v2, @v3;
+------+------+------+
| @v1  | @v2  | @v3  |
+------+------+------+
| A    |   65 |   65 |
+------+------+------+
```

If the value of a user variable is selected in a result set, it is returned to the client as a string.

If you refer to a variable that has not been initialized, it has a value of `NULL` and a type of string.

User variables may be used in most contexts where expressions are permitted. This does not currently include contexts that explicitly require a literal value, such as in the `LIMIT` clause of a `SELECT` statement, or the `IGNORE N LINES` clause of a `LOAD DATA` statement.

You can also assign a value to a user variable in statements other than `SET`. In this case, the assignment operator must be `:=` and not `=` because the latter is treated as the comparison operator `=` in statements other than `SET`:

```
mysql> SET @t1=1, @t2=2, @t3:=4;
mysql> SELECT @t1, @t2, @t3, @t4 := @t1+@t2+@t3;
+------+------+------+--------------------+
| @t1  | @t2  | @t3  | @t4 := @t1+@t2+@t3 |
+------+------+------+--------------------+
|    1 |    2 |    4 |                  7 |
+------+------+------+--------------------+
```

As a general rule, other than in `SET` statements, you should never assign a value to a user variable and read the value within the same statement. For example, to increment a variable, this is okay:

```
SET @a = @a + 1;
```

For other statements, such as `SELECT`, you might get the results you expect, but this is not guaranteed. In the following statement, you might think that MySQL evaluates `@a` first and then performs an assignment second:

```
SELECT @a, @a:=@a+1, ...;
```

However, the order of evaluation for expressions involving user variables is undefined.

Another issue with assigning a value to a variable and reading the value within the same non-`SET` statement is that the default result type of a variable is based on its type at the start of the statement. The following example illustrates this:

```
mysql> SET @a='test';
mysql> SELECT @a,(@a:=20) FROM tbl_name;
```

For this `SELECT` statement, MySQL reports to the client that column one is a string and converts all accesses of `@a` to strings, even though @a is set to a number for the second row. After the `SELECT` statement executes, `@a` is regarded as a number for the next statement.

To avoid problems with this behavior, either do not assign a value to and read the value of the same variable within a single statement, or else set the variable to `0`, `0.0`, or `''` to define its type before you use it.

In a `SELECT` statement, each select expression is evaluated only when sent to the client. This means that in a `HAVING`, `GROUP BY`, or `ORDER BY` clause, referring to a variable that is assigned a value in the select expression list does *not* work as expected:

```
mysql> SELECT (@aa:=id) AS a, (@aa+3) AS b FROM tbl_name HAVING b=5;
```

The reference to `b` in the `HAVING` clause refers to an alias for an expression in the select list that uses `@aa`. This does not work as expected: `@aa` contains the value of `id` from the previous selected row, not from the current row.

User variables are intended to provide data values. They cannot be used directly in an SQL statement as an identifier or as part of an identifier, such as in contexts where a table or database name is expected, or as a reserved word such as `SELECT`. This is true even if the variable is quoted, as shown in the following example:

```
mysql> SELECT c1 FROM t;
+----+
| c1 |
+----+
|  0 |
+----+
|  1 |
+----+
2 rows in set (0.00 sec)
```

```
mysql> SET @col = "c1";
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @col FROM t;
+------+
| @col |
+------+
| c1   |
+------+
1 row in set (0.00 sec)

mysql> SELECT `@col` FROM t;
ERROR 1054 (42S22): Unknown column '@col' in 'field list'

mysql> SET @col = "`c1`";
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @col FROM t;
+------+
| @col |
+------+
| `c1` |
+------+
1 row in set (0.00 sec)
```

An exception to this principle that user variables cannot be used to provide identifiers, is when you are constructing a string for use as a prepared statement to execute later. In this case, user variables can be used to provide any part of the statement. The following example illustrates how this can be done:

```
mysql> SET @c = "c1";
Query OK, 0 rows affected (0.00 sec)

mysql> SET @s = CONCAT("SELECT ", @c, " FROM t");
Query OK, 0 rows affected (0.00 sec)

mysql> PREPARE stmt FROM @s;
Query OK, 0 rows affected (0.04 sec)
Statement prepared

mysql> EXECUTE stmt;
+----+
| c1 |
+----+
|  0 |
+----+
|  1 |
+----+
2 rows in set (0.00 sec)

mysql> DEALLOCATE PREPARE stmt;
Query OK, 0 rows affected (0.00 sec)
```

See Section 13.5, “Prepared Statements”, for more information.

A similar technique can be used in application programs to construct SQL statements using program variables, as shown here using PHP 5:

```
<?php
  $mysqli = new mysqli("localhost", "user", "pass", "test");

  if( mysqli_connect_errno() )
    die("Connection failed: %s\n", mysqli_connect_error());

  $col = "c1";

  $query = "SELECT $col FROM t";

  $result = $mysqli->query($query);

  while($row = $result->fetch_assoc())
  {
```

```
    echo "<p>" . $row["$col"] . "</p>\n";
  }

  $result->close();

  $mysqli->close();
?>
```

Assembling an SQL statement in this fashion is sometimes known as “Dynamic SQL”.

9.5 Expressions

This section lists the grammar rules that expressions must follow in MySQL and provides additional information about the types of terms that may appear in expressions.

- Expression Syntax
- Expression Term Notes
- Temporal Intervals

Expression Syntax

The following grammar rules define expression syntax in MySQL. The grammar shown here is based on that given in the `sql/sql_yacc.yy` file of MySQL source distributions. For additional information about some of the expression terms, see Expression Term Notes.

```
expr:
    expr OR expr
  | expr || expr
  | expr XOR expr
  | expr AND expr
  | expr && expr
  | NOT expr
  | ! expr
  | boolean_primary IS [NOT] {TRUE | FALSE | UNKNOWN}
  | boolean_primary

boolean_primary:
    boolean_primary IS [NOT] NULL
  | boolean_primary <=> predicate
  | boolean_primary comparison_operator predicate
  | boolean_primary comparison_operator {ALL | ANY} (subquery)
  | predicate

comparison_operator: = | >= | > | <= | < | <> | !=

predicate:
    bit_expr [NOT] IN (subquery)
  | bit_expr [NOT] IN (expr [, expr] ...)
  | bit_expr [NOT] BETWEEN bit_expr AND predicate
  | bit_expr SOUNDS LIKE bit_expr
  | bit_expr [NOT] LIKE simple_expr [ESCAPE simple_expr]
  | bit_expr [NOT] REGEXP bit_expr
  | bit_expr

bit_expr:
    bit_expr | bit_expr
  | bit_expr & bit_expr
  | bit_expr << bit_expr
  | bit_expr >> bit_expr
  | bit_expr + bit_expr
  | bit_expr - bit_expr
  | bit_expr * bit_expr
  | bit_expr / bit_expr
  | bit_expr DIV bit_expr
  | bit_expr MOD bit_expr
  | bit_expr % bit_expr
```

```
  | bit_expr ^ bit_expr
  | bit_expr + interval_expr
  | bit_expr - interval_expr
  | simple_expr

simple_expr:
    literal
  | identifier
  | function_call
  | simple_expr COLLATE collation_name
  | param_marker
  | variable
  | simple_expr || simple_expr
  | + simple_expr
  | - simple_expr
  | ~ simple_expr
  | ! simple_expr
  | BINARY simple_expr
  | (expr [, expr] ...)
  | ROW (expr, expr [, expr] ...)
  | (subquery)
  | EXISTS (subquery)
  | {identifier expr}
  | match_expr
  | case_expr
  | interval_expr
```

For operator precedence, see Section 12.4.1, “Operator Precedence”. The precedence and meaning of some operators depends on the SQL mode:

- By default, `||` is a logical `OR` operator. With `PIPES_AS_CONCAT` enabled, `||` is string concatenation, with a precedence between `^` and the unary operators.
- By default, `!` has a higher precedence than `NOT`. With `HIGH_NOT_PRECEDENCE` enabled, `!` and `NOT` have the same precedence.

See Section 5.1.10, “Server SQL Modes”.

Expression Term Notes

For literal value syntax, see Section 9.1, “Literal Values”.

For identifier syntax, see Section 9.2, “Schema Object Names”.

Variables can be user variables, system variables, or stored program local variables or parameters:

- User variables: Section 9.4, “User-Defined Variables”
- System variables: Section 5.1.8, “Using System Variables”
- Stored program local variables: Section 13.6.4.1, “Local Variable DECLARE Statement”
- Stored program parameters: Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”

`param_marker` is `?` as used in prepared statements for placeholders. See Section 13.5.1, “PREPARE Statement”.

`(`*`subquery`*`)` indicates a subquery that returns a single value; that is, a scalar subquery. See Section 13.2.10.1, “The Subquery as Scalar Operand”.

`{`*`identifier expr`*`}` is ODBC escape syntax and is accepted for ODBC compatibility. The value is *`expr`*. The `{` and `}` curly braces in the syntax should be written literally; they are not metasyntax as used elsewhere in syntax descriptions.

`match_expr` indicates a `MATCH` expression. See Section 12.10, “Full-Text Search Functions”.

`case_expr` indicates a `CASE` expression. See Section 12.5, “Flow Control Functions”.

`interval_expr` represents a temporal interval. See Temporal Intervals.

Temporal Intervals

`interval_expr` in expressions represents a temporal interval. Intervals have this syntax:

```
INTERVAL expr unit
```

`expr` represents a quantity. *`unit`* represents the unit for interpreting the quantity; it is a specifier such as `HOUR`, `DAY`, or `WEEK`. The `INTERVAL` keyword and the *`unit`* specifier are not case-sensitive.

The following table shows the expected form of the *`expr`* argument for each *`unit`* value.

Table 9.2 Temporal Interval Expression and Unit Arguments

***`unit`* Value**	**Expected *`expr`* Format**
`MICROSECOND`	`MICROSECONDS`
`SECOND`	`SECONDS`
`MINUTE`	`MINUTES`
`HOUR`	`HOURS`
`DAY`	`DAYS`
`WEEK`	`WEEKS`
`MONTH`	`MONTHS`
`QUARTER`	`QUARTERS`
`YEAR`	`YEARS`
`SECOND_MICROSECOND`	`'SECONDS.MICROSECONDS'`
`MINUTE_MICROSECOND`	`'MINUTES:SECONDS.MICROSECONDS'`
`MINUTE_SECOND`	`'MINUTES:SECONDS'`
`HOUR_MICROSECOND`	`'HOURS:MINUTES:SECONDS.MICROSECONDS'`
`HOUR_SECOND`	`'HOURS:MINUTES:SECONDS'`
`HOUR_MINUTE`	`'HOURS:MINUTES'`
`DAY_MICROSECOND`	`'DAYS HOURS:MINUTES:SECONDS.MICROSECONDS'`
`DAY_SECOND`	`'DAYS HOURS:MINUTES:SECONDS'`
`DAY_MINUTE`	`'DAYS HOURS:MINUTES'`
`DAY_HOUR`	`'DAYS HOURS'`
`YEAR_MONTH`	`'YEARS-MONTHS'`

MySQL permits any punctuation delimiter in the *`expr`* format. Those shown in the table are the suggested delimiters.

Temporal intervals are used for certain functions, such as `DATE_ADD()` and `DATE_SUB()`:

```
mysql> SELECT DATE_ADD('2018-05-01',INTERVAL 1 DAY);
        -> '2018-05-02'
mysql> SELECT DATE_SUB('2018-05-01',INTERVAL 1 YEAR);
        -> '2017-05-01'
mysql> SELECT DATE_ADD('2020-12-31 23:59:59',
    ->                 INTERVAL 1 SECOND);
        -> '2021-01-01 00:00:00'
mysql> SELECT DATE_ADD('2018-12-31 23:59:59',
    ->                 INTERVAL 1 DAY);
```

```
        -> '2019-01-01 23:59:59'
mysql> SELECT DATE_ADD('2100-12-31 23:59:59',
    ->                 INTERVAL '1:1' MINUTE_SECOND);
        -> '2101-01-01 00:01:00'
mysql> SELECT DATE_SUB('2025-01-01 00:00:00',
    ->                 INTERVAL '1 1:1:1' DAY_SECOND);
        -> '2024-12-30 22:58:59'
mysql> SELECT DATE_ADD('1900-01-01 00:00:00',
    ->                 INTERVAL '-1 10' DAY_HOUR);
        -> '1899-12-30 14:00:00'
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
        -> '1997-12-02'
mysql> SELECT DATE_ADD('1992-12-31 23:59:59.000002',
    ->            INTERVAL '1.999999' SECOND_MICROSECOND);
        -> '1993-01-01 00:00:01.000001'
```

Temporal arithmetic also can be performed in expressions using `INTERVAL` together with the `+` or `-` operator:

```
date + INTERVAL expr unit
date - INTERVAL expr unit
```

`INTERVAL expr unit` is permitted on either side of the `+` operator if the expression on the other side is a date or datetime value. For the `-` operator, `INTERVAL expr unit` is permitted only on the right side, because it makes no sense to subtract a date or datetime value from an interval.

```
mysql> SELECT '2018-12-31 23:59:59' + INTERVAL 1 SECOND;
        -> '2019-01-01 00:00:00'
mysql> SELECT INTERVAL 1 DAY + '2018-12-31';
        -> '2019-01-01'
mysql> SELECT '2025-01-01' - INTERVAL 1 SECOND;
        -> '2024-12-31 23:59:59'
```

The `EXTRACT()` function uses the same kinds of *`unit`* specifiers as `DATE_ADD()` or `DATE_SUB()`, but extracts parts from the date rather than performing date arithmetic:

```
mysql> SELECT EXTRACT(YEAR FROM '2019-07-02');
        -> 2019
mysql> SELECT EXTRACT(YEAR_MONTH FROM '2019-07-02 01:02:03');
        -> 201907
```

Temporal intervals can be used in `CREATE EVENT` statements:

```
CREATE EVENT myevent
    ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 HOUR
    DO
      UPDATE myschema.mytable SET mycol = mycol + 1;
```

If you specify an interval value that is too short (does not include all the interval parts that would be expected from the *`unit`* keyword), MySQL assumes that you have left out the leftmost parts of the interval value. For example, if you specify a *`unit`* of `DAY_SECOND`, the value of *`expr`* is expected to have days, hours, minutes, and seconds parts. If you specify a value like `'1:10'`, MySQL assumes that the days and hours parts are missing and the value represents minutes and seconds. In other words, `'1:10' DAY_SECOND` is interpreted in such a way that it is equivalent to `'1:10' MINUTE_SECOND`. This is analogous to the way that MySQL interprets `TIME` values as representing elapsed time rather than as a time of day.

`expr` is treated as a string, so be careful if you specify a nonstring value with `INTERVAL`. For example, with an interval specifier of `HOUR_MINUTE`, '6/4' is treated as 6 hours, four minutes, whereas `6/4` evaluates to `1.5000` and is treated as 1 hour, 5000 minutes:

```
mysql> SELECT '6/4', 6/4;
        -> 1.5000
mysql> SELECT DATE_ADD('2019-01-01', INTERVAL '6/4' HOUR_MINUTE);
        -> '2019-01-01 06:04:00'
mysql> SELECT DATE_ADD('2019-01-01', INTERVAL 6/4 HOUR_MINUTE);
        -> '2019-01-04 12:20:00'
```

To ensure interpretation of the interval value as you expect, a `CAST()` operation may be used. To treat `6/4` as 1 hour, 5 minutes, cast it to a `DECIMAL` value with a single fractional digit:

```
mysql> SELECT CAST(6/4 AS DECIMAL(3,1));
        -> 1.5
mysql> SELECT DATE_ADD('1970-01-01 12:00:00',
    ->                 INTERVAL CAST(6/4 AS DECIMAL(3,1)) HOUR_MINUTE);
        -> '1970-01-01 13:05:00'
```

If you add to or subtract from a date value something that contains a time part, the result is automatically converted to a datetime value:

```
mysql> SELECT DATE_ADD('2023-01-01', INTERVAL 1 DAY);
        -> '2023-01-02'
mysql> SELECT DATE_ADD('2023-01-01', INTERVAL 1 HOUR);
        -> '2023-01-01 01:00:00'
```

If you add `MONTH`, `YEAR_MONTH`, or `YEAR` and the resulting date has a day that is larger than the maximum day for the new month, the day is adjusted to the maximum days in the new month:

```
mysql> SELECT DATE_ADD('2019-01-30', INTERVAL 1 MONTH);
        -> '2019-02-28'
```

Date arithmetic operations require complete dates and do not work with incomplete dates such as `'2016-07-00'` or badly malformed dates:

```
mysql> SELECT DATE_ADD('2016-07-00', INTERVAL 1 DAY);
        -> NULL
mysql> SELECT '2005-03-32' + INTERVAL 1 MONTH;
        -> NULL
```

9.6 Comments

MySQL Server supports three comment styles:

- From a `#` character to the end of the line.
- From a `--` sequence to the end of the line. In MySQL, the `--` (double-dash) comment style requires the second dash to be followed by at least one whitespace or control character (such as a space, tab, newline, and so on). This syntax differs slightly from standard SQL comment syntax, as discussed in Section 1.7.2.4, “'--' as the Start of a Comment”.
- From a `/*` sequence to the following `*/` sequence, as in the C programming language. This syntax enables a comment to extend over multiple lines because the beginning and closing sequences need not be on the same line.

The following example demonstrates all three comment styles:

```
mysql> SELECT 1+1;     # This comment continues to the end of line
mysql> SELECT 1+1;     -- This comment continues to the end of line
mysql> SELECT 1 /* this is an in-line comment */ + 1;
mysql> SELECT 1+
/*
this is a
multiple-line comment
*/
1;
```

Nested comments are not supported. (Under some conditions, nested comments might be permitted, but usually are not, and users should avoid them.)

MySQL Server supports certain variants of C-style comments. These enable you to write code that includes MySQL extensions, but is still portable, by using comments of the following form:

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

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

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

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

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

The comment syntax just described applies to how the `mysqld` server parses SQL statements. The `mysql` client program also performs some parsing of statements before sending them to the server. (It does this to determine statement boundaries within a multiple-statement input line.) For information about differences between the server and `mysql` client parsers, see Section 4.5.1.6, “mysql Client Tips”.

Comments in `/*!12345 ... */` format are not stored on the server. If this format is used to comment stored programs, the comments are not retained in the program body.

The use of short-form `mysql` commands such as `\C` within multiple-line `/* ... */` comments is not supported.

Chapter 10 Character Sets, Collations, Unicode

Table of Contents

10.1 Character Sets and Collations in General 1176
10.2 Character Sets and Collations in MySQL 1177
10.2.1 Character Set Repertoire 1178
10.2.2 UTF-8 for Metadata 1180
10.3 Specifying Character Sets and Collations 1181
10.3.1 Collation Naming Conventions 1181
10.3.2 Server Character Set and Collation 1182
10.3.3 Database Character Set and Collation 1183
10.3.4 Table Character Set and Collation 1184
10.3.5 Column Character Set and Collation 1185
10.3.6 Character String Literal Character Set and Collation 1186
10.3.7 The National Character Set 1188
10.3.8 Character Set Introducers 1188
10.3.9 Examples of Character Set and Collation Assignment 1190
10.3.10 Compatibility with Other DBMSs 1190
10.4 Connection Character Sets and Collations 1191
10.5 Configuring Application Character Set and Collation 1197
10.6 Error Message Character Set 1198
10.7 Column Character Set Conversion 1199
10.8 Collation Issues 1200
10.8.1 Using COLLATE in SQL Statements 1200
10.8.2 COLLATE Clause Precedence 1201
10.8.3 Character Set and Collation Compatibility 1201
10.8.4 Collation Coercibility in Expressions 1201
10.8.5 The binary Collation Compared to _bin Collations 1203
10.8.6 Examples of the Effect of Collation 1205
10.8.7 Using Collation in INFORMATION_SCHEMA Searches 1206
10.9 Unicode Support 1208
10.9.1 The utf8mb4 Character Set (4-Byte UTF-8 Unicode Encoding) 1209
10.9.2 The utf8mb3 Character Set (3-Byte UTF-8 Unicode Encoding) 1210
10.9.3 The utf8 Character Set (Alias for utf8mb3) 1210
10.9.4 The ucs2 Character Set (UCS-2 Unicode Encoding) 1211
10.9.5 The utf16 Character Set (UTF-16 Unicode Encoding) 1211
10.9.6 The utf16le Character Set (UTF-16LE Unicode Encoding) 1211
10.9.7 The utf32 Character Set (UTF-32 Unicode Encoding) 1211
10.9.8 Converting Between 3-Byte and 4-Byte Unicode Character Sets 1212
10.10 Supported Character Sets and Collations 1214
10.10.1 Unicode Character Sets 1215
10.10.2 West European Character Sets 1220
10.10.3 Central European Character Sets 1221
10.10.4 South European and Middle East Character Sets 1222
10.10.5 Baltic Character Sets 1223
10.10.6 Cyrillic Character Sets 1223
10.10.7 Asian Character Sets 1224
10.10.8 The Binary Character Set 1227
10.11 Restrictions on Character Sets 1228
10.12 Setting the Error Message Language 1228
10.13 Adding a Character Set 1229
10.13.1 Character Definition Arrays 1231
10.13.2 String Collating Support for Complex Character Sets 1232
10.13.3 Multi-Byte Character Support for Complex Character Sets 1232
10.14 Adding a Collation to a Character Set 1232

10.14.1 Collation Implementation Types 1233
10.14.2 Choosing a Collation ID 1236
10.14.3 Adding a Simple Collation to an 8-Bit Character Set 1237
10.14.4 Adding a UCA Collation to a Unicode Character Set 1238
10.15 Character Set Configuration 1244
10.16 MySQL Server Locale Support 1245

MySQL includes character set support that enables you to store data using a variety of character sets and perform comparisons according to a variety of collations. The default MySQL server character set and collation are `latin1` and `latin1_swedish_ci`, but you can specify character sets at the server, database, table, column, and string literal levels.

This chapter discusses the following topics:

- What are character sets and collations?
- The multiple-level default system for character set assignment.
- Syntax for specifying character sets and collations.
- Affected functions and operations.
- Unicode support.
- The character sets and collations that are available, with notes.
- Selecting the language for error messages.
- Selecting the locale for day and month names.

Character set issues affect not only data storage, but also communication between client programs and the MySQL server. If you want the client program to communicate with the server using a character set different from the default, you'll need to indicate which one. For example, to use the `utf8` Unicode character set, issue this statement after connecting to the server:

```
SET NAMES 'utf8';
```

For more information about configuring character sets for application use and character set-related issues in client/server communication, see Section 10.5, “Configuring Application Character Set and Collation”, and Section 10.4, “Connection Character Sets and Collations”.

10.1 Character Sets and Collations in General

A *character set* is a set of symbols and encodings. A *collation* is a set of rules for comparing characters in a character set. Let's make the distinction clear with an example of an imaginary character set.

Suppose that we have an alphabet with four letters: `A`, `B`, `a`, `b`. We give each letter a number: `A` = 0, `B` = 1, `a` = 2, `b` = 3. The letter `A` is a symbol, the number 0 is the *encoding* for `A`, and the combination of all four letters and their encodings is a *character set*.

Suppose that we want to compare two string values, `A` and `B`. The simplest way to do this is to look at the encodings: 0 for `A` and 1 for `B`. Because 0 is less than 1, we say `A` is less than `B`. What we've just done is apply a collation to our character set. The collation is a set of rules (only one rule in this case): “compare the encodings.” We call this simplest of all possible collations a *binary* collation.

But what if we want to say that the lowercase and uppercase letters are equivalent? Then we would have at least two rules: (1) treat the lowercase letters `a` and `b` as equivalent to `A` and `B`; (2) then compare the encodings. We call this a *case-insensitive* collation. It is a little more complex than a binary collation.

In real life, most character sets have many characters: not just `A` and `B` but whole alphabets, sometimes multiple alphabets or eastern writing systems with thousands of characters, along with

many special symbols and punctuation marks. Also in real life, most collations have many rules, not just for whether to distinguish lettercase, but also for whether to distinguish accents (an “accent” is a mark attached to a character as in German Ö), and for multiple-character mappings (such as the rule that Ö = OE in one of the two German collations).

MySQL can do these things for you:

- Store strings using a variety of character sets.
- Compare strings using a variety of collations.
- Mix strings with different character sets or collations in the same server, the same database, or even the same table.
- Enable specification of character set and collation at any level.

To use these features effectively, you must know what character sets and collations are available, how to change the defaults, and how they affect the behavior of string operators and functions.

10.2 Character Sets and Collations in MySQL

MySQL Server supports multiple character sets. To display the available character sets, use the INFORMATION_SCHEMA CHARACTER_SETS table or the SHOW CHARACTER SET statement. A partial listing follows. For more complete information, see Section 10.10, “Supported Character Sets and Collations”.

```
mysql> SHOW CHARACTER SET;
+----------+---------------------------------+---------------------+--------+
| Charset  | Description                     | Default collation   | Maxlen |
+----------+---------------------------------+---------------------+--------+
| big5     | Big5 Traditional Chinese        | big5_chinese_ci     |      2 |
...
| latin1   | cp1252 West European            | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European     | latin2_general_ci   |      1 |
...
| utf8     | UTF-8 Unicode                   | utf8_general_ci     |      3 |
| ucs2     | UCS-2 Unicode                   | ucs2_general_ci     |      2 |
...
| utf8mb4  | UTF-8 Unicode                   | utf8mb4_general_ci  |      4 |
...
| binary   | Binary pseudo charset           | binary              |      1 |
...
```

By default, the SHOW CHARACTER SET statement displays all available character sets. It takes an optional LIKE or WHERE clause that indicates which character set names to match. For example:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+---------+-----------------------------+-------------------+--------+
| Charset | Description                 | Default collation | Maxlen |
+---------+-----------------------------+-------------------+--------+
| latin1  | cp1252 West European        | latin1_swedish_ci |      1 |
| latin2  | ISO 8859-2 Central European | latin2_general_ci |      1 |
| latin5  | ISO 8859-9 Turkish          | latin5_turkish_ci |      1 |
| latin7  | ISO 8859-13 Baltic          | latin7_general_ci |      1 |
+---------+-----------------------------+-------------------+--------+
```

A given character set always has at least one collation, and most character sets have several. To list the display collations for a character set, use the INFORMATION_SCHEMA COLLATIONS table or the SHOW COLLATION statement.

By default, the SHOW COLLATION statement displays all available collations. It takes an optional LIKE or WHERE clause that indicates which collation names to display. For example, to see the collations for the default character set, latin1 (cp1252 West European), use this statement:

```
mysql> SHOW COLLATION WHERE Charset = 'latin1';
+-------------------+---------+----+---------+----------+---------+
```

```
| Collation         | Charset | Id | Default | Compiled | Sortlen |
+-------------------+---------+----+---------+----------+---------+
| latin1_german1_ci | latin1  |  5 |         | Yes      |       1 |
| latin1_swedish_ci | latin1  |  8 | Yes     | Yes      |       1 |
| latin1_danish_ci  | latin1  | 15 |         | Yes      |       1 |
| latin1_german2_ci | latin1  | 31 |         | Yes      |       2 |
| latin1_bin        | latin1  | 47 |         | Yes      |       1 |
| latin1_general_ci | latin1  | 48 |         | Yes      |       1 |
| latin1_general_cs | latin1  | 49 |         | Yes      |       1 |
| latin1_spanish_ci | latin1  | 94 |         | Yes      |       1 |
+-------------------+---------+----+---------+----------+---------+
```

The `latin1` collations have the following meanings.

Collation	Meaning
`latin1_bin`	Binary according to `latin1` encoding
`latin1_danish_ci`	Danish/Norwegian
`latin1_general_ci`	Multilingual (Western European)
`latin1_general_cs`	Multilingual (ISO Western European), case-sensitive
`latin1_german1_ci`	German DIN-1 (dictionary order)
`latin1_german2_ci`	German DIN-2 (phone book order)
`latin1_spanish_ci`	Modern Spanish
`latin1_swedish_ci`	Swedish/Finnish

Collations have these general characteristics:

- Two different character sets cannot have the same collation.
- Each character set has a *default collation*. For example, the default collations for `latin1` and `utf8` are `latin1_swedish_ci` and `utf8_general_ci`, respectively. The `INFORMATION_SCHEMA CHARACTER_SETS` table and the `SHOW CHARACTER SET` statement indicate the default collation for each character set. The `INFORMATION_SCHEMA COLLATIONS` table and the `SHOW COLLATION` statement have a column that indicates for each collation whether it is the default for its character set (`Yes` if so, empty if not).
- Collation names start with the name of the character set with which they are associated, generally followed by one or more suffixes indicating other collation characteristics. For additional information about naming conventions, see Section 10.3.1, “Collation Naming Conventions”.

When a character set has multiple collations, it might not be clear which collation is most suitable for a given application. To avoid choosing an inappropriate collation, perform some comparisons with representative data values to make sure that a given collation sorts values the way you expect.

10.2.1 Character Set Repertoire

The *repertoire* of a character set is the collection of characters in the set.

String expressions have a repertoire attribute, which can have two values:

- `ASCII`: The expression can contain only ASCII characters; that is, characters in the Unicode range `U+0000` to `U+007F`.
- `UNICODE`: The expression can contain characters in the Unicode range `U+0000` to `U+10FFFF`. This includes characters in the Basic Multilingual Plane (BMP) range (`U+0000` to `U+FFFF`) and supplementary characters outside the BMP range (`U+10000` to `U+10FFFF`).

The `ASCII` range is a subset of `UNICODE` range, so a string with `ASCII` repertoire can be converted safely without loss of information to the character set of any string with `UNICODE` repertoire. It can also be converted safely to any character set that is a superset of the `ascii` character set. (All MySQL

character sets are supersets of `ascii` with the exception of `swe7`, which reuses some punctuation characters for Swedish accented characters.)

The use of repertoire enables character set conversion in expressions for many cases where MySQL would otherwise return an “illegal mix of collations” error when the rules for collation coercibility are insufficient to resolve ambiguities. (For information about coercibility, see Section 10.8.4, “Collation Coercibility in Expressions”.)

The following discussion provides examples of expressions and their repertoires, and describes how the use of repertoire changes string expression evaluation:

- The repertoire for a string constant depends on string content and may differ from the repertoire of the string character set. Consider these statements:

  ```
  SET NAMES utf8; SELECT 'abc';
  SELECT _utf8'def';
  SELECT N'MySQL';
  ```

  Although the character set is `utf8` in each of the preceding cases, the strings do not actually contain any characters outside the ASCII range, so their repertoire is `ASCII` rather than `UNICODE`.

- A column having the `ascii` character set has `ASCII` repertoire because of its character set. In the following table, `c1` has `ASCII` repertoire:

  ```
  CREATE TABLE t1 (c1 CHAR(1) CHARACTER SET ascii);
  ```

  The following example illustrates how repertoire enables a result to be determined in a case where an error occurs without repertoire:

  ```
  CREATE TABLE t1 (
    c1 CHAR(1) CHARACTER SET latin1,
    c2 CHAR(1) CHARACTER SET ascii
  );
  INSERT INTO t1 VALUES ('a','b');
  SELECT CONCAT(c1,c2) FROM t1;
  ```

  Without repertoire, this error occurs:

  ```
  ERROR 1267 (HY000): Illegal mix of collations (latin1_swedish_ci,IMPLICIT)
  and (ascii_general_ci,IMPLICIT) for operation 'concat'
  ```

  Using repertoire, subset to superset (`ascii` to `latin1`) conversion can occur and a result is returned:

  ```
  +---------------+
  | CONCAT(c1,c2) |
  +---------------+
  | ab            |
  +---------------+
  ```

- Functions with one string argument inherit the repertoire of their argument. The result of `UPPER(_utf8'abc')` has `ASCII` repertoire because its argument has `ASCII` repertoire. (Despite the `_utf8` introducer, the string `'abc'` contains no characters outside the ASCII range.)

- For functions that return a string but do not have string arguments and use `character_set_connection` as the result character set, the result repertoire is `ASCII` if `character_set_connection` is `ascii`, and `UNICODE` otherwise:

  ```
  FORMAT(numeric_column, 4);
  ```

  Use of repertoire changes how MySQL evaluates the following example:

  ```
  SET NAMES ascii;
  CREATE TABLE t1 (a INT, b VARCHAR(10) CHARACTER SET latin1);
  INSERT INTO t1 VALUES (1,'b');
  SELECT CONCAT(FORMAT(a, 4), b) FROM t1;
  ```

Without repertoire, this error occurs:

```
ERROR 1267 (HY000): Illegal mix of collations (ascii_general_ci,COERCIBLE)
and (latin1_swedish_ci,IMPLICIT) for operation 'concat'
```

With repertoire, a result is returned:

```
+-------------------------+
| CONCAT(FORMAT(a, 4), b) |
+-------------------------+
| 1.0000b                 |
+-------------------------+
```

- Functions with two or more string arguments use the “widest” argument repertoire for the result repertoire, where `UNICODE` is wider than `ASCII`. Consider the following `CONCAT()` calls:

  ```
  CONCAT(_ucs2 X'0041', _ucs2 X'0042')
  CONCAT(_ucs2 X'0041', _ucs2 X'00C2')
  ```

  For the first call, the repertoire is `ASCII` because both arguments are within the ASCII range. For the second call, the repertoire is `UNICODE` because the second argument is outside the ASCII range.

- The repertoire for function return values is determined based on the repertoire of only those arguments that affect the result's character set and collation.

  ```
  IF(column1 < column2, 'smaller', 'greater')
  ```

  The result repertoire is `ASCII` because the two string arguments (the second argument and the third argument) both have `ASCII` repertoire. The first argument does not matter for the result repertoire, even if the expression uses string values.

10.2.2 UTF-8 for Metadata

Metadata is “the data about the data.” Anything that *describes* the database—as opposed to being the *contents* of the database—is metadata. Thus column names, database names, user names, version names, and most of the string results from `SHOW` are metadata. This is also true of the contents of tables in `INFORMATION_SCHEMA` because those tables by definition contain information about database objects.

Representation of metadata must satisfy these requirements:

- All metadata must be in the same character set. Otherwise, neither the `SHOW` statements nor `SELECT` statements for tables in `INFORMATION_SCHEMA` would work properly because different rows in the same column of the results of these operations would be in different character sets.
- Metadata must include all characters in all languages. Otherwise, users would not be able to name columns and tables using their own languages.

To satisfy both requirements, MySQL stores metadata in a Unicode character set, namely UTF-8. This does not cause any disruption if you never use accented or non-Latin characters. But if you do, you should be aware that metadata is in UTF-8.

The metadata requirements mean that the return values of the `USER()`, `CURRENT_USER()`, `SESSION_USER()`, `SYSTEM_USER()`, `DATABASE()`, and `VERSION()` functions have the UTF-8 character set by default.

The server sets the `character_set_system` system variable to the name of the metadata character set:

```
mysql> SHOW VARIABLES LIKE 'character_set_system';
+----------------------+-------+
| Variable_name        | Value |
+----------------------+-------+
```

```
| character_set_system | utf8  |
+----------------------+-------+
```

Storage of metadata using Unicode does *not* mean that the server returns headers of columns and the results of `DESCRIBE` functions in the `character_set_system` character set by default. When you use `SELECT column1 FROM t`, the name `column1` itself is returned from the server to the client in the character set determined by the value of the `character_set_results` system variable, which has a default value of `utf8`. If you want the server to pass metadata results back in a different character set, use the `SET NAMES` statement to force the server to perform character set conversion. `SET NAMES` sets the `character_set_results` and other related system variables. (See Section 10.4, “Connection Character Sets and Collations”.) Alternatively, a client program can perform the conversion after receiving the result from the server. It is more efficient for the client to perform the conversion, but this option is not always available for all clients.

If `character_set_results` is set to `NULL`, no conversion is performed and the server returns metadata using its original character set (the set indicated by `character_set_system`).

Error messages returned from the server to the client are converted to the client character set automatically, as with metadata.

If you are using (for example) the `USER()` function for comparison or assignment within a single statement, don't worry. MySQL performs some automatic conversion for you.

```
SELECT * FROM t1 WHERE USER() = latin1_column;
```

This works because the contents of `latin1_column` are automatically converted to UTF-8 before the comparison.

```
INSERT INTO t1 (latin1_column) SELECT USER();
```

This works because the contents of `USER()` are automatically converted to `latin1` before the assignment.

Although automatic conversion is not in the SQL standard, the standard does say that every character set is (in terms of supported characters) a “subset” of Unicode. Because it is a well-known principle that “what applies to a superset can apply to a subset,” we believe that a collation for Unicode can apply for comparisons with non-Unicode strings. For more information about coercion of strings, see Section 10.8.4, “Collation Coercibility in Expressions”.

10.3 Specifying Character Sets and Collations

There are default settings for character sets and collations at four levels: server, database, table, and column. The description in the following sections may appear complex, but it has been found in practice that multiple-level defaulting leads to natural and obvious results.

`CHARACTER SET` is used in clauses that specify a character set. `CHARSET` can be used as a synonym for `CHARACTER SET`.

Character set issues affect not only data storage, but also communication between client programs and the MySQL server. If you want the client program to communicate with the server using a character set different from the default, you'll need to indicate which one. For example, to use the `utf8` Unicode character set, issue this statement after connecting to the server:

```
SET NAMES 'utf8';
```

For more information about character set-related issues in client/server communication, see Section 10.4, “Connection Character Sets and Collations”.

10.3.1 Collation Naming Conventions

MySQL collation names follow these conventions:

- A collation name starts with the name of the character set with which it is associated, generally followed by one or more suffixes indicating other collation characteristics. For example, `utf8_general_ci` and `latin1_swedish_ci` are collations for the `utf8` and `latin1` character sets, respectively. The `binary` character set has a single collation, also named `binary`, with no suffixes.
- A language-specific collation includes a language name. For example, `utf8_turkish_ci` and `utf8_hungarian_ci` sort characters for the `utf8` character set using the rules of Turkish and Hungarian, respectively.
- Collation suffixes indicate whether a collation is case-sensitive, accent-sensitive, or kana-sensitive (or some combination thereof), or binary. The following table shows the suffixes used to indicate these characteristics.

  Table 10.1 Collation Suffix Meanings

Suffix	Meaning
`_ai`	Accent-insensitive
`_as`	Accent-sensitive
`_ci`	Case-insensitive
`_cs`	Case-sensitive
`_bin`	Binary

  For nonbinary collation names that do not specify accent sensitivity, it is determined by case sensitivity. If a collation name does not contain `_ai` or `_as`, `_ci` in the name implies `_ai` and `_cs` in the name implies `_as`. For example, `latin1_general_ci` is explicitly case-insensitive and implicitly accent-insensitive, and `latin1_general_cs` is explicitly case-sensitive and implicitly accent-sensitive.

  For the `binary` collation of the `binary` character set, comparisons are based on numeric byte values. For the `_bin` collation of a nonbinary character set, comparisons are based on numeric character code values, which differ from byte values for multibyte characters. For information about the differences between the `binary` collation of the `binary` character set and the `_bin` collations of nonbinary character sets, see Section 10.8.5, “The binary Collation Compared to _bin Collations”.

- Collation names for Unicode character sets may include a version number to indicate the version of the Unicode Collation Algorithm (UCA) on which the collation is based. UCA-based collations without a version number in the name use the version-4.0.0 UCA weight keys. For example:
  - `utf8_unicode_520_ci` is based on UCA 5.2.0 weight keys (http://www.unicode.org/Public/UCA/5.2.0/allkeys.txt).
  - `utf8_unicode_ci` (with no version named) is based on UCA 4.0.0 weight keys (http://www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt).
- For Unicode character sets, the *`xxx`*`_general_mysql500_ci` collations preserve the pre-5.1.24 ordering of the original *`xxx`*`_general_ci` collations and permit upgrades for tables created before MySQL 5.1.24 (Bug #27877).

10.3.2 Server Character Set and Collation

MySQL Server has a server character set and a server collation. By default, these are `latin1` and `latin1_swedish_ci`, but they can be set explicitly at server startup on the command line or in an option file and changed at runtime.

Initially, the server character set and collation depend on the options that you use when you start `mysqld`. You can use `--character-set-server` for the character set. Along with it, you can add `--collation-server` for the collation. If you don't specify a character set, that is the same as saying

`--character-set-server=latin1`. If you specify only a character set (for example, `latin1`) but not a collation, that is the same as saying `--character-set-server=latin1 --collation-server=latin1_swedish_ci` because `latin1_swedish_ci` is the default collation for `latin1`. Therefore, the following three commands all have the same effect:

```
mysqld
mysqld --character-set-server=latin1
mysqld --character-set-server=latin1 \
  --collation-server=latin1_swedish_ci
```

One way to change the settings is by recompiling. To change the default server character set and collation when building from sources, use the `DEFAULT_CHARSET` and `DEFAULT_COLLATION` options for `CMake`. For example:

```
cmake . -DDEFAULT_CHARSET=latin1
```

Or:

```
cmake . -DDEFAULT_CHARSET=latin1 \
  -DDEFAULT_COLLATION=latin1_german1_ci
```

Both `mysqld` and `CMake` verify that the character set/collation combination is valid. If not, each program displays an error message and terminates.

The server character set and collation are used as default values if the database character set and collation are not specified in `CREATE DATABASE` statements. They have no other purpose.

The current server character set and collation can be determined from the values of the `character_set_server` and `collation_server` system variables. These variables can be changed at runtime.

10.3.3 Database Character Set and Collation

Every database has a database character set and a database collation. The `CREATE DATABASE` and `ALTER DATABASE` statements have optional clauses for specifying the database character set and collation:

```
CREATE DATABASE db_name
    [[DEFAULT] CHARACTER SET charset_name]
    [[DEFAULT] COLLATE collation_name]

ALTER DATABASE db_name
    [[DEFAULT] CHARACTER SET charset_name]
    [[DEFAULT] COLLATE collation_name]
```

The keyword `SCHEMA` can be used instead of `DATABASE`.

All database options are stored in a text file named `db.opt` that can be found in the database directory.

The `CHARACTER SET` and `COLLATE` clauses make it possible to create databases with different character sets and collations on the same MySQL server.

Example:

```
CREATE DATABASE db_name CHARACTER SET latin1 COLLATE latin1_swedish_ci;
```

MySQL chooses the database character set and database collation in the following manner:

- If both `CHARACTER SET charset_name` and `COLLATE collation_name` are specified, character set `charset_name` and collation `collation_name` are used.
- If `CHARACTER SET charset_name` is specified without `COLLATE`, character set `charset_name` and its default collation are used. To see the default collation for each character set, use the `SHOW CHARACTER SET` statement or query the `INFORMATION_SCHEMA CHARACTER_SETS` table.

- If `COLLATE collation_name` is specified without `CHARACTER SET`, the character set associated with `collation_name` and collation `collation_name` are used.
- Otherwise (neither `CHARACTER SET` nor `COLLATE` is specified), the server character set and server collation are used.

The character set and collation for the default database can be determined from the values of the `character_set_database` and `collation_database` system variables. The server sets these variables whenever the default database changes. If there is no default database, the variables have the same value as the corresponding server-level system variables, `character_set_server` and `collation_server`.

To see the default character set and collation for a given database, use these statements:

```
USE db_name;
SELECT @@character_set_database, @@collation_database;
```

Alternatively, to display the values without changing the default database:

```
SELECT DEFAULT_CHARACTER_SET_NAME, DEFAULT_COLLATION_NAME
FROM INFORMATION_SCHEMA.SCHEMATA WHERE SCHEMA_NAME = 'db_name';
```

The database character set and collation affect these aspects of server operation:

- For `CREATE TABLE` statements, the database character set and collation are used as default values for table definitions if the table character set and collation are not specified. To override this, provide explicit `CHARACTER SET` and `COLLATE` table options.
- For `LOAD DATA` statements that include no `CHARACTER SET` clause, the server uses the character set indicated by the `character_set_database` system variable to interpret the information in the file. To override this, provide an explicit `CHARACTER SET` clause.
- For stored routines (procedures and functions), the database character set and collation in effect at routine creation time are used as the character set and collation of character data parameters for which the declaration includes no `CHARACTER SET` or a `COLLATE` attribute. To override this, provide `CHARACTER SET` and `COLLATE` explicitly.

10.3.4 Table Character Set and Collation

Every table has a table character set and a table collation. The `CREATE TABLE` and `ALTER TABLE` statements have optional clauses for specifying the table character set and collation:

```
CREATE TABLE tbl_name (column_list)
    [[DEFAULT] CHARACTER SET charset_name]
    [COLLATE collation_name]]

ALTER TABLE tbl_name
    [[DEFAULT] CHARACTER SET charset_name]
    [COLLATE collation_name]
```

Example:

```
CREATE TABLE t1 ( ... )
CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

MySQL chooses the table character set and collation in the following manner:

- If both `CHARACTER SET charset_name` and `COLLATE collation_name` are specified, character set `charset_name` and collation `collation_name` are used.
- If `CHARACTER SET charset_name` is specified without `COLLATE`, character set `charset_name` and its default collation are used. To see the default collation for each character set, use the `SHOW CHARACTER SET` statement or query the `INFORMATION_SCHEMA CHARACTER_SETS` table.

- If `COLLATE collation_name` is specified without `CHARACTER SET`, the character set associated with `collation_name` and collation `collation_name` are used.
- Otherwise (neither `CHARACTER SET` nor `COLLATE` is specified), the database character set and collation are used.

The table character set and collation are used as default values for column definitions if the column character set and collation are not specified in individual column definitions. The table character set and collation are MySQL extensions; there are no such things in standard SQL.

10.3.5 Column Character Set and Collation

Every “character” column (that is, a column of type `CHAR`, `VARCHAR`, a `TEXT` type, or any synonym) has a column character set and a column collation. Column definition syntax for `CREATE TABLE` and `ALTER TABLE` has optional clauses for specifying the column character set and collation:

```
col_name {CHAR | VARCHAR | TEXT} (col_length)
    [CHARACTER SET charset_name]
    [COLLATE collation_name]
```

These clauses can also be used for `ENUM` and `SET` columns:

```
col_name {ENUM | SET} (val_list)
    [CHARACTER SET charset_name]
    [COLLATE collation_name]
```

Examples:

```
CREATE TABLE t1
(
    col1 VARCHAR(5)
      CHARACTER SET latin1
      COLLATE latin1_german1_ci
);

ALTER TABLE t1 MODIFY
    col1 VARCHAR(5)
      CHARACTER SET latin1
      COLLATE latin1_swedish_ci;
```

MySQL chooses the column character set and collation in the following manner:

- If both `CHARACTER SET charset_name` and `COLLATE collation_name` are specified, character set `charset_name` and collation `collation_name` are used.

  ```
  CREATE TABLE t1
  (
      col1 CHAR(10) CHARACTER SET utf8 COLLATE utf8_unicode_ci
  ) CHARACTER SET latin1 COLLATE latin1_bin;
  ```

  The character set and collation are specified for the column, so they are used. The column has character set `utf8` and collation `utf8_unicode_ci`.

- If `CHARACTER SET charset_name` is specified without `COLLATE`, character set `charset_name` and its default collation are used.

  ```
  CREATE TABLE t1
  (
      col1 CHAR(10) CHARACTER SET utf8
  ) CHARACTER SET latin1 COLLATE latin1_bin;
  ```

  The character set is specified for the column, but the collation is not. The column has character set `utf8` and the default collation for `utf8`, which is `utf8_general_ci`. To see the default collation for each character set, use the `SHOW CHARACTER SET` statement or query the `INFORMATION_SCHEMA CHARACTER_SETS` table.

- If `COLLATE collation_name` is specified without `CHARACTER SET`, the character set associated with `collation_name` and collation `collation_name` are used.

```
CREATE TABLE t1
(
    col1 CHAR(10) COLLATE utf8_polish_ci
) CHARACTER SET latin1 COLLATE latin1_bin;
```

  The collation is specified for the column, but the character set is not. The column has collation `utf8_polish_ci` and the character set is the one associated with the collation, which is `utf8`.

- Otherwise (neither `CHARACTER SET` nor `COLLATE` is specified), the table character set and collation are used.

```
CREATE TABLE t1
(
    col1 CHAR(10)
) CHARACTER SET latin1 COLLATE latin1_bin;
```

  Neither the character set nor collation is specified for the column, so the table defaults are used. The column has character set `latin1` and collation `latin1_bin`.

The `CHARACTER SET` and `COLLATE` clauses are standard SQL.

If you use `ALTER TABLE` to convert a column from one character set to another, MySQL attempts to map the data values, but if the character sets are incompatible, there may be data loss.

10.3.6 Character String Literal Character Set and Collation

Every character string literal has a character set and a collation.

For the simple statement `SELECT 'string'`, the string has the connection default character set and collation defined by the `character_set_connection` and `collation_connection` system variables.

A character string literal may have an optional character set introducer and `COLLATE` clause, to designate it as a string that uses a particular character set and collation:

```
[_charset_name]'string' [COLLATE collation_name]
```

The `_charset_name` expression is formally called an *introducer*. It tells the parser, “the string that follows uses character set `charset_name`.” An introducer does not change the string to the introducer character set like `CONVERT()` would do. It does not change the string value, although padding may occur. The introducer is just a signal. See Section 10.3.8, “Character Set Introducers”.

Examples:

```
SELECT 'abc';
SELECT _latin1'abc';
SELECT _binary'abc';
SELECT _utf8'abc' COLLATE utf8_danish_ci;
```

Character set introducers and the `COLLATE` clause are implemented according to standard SQL specifications.

MySQL determines the character set and collation of a character string literal in the following manner:

- If both `_charset_name` and `COLLATE collation_name` are specified, character set `charset_name` and collation `collation_name` are used. `collation_name` must be a permitted collation for `charset_name`.

- If `_charset_name` is specified but `COLLATE` is not specified, character set `charset_name` and its default collation are used. To see the default collation for each character set, use the `SHOW CHARACTER SET` statement or query the `INFORMATION_SCHEMA CHARACTER_SETS` table.

- If *`_charset_name`* is not specified but `COLLATE` *`collation_name`* is specified, the connection default character set given by the `character_set_connection` system variable and collation *`collation_name`* are used. *`collation_name`* must be a permitted collation for the connection default character set.
- Otherwise (neither *`_charset_name`* nor `COLLATE` *`collation_name`* is specified), the connection default character set and collation given by the `character_set_connection` and `collation_connection` system variables are used.

Examples:

- A nonbinary string with `latin1` character set and `latin1_german1_ci` collation:

```
SELECT _latin1'Müller' COLLATE latin1_german1_ci;
```

- A nonbinary string with `utf8` character set and its default collation (that is, `utf8_general_ci`):

```
SELECT _utf8'Müller';
```

- A binary string with `binary` character set and its default collation (that is, `binary`):

```
SELECT _binary'Müller';
```

- A nonbinary string with the connection default character set and `utf8_general_ci` collation (fails if the connection character set is not `utf8`):

```
SELECT 'Müller' COLLATE utf8_general_ci;
```

- A string with the connection default character set and collation:

```
SELECT 'Müller';
```

An introducer indicates the character set for the following string, but does not change how the parser performs escape processing within the string. Escapes are always interpreted by the parser according to the character set given by `character_set_connection`.

The following examples show that escape processing occurs using `character_set_connection` even in the presence of an introducer. The examples use `SET NAMES` (which changes `character_set_connection`, as discussed in Section 10.4, “Connection Character Sets and Collations”), and display the resulting strings using the `HEX()` function so that the exact string contents can be seen.

Example 1:

```
mysql> SET NAMES latin1;
mysql> SELECT HEX('à\n'), HEX(_sjis'à\n');
+------------+-----------------+
| HEX('à\n')  | HEX(_sjis'à\n')  |
+------------+-----------------+
| E00A       | E00A            |
+------------+-----------------+
```

Here, `à` (hexadecimal value `E0`) is followed by `\n`, the escape sequence for newline. The escape sequence is interpreted using the `character_set_connection` value of `latin1` to produce a literal newline (hexadecimal value `0A`). This happens even for the second string. That is, the `_sjis` introducer does not affect the parser's escape processing.

Example 2:

```
mysql> SET NAMES sjis;
mysql> SELECT HEX('à\n'), HEX(_latin1'à\n');
+------------+------------------+
| HEX('à\n')  | HEX(_latin1'à\n')  |
+------------+------------------+
| E05C6E     | E05C6E           |
+------------+------------------+
```

Here, `character_set_connection` is `sjis`, a character set in which the sequence of `à` followed by `\` (hexadecimal values `05` and `5C`) is a valid multibyte character. Hence, the first two bytes of the string are interpreted as a single `sjis` character, and the `\` is not interpreted as an escape character. The following `n` (hexadecimal value `6E`) is not interpreted as part of an escape sequence. This is true even for the second string; the `_latin1` introducer does not affect escape processing.

10.3.7 The National Character Set

Standard SQL defines `NCHAR` or `NATIONAL CHAR` as a way to indicate that a `CHAR` column should use some predefined character set. MySQL uses `utf8` as this predefined character set. For example, these data type declarations are equivalent:

```
CHAR(10) CHARACTER SET utf8
NATIONAL CHARACTER(10)
NCHAR(10)
```

As are these:

```
VARCHAR(10) CHARACTER SET utf8
NATIONAL VARCHAR(10)
NVARCHAR(10)
NCHAR VARCHAR(10)
NATIONAL CHARACTER VARYING(10)
NATIONAL CHAR VARYING(10)
```

You can use `N'literal'` (or `n'literal'`) to create a string in the national character set. These statements are equivalent:

```
SELECT N'some text';
SELECT n'some text';
SELECT _utf8'some text';
```

10.3.8 Character Set Introducers

A character string literal, hexadecimal literal, or bit-value literal may have an optional character set introducer and `COLLATE` clause, to designate it as a string that uses a particular character set and collation:

```
[_charset_name] literal [COLLATE collation_name]
```

The *`_charset_name`* expression is formally called an *introducer*. It tells the parser, “the string that follows uses character set *`charset_name`*.” An introducer does not change the string to the introducer character set like `CONVERT()` would do. It does not change the string value, although padding may occur. The introducer is just a signal.

For character string literals, space between the introducer and the string is permitted but optional.

For character set literals, an introducer indicates the character set for the following string, but does not change how the parser performs escape processing within the string. Escapes are always interpreted by the parser according to the character set given by `character_set_connection`. For additional discussion and examples, see Section 10.3.6, “Character String Literal Character Set and Collation”.

Examples:

```
SELECT 'abc';
SELECT _latin1'abc';
SELECT _binary'abc';
SELECT _utf8'abc' COLLATE utf8_danish_ci;

SELECT _latin1 X'4D7953514C';
SELECT _utf8 0x4D7953514C COLLATE utf8_danish_ci;

SELECT _latin1 b'1000001';
SELECT _utf8 0b1000001 COLLATE utf8_danish_ci;
```

Character set introducers and the COLLATE clause are implemented according to standard SQL specifications.

Character string literals can be designated as binary strings by using the _binary introducer. Hexadecimal literals and bit-value literals are binary strings by default, so _binary is permitted, but unnecessary.

MySQL determines the character set and collation of a character string literal, hexadecimal literal, or bit-value literal in the following manner:

- If both *_charset_name* and COLLATE *collation_name* are specified, character set *charset_name* and collation *collation_name* are used. *collation_name* must be a permitted collation for *charset_name*.
- If *_charset_name* is specified but COLLATE is not specified, character set *charset_name* and its default collation are used. To see the default collation for each character set, use the SHOW CHARACTER SET statement or query the INFORMATION_SCHEMA CHARACTER_SETS table.
- If *_charset_name* is not specified but COLLATE *collation_name* is specified:
  - For a character string literal, the connection default character set given by the character_set_connection system variable and collation *collation_name* are used. *collation_name* must be a permitted collation for the connection default character set.
  - For a hexadecimal literal or bit-value literal, the only permitted collation is binary because these types of literals are binary strings by default.
- Otherwise (neither *_charset_name* nor COLLATE *collation_name* is specified):
  - For a character string literal, the connection default character set and collation given by the character_set_connection and collation_connection system variables are used.
  - For a hexadecimal literal or bit-value literal, the character set and collation are binary.

Examples:

- Nonbinary strings with latin1 character set and latin1_german1_ci collation:

```
SELECT _latin1'Müller' COLLATE latin1_german1_ci;
SELECT _latin1 X'0A0D' COLLATE latin1_german1_ci;
SELECT _latin1 b'0110' COLLATE latin1_german1_ci;
```

- Nonbinary strings with utf8 character set and its default collation (that is, utf8_general_ci):

```
SELECT _utf8'Müller';
SELECT _utf8 X'0A0D';
SELECT _utf8 b'0110';
```

- Binary strings with binary character set and its default collation (that is, binary):

```
SELECT _binary'Müller';
SELECT X'0A0D';
SELECT b'0110';
```

  The hexadecimal literal and bit-value literal need no introducer because they are binary strings by default.

- A nonbinary string with the connection default character set and utf8_general_ci collation (fails if the connection character set is not utf8):

```
SELECT 'Müller' COLLATE utf8_general_ci;
```

  This construction (COLLATE only) does not work for hexadecimal literals or bit literals because their character set is binary no matter the connection character set, and binary is not compatible

with the `utf8_general_ci` collation. The only permitted `COLLATE` clause in the absence of an introducer is `COLLATE binary`.

- A string with the connection default character set and collation:

```
SELECT 'Müller';
```

10.3.9 Examples of Character Set and Collation Assignment

The following examples show how MySQL determines default character set and collation values.

Example 1: Table and Column Definition

```
CREATE TABLE t1
(
    c1 CHAR(10) CHARACTER SET latin1 COLLATE latin1_german1_ci
) DEFAULT CHARACTER SET latin2 COLLATE latin2_bin;
```

Here we have a column with a `latin1` character set and a `latin1_german1_ci` collation. The definition is explicit, so that is straightforward. Notice that there is no problem with storing a `latin1` column in a `latin2` table.

Example 2: Table and Column Definition

```
CREATE TABLE t1
(
    c1 CHAR(10) CHARACTER SET latin1
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

This time we have a column with a `latin1` character set and a default collation. Although it might seem natural, the default collation is not taken from the table level. Instead, because the default collation for `latin1` is always `latin1_swedish_ci`, column `c1` has a collation of `latin1_swedish_ci` (not `latin1_danish_ci`).

Example 3: Table and Column Definition

```
CREATE TABLE t1
(
    c1 CHAR(10)
) DEFAULT CHARACTER SET latin1 COLLATE latin1_danish_ci;
```

We have a column with a default character set and a default collation. In this circumstance, MySQL checks the table level to determine the column character set and collation. Consequently, the character set for column `c1` is `latin1` and its collation is `latin1_danish_ci`.

Example 4: Database, Table, and Column Definition

```
CREATE DATABASE d1
    DEFAULT CHARACTER SET latin2 COLLATE latin2_czech_ci;
USE d1;
CREATE TABLE t1
(
    c1 CHAR(10)
);
```

We create a column without specifying its character set and collation. We're also not specifying a character set and a collation at the table level. In this circumstance, MySQL checks the database level to determine the table settings, which thereafter become the column settings.) Consequently, the character set for column `c1` is `latin2` and its collation is `latin2_czech_ci`.

10.3.10 Compatibility with Other DBMSs

For MaxDB compatibility these two statements are the same:

```
CREATE TABLE t1 (f1 CHAR(N) UNICODE);
CREATE TABLE t1 (f1 CHAR(N) CHARACTER SET ucs2);
```

10.4 Connection Character Sets and Collations

A “connection” is what a client program makes when it connects to the server, to begin a session within which it interacts with the server. The client sends SQL statements, such as queries, over the session connection. The server sends responses, such as result sets or error messages, over the connection back to the client.

- Connection Character Set and Collation System Variables
- Impermissible Client Character Sets
- Client Program Connection Character Set Configuration
- SQL Statements for Connection Character Set Configuration
- Connection Character Set Error Handling

Connection Character Set and Collation System Variables

Several character set and collation system variables relate to a client's interaction with the server. Some of these have been mentioned in earlier sections:

- The `character_set_server` and `collation_server` system variables indicate the server character set and collation. See Section 10.3.2, “Server Character Set and Collation”.
- The `character_set_database` and `collation_database` system variables indicate the character set and collation of the default database. See Section 10.3.3, “Database Character Set and Collation”.

Additional character set and collation system variables are involved in handling traffic for the connection between a client and the server. Every client has session-specific connection-related character set and collation system variables. These session system variable values are initialized at connect time, but can be changed within the session.

Several questions about character set and collation handling for client connections can be answered in terms of system variables:

- What character set are statements in when they leave the client?

  The server takes the `character_set_client` system variable to be the character set in which statements are sent by the client.

  Note

  Some character sets cannot be used as the client character set. See Impermissible Client Character Sets.

- What character set should the server translate statements to after receiving them?

  To determine this, the server uses the `character_set_connection` and `collation_connection` system variables:

  - The server converts statements sent by the client from `character_set_client` to `character_set_connection`. Exception: For string literals that have an introducer such as `_utf8mb4` or `_latin2`, the introducer determines the character set. See Section 10.3.8, “Character Set Introducers”.
  - `collation_connection` is important for comparisons of literal strings. For comparisons of strings with column values, `collation_connection` does not matter because columns have their own collation, which has a higher collation precedence (see Section 10.8.4, “Collation Coercibility in Expressions”).

- What character set should the server translate query results to before shipping them back to the client?

  The `character_set_results` system variable indicates the character set in which the server returns query results to the client. This includes result data such as column values, result metadata such as column names, and error messages.

  To tell the server to perform no conversion of result sets or error messages, set `character_set_results` to `NULL` or `binary`:

  ```
  SET character_set_results = NULL;
  SET character_set_results = binary;
  ```

  For more information about character sets and error messages, see Section 10.6, “Error Message Character Set”.

To see the values of the character set and collation system variables that apply to the current session, use this statement:

```
SELECT * FROM INFORMATION_SCHEMA.SESSION_VARIABLES
WHERE VARIABLE_NAME IN (
'character_set_client', 'character_set_connection',
'character_set_results', 'collation_connection'
) ORDER BY VARIABLE_NAME;
```

The following simpler statements also display the connection variables, but include other related variables as well. They can be useful to see *all* character set and collation system variables:

```
SHOW SESSION VARIABLES LIKE 'character\_set\_%';
SHOW SESSION VARIABLES LIKE 'collation\_%';
```

Clients can fine-tune the settings for these variables, or depend on the defaults (in which case, you can skip the rest of this section). If you do not use the defaults, you must change the character settings *for each connection to the server.*

Impermissible Client Character Sets

The `character_set_client` system variable cannot be set to certain character sets:

```
ucs2
utf16
utf16le
utf32
```

Attempting to use any of those character sets as the client character set produces an error:

```
mysql> SET character_set_client = 'ucs2';
ERROR 1231 (42000): Variable 'character_set_client'
can't be set to the value of 'ucs2'
```

The same error occurs if any of those character sets are used in the following contexts, all of which result in an attempt to set `character_set_client` to the named character set:

- The `--default-character-set=`*`charset_name`* command option used by MySQL client programs such as `mysql` and `mysqladmin`.
- The `SET NAMES '`*`charset_name`*`'` statement.
- The `SET CHARACTER SET '`*`charset_name`*`'` statement.

Client Program Connection Character Set Configuration

When a client connects to the server, it indicates which character set it wants to use for communication with the server. (Actually, the client indicates the default collation for that character set, from

which the server can determine the character set.) The server uses this information to set the `character_set_client`, `character_set_results`, `character_set_connection` system variables to the character set, and `collation_connection` to the character set default collation. In effect, the server performs the equivalent of a `SET NAMES` operation.

If the server does not support the requested character set or collation, it falls back to using the server character set and collation to configure the connection. For additional detail about this fallback behavior, see Connection Character Set Error Handling.

The `mysql`, `mysqladmin`, `mysqlcheck`, `mysqlimport`, and `mysqlshow` client programs determine the default character set to use as follows:

- In the absence of other information, each client uses the compiled-in default character set, usually `latin1`.
- Each client can autodetect which character set to use based on the operating system setting, such as the value of the `LANG` or `LC_ALL` locale environment variable on Unix systems or the code page setting on Windows systems. For systems on which the locale is available from the OS, the client uses it to set the default character set rather than using the compiled-in default. For example, setting `LANG` to `ru_RU.KOI8-R` causes the `koi8r` character set to be used. Thus, users can configure the locale in their environment for use by MySQL clients.

  The OS character set is mapped to the closest MySQL character set if there is no exact match. If the client does not support the matching character set, it uses the compiled-in default. For example, `ucs2` is not supported as a connection character set, so it maps to the compiled-in default.

  C applications can use character set autodetection based on the OS setting by invoking `mysql_options()` as follows before connecting to the server:

```
mysql_options(mysql,
              MYSQL_SET_CHARSET_NAME,
              MYSQL_AUTODETECT_CHARSET_NAME);
```

- Each client supports a `--default-character-set` option, which enables users to specify the character set explicitly to override whatever default the client otherwise determines.

Note

Some character sets cannot be used as the client character set. Attempting to use them with `--default-character-set` produces an error. See Impermissible Client Character Sets.

With the `mysql` client, to use a character set different from the default, you could explicitly execute a `SET NAMES` statement every time you connect to the server (see Client Program Connection Character Set Configuration). To accomplish the same result more easily, specify the character set in your option file. For example, the following option file setting changes the three connection-related character set system variables set to `koi8r` each time you invoke `mysql`:

```
[mysql]
default-character-set=koi8r
```

If you are using the `mysql` client with auto-reconnect enabled (which is not recommended), it is preferable to use the `charset` command rather than `SET NAMES`. For example:

```
mysql> charset koi8r
Charset changed
```

The `charset` command issues a `SET NAMES` statement, and also changes the default character set that `mysql` uses when it reconnects after the connection has dropped.

When configuration client programs, you must also consider the environment within which they execute. See Section 10.5, “Configuring Application Character Set and Collation”.

SQL Statements for Connection Character Set Configuration

After a connection has been established, clients can change the character set and collation system variables for the current session. These variables can be changed individually using `SET` statements, but two more convenient statements affect the connection-related character set sytem variables as a group:

- `SET NAMES 'charset_name' [COLLATE 'collation_name']`

  `SET NAMES` indicates what character set the client uses to send SQL statements to the server. Thus, `SET NAMES 'cp1251'` tells the server, "future incoming messages from this client are in character set `cp1251`." It also specifies the character set that the server should use for sending results back to the client. (For example, it indicates what character set to use for column values if you use a `SELECT` statement that produces a result set.)

  A `SET NAMES 'charset_name'` statement is equivalent to these three statements:

  ```
  SET character_set_client = charset_name;
  SET character_set_results = charset_name;
  SET character_set_connection = charset_name;
  ```

  Setting `character_set_connection` to *`charset_name`* also implicitly sets `collation_connection` to the default collation for *`charset_name`*. It is unnecessary to set that collation explicitly. To specify a particular collation to use for `collation_connection`, add a `COLLATE` clause:

  ```
  SET NAMES 'charset_name' COLLATE 'collation_name'
  ```

- `SET CHARACTER SET 'charset_name'`

  `SET CHARACTER SET` is similar to `SET NAMES` but sets `character_set_connection` and `collation_connection` to `character_set_database` and `collation_database` (which, as mentioned previously, indicate the character set and collation of the default database).

  A `SET CHARACTER SET charset_name` statement is equivalent to these three statements:

  ```
  SET character_set_client = charset_name;
  SET character_set_results = charset_name;
  SET collation_connection = @@collation_database;
  ```

  Setting `collation_connection` also implicitly sets `character_set_connection` to the character set associated with the collation (equivalent to executing `SET character_set_connection = @@character_set_database`). It is unnecessary to set `character_set_connection` explicitly.

Note

Some character sets cannot be used as the client character set. Attempting to use them with `SET NAMES` or `SET CHARACTER SET` produces an error. See Impermissible Client Character Sets.

Example: Suppose that `column1` is defined as `CHAR(5) CHARACTER SET latin2`. If you do not say `SET NAMES` or `SET CHARACTER SET`, then for `SELECT column1 FROM t`, the server sends back all the values for `column1` using the character set that the client specified when it connected. On the other hand, if you say `SET NAMES 'latin1'` or `SET CHARACTER SET 'latin1'` before issuing the `SELECT` statement, the server converts the `latin2` values to `latin1` just before sending results back. Conversion may be lossy for characters that are not in both character sets.

Connection Character Set Error Handling

Attempts to use an inappropriate connection character set or collation can produce an error, or cause the server to fall back to its default character set and collation for a given connection. This section

describes problems that can occur when configuring the connection character set. These problems can occur when establishing a connection or when changing the character set within an established connection.

- Connect-Time Error Handling
- Runtime Error Handling

Connect-Time Error Handling

Some character sets cannot be used as the client character set; see Impermissible Client Character Sets. If you specify a character set that is valid but not permitted as a client character set, the server returns an error:

```
shell> mysql --default-character-set=ucs2
ERROR 1231 (42000): Variable 'character_set_client' can't be set to
the value of 'ucs2'
```

If you specify a character set that the client does not recognize, it produces an error:

```
shell> mysql --default-character-set=bogus
mysql: Character set 'bogus' is not a compiled character set and is
not specified in the '/usr/local/mysql/share/charsets/Index.xml' file
ERROR 2019 (HY000): Can't initialize character set bogus
(path: /usr/local/mysql/share/charsets/)
```

If you specify a character set that the client recognizes but the server does not, the server falls back to its default character set and collation. Suppose that the server is configured to use `latin1` and `latin1_swedish_ci` as its defaults, and that it does not recognize `gb18030` as a valid character set. A client that specifies `--default-character-set=gb18030` is able to connect to the server, but the resulting character set is not what the client wants:

```
mysql> SHOW SESSION VARIABLES LIKE 'character\_set\_%';
+--------------------------+--------+
| Variable_name            | Value  |
+--------------------------+--------+
| character_set_client     | latin1 |
| character_set_connection | latin1 |
...
| character_set_results    | latin1 |
...
+--------------------------+--------+
mysql> SHOW SESSION VARIABLES LIKE 'collation_connection';
+----------------------+-------------------+
| Variable_name        | Value             |
+----------------------+-------------------+
| collation_connection | latin1_swedish_ci |
+----------------------+-------------------+
```

You can see that the connection system variables have been set to reflect a character set and collation of `latin1` and `latin1_swedish_ci`. This occurs because the server cannot satisfy the client character set request and falls back to its defaults.

In this case, the client cannot use the character set that it wants because the server does not support it. The client must either be willing to use a different character set, or connect to a different server that supports the desired character set.

The same problem occurs in a more subtle context: When the client tells the server to use a character set that the server recognizes, but the default collation for that character set on the client side is not known on the server side. This occurs, for example, when a MySQL 8.0 client wants to connect to a MySQL 5.7 server using `utf8mb4` as the client character set. A client that specifies `--default-character-set=utf8mb4` is able to connect to the server. However, as in the previous example, the server falls back to its default character set and collation, not what the client requested:

```
mysql> SHOW SESSION VARIABLES LIKE 'character\_set\_%';
+--------------------------+--------+
| Variable_name            | Value  |
```

```
+--------------------------+--------+
| character_set_client     | latin1 |
| character_set_connection | latin1 |
...
| character_set_results    | latin1 |
...
+--------------------------+--------+
mysql> SHOW SESSION VARIABLES LIKE 'collation_connection';
+----------------------+-------------------+
| Variable_name        | Value             |
+----------------------+-------------------+
| collation_connection | latin1_swedish_ci |
+----------------------+-------------------+
```

Why does this occur? After all, `utf8mb4` is known to the 8.0 client and the 5.7 server, so both of them recognize it. To understand this behavior, it is necessary to understand that when the client tells the server which character set it wants to use, it really tells the server the default collation for that character set. Therefore, the aforementioned behavior occurs due to a combination of factors:

- The default collation for `utf8mb4` differs between MySQL 5.7 and 8.0 (`utf8mb4_general_ci` for 5.7, `utf8mb4_0900_ai_ci` for 8.0).
- When the 8.0 client requests a character set of `utf8mb4`, what it sends to the server is the default 8.0 `utf8mb4` collation; that is, the `utf8mb4_0900_ai_ci`.
- `utf8mb4_0900_ai_ci` is implemented only as of MySQL 8.0, so the 5.7 server does not recognize it.
- Because the 5.7 server does not recognize `utf8mb4_0900_ai_ci`, it cannot satisfy the client character set request, and falls back to its default character set and collation (`latin1` and `latin1_swedish_ci`).

In this case, the client can still use `utf8mb4` by issuing a `SET NAMES 'utf8mb4'` statement after connecting. The resulting collation is the 5.7 default `utf8mb4` collation; that is, `utf8mb4_general_ci`. If the client additionally wants a collation of `utf8mb4_0900_ai_ci`, it cannot achieve that because the server does not recognize that collation. The client must either be willing to use a different `utf8mb4` collation, or connect to a server from MySQL 8.0 or higher.

Runtime Error Handling

Within an established connection, the client can request a change of connection character set and collation with `SET NAMES` or `SET CHARACTER SET`.

Some character sets cannot be used as the client character set; see Impermissible Client Character Sets. If you specify a character set that is valid but not permitted as a client character set, the server returns an error:

```
mysql> SET NAMES 'ucs2';
ERROR 1231 (42000): Variable 'character_set_client' can't be set to
the value of 'ucs2'
```

If the server does not recognize the character set (or the collation), it produces an error:

```
mysql> SET NAMES 'bogus';
ERROR 1115 (42000): Unknown character set: 'bogus'

mysql> SET NAMES 'utf8mb4' COLLATE 'bogus';
ERROR 1273 (HY000): Unknown collation: 'bogus'
```

Tip

A client that wants to verify whether its requested character set was honored by the server can execute the following statement after connecting and checking that the result is the expected character set:

```
SELECT @@character_set_client;
```

10.5 Configuring Application Character Set and Collation

For applications that store data using the default MySQL character set and collation (`latin1`, `latin1_swedish_ci`), no special configuration should be needed. If applications require data storage using a different character set or collation, you can configure character set information several ways:

- Specify character settings per database. For example, applications that use one database might use the default of `latin1`, whereas applications that use another database might use `sjis`.
- Specify character settings at server startup. This causes the server to use the given settings for all applications that do not make other arrangements.
- Specify character settings at configuration time, if you build MySQL from source. This causes the server to use the given settings as the defaults for all applications, without having to specify them at server startup.

When different applications require different character settings, the per-database technique provides a good deal of flexibility. If most or all applications use the same character set, specifying character settings at server startup or configuration time may be most convenient.

For the per-database or server-startup techniques, the settings control the character set for data storage. Applications must also tell the server which character set to use for client/server communications, as described in the following instructions.

The examples shown here assume use of the `utf8` character set and `utf8_general_ci` collation in particular contexts as an alternative to the defaults of `latin1` and `latin1_swedish_ci`.

- **Specify character settings per database.** To create a database such that its tables use a given default character set and collation for data storage, use a `CREATE DATABASE` statement like this:

  ```
  CREATE DATABASE mydb
    CHARACTER SET utf8
    COLLATE utf8_general_ci;
  ```

  Tables created in the database use `utf8` and `utf8_general_ci` by default for any character columns.

  Applications that use the database should also configure their connection to the server each time they connect. This can be done by executing a `SET NAMES 'utf8'` statement after connecting. The statement can be used regardless of connection method (the `mysql` client, PHP scripts, and so forth).

  In some cases, it may be possible to configure the connection to use the desired character set some other way. For example, to connect using `mysql`, you can specify the `--default-character-set=utf8` command-line option to achieve the same effect as `SET NAMES 'utf8'`.

  For more information about configuring client connections, see Section 10.4, “Connection Character Sets and Collations”.

  Note

  If you use `ALTER DATABASE` to change the database default character set or collation, existing stored routines in the database that use those defaults must be dropped and recreated so that they use the new defaults. (In a stored routine, variables with character data types use the database defaults if the character set or collation are not specified explicitly. See Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”.)

- **Specify character settings at server startup.** To select a character set and collation at server startup, use the `--character-set-server` and `--collation-server` options. For example, to specify the options in an option file, include these lines:

```
[mysqld]
character-set-server=utf8
collation-server=utf8_general_ci
```

These settings apply server-wide and apply as the defaults for databases created by any application, and for tables created in those databases.

It is still necessary for applications to configure their connection using `SET NAMES` or equivalent after they connect, as described previously. You might be tempted to start the server with the `--init_connect="SET NAMES 'utf8'"` option to cause `SET NAMES` to be executed automatically for each client that connects. However, this may yield inconsistent results because the `init_connect` value is not executed for users who have the `SUPER` privilege.

- **Specify character settings at MySQL configuration time.** To select a character set and collation if you configure and build MySQL from source, use the `DEFAULT_CHARSET` and `DEFAULT_COLLATION` `CMake` options:

```
cmake . -DDEFAULT_CHARSET=utf8 \
  -DDEFAULT_COLLATION=utf8_general_ci
```

The resulting server uses `utf8` and `utf8_general_ci` as the default for databases and tables and for client connections. It is unnecessary to use `--character-set-server` and `--collation-server` to specify those defaults at server startup. It is also unnecessary for applications to configure their connection using `SET NAMES` or equivalent after they connect to the server.

Regardless of how you configure the MySQL character set for application use, you must also consider the environment within which those applications execute. For example, if you send statements using UTF-8 text taken from a file that you create in an editor, you should edit the file with the locale of your environment set to UTF-8 so that the file encoding is correct and so that the operating system handles it correctly. If you use the `mysql` client from within a terminal window, the window must be configured to use UTF-8 or characters may not display properly. For a script that executes in a Web environment, the script must handle character encoding properly for its interaction with the MySQL server, and it must generate pages that correctly indicate the encoding so that browsers know how to display the content of the pages. For example, you can include this `<meta>` tag within your `<head>` element:

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
```

10.6 Error Message Character Set

This section describes how the MySQL server uses character sets for constructing error messages. For information about the language of error messages (rather than the character set), see Section 10.12, “Setting the Error Message Language”. For general information about configuring error logging, see Section 5.4.2, “The Error Log”.

- Character Set for Error Message Construction
- Character Set for Error Message Disposition

Character Set for Error Message Construction

The server constructs error messages as follows:

- The message template uses UTF-8 (`utf8mb3`).
- Parameters in the message template are replaced with values that apply to a specific error occurrence:
  - Identifiers such as table or column names use UTF-8 internally so they are copied as is.
  - Character (nonbinary) string values are converted from their character set to UTF-8.

- Binary string values are copied as is for bytes in the range `0x20` to `0x7E`, and using `\x` hexadecimal encoding for bytes outside that range. For example, if a duplicate-key error occurs for an attempt to insert `0x41CF9F` into a `VARBINARY` unique column, the resulting error message uses UTF-8 with some bytes hexadecimal encoded:

  ```
  Duplicate entry 'A\xC3\x9F' for key 1
  ```

Character Set for Error Message Disposition

An error message, once constructed, can be written by the server to the error log or sent to clients:

- If the server writes the error message to the error log, it writes it in UTF-8, as constructed, without conversion to another character set.
- If the server sends the error message to a client program, the server converts it from UTF-8 to the character set specified by the `character_set_results` system variable. If `character_set_results` has a value of `NULL` or `binary`, no conversion occurs. No conversion occurs if the variable value is `utf8mb3` or `utf8mb4`, either, because those character sets have a repertoire that includes all UTF-8 characters used in message construction.

  If characters cannot be represented in `character_set_results`, some encoding may occur during the conversion. The encoding uses Unicode code point values:

  - Characters in the Basic Multilingual Plane (BMP) range (`0x0000` to `0xFFFF`) are written using `\nnnn` notation.
  - Characters outside the BMP range (`0x10000` to `0x10FFFF`) are written using `\+nnnnnn` notation.

  Clients can set `character_set_results` to control the character set in which they receive error messages. The variable can be set directly, or indirectly by means such as `SET NAMES`. For more information about `character_set_results`, see Section 10.4, “Connection Character Sets and Collations”.

10.7 Column Character Set Conversion

To convert a binary or nonbinary string column to use a particular character set, use `ALTER TABLE`. For successful conversion to occur, one of the following conditions must apply:

- If the column has a binary data type (`BINARY`, `VARBINARY`, `BLOB`), all the values that it contains must be encoded using a single character set (the character set you're converting the column to). If you use a binary column to store information in multiple character sets, MySQL has no way to know which values use which character set and cannot convert the data properly.
- If the column has a nonbinary data type (`CHAR`, `VARCHAR`, `TEXT`), its contents should be encoded in the column character set, not some other character set. If the contents are encoded in a different character set, you can convert the column to use a binary data type first, and then to a nonbinary column with the desired character set.

Suppose that a table `t` has a binary column named `col1` defined as `VARBINARY(50)`. Assuming that the information in the column is encoded using a single character set, you can convert it to a nonbinary column that has that character set. For example, if `col1` contains binary data representing characters in the `greek` character set, you can convert it as follows:

```
ALTER TABLE t MODIFY col1 VARCHAR(50) CHARACTER SET greek;
```

If your original column has a type of `BINARY(50)`, you could convert it to `CHAR(50)`, but the resulting values are padded with `0x00` bytes at the end, which may be undesirable. To remove these bytes, use the `TRIM()` function:

```
UPDATE t SET col1 = TRIM(TRAILING 0x00 FROM col1);
```

Suppose that table `t` has a nonbinary column named `col1` defined as `CHAR(50) CHARACTER SET latin1` but you want to convert it to use `utf8` so that you can store values from many languages. The following statement accomplishes this:

```
ALTER TABLE t MODIFY col1 CHAR(50) CHARACTER SET utf8;
```

Conversion may be lossy if the column contains characters that are not in both character sets.

A special case occurs if you have old tables from before MySQL 4.1 where a nonbinary column contains values that actually are encoded in a character set different from the server's default character set. For example, an application might have stored `sjis` values in a column, even though MySQL's default character set was different. It is possible to convert the column to use the proper character set but an additional step is required. Suppose that the server's default character set was `latin1` and `col1` is defined as `CHAR(50)` but its contents are `sjis` values. The first step is to convert the column to a binary data type, which removes the existing character set information without performing any character conversion:

```
ALTER TABLE t MODIFY col1 BLOB;
```

The next step is to convert the column to a nonbinary data type with the proper character set:

```
ALTER TABLE t MODIFY col1 CHAR(50) CHARACTER SET sjis;
```

This procedure requires that the table not have been modified already with statements such as `INSERT` or `UPDATE` after an upgrade to MySQL 4.1 or higher. In that case, MySQL would store new values in the column using `latin1`; the column contains a mix of `sjis` and `latin1` values and cannot be converted properly.

If you specified attributes when creating a column initially, you should also specify them when altering the table with `ALTER TABLE`. For example, if you specified `NOT NULL` and an explicit `DEFAULT` value, you should also provide them in the `ALTER TABLE` statement. Otherwise, the resulting column definition does not include those attributes.

To convert all character columns in a table, the `ALTER TABLE ... CONVERT TO CHARACTER SET` *`charset`* statement may be useful. See Section 13.1.7, “ALTER TABLE Statement”.

10.8 Collation Issues

The following sections discuss various aspects of character set collations.

10.8.1 Using COLLATE in SQL Statements

With the `COLLATE` clause, you can override whatever the default collation is for a comparison. `COLLATE` may be used in various parts of SQL statements. Here are some examples:

- With `ORDER BY`:

  ```
  SELECT k
  FROM t1
  ORDER BY k COLLATE latin1_german2_ci;
  ```

- With `AS`:

  ```
  SELECT k COLLATE latin1_german2_ci AS k1
  FROM t1
  ORDER BY k1;
  ```

- With `GROUP BY`:

  ```
  SELECT k
  FROM t1
  GROUP BY k COLLATE latin1_german2_ci;
  ```

- With aggregate functions:

```
SELECT MAX(k COLLATE latin1_german2_ci)
FROM t1;
```

- With `DISTINCT`:

```
SELECT DISTINCT k COLLATE latin1_german2_ci
FROM t1;
```

- With `WHERE`:

```
    SELECT *
    FROM t1
    WHERE _latin1 'Müller' COLLATE latin1_german2_ci = k;
```

```
    SELECT *
    FROM t1
    WHERE k LIKE _latin1 'Müller' COLLATE latin1_german2_ci;
```

- With `HAVING`:

```
SELECT k
FROM t1
GROUP BY k
HAVING k = _latin1 'Müller' COLLATE latin1_german2_ci;
```

10.8.2 COLLATE Clause Precedence

The `COLLATE` clause has high precedence (higher than `||`), so the following two expressions are equivalent:

```
x || y COLLATE z
x || (y COLLATE z)
```

10.8.3 Character Set and Collation Compatibility

Each character set has one or more collations, but each collation is associated with one and only one character set. Therefore, the following statement causes an error message because the `latin2_bin` collation is not legal with the `latin1` character set:

```
mysql> SELECT _latin1 'x' COLLATE latin2_bin;
ERROR 1253 (42000): COLLATION 'latin2_bin' is not valid
for CHARACTER SET 'latin1'
```

10.8.4 Collation Coercibility in Expressions

In the great majority of statements, it is obvious what collation MySQL uses to resolve a comparison operation. For example, in the following cases, it should be clear that the collation is the collation of column `x`:

```
SELECT x FROM T ORDER BY x;
SELECT x FROM T WHERE x = x;
SELECT DISTINCT x FROM T;
```

However, with multiple operands, there can be ambiguity. For example, this statement performs a comparison between the column `x` and the string literal `'Y'`:

```
SELECT x FROM T WHERE x = 'Y';
```

If `x` and `'Y'` have the same collation, there is no ambiguity about the collation to use for the comparison. But if they have different collations, should the comparison use the collation of `x`, or of `'Y'`? Both `x` and `'Y'` have collations, so which collation takes precedence?

A mix of collations may also occur in contexts other than comparison. For example, a multiple-argument concatenation operation such as `CONCAT(x,'Y')` combines its arguments to produce a single string. What collation should the result have?

To resolve questions like these, MySQL checks whether the collation of one item can be coerced to the collation of the other. MySQL assigns coercibility values as follows:

- An explicit `COLLATE` clause has a coercibility of 0 (not coercible at all).
- The concatenation of two strings with different collations has a coercibility of 1.
- The collation of a column or a stored routine parameter or local variable has a coercibility of 2.
- A “system constant” (the string returned by functions such as `USER()` or `VERSION()`) has a coercibility of 3.
- The collation of a literal has a coercibility of 4.
- The collation of a numeric or temporal value has a coercibility of 5.
- `NULL` or an expression that is derived from `NULL` has a coercibility of 6.

MySQL uses coercibility values with the following rules to resolve ambiguities:

- Use the collation with the lowest coercibility value.
- If both sides have the same coercibility, then:
  - If both sides are Unicode, or both sides are not Unicode, it is an error.
  - If one of the sides has a Unicode character set, and another side has a non-Unicode character set, the side with Unicode character set wins, and automatic character set conversion is applied to the non-Unicode side. For example, the following statement does not return an error:

    ```
    SELECT CONCAT(utf8_column, latin1_column) FROM t1;
    ```

    It returns a result that has a character set of `utf8` and the same collation as `utf8_column`. Values of `latin1_column` are automatically converted to `utf8` before concatenating.
  - For an operation with operands from the same character set but that mix a `_bin` collation and a `_ci` or `_cs` collation, the `_bin` collation is used. This is similar to how operations that mix nonbinary and binary strings evaluate the operands as binary strings, applied to collations rather than data types.

Although automatic conversion is not in the SQL standard, the standard does say that every character set is (in terms of supported characters) a “subset” of Unicode. Because it is a well-known principle that “what applies to a superset can apply to a subset,” we believe that a collation for Unicode can apply for comparisons with non-Unicode strings. More generally, MySQL uses the concept of character set repertoire, which can sometimes be used to determine subset relationships among character sets and enable conversion of operands in operations that would otherwise produce an error. See Section 10.2.1, “Character Set Repertoire”.

The following table illustrates some applications of the preceding rules.

Comparison	Collation Used
`column1 = 'A'`	Use collation of `column1`
`column1 = 'A' COLLATE x`	Use collation of `'A' COLLATE x`
`column1 COLLATE x = 'A' COLLATE y`	Error

To determine the coercibility of a string expression, use the `COERCIBILITY()` function (see Section 12.16, “Information Functions”):

```
mysql> SELECT COERCIBILITY(_utf8'A' COLLATE utf8_bin);
        -> 0
mysql> SELECT COERCIBILITY(VERSION());
        -> 3
```

```
mysql> SELECT COERCIBILITY('A');
        -> 4
mysql> SELECT COERCIBILITY(1000);
        -> 5
mysql> SELECT COERCIBILITY(NULL);
        -> 6
```

For implicit conversion of a numeric or temporal value to a string, such as occurs for the argument `1` in the expression `CONCAT(1, 'abc')`, the result is a character (nonbinary) string that has a character set and collation determined by the `character_set_connection` and `collation_connection` system variables. See Section 12.3, “Type Conversion in Expression Evaluation”.

10.8.5 The binary Collation Compared to _bin Collations

This section describes how the `binary` collation for binary strings compares to `_bin` collations for nonbinary strings.

Binary strings (as stored using the `BINARY`, `VARBINARY`, and `BLOB` data types) have a character set and collation named `binary`. Binary strings are sequences of bytes and the numeric values of those bytes determine comparison and sort order. See Section 10.10.8, “The Binary Character Set”.

Nonbinary strings (as stored using the `CHAR`, `VARCHAR`, and `TEXT` data types) have a character set and collation other than `binary`. A given nonbinary character set can have several collations, each of which defines a particular comparison and sort order for the characters in the set. One of these is the binary collation, indicated by a `_bin` suffix in the collation name. For example, the binary collation for `utf8` and `latin1` is named `utf8_bin` and `latin1_bin`, respectively.

The `binary` collation differs from `_bin` collations in several respects, discussed in the following sections:

- The Unit for Comparison and Sorting
- Character Set Conversion
- Lettercase Conversion
- Trailing Space Handling in Comparisons
- Trailing Space Handling for Inserts and Retrievals

The Unit for Comparison and Sorting

Binary strings are sequences of bytes. For the `binary` collation, comparison and sorting are based on numeric byte values. Nonbinary strings are sequences of characters, which might be multibyte. Collations for nonbinary strings define an ordering of the character values for comparison and sorting. For `_bin` collations, this ordering is based on numeric character code values, which is similar to ordering for binary strings except that character code values might be multibyte.

Character Set Conversion

A nonbinary string has a character set and is automatically converted to another character set in many cases, even when the string has a `_bin` collation:

- When assigning column values to another column that has a different character set:

```
UPDATE t1 SET utf8_bin_column=latin1_column;
INSERT INTO t1 (latin1_column) SELECT utf8_bin_column FROM t2;
```

- When assigning column values for `INSERT` or `UPDATE` using a string literal:

```
SET NAMES latin1;
INSERT INTO t1 (utf8_bin_column) VALUES ('string-in-latin1');
```

- When sending results from the server to a client:

```
SET NAMES latin1;
SELECT utf8_bin_column FROM t2;
```

For binary string columns, no conversion occurs. For cases similar to those preceding, the string value is copied byte-wise.

Lettercase Conversion

Collations for nonbinary character sets provide information about lettercase of characters, so characters in a nonbinary string can be converted from one lettercase to another, even for `_bin` collations that ignore lettercase for ordering:

```
mysql> SET NAMES utf8mb4 COLLATE utf8mb4_bin;
mysql> SELECT LOWER('aA'), UPPER('zZ');
+-------------+-------------+
| LOWER('aA') | UPPER('zZ') |
+-------------+-------------+
| aa          | ZZ          |
+-------------+-------------+
```

The concept of lettercase does not apply to bytes in a binary string. To perform lettercase conversion, the string must first be converted to a nonbinary string using a character set appropriate for the data stored in the string:

```
mysql> SET NAMES binary;
mysql> SELECT LOWER('aA'), LOWER(CONVERT('aA' USING utf8mb4));
+-------------+-----------------------------------+
| LOWER('aA') | LOWER(CONVERT('aA' USING utf8mb4)) |
+-------------+-----------------------------------+
| aA          | aa                                |
+-------------+-----------------------------------+
```

Trailing Space Handling in Comparisons

Nonbinary strings have `PAD SPACE` behavior for all collations, including `_bin` collations. Trailing spaces are insignificant in comparisons:

```
mysql> SET NAMES utf8 COLLATE utf8_bin;
mysql> SELECT 'a ' = 'a';
+------------+
| 'a ' = 'a' |
+------------+
|          1 |
+------------+
```

For binary strings, all bytes are significant in comparisons, including trailing spaces:

```
mysql> SET NAMES binary;
mysql> SELECT 'a ' = 'a';
+------------+
| 'a ' = 'a' |
+------------+
|          0 |
+------------+
```

Trailing Space Handling for Inserts and Retrievals

`CHAR(N)` columns store nonbinary strings *`N`* characters long. For inserts, values shorter than *`N`* characters are extended with spaces. For retrievals, trailing spaces are removed.

`BINARY(N)` columns store binary strings *`N`* bytes long. For inserts, values shorter than *`N`* bytes are extended with `0x00` bytes. For retrievals, nothing is removed; a value of the declared length is always returned.

```
mysql> CREATE TABLE t1 (
         a CHAR(10) CHARACTER SET utf8 COLLATE utf8_bin,
         b BINARY(10)
```

```
      );
mysql> INSERT INTO t1 VALUES ('x','x');
mysql> INSERT INTO t1 VALUES ('x ','x ');
mysql> SELECT a, b, HEX(a), HEX(b) FROM t1;
+------+------------+--------+----------------------+
| a    | b          | HEX(a) | HEX(b)               |
+------+------------+--------+----------------------+
| x    | x          | 78     | 78000000000000000000 |
| x    | x          | 78     | 78200000000000000000 |
+------+------------+--------+----------------------+
```

10.8.6 Examples of the Effect of Collation

Example 1: Sorting German Umlauts

Suppose that column `X` in table `T` has these `latin1` column values:

```
Muffler
Müller
MX Systems
MySQL
```

Suppose also that the column values are retrieved using the following statement:

```
SELECT X FROM T ORDER BY X COLLATE collation_name;
```

The following table shows the resulting order of the values if we use `ORDER BY` with different collations.

latin1_swedish_ci	latin1_german1_ci	latin1_german2_ci
Muffler	Muffler	Müller
MX Systems	Müller	Muffler
Müller	MX Systems	MX Systems
MySQL	MySQL	MySQL

The character that causes the different sort orders in this example is the U with two dots over it (ü), which the Germans call “U-umlaut.”

- The first column shows the result of the `SELECT` using the Swedish/Finnish collating rule, which says that U-umlaut sorts with Y.
- The second column shows the result of the `SELECT` using the German DIN-1 rule, which says that U-umlaut sorts with U.
- The third column shows the result of the `SELECT` using the German DIN-2 rule, which says that U-umlaut sorts with UE.

Example 2: Searching for German Umlauts

Suppose that you have three tables that differ only by the character set and collation used:

```
mysql> SET NAMES utf8;
mysql> CREATE TABLE german1 (
         c CHAR(10)
       ) CHARACTER SET latin1 COLLATE latin1_german1_ci;
mysql> CREATE TABLE german2 (
         c CHAR(10)
       ) CHARACTER SET latin1 COLLATE latin1_german2_ci;
mysql> CREATE TABLE germanutf8 (
         c CHAR(10)
       ) CHARACTER SET utf8 COLLATE utf8_unicode_ci;
```

Each table contains two records:

```
mysql> INSERT INTO german1 VALUES ('Bar'), ('Bär');
mysql> INSERT INTO german2 VALUES ('Bar'), ('Bär');
```

```
mysql> INSERT INTO germanutf8 VALUES ('Bar'), ('Bär');
```

Two of the above collations have an `A = Ä` equality, and one has no such equality (`latin1_german2_ci`). For that reason, you'll get these results in comparisons:

```
mysql> SELECT * FROM german1 WHERE c = 'Bär';
+------+
| c    |
+------+
| Bar  |
| Bär  |
+------+
mysql> SELECT * FROM german2 WHERE c = 'Bär';
+------+
| c    |
+------+
| Bär  |
+------+
mysql> SELECT * FROM germanutf8 WHERE c = 'Bär';
+------+
| c    |
+------+
| Bar  |
| Bär  |
+------+
```

This is not a bug but rather a consequence of the sorting properties of `latin1_german1_ci` and `utf8_unicode_ci` (the sorting shown is done according to the German DIN 5007 standard).

10.8.7 Using Collation in INFORMATION_SCHEMA Searches

String columns in `INFORMATION_SCHEMA` tables have a collation of `utf8_general_ci`, which is case-insensitive. However, for values that correspond to objects that are represented in the file system, such as databases and tables, searches in `INFORMATION_SCHEMA` string columns can be case-sensitive or case-insensitive, depending on the characteristics of the underlying file system and the value of the `lower_case_table_names` system variable. For example, searches may be case-sensitive if the file system is case-sensitive. This section describes this behavior and how to modify it if necessary; see also Bug #34921.

Suppose that a query searches the `SCHEMATA.SCHEMA_NAME` column for the `test` database. On Linux, file systems are case-sensitive, so comparisons of `SCHEMATA.SCHEMA_NAME` with `'test'` match, but comparisons with `'TEST'` do not:

```
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
       WHERE SCHEMA_NAME = 'test';
+-------------+
| SCHEMA_NAME |
+-------------+
| test        |
+-------------+

mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
       WHERE SCHEMA_NAME = 'TEST';
Empty set (0.00 sec)
```

These results occur with the `lower_case_table_names` system variable set to 0. Changing the value of `lower_case_table_names` to 1 or 2 causes the second query to return the same (nonempty) result as the first query.

On Windows or macOS, file systems are not case-sensitive, so comparisons match both `'test'` and `'TEST'`:

```
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
       WHERE SCHEMA_NAME = 'test';
+-------------+
| SCHEMA_NAME |
+-------------+
```

```
| test        |
+-------------+

mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
       WHERE SCHEMA_NAME = 'TEST';
+-------------+
| SCHEMA_NAME |
+-------------+
| TEST        |
+-------------+
```

The value of `lower_case_table_names` makes no difference in this context.

The preceding behavior occurs because the `utf8_general_ci` collation is not used for `INFORMATION_SCHEMA` queries when searching for values that correspond to objects represented in the file system. It is a result of file system-scanning optimizations implemented for `INFORMATION_SCHEMA` searches. For information about these optimizations, see Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”.

If the result of a string operation on an `INFORMATION_SCHEMA` column differs from expectations, a workaround is to use an explicit `COLLATE` clause to force a suitable collation (see Section 10.8.1, “Using COLLATE in SQL Statements”). For example, to perform a case-insensitive search, use `COLLATE` with the `INFORMATION_SCHEMA` column name:

```
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
       WHERE SCHEMA_NAME COLLATE utf8_general_ci = 'test';
+-------------+
| SCHEMA_NAME |
+-------------+
| test        |
+-------------+

mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
       WHERE SCHEMA_NAME COLLATE utf8_general_ci = 'TEST';
+-------------+
| SCHEMA_NAME |
+-------------+
| test        |
+-------------+
```

In the preceding queries, it is important to apply the `COLLATE` clause to the `INFORMATION_SCHEMA` column name. Applying `COLLATE` to the comparison value has no effect.

You can also use the `UPPER()` or `LOWER()` function:

```
WHERE UPPER(SCHEMA_NAME) = 'TEST'
WHERE LOWER(SCHEMA_NAME) = 'test'
```

Although a case-insensitive comparison can be performed even on platforms with case-sensitive file systems, as just shown, it is not necessarily always the right thing to do. On such platforms, it is possible to have multiple objects with names that differ only in lettercase. For example, tables named `city`, `CITY`, and `City` can all exist simultaneously. Consider whether a search should match all such names or just one and write queries accordingly. The first of the following comparisons (with `utf8_bin`) is case-sensitive; the others are not:

```
WHERE TABLE_NAME COLLATE utf8_bin = 'City'
WHERE TABLE_NAME COLLATE utf8_general_ci = 'city'
WHERE UPPER(TABLE_NAME) = 'CITY'
WHERE LOWER(TABLE_NAME) = 'city'
```

Searches in `INFORMATION_SCHEMA` string columns for values that refer to `INFORMATION_SCHEMA` itself do use the `utf8_general_ci` collation because `INFORMATION_SCHEMA` is a “virtual” database not represented in the file system. For example, comparisons with `SCHEMATA.SCHEMA_NAME` match `'information_schema'` or `'INFORMATION_SCHEMA'` regardless of platform:

```
mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
       WHERE SCHEMA_NAME = 'information_schema';
```

```
+--------------------+
| SCHEMA_NAME        |
+--------------------+
| information_schema |
+--------------------+

mysql> SELECT SCHEMA_NAME FROM INFORMATION_SCHEMA.SCHEMATA
       WHERE SCHEMA_NAME = 'INFORMATION_SCHEMA';
+--------------------+
| SCHEMA_NAME        |
+--------------------+
| information_schema |
+--------------------+
```

10.9 Unicode Support

The Unicode Standard includes characters from the Basic Multilingual Plane (BMP) and supplementary characters that lie outside the BMP. This section describes support for Unicode in MySQL. For information about the Unicode Standard itself, visit the Unicode Consortium website.

BMP characters have these characteristics:

- Their code point values are between 0 and 65535 (or `U+0000` and `U+FFFF`).
- They can be encoded in a variable-length encoding using 8, 16, or 24 bits (1 to 3 bytes).
- They can be encoded in a fixed-length encoding using 16 bits (2 bytes).
- They are sufficient for almost all characters in major languages.

Supplementary characters lie outside the BMP:

- Their code point values are between `U+10000` and `U+10FFFF`).
- Unicode support for supplementary characters requires character sets that have a range outside BMP characters and therefore take more space than BMP characters (up to 4 bytes per character).

The UTF-8 (Unicode Transformation Format with 8-bit units) method for encoding Unicode data is implemented according to RFC 3629, which describes encoding sequences that take from one to four bytes. The idea of UTF-8 is that various Unicode characters are encoded using byte sequences of different lengths:

- Basic Latin letters, digits, and punctuation signs use one byte.
- Most European and Middle East script letters fit into a 2-byte sequence: extended Latin letters (with tilde, macron, acute, grave and other accents), Cyrillic, Greek, Armenian, Hebrew, Arabic, Syriac, and others.
- Korean, Chinese, and Japanese ideographs use 3-byte or 4-byte sequences.

MySQL supports these Unicode character sets:

- `utf8mb4`: A UTF-8 encoding of the Unicode character set using one to four bytes per character.
- `utf8mb3`: A UTF-8 encoding of the Unicode character set using one to three bytes per character.
- `utf8`: An alias for `utf8mb3`.
- `ucs2`: The UCS-2 encoding of the Unicode character set using two bytes per character.
- `utf16`: The UTF-16 encoding for the Unicode character set using two or four bytes per character. Like `ucs2` but with an extension for supplementary characters.
- `utf16le`: The UTF-16LE encoding for the Unicode character set. Like `utf16` but little-endian rather than big-endian.

- `utf32`: The UTF-32 encoding for the Unicode character set using four bytes per character.

Table 10.2, “Unicode Character Set General Characteristics”, summarizes the general characteristics of Unicode character sets supported by MySQL.

Table 10.2 Unicode Character Set General Characteristics

Character Set	Supported Characters	Required Storage Per Character
`utf8mb3`, `utf8`	BMP only	1, 2, or 3 bytes
`ucs2`	BMP only	2 bytes
`utf8mb4`	BMP and supplementary	1, 2, 3, or 4 bytes
`utf16`	BMP and supplementary	2 or 4 bytes
`utf16le`	BMP and supplementary	2 or 4 bytes
`utf32`	BMP and supplementary	4 bytes

Characters outside the BMP compare as REPLACEMENT CHARACTER and convert to `'?'` when converted to a Unicode character set that supports only BMP characters (`utf8mb3` or `ucs2`).

If you use character sets that support supplementary characters and thus are “wider” than the BMP-only `utf8mb3` and `ucs2` character sets, there are potential incompatibility issues for your applications; see Section 10.9.8, “Converting Between 3-Byte and 4-Byte Unicode Character Sets”. That section also describes how to convert tables from the (3-byte) `utf8mb3` to the (4-byte) `utf8mb4`, and what constraints may apply in doing so.

A similar set of collations is available for most Unicode character sets. For example, each has a Danish collation, the names of which are `utf8mb4_danish_ci`, `utf8mb3_danish_ci`, `utf8_danish_ci`, `ucs2_danish_ci`, `utf16_danish_ci`, and `utf32_danish_ci`. The exception is `utf16le`, which has only two collations. For information about Unicode collations and their differentiating properties, including collation properties for supplementary characters, see Section 10.10.1, “Unicode Character Sets”.

The MySQL implementation of UCS-2, UTF-16, and UTF-32 stores characters in big-endian byte order and does not use a byte order mark (BOM) at the beginning of values. Other database systems might use little-endian byte order or a BOM. In such cases, conversion of valuesmust be performed when transferring data between those systems and MySQL. The implementation of UTF-16LE is little-endian.

MySQL uses no BOM for UTF-8 values.

Client applications that communicate with the server using Unicode should set the client character set accordingly (for example, by issuing a `SET NAMES 'utf8mb4'` statement). Some character sets cannot be used as the client character set. Attempting to use them with `SET NAMES` or `SET CHARACTER SET` produces an error. See Impermissible Client Character Sets.

The following sections provide additional detail on the Unicode character sets in MySQL.

10.9.1 The utf8mb4 Character Set (4-Byte UTF-8 Unicode Encoding)

The `utfmb4` character set has these characteristics:

- Supports BMP and supplementary characters.
- Requires a maximum of four bytes per multibyte character.

`utf8mb4` contrasts with the `utf8mb3` character set, which supports only BMP characters and uses a maximum of three bytes per character:

- For a BMP character, `utf8mb4` and `utf8mb3` have identical storage characteristics: same code values, same encoding, same length.

- For a supplementary character, `utf8mb4` requires four bytes to store it, whereas `utf8mb3` cannot store the character at all. When converting `utf8mb3` columns to `utf8mb4`, you need not worry about converting supplementary characters because there are none.

`utf8mb4` is a superset of `utf8mb3`, so for an operation such as the following concatenation, the result has character set `utf8mb4` and the collation of `utf8mb4_col`:

```
SELECT CONCAT(utf8mb3_col, utf8mb4_col);
```

Similarly, the following comparison in the `WHERE` clause works according to the collation of `utf8mb4_col`:

```
SELECT * FROM utf8mb3_tbl, utf8mb4_tbl
WHERE utf8mb3_tbl.utf8mb3_col = utf8mb4_tbl.utf8mb4_col;
```

For information about data type storage as it relates to multibyte character sets, see String Type Storage Requirements.

10.9.2 The utf8mb3 Character Set (3-Byte UTF-8 Unicode Encoding)

The `utf8mb3` character set has these characteristics:

- Supports BMP characters only (no support for supplementary characters)
- Requires a maximum of three bytes per multibyte character.

Applications that use UTF-8 data but require supplementary character support should use `utf8mb4` rather than `utf8mb3` (see Section 10.9.1, “The utf8mb4 Character Set (4-Byte UTF-8 Unicode Encoding)”).

Exactly the same set of characters is available in `utf8mb3` and `ucs2`. That is, they have the same repertoire.

`utf8` is an alias for `utf8mb3`; the character limit is implicit, rather than explicit in the name.

`utf8mb3` can be used in `CHARACTER SET` clauses, and `utf8mb3_`*`collation_substring`* in `COLLATE` clauses, where *`collation_substring`* is `bin`, `czech_ci`, `danish_ci`, `esperanto_ci`, `estonian_ci`, and so forth. For example:

```
CREATE TABLE t (s1 CHAR(1) CHARACTER SET utf8mb3;
SELECT * FROM t WHERE s1 COLLATE utf8mb3_general_ci = 'x';
DECLARE x VARCHAR(5) CHARACTER SET utf8mb3 COLLATE utf8mb3_danish_ci;
SELECT CAST('a' AS CHAR CHARACTER SET utf8) COLLATE utf8_czech_ci;
```

MySQL immediately converts instances of `utf8mb3` in statements to `utf8`, so in statements such as `SHOW CREATE TABLE` or `SELECT CHARACTER_SET_NAME FROM INFORMATION_SCHEMA.COLUMNS` or `SELECT COLLATION_NAME FROM INFORMATION_SCHEMA.COLUMNS`, users see the name `utf8` or `utf8_`*`collation_substring`*.

`utf8mb3` is also valid in contexts other than `CHARACTER SET` clauses. For example:

```
mysqld --character-set-server=utf8mb3
```

```
SET NAMES 'utf8mb3'; /* and other SET statements that have similar effect */
SELECT _utf8mb3 'a';
```

For information about data type storage as it relates to multibyte character sets, see String Type Storage Requirements.

10.9.3 The utf8 Character Set (Alias for utf8mb3)

`utf8` is an alias for the `utf8mb3` character set. For more information, see Section 10.9.2, “The utf8mb3 Character Set (3-Byte UTF-8 Unicode Encoding)”.

10.9.4 The ucs2 Character Set (UCS-2 Unicode Encoding)

In UCS-2, every character is represented by a 2-byte Unicode code with the most significant byte first. For example: `LATIN CAPITAL LETTER A` has the code `0x0041` and it is stored as a 2-byte sequence: `0x00 0x41`. `CYRILLIC SMALL LETTER YERU` (Unicode `0x044B`) is stored as a 2-byte sequence: `0x04 0x4B`. For Unicode characters and their codes, please refer to the Unicode Consortium website.

The `ucs2` character set has these characteristics:

- Supports BMP characters only (no support for supplementary characters)
- Uses a fixed-length 16-bit encoding and requires two bytes per character.

10.9.5 The utf16 Character Set (UTF-16 Unicode Encoding)

The `utf16` character set is the `ucs2` character set with an extension that enables encoding of supplementary characters:

- For a BMP character, `utf16` and `ucs2` have identical storage characteristics: same code values, same encoding, same length.
- For a supplementary character, `utf16` has a special sequence for representing the character using 32 bits. This is called the “surrogate” mechanism: For a number greater than `0xffff`, take 10 bits and add them to `0xd800` and put them in the first 16-bit word, take 10 more bits and add them to `0xdc00` and put them in the next 16-bit word. Consequently, all supplementary characters require 32 bits, where the first 16 bits are a number between `0xd800` and `0xdbff`, and the last 16 bits are a number between `0xdc00` and `0xdfff`. Examples are in Section 15.5 Surrogates Area of the Unicode 4.0 document.

Because `utf16` supports surrogates and `ucs2` does not, there is a validity check that applies only in `utf16`: You cannot insert a top surrogate without a bottom surrogate, or vice versa. For example:

```
INSERT INTO t (ucs2_column) VALUES (0xd800); /* legal */
INSERT INTO t (utf16_column)VALUES (0xd800); /* illegal */
```

There is no validity check for characters that are technically valid but are not true Unicode (that is, characters that Unicode considers to be “unassigned code points” or “private use” characters or even “illegals” like `0xffff`). For example, since `U+F8FF` is the Apple Logo, this is legal:

```
INSERT INTO t (utf16_column)VALUES (0xf8ff); /* legal */
```

Such characters cannot be expected to mean the same thing to everyone.

Because MySQL must allow for the worst case (that one character requires four bytes) the maximum length of a `utf16` column or index is only half of the maximum length for a `ucs2` column or index. For example, the maximum length of a `MEMORY` table index key is 3072 bytes, so these statements create tables with the longest permitted indexes for `ucs2` and `utf16` columns:

```
CREATE TABLE tf (s1 VARCHAR(1536) CHARACTER SET ucs2) ENGINE=MEMORY;
CREATE INDEX i ON tf (s1);
CREATE TABLE tg (s1 VARCHAR(768) CHARACTER SET utf16) ENGINE=MEMORY;
CREATE INDEX i ON tg (s1);
```

10.9.6 The utf16le Character Set (UTF-16LE Unicode Encoding)

This is the same as `utf16` but is little-endian rather than big-endian.

10.9.7 The utf32 Character Set (UTF-32 Unicode Encoding)

The `utf32` character set is fixed length (like `ucs2` and unlike `utf16`). `utf32` uses 32 bits for every character, unlike `ucs2` (which uses 16 bits for every character), and unlike `utf16` (which uses 16 bits for some characters and 32 bits for others).

`utf32` takes twice as much space as `ucs2` and more space than `utf16`, but `utf32` has the same advantage as `ucs2` that it is predictable for storage: The required number of bytes for `utf32` equals the number of characters times 4. Also, unlike `utf16`, there are no tricks for encoding in `utf32`, so the stored value equals the code value.

To demonstrate how the latter advantage is useful, here is an example that shows how to determine a `utf8mb4` value given the `utf32` code value:

```
/* Assume code value = 100cc LINEAR B WHEELED CHARIOT */
CREATE TABLE tmp (utf32_col CHAR(1) CHARACTER SET utf32,
                  utf8mb4_col CHAR(1) CHARACTER SET utf8mb4);
INSERT INTO tmp VALUES (0x000100cc,NULL);
UPDATE tmp SET utf8mb4_col = utf32_col;
SELECT HEX(utf32_col),HEX(utf8mb4_col) FROM tmp;
```

MySQL is very forgiving about additions of unassigned Unicode characters or private-use-area characters. There is in fact only one validity check for `utf32`: No code value may be greater than `0x10ffff`. For example, this is illegal:

```
INSERT INTO t (utf32_column) VALUES (0x110000); /* illegal */
```

10.9.8 Converting Between 3-Byte and 4-Byte Unicode Character Sets

This section describes issues that you may face when converting character data between the `utf8mb3` and `utf8mb4` character sets.

Note

This discussion focuses primarily on converting between `utf8mb3` and `utf8mb4`, but similar principles apply to converting between the `ucs2` character set and character sets such as `utf16` or `utf32`.

The `utf8mb3` and `utf8mb4` character sets differ as follows:

- `utf8mb3` supports only characters in the Basic Multilingual Plane (BMP). `utf8mb4` additionally supports supplementary characters that lie outside the BMP.
- `utf8mb3` uses a maximum of three bytes per character. `utf8mb4` uses a maximum of four bytes per character.

Note

This discussion refers to the `utf8mb3` and `utf8mb4` character set names to be explicit about referring to 3-byte and 4-byte UTF-8 character set data. The exception is that in table definitions, `utf8` is used because MySQL converts instances of `utf8mb3` specified in such definitions to `utf8`, which is an alias for `utf8mb3`.

One advantage of converting from `utf8mb3` to `utf8mb4` is that this enables applications to use supplementary characters. One tradeoff is that this may increase data storage space requirements.

In terms of table content, conversion from `utf8mb3` to `utf8mb4` presents no problems:

- For a BMP character, `utf8mb4` and `utf8mb3` have identical storage characteristics: same code values, same encoding, same length.
- For a supplementary character, `utf8mb4` requires four bytes to store it, whereas `utf8mb3` cannot store the character at all. When converting `utf8mb3` columns to `utf8mb4`, you need not worry about converting supplementary characters because there are none.

In terms of table structure, these are the primary potential incompatibilities:

- For the variable-length character data types (`VARCHAR` and the `TEXT` types), the maximum permitted length in characters is less for `utf8mb4` columns than for `utf8mb3` columns.

- For all character data types (CHAR, VARCHAR, and the TEXT types), the maximum number of characters that can be indexed is less for utf8mb4 columns than for utf8mb3 columns.

Consequently, to convert tables from utf8mb3 to utf8mb4, it may be necessary to change some column or index definitions.

Tables can be converted from utf8mb3 to utf8mb4 by using ALTER TABLE. Suppose that a table has this definition:

```
CREATE TABLE t1 (
  col1 CHAR(10) CHARACTER SET utf8 COLLATE utf8_unicode_ci NOT NULL,
  col2 CHAR(10) CHARACTER SET utf8 COLLATE utf8_bin NOT NULL
) CHARACTER SET utf8;
```

The following statement converts t1 to use utf8mb4:

```
ALTER TABLE t1
  DEFAULT CHARACTER SET utf8mb4,
  MODIFY col1 CHAR(10)
    CHARACTER SET utf8mb4 COLLATE utf8mb4_unicode_ci NOT NULL,
  MODIFY col2 CHAR(10)
    CHARACTER SET utf8mb4 COLLATE utf8mb4_bin NOT NULL;
```

The catch when converting from utf8mb3 to utf8mb4 is that the maximum length of a column or index key is unchanged in terms of *bytes*. Therefore, it is smaller in terms of *characters* because the maximum length of a character is four bytes instead of three. For the CHAR, VARCHAR, and TEXT data types, watch for these issues when converting your MySQL tables:

- Check all definitions of utf8mb3 columns and make sure they do not exceed the maximum length for the storage engine.
- Check all indexes on utf8mb3 columns and make sure they do not exceed the maximum length for the storage engine. Sometimes the maximum can change due to storage engine enhancements.

If the preceding conditions apply, you must either reduce the defined length of columns or indexes, or continue to use utf8mb3 rather than utf8mb4.

Here are some examples where structural changes may be needed:

- A TINYTEXT column can hold up to 255 bytes, so it can hold up to 85 3-byte or 63 4-byte characters. Suppose that you have a TINYTEXT column that uses utf8mb3 but must be able to contain more than 63 characters. You cannot convert it to utf8mb4 unless you also change the data type to a longer type such as TEXT.

  Similarly, a very long VARCHAR column may need to be changed to one of the longer TEXT types if you want to convert it from utf8mb3 to utf8mb4.

- InnoDB has a maximum index length of 767 bytes for tables that use COMPACT or REDUNDANT row format, so for utf8mb3 or utf8mb4 columns, you can index a maximum of 255 or 191 characters, respectively. If you currently have utf8mb3 columns with indexes longer than 191 characters, you must index a smaller number of characters.

  In an InnoDB table that uses COMPACT or REDUNDANT row format, these column and index definitions are legal:

  ```
  col1 VARCHAR(500) CHARACTER SET utf8, INDEX (col1(255))
  ```

  To use utf8mb4 instead, the index must be smaller:

  ```
  col1 VARCHAR(500) CHARACTER SET utf8mb4, INDEX (col1(191))
  ```

Note

For InnoDB tables that use COMPRESSED or DYNAMIC row format, you can enable the innodb_large_prefix option to permit index key

> prefixes longer than 767 bytes (up to 3072 bytes). Creating such tables also requires the option values `innodb_file_format=barracuda` and `innodb_file_per_table=true`.) In this case, enabling the `innodb_large_prefix` option enables you to index a maximum of 1024 or 768 characters for `utf8mb3` or `utf8mb4` columns, respectively. For related information, see Section 14.22, “InnoDB Limits”.

The preceding types of changes are most likely to be required only if you have very long columns or indexes. Otherwise, you should be able to convert your tables from `utf8mb3` to `utf8mb4` without problems, using `ALTER TABLE` as described previously.

The following items summarize other potential incompatibilities:

- `SET NAMES 'utf8mb4'` causes use of the 4-byte character set for connection character sets. As long as no 4-byte characters are sent from the server, there should be no problems. Otherwise, applications that expect to receive a maximum of three bytes per character may have problems. Conversely, applications that expect to send 4-byte characters must ensure that the server understands them.

- For replication, if character sets that support supplementary characters are to be used on the source, all replicas must understand them as well.

  Also, keep in mind the general principle that if a table has different definitions on the source and replica, this can lead to unexpected results. For example, the differences in maximum index key length make it risky to use `utf8mb3` on the source and `utf8mb4` on the replica.

If you have converted to `utf8mb4`, `utf16`, `utf16le`, or `utf32`, and then decide to convert back to `utf8mb3` or `ucs2` (for example, to downgrade to an older version of MySQL), these considerations apply:

- `utf8mb3` and `ucs2` data should present no problems.

- The server must be recent enough to recognize definitions referring to the character set from which you are converting.

- For object definitions that refer to the `utf8mb4` character set, you can dump them with `mysqldump` prior to downgrading, edit the dump file to change instances of `utf8mb4` to `utf8`, and reload the file in the older server, as long as there are no 4-byte characters in the data. The older server sees `utf8` in the dump file object definitions and creates new objects that use the (3-byte) `utf8` character set.

10.10 Supported Character Sets and Collations

This section indicates which character sets MySQL supports. There is one subsection for each group of related character sets. For each character set, the permissible collations are listed.

To list the available character sets and their default collations, use the `SHOW CHARACTER SET` statement or query the `INFORMATION_SCHEMA CHARACTER_SETS` table. For example:

```
mysql> SHOW CHARACTER SET;
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |      2 |
| dec8     | DEC West European           | dec8_swedish_ci     |      1 |
| cp850    | DOS West European           | cp850_general_ci    |      1 |
| hp8      | HP West European            | hp8_english_ci      |      1 |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |      1 |
| latin1   | cp1252 West European        | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |      1 |
| swe7     | 7bit Swedish                | swe7_swedish_ci     |      1 |
| ascii    | US ASCII                    | ascii_general_ci    |      1 |
| ujis     | EUC-JP Japanese             | ujis_japanese_ci    |      3 |
```

```
| sjis     | Shift-JIS Japanese          | sjis_japanese_ci    |      2 |
| hebrew   | ISO 8859-8 Hebrew           | hebrew_general_ci   |      1 |
| tis620   | TIS620 Thai                 | tis620_thai_ci      |      1 |
| euckr    | EUC-KR Korean               | euckr_korean_ci     |      2 |
| koi8u    | KOI8-U Ukrainian            | koi8u_general_ci    |      1 |
| gb2312   | GB2312 Simplified Chinese   | gb2312_chinese_ci   |      2 |
| greek    | ISO 8859-7 Greek            | greek_general_ci    |      1 |
| cp1250   | Windows Central European    | cp1250_general_ci   |      1 |
| gbk      | GBK Simplified Chinese      | gbk_chinese_ci      |      2 |
| latin5   | ISO 8859-9 Turkish          | latin5_turkish_ci   |      1 |
| armscii8 | ARMSCII-8 Armenian          | armscii8_general_ci |      1 |
| utf8     | UTF-8 Unicode               | utf8_general_ci     |      3 |
| ucs2     | UCS-2 Unicode               | ucs2_general_ci     |      2 |
| cp866    | DOS Russian                 | cp866_general_ci    |      1 |
| keybcs2  | DOS Kamenicky Czech-Slovak  | keybcs2_general_ci  |      1 |
| macce    | Mac Central European        | macce_general_ci    |      1 |
| macroman | Mac West European           | macroman_general_ci |      1 |
| cp852    | DOS Central European        | cp852_general_ci    |      1 |
| latin7   | ISO 8859-13 Baltic          | latin7_general_ci   |      1 |
| utf8mb4  | UTF-8 Unicode               | utf8mb4_general_ci  |      4 |
| cp1251   | Windows Cyrillic            | cp1251_general_ci   |      1 |
| utf16    | UTF-16 Unicode              | utf16_general_ci    |      4 |
| utf16le  | UTF-16LE Unicode            | utf16le_general_ci  |      4 |
| cp1256   | Windows Arabic              | cp1256_general_ci   |      1 |
| cp1257   | Windows Baltic              | cp1257_general_ci   |      1 |
| utf32    | UTF-32 Unicode              | utf32_general_ci    |      4 |
| binary   | Binary pseudo charset       | binary              |      1 |
| geostd8  | GEOSTD8 Georgian            | geostd8_general_ci  |      1 |
| cp932    | SJIS for Windows Japanese   | cp932_japanese_ci   |      2 |
| eucjpms  | UJIS for Windows Japanese   | eucjpms_japanese_ci |      3 |
+----------+-----------------------------+---------------------+--------+
```

In cases where a character set has multiple collations, it might not be clear which collation is most suitable for a given application. To avoid choosing the wrong collation, it can be helpful to perform some comparisons with representative data values to make sure that a given collation sorts values the way you expect.

10.10.1 Unicode Character Sets

This section describes the collations available for Unicode character sets and their differentiating properties. For general information about Unicode, see Section 10.9, “Unicode Support”.

MySQL supports multiple Unicode character sets:

- `utf8mb4`: A UTF-8 encoding of the Unicode character set using one to four bytes per character.
- `utf8mb3`: A UTF-8 encoding of the Unicode character set using one to three bytes per character.
- `utf8`: An alias for `utf8mb3`.
- `ucs2`: The UCS-2 encoding of the Unicode character set using two bytes per character.
- `utf16`: The UTF-16 encoding for the Unicode character set using two or four bytes per character. Like `ucs2` but with an extension for supplementary characters.
- `utf16le`: The UTF-16LE encoding for the Unicode character set. Like `utf16` but little-endian rather than big-endian.
- `utf32`: The UTF-32 encoding for the Unicode character set using four bytes per character.

`utf8mb4`, `utf16`, `utf16le`, and `utf32` support Basic Multilingual Plane (BMP) characters and supplementary characters that lie outside the BMP. `utf8` and `ucs2` support only BMP characters.

Most Unicode character sets have a general collation (indicated by `_general` in the name or by the absence of a language specifier), a binary collation (indicated by `_bin` in the name), and several language-specific collations (indicated by language specifiers). For example, for

`utf8mb4`, `utf8mb4_general_ci` and `utf8mb4_bin` are its general and binary collations, and `utf8mb4_danish_ci` is one of its language-specific collations.

Collation support for `utf16le` is limited. The only collations available are `utf16le_general_ci` and `utf16le_bin`. These are similar to `utf16_general_ci` and `utf16_bin`.

- Unicode Collation Algorithm (UCA) Versions
- Language-Specific Collations
- _general_ci Versus _unicode_ci Collations
- Character Collating Weights
- Miscellaneous Information

Unicode Collation Algorithm (UCA) Versions

MySQL implements the *`xxx`*`_unicode_ci` collations according to the Unicode Collation Algorithm (UCA) described at http://www.unicode.org/reports/tr10/. The collation uses the version-4.0.0 UCA weight keys: http://www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt. The *`xxx`*`_unicode_ci` collations have only partial support for the Unicode Collation Algorithm. Some characters are not supported, and combining marks are not fully supported. This affects primarily Vietnamese, Yoruba, and some smaller languages such as Navajo. A combined character is considered different from the same character written with a single unicode character in string comparisons, and the two characters are considered to have a different length (for example, as returned by the `CHAR_LENGTH()` function or in result set metadata).

Unicode collations based on UCA versions higher than 4.0.0 include the version in the collation name. Thus, `utf8_unicode_520_ci` is based on UCA 5.2.0 weight keys (http://www.unicode.org/Public/UCA/5.2.0/allkeys.txt).

The `LOWER()` and `UPPER()` functions perform case folding according to the collation of their argument. A character that has uppercase and lowercase versions only in a Unicode version higher than 4.0.0 is converted by these functions only if the argument collation uses a high enough UCA version.

Language-Specific Collations

MySQL implements language-specific Unicode collations if the ordering based only on the Unicode Collation Algorithm (UCA) does not work well for a language. Language-specific collations are UCA-based, with additional language tailoring rules. Examples of such rules appear later in this section. For questions about particular language orderings, unicode.org provides Common Locale Data Repository (CLDR) collation charts at http://www.unicode.org/cldr/charts/30/collation/index.html.

A language name shown in the following table indicates a language-specific collation. Unicode character sets may include collations for one or more of these languages.

Table 10.3 Unicode Collation Language Specifiers

Language	Language Specifier
Classical Latin	`roman`
Croatian	`croatian`
Czech	`czech`
Danish	`danish`
Esperanto	`esperanto`
Estonian	`estonian`
German phone book order	`german2`
Hungarian	`hungarian`

Language	Language Specifier
Icelandic	`icelandic`
Latvian	`latvian`
Lithuanian	`lithuanian`
Persian	`persian`
Polish	`polish`
Romanian	`romanian`
Sinhala	`sinhala`
Slovak	`slovak`
Slovenian	`slovenian`
Modern Spanish	`spanish`
Traditional Spanish	`spanish2`
Swedish	`swedish`
Turkish	`turkish`
Vietnamese	`vietnamese`

Croatian collations are tailored for these Croatian letters: `Č`, `Ć`, `Dž`, `Đ`, `Lj`, `Nj`, `Š`, `Ž`.

Danish collations may also be used for Norwegian.

For Classical Latin collations, `I` and `J` compare as equal, and `U` and `V` compare as equal.

Spanish collations are available for modern and traditional Spanish. For both, `ñ` (n-tilde) is a separate letter between `n` and `o`. In addition, for traditional Spanish, `ch` is a separate letter between `c` and `d`, and `ll` is a separate letter between `l` and `m`.

Traditional Spanish collations may also be used for Asturian and Galician.

Swedish collations include Swedish rules. For example, in Swedish, the following relationship holds, which is not something expected by a German or French speaker:

```
Ü = Y < Ö
```

_general_ci Versus _unicode_ci Collations

For any Unicode character set, operations performed using the *`xxx`*`_general_ci` collation are faster than those for the *`xxx`*`_unicode_ci` collation. For example, comparisons for the `utf8_general_ci` collation are faster, but slightly less correct, than comparisons for `utf8_unicode_ci`. The reason is that `utf8_unicode_ci` supports mappings such as expansions; that is, when one character compares as equal to combinations of other characters. For example, ß is equal to `ss` in German and some other languages. `utf8_unicode_ci` also supports contractions and ignorable characters. `utf8_general_ci` is a legacy collation that does not support expansions, contractions, or ignorable characters. It can make only one-to-one comparisons between characters.

To further illustrate, the following equalities hold in both `utf8_general_ci` and `utf8_unicode_ci` (for the effect of this in comparisons or searches, see Section 10.8.6, “Examples of the Effect of Collation”):

```
Ä = A
Ö = O
Ü = U
```

A difference between the collations is that this is true for `utf8_general_ci`:

```
ß = s
```

Whereas this is true for `utf8_unicode_ci`, which supports the German DIN-1 ordering (also known as dictionary order):

```
ß = ss
```

MySQL implements `utf8` language-specific collations if the ordering with `utf8_unicode_ci` does not work well for a language. For example, `utf8_unicode_ci` works fine for German dictionary order and French, so there is no need to create special `utf8` collations.

`utf8_general_ci` also is satisfactory for both German and French, except that `ß` is equal to `s`, and not to `ss`. If this is acceptable for your application, you should use `utf8_general_ci` because it is faster. If this is not acceptable (for example, if you require German dictionary order), use `utf8_unicode_ci` because it is more accurate.

If you require German DIN-2 (phone book) ordering, use the `utf8_german2_ci` collation, which compares the following sets of characters equal:

```
Ä = Æ = AE
Ö = Œ = OE
Ü = UE
ß = ss
```

`utf8_german2_ci` is similar to `latin1_german2_ci`, but the latter does not compare `Æ` equal to `AE` or `Œ` equal to `OE`. There is no `utf8_german_ci` corresponding to `latin1_german_ci` for German dictionary order because `utf8_general_ci` suffices.

Character Collating Weights

A character's collating weight is determined as follows:

- For all Unicode collations except the `_bin` (binary) collations, MySQL performs a table lookup to find a character's collating weight.
- For `_bin` collations, the weight is based on the code point, possibly with leading zero bytes added.

Collating weights can be displayed using the `WEIGHT_STRING()` function. (See Section 12.8, “String Functions and Operators”.) If a collation uses a weight lookup table, but a character is not in the table (for example, because it is a “new” character), collating weight determination becomes more complex:

- For BMP characters in general collations (*`xxx`*`_general_ci`), the weight is the code point.
- For BMP characters in UCA collations (for example, *`xxx`*`_unicode_ci` and language-specific collations), the following algorithm applies:

```
if (code >= 0x3400 && code <= 0x4DB5)
  base= 0xFB80; /* CJK Ideograph Extension */
else if (code >= 0x4E00 && code <= 0x9FA5)
  base= 0xFB40; /* CJK Ideograph */
else
  base= 0xFBC0; /* All other characters */
aaaa= base +  (code >> 15);
bbbb= (code & 0x7FFF) | 0x8000;
```

The result is a sequence of two collating elements, `aaaa` followed by `bbbb`. For example:

```
mysql> SELECT HEX(WEIGHT_STRING(_ucs2 0x04CF COLLATE ucs2_unicode_ci));
+-------------------------------------------------------+
| HEX(WEIGHT_STRING(_ucs2 0x04CF COLLATE ucs2_unicode_ci)) |
+-------------------------------------------------------+
| FBC084CF                                              |
+-------------------------------------------------------+
```

Thus, `U+04cf CYRILLIC SMALL LETTER PALOCHKA` is, with all UCA 4.0.0 collations, greater than `U+04c0 CYRILLIC LETTER PALOCHKA`. With UCA 5.2.0 collations, all palochkas sort together.

- For supplementary characters in general collations, the weight is the weight for `0xfffd REPLACEMENT CHARACTER`. For supplementary characters in UCA 4.0.0 collations, their collating weight is `0xfffd`. That is, to MySQL, all supplementary characters are equal to each other, and greater than almost all BMP characters.

  An example with Deseret characters and `COUNT(DISTINCT)`:

  ```
  CREATE TABLE t (s1 VARCHAR(5) CHARACTER SET utf32 COLLATE utf32_unicode_ci);
  INSERT INTO t VALUES (0xfffd);   /* REPLACEMENT CHARACTER */
  INSERT INTO t VALUES (0x010412); /* DESERET CAPITAL LETTER BEE */
  INSERT INTO t VALUES (0x010413); /* DESERET CAPITAL LETTER TEE */
  SELECT COUNT(DISTINCT s1) FROM t;
  ```

  The result is 2 because in the MySQL *`xxx`*`_unicode_ci` collations, the replacement character has a weight of `0x0dc6`, whereas Deseret Bee and Deseret Tee both have a weight of `0xfffd`. (Were the `utf32_general_ci` collation used instead, the result is 1 because all three characters have a weight of `0xfffd` in that collation.)

  An example with cuneiform characters and `WEIGHT_STRING()`:

  ```
  /*
  The four characters in the INSERT string are
  00000041  # LATIN CAPITAL LETTER A
  0001218F  # CUNEIFORM SIGN KAB
  000121A7  # CUNEIFORM SIGN KISH
  00000042  # LATIN CAPITAL LETTER B
  */
  CREATE TABLE t (s1 CHAR(4) CHARACTER SET utf32 COLLATE utf32_unicode_ci);
  INSERT INTO t VALUES (0x000004100001218f000121a700000042);
  SELECT HEX(WEIGHT_STRING(s1)) FROM t;
  ```

  The result is:

  ```
  0E33 FFFD FFFD 0E4A
  ```

  `0E33` and `0E4A` are primary weights as in UCA 4.0.0. `FFFD` is the weight for KAB and also for KISH.

  The rule that all supplementary characters are equal to each other is nonoptimal but is not expected to cause trouble. These characters are very rare, so it is very rare that a multi-character string consists entirely of supplementary characters. In Japan, since the supplementary characters are obscure Kanji ideographs, the typical user does not care what order they are in, anyway. If you really want rows sorted by the MySQL rule and secondarily by code point value, it is easy:

  ```
  ORDER BY s1 COLLATE utf32_unicode_ci, s1 COLLATE utf32_bin
  ```

- For supplementary characters based on UCA versions higher than 4.0.0 (for example, *`xxx`*`_unicode_520_ci`), supplementary characters do not necessarily all have the same collating weight. Some have explicit weights from the UCA `allkeys.txt` file. Others have weights calculated from this algorithm:

  ```
  aaaa= base +  (code >> 15);
  bbbb= (code & 0x7FFF) | 0x8000;
  ```

There is a difference between “ordering by the character's code value” and “ordering by the character's binary representation,” a difference that appears only with `utf16_bin`, because of surrogates.

Suppose that `utf16_bin` (the binary collation for `utf16`) was a binary comparison “byte by byte” rather than “character by character.” If that were so, the order of characters in `utf16_bin` would differ from the order in `utf8_bin`. For example, the following chart shows two rare characters. The first character is in the range `E000`-`FFFF`, so it is greater than a surrogate but less than a supplementary. The second character is a supplementary.

```
Code point  Character                  utf8        utf16
----------  ---------                  ----        -----
```

```
0FF9D       HALFWIDTH KATAKANA LETTER N  EF BE 9D     FF 9D
10384       UGARITIC LETTER DELTA        F0 90 8E 84  D8 00 DF 84
```

The two characters in the chart are in order by code point value because `0xff9d` < `0x10384`. And they are in order by `utf8` value because `0xef` < `0xf0`. But they are not in order by `utf16` value, if we use byte-by-byte comparison, because `0xff` > `0xd8`.

So MySQL's `utf16_bin` collation is not “byte by byte.” It is “by code point.” When MySQL sees a supplementary-character encoding in `utf16`, it converts to the character's code-point value, and then compares. Therefore, `utf8_bin` and `utf16_bin` are the same ordering. This is consistent with the SQL:2008 standard requirement for a UCS_BASIC collation: “UCS_BASIC is a collation in which the ordering is determined entirely by the Unicode scalar values of the characters in the strings being sorted. It is applicable to the UCS character repertoire. Since every character repertoire is a subset of the UCS repertoire, the UCS_BASIC collation is potentially applicable to every character set. NOTE 11: The Unicode scalar value of a character is its code point treated as an unsigned integer.”

If the character set is `ucs2`, comparison is byte-by-byte, but `ucs2` strings should not contain surrogates, anyway.

Miscellaneous Information

The *`xxx`*`_general_mysql500_ci` collations preserve the pre-5.1.24 ordering of the original *`xxx`*`_general_ci` collations and permit upgrades for tables created before MySQL 5.1.24 (Bug #27877).

10.10.2 West European Character Sets

Western European character sets cover most West European languages, such as French, Spanish, Catalan, Basque, Portuguese, Italian, Albanian, Dutch, German, Danish, Swedish, Norwegian, Finnish, Faroese, Icelandic, Irish, Scottish, and English.

- `ascii` (US ASCII) collations:
  - `ascii_bin`
  - `ascii_general_ci` (default)
- `cp850` (DOS West European) collations:
  - `cp850_bin`
  - `cp850_general_ci` (default)
- `dec8` (DEC Western European) collations:
  - `dec8_bin`
  - `dec8_swedish_ci` (default)
- `hp8` (HP Western European) collations:
  - `hp8_bin`
  - `hp8_english_ci` (default)
- `latin1` (cp1252 West European) collations:
  - `latin1_bin`
  - `latin1_danish_ci`
  - `latin1_general_ci`

- latin1_general_cs
- latin1_german1_ci
- latin1_german2_ci
- latin1_spanish_ci
- latin1_swedish_ci (default)

latin1 is the default character set. MySQL's latin1 is the same as the Windows cp1252 character set. This means it is the same as the official ISO 8859-1 or IANA (Internet Assigned Numbers Authority) latin1, except that IANA latin1 treats the code points between 0x80 and 0x9f as "undefined," whereas cp1252, and therefore MySQL's latin1, assign characters for those positions. For example, 0x80 is the Euro sign. For the "undefined" entries in cp1252, MySQL translates 0x81 to Unicode 0x0081, 0x8d to 0x008d, 0x8f to 0x008f, 0x90 to 0x0090, and 0x9d to 0x009d.

The latin1_swedish_ci collation is the default that probably is used by the majority of MySQL customers. Although it is frequently said that it is based on the Swedish/Finnish collation rules, there are Swedes and Finns who disagree with this statement.

The latin1_german1_ci and latin1_german2_ci collations are based on the DIN-1 and DIN-2 standards, where DIN stands for *Deutsches Institut für Normung* (the German equivalent of ANSI). DIN-1 is called the "dictionary collation" and DIN-2 is called the "phone book collation." For an example of the effect this has in comparisons or when doing searches, see Section 10.8.6, "Examples of the Effect of Collation".

- latin1_german1_ci (dictionary) rules:

```
Ä = A
Ö = O
Ü = U
ß = s
```

- latin1_german2_ci (phone-book) rules:

```
Ä = AE
Ö = OE
Ü = UE
ß = ss
```

In the latin1_spanish_ci collation, ñ (n-tilde) is a separate letter between n and o.

- macroman (Mac West European) collations:
  - macroman_bin
  - macroman_general_ci (default)
- swe7 (7bit Swedish) collations:
  - swe7_bin
  - swe7_swedish_ci (default)

10.10.3 Central European Character Sets

MySQL provides some support for character sets used in the Czech Republic, Slovakia, Hungary, Romania, Slovenia, Croatia, Poland, and Serbia (Latin).

- cp1250 (Windows Central European) collations:

  - cp1250_bin
  - cp1250_croatian_ci
  - cp1250_czech_cs
  - cp1250_general_ci (default)
  - cp1250_polish_ci
- cp852 (DOS Central European) collations:
  - cp852_bin
  - cp852_general_ci (default)
- keybcs2 (DOS Kamenicky Czech-Slovak) collations:
  - keybcs2_bin
  - keybcs2_general_ci (default)
- latin2 (ISO 8859-2 Central European) collations:
  - latin2_bin
  - latin2_croatian_ci
  - latin2_czech_cs
  - latin2_general_ci (default)
  - latin2_hungarian_ci
- macce (Mac Central European) collations:
  - macce_bin
  - macce_general_ci (default)

10.10.4 South European and Middle East Character Sets

South European and Middle Eastern character sets supported by MySQL include Armenian, Arabic, Georgian, Greek, Hebrew, and Turkish.

- armscii8 (ARMSCII-8 Armenian) collations:
  - armscii8_bin
  - armscii8_general_ci (default)
- cp1256 (Windows Arabic) collations:
  - cp1256_bin
  - cp1256_general_ci (default)
- geostd8 (GEOSTD8 Georgian) collations:
  - geostd8_bin
  - geostd8_general_ci (default)

- `greek` (ISO 8859-7 Greek) collations:
  - `greek_bin`
  - `greek_general_ci` (default)
- `hebrew` (ISO 8859-8 Hebrew) collations:
  - `hebrew_bin`
  - `hebrew_general_ci` (default)
- `latin5` (ISO 8859-9 Turkish) collations:
  - `latin5_bin`
  - `latin5_turkish_ci` (default)

10.10.5 Baltic Character Sets

The Baltic character sets cover Estonian, Latvian, and Lithuanian languages.

- `cp1257` (Windows Baltic) collations:
  - `cp1257_bin`
  - `cp1257_general_ci` (default)
  - `cp1257_lithuanian_ci`
- `latin7` (ISO 8859-13 Baltic) collations:
  - `latin7_bin`
  - `latin7_estonian_cs`
  - `latin7_general_ci` (default)
  - `latin7_general_cs`

10.10.6 Cyrillic Character Sets

The Cyrillic character sets and collations are for use with Belarusian, Bulgarian, Russian, Ukrainian, and Serbian (Cyrillic) languages.

- `cp1251` (Windows Cyrillic) collations:
  - `cp1251_bin`
  - `cp1251_bulgarian_ci`
  - `cp1251_general_ci` (default)
  - `cp1251_general_cs`
  - `cp1251_ukrainian_ci`
- `cp866` (DOS Russian) collations:
  - `cp866_bin`
  - `cp866_general_ci` (default)

- `koi8r` (KOI8-R Relcom Russian) collations:
  - `koi8r_bin`
  - `koi8r_general_ci` (default)
- `koi8u` (KOI8-U Ukrainian) collations:
  - `koi8u_bin`
  - `koi8u_general_ci` (default)

10.10.7 Asian Character Sets

The Asian character sets that we support include Chinese, Japanese, Korean, and Thai. These can be complicated. For example, the Chinese sets must allow for thousands of different characters. See Section 10.10.7.1, “The cp932 Character Set”, for additional information about the `cp932` and `sjis` character sets.

For answers to some common questions and problems relating support for Asian character sets in MySQL, see Section A.11, “MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”.

- `big5` (Big5 Traditional Chinese) collations:
  - `big5_bin`
  - `big5_chinese_ci` (default)
- `cp932` (SJIS for Windows Japanese) collations:
  - `cp932_bin`
  - `cp932_japanese_ci` (default)
- `eucjpms` (UJIS for Windows Japanese) collations:
  - `eucjpms_bin`
  - `eucjpms_japanese_ci` (default)
- `euckr` (EUC-KR Korean) collations:
  - `euckr_bin`
  - `euckr_korean_ci` (default)
- `gb2312` (GB2312 Simplified Chinese) collations:
  - `gb2312_bin`
  - `gb2312_chinese_ci` (default)
- `gbk` (GBK Simplified Chinese) collations:
  - `gbk_bin`
  - `gbk_chinese_ci` (default)
- `sjis` (Shift-JIS Japanese) collations:
  - `sjis_bin`
  - `sjis_japanese_ci` (default)

- `tis620` (TIS620 Thai) collations:
  - `tis620_bin`
  - `tis620_thai_ci` (default)
- `ujis` (EUC-JP Japanese) collations:
  - `ujis_bin`
  - `ujis_japanese_ci` (default)

The `big5_chinese_ci` collation sorts on number of strokes.

10.10.7.1 The cp932 Character Set

Why is `cp932` needed?

In MySQL, the `sjis` character set corresponds to the `Shift_JIS` character set defined by IANA, which supports JIS X0201 and JIS X0208 characters. (See http://www.iana.org/assignments/character-sets.)

However, the meaning of “SHIFT JIS” as a descriptive term has become very vague and it often includes the extensions to `Shift_JIS` that are defined by various vendors.

For example, “SHIFT JIS” used in Japanese Windows environments is a Microsoft extension of `Shift_JIS` and its exact name is `Microsoft Windows Codepage : 932` or `cp932`. In addition to the characters supported by `Shift_JIS`, `cp932` supports extension characters such as NEC special characters, NEC selected—IBM extended characters, and IBM selected characters.

Many Japanese users have experienced problems using these extension characters. These problems stem from the following factors:

- MySQL automatically converts character sets.
- Character sets are converted using Unicode (`ucs2`).
- The `sjis` character set does not support the conversion of these extension characters.
- There are several conversion rules from so-called “SHIFT JIS” to Unicode, and some characters are converted to Unicode differently depending on the conversion rule. MySQL supports only one of these rules (described later).

The MySQL `cp932` character set is designed to solve these problems.

Because MySQL supports character set conversion, it is important to separate IANA `Shift_JIS` and `cp932` into two different character sets because they provide different conversion rules.

How does `cp932` differ from `sjis`?

The `cp932` character set differs from `sjis` in the following ways:

- `cp932` supports NEC special characters, NEC selected—IBM extended characters, and IBM selected characters.
- Some `cp932` characters have two different code points, both of which convert to the same Unicode code point. When converting from Unicode back to `cp932`, one of the code points must be selected. For this “round trip conversion,” the rule recommended by Microsoft is used. (See http://support.microsoft.com/kb/170559/EN-US/.)

  The conversion rule works like this:

- If the character is in both JIS X 0208 and NEC special characters, use the code point of JIS X 0208.
- If the character is in both NEC special characters and IBM selected characters, use the code point of NEC special characters.
- If the character is in both IBM selected characters and NEC selected—IBM extended characters, use the code point of IBM extended characters.

The table shown at https://msdn.microsoft.com/en-us/goglobal/cc305152.aspx provides information about the Unicode values of `cp932` characters. For `cp932` table entries with characters under which a four-digit number appears, the number represents the corresponding Unicode (`ucs2`) encoding. For table entries with an underlined two-digit value appears, there is a range of `cp932` character values that begin with those two digits. Clicking such a table entry takes you to a page that displays the Unicode value for each of the `cp932` characters that begin with those digits.

The following links are of special interest. They correspond to the encodings for the following sets of characters:

- NEC special characters (lead byte `0x87`):

```
https://msdn.microsoft.com/en-us/goglobal/gg674964
```

- NEC selected—IBM extended characters (lead byte `0xED` and `0xEE`):

```
https://msdn.microsoft.com/en-us/goglobal/gg671837
https://msdn.microsoft.com/en-us/goglobal/gg671838
```

- IBM selected characters (lead byte `0xFA`, `0xFB`, `0xFC`):

```
https://msdn.microsoft.com/en-us/goglobal/gg671839
https://msdn.microsoft.com/en-us/goglobal/gg671840
https://msdn.microsoft.com/en-us/goglobal/gg671841
```

- `cp932` supports conversion of user-defined characters in combination with `eucjpms`, and solves the problems with `sjis`/`ujis` conversion. For details, please refer to http://www.sljfaq.org/afaq/encodings.html.

For some characters, conversion to and from `ucs2` is different for `sjis` and `cp932`. The following tables illustrate these differences.

Conversion to `ucs2`:

`sjis`/`cp932` Value	**`sjis` -> `ucs2` Conversion**	**`cp932` -> `ucs2` Conversion**
5C	005C	005C
7E	007E	007E
815C	2015	2015
815F	005C	FF3C
8160	301C	FF5E
8161	2016	2225
817C	2212	FF0D
8191	00A2	FFE0
8192	00A3	FFE1
81CA	00AC	FFE2

Conversion from `ucs2`:

ucs2 value	ucs2 -> sjis Conversion	ucs2 -> cp932 Conversion
005C	815F	5C
007E	7E	7E
00A2	8191	3F
00A3	8192	3F
00AC	81CA	3F
2015	815C	815C
2016	8161	3F
2212	817C	3F
2225	3F	8161
301C	8160	3F
FF0D	3F	817C
FF3C	3F	815F
FF5E	3F	8160
FFE0	3F	8191
FFE1	3F	8192
FFE2	3F	81CA

Users of any Japanese character sets should be aware that using `--character-set-client-handshake` (or `--skip-character-set-client-handshake`) has an important effect. See Section 5.1.6, “Server Command Options”.

10.10.8 The Binary Character Set

The `binary` character set is the character set for binary strings, which are sequences of bytes. The `binary` character set has one collation, also named `binary`. Comparison and sorting are based on numeric byte values, rather than on numeric character code values (which for multibyte characters differ from numeric byte values). For information about the differences between the `binary` collation of the `binary` character set and the `_bin` collations of nonbinary character sets, see Section 10.8.5, “The binary Collation Compared to _bin Collations”.

For the `binary` character set, the concepts of lettercase and accent equivalence do not apply:

- For single-byte characters stored as binary strings, character and byte boundaries are the same, so lettercase and accent differences are significant in comparisons. That is, the `binary` collation is case-sensitive and accent-sensitive.

```
mysql> SET NAMES 'binary';
mysql> SELECT CHARSET('abc'), COLLATION('abc');
+----------------+------------------+
| CHARSET('abc') | COLLATION('abc') |
+----------------+------------------+
| binary         | binary           |
+----------------+------------------+
mysql> SELECT 'abc' = 'ABC', 'a' = 'ä';
+---------------+------------+
| 'abc' = 'ABC' | 'a' = 'ä'  |
+---------------+------------+
|             0 |          0 |
+---------------+------------+
```

- For multibyte characters stored as binary strings, character and byte boundaries differ. Character boundaries are lost, so comparisons that depend on them are not meaningful.

To perform lettercase conversion of a binary string, first convert it to a nonbinary string using a character set appropriate for the data stored in the string:

```
mysql> SET @str = BINARY 'New York';
mysql> SELECT LOWER(@str), LOWER(CONVERT(@str USING utf8mb4));
+-------------+-----------------------------------+
| LOWER(@str) | LOWER(CONVERT(@str USING utf8mb4)) |
+-------------+-----------------------------------+
| New York    | new york                          |
+-------------+-----------------------------------+
```

To convert a string expression to a binary string, these constructs are equivalent:

```
BINARY expr
CAST(expr AS BINARY)
CONVERT(expr USING BINARY)
```

If a value is a character string literal, the `_binary` introducer may be used to designate it as a binary string. For example:

```
_binary 'a'
```

The `_binary` introducer is permitted for hexadecimal literals and bit-value literals as well, but unnecessary; such literals are binary strings by default.

For more information about introducers, see Section 10.3.8, “Character Set Introducers”.

10.11 Restrictions on Character Sets

- Identifiers are stored in `mysql` database tables (`user`, `db`, and so forth) using `utf8`, but identifiers can contain only characters in the Basic Multilingual Plane (BMP). Supplementary characters are not permitted in identifiers.
- The `ucs2`, `utf16`, `utf16le`, and `utf32` character sets have the following restrictions:
  - None of them can be used as the client character set. See Impermissible Client Character Sets.
  - It is currently not possible to use `LOAD DATA` to load data files that use these character sets.
  - `FULLTEXT` indexes cannot be created on a column that uses any of these character sets. However, you can perform `IN BOOLEAN MODE` searches on the column without an index.
  - The use of `ENCRYPT()` with these character sets is not recommended because the underlying system call expects a string terminated by a zero byte.
- The `REGEXP` and `RLIKE` operators work in byte-wise fashion, so they are not multibyte safe and may produce unexpected results with multibyte character sets. In addition, these operators compare characters by their byte values and accented characters may not compare as equal even if a given collation treats them as equal.

10.12 Setting the Error Message Language

By default, `mysqld` produces error messages in English, but they can be displayed instead in any of several other languages: Czech, Danish, Dutch, Estonian, French, German, Greek, Hungarian, Italian, Japanese, Korean, Norwegian, Norwegian-ny, Polish, Portuguese, Romanian, Russian, Slovak, Spanish, or Swedish. This applies to messages the server writes to the error log and sends to clients.

To select the language in which the server writes error messages, follow the instructions in this section. For information about changing the character set for error messages (rather than the language), see Section 10.6, “Error Message Character Set”. For general information about configuring error logging, see Section 5.4.2, “The Error Log”.

The server searches for the error message file using these rules:

- It looks for the file in a directory constructed from two system variable values, `lc_messages_dir` and `lc_messages`, with the latter converted to a language name. Suppose that you start the server using this command:

  ```
  mysqld --lc_messages_dir=/usr/share/mysql --lc_messages=fr_FR
  ```

  In this case, `mysqld` maps the locale `fr_FR` to the language `french` and looks for the error file in the `/usr/share/mysql/french` directory.

  By default, the language files are located in the `share/mysql/LANGUAGE` directory under the MySQL base directory.

- If the message file cannot be found in the directory constructed as just described, the server ignores the `lc_messages` value and uses only the `lc_messages_dir` value as the location in which to look.

The `lc_messages_dir` system variable can be set only at server startup and has only a global read-only value at runtime. `lc_messages` can be set at server startup and has global and session values that can be modified at runtime. Thus, the error message language can be changed while the server is running, and each client can have its own error message language by setting its session `lc_messages` value to the desired locale name. For example, if the server is using the `fr_FR` locale for error messages, a client can execute this statement to receive error messages in English:

```
SET lc_messages = 'en_US';
```

10.13 Adding a Character Set

This section discusses the procedure for adding a character set to MySQL. The proper procedure depends on whether the character set is simple or complex:

- If the character set does not need special string collating routines for sorting and does not need multibyte character support, it is simple.
- If the character set needs either of those features, it is complex.

For example, `greek` and `swe7` are simple character sets, whereas `big5` and `czech` are complex character sets.

To use the following instructions, you must have a MySQL source distribution. In the instructions, *`MYSET`* represents the name of the character set that you want to add.

1. Add a `<charset>` element for *`MYSET`* to the `sql/share/charsets/Index.xml` file. Use the existing contents in the file as a guide to adding new contents. A partial listing for the `latin1` `<charset>` element follows:

   ```
   <charset name="latin1">
     <family>Western</family>
     <description>cp1252 West European</description>
     ...
     <collation name="latin1_swedish_ci" id="8" order="Finnish, Swedish">
       <flag>primary</flag>
       <flag>compiled</flag>
     </collation>
     <collation name="latin1_danish_ci" id="15" order="Danish"/>
     ...
     <collation name="latin1_bin" id="47" order="Binary">
       <flag>binary</flag>
       <flag>compiled</flag>
     </collation>
     ...
   </charset>
   ```

   The `<charset>` element must list all the collations for the character set. These must include at least a binary collation and a default (primary) collation. The default collation is often named using

a suffix of `general_ci` (general, case-insensitive). It is possible for the binary collation to be the default collation, but usually they are different. The default collation should have a `primary` flag. The binary collation should have a `binary` flag.

You must assign a unique ID number to each collation. The range of IDs from 1024 to 2047 is reserved for user-defined collations. To find the maximum of the currently used collation IDs, use this query:

```
SELECT MAX(ID) FROM INFORMATION_SCHEMA.COLLATIONS;
```

2. This step depends on whether you are adding a simple or complex character set. A simple character set requires only a configuration file, whereas a complex character set requires C source file that defines collation functions, multibyte functions, or both.

   For a simple character set, create a configuration file, *`MYSET`*`.xml`, that describes the character set properties. Create this file in the `sql/share/charsets` directory. You can use a copy of `latin1.xml` as the basis for this file. The syntax for the file is very simple:

   - Comments are written as ordinary XML comments (`<!-- text -->`).
   - Words within `<map>` array elements are separated by arbitrary amounts of whitespace.
   - Each word within `<map>` array elements must be a number in hexadecimal format.
   - The `<map>` array element for the `<ctype>` element has 257 words. The other `<map>` array elements after that have 256 words. See Section 10.13.1, “Character Definition Arrays”.
   - For each collation listed in the `<charset>` element for the character set in `Index.xml`, *`MYSET`*`.xml` must contain a `<collation>` element that defines the character ordering.

   For a complex character set, create a C source file that describes the character set properties and defines the support routines necessary to properly perform operations on the character set:

   - Create the file `ctype-`*`MYSET`*`.c` in the `strings` directory. Look at one of the existing `ctype-*.c` files (such as `ctype-big5.c`) to see what needs to be defined. The arrays in your file must have names like `ctype_`*`MYSET`*, `to_lower_`*`MYSET`*, and so on. These correspond to the arrays for a simple character set. See Section 10.13.1, “Character Definition Arrays”.
   - For each `<collation>` element listed in the `<charset>` element for the character set in `Index.xml`, the `ctype-`*`MYSET`*`.c` file must provide an implementation of the collation.
   - If the character set requires string collating functions, see Section 10.13.2, “String Collating Support for Complex Character Sets”.
   - If the character set requires multibyte character support, see Section 10.13.3, “Multi-Byte Character Support for Complex Character Sets”.

3. Modify the configuration information. Use the existing configuration information as a guide to adding information for *`MYSYS`*. The example here assumes that the character set has default and binary collations, but more lines are needed if *`MYSET`* has additional collations.

   a. Edit `mysys/charset-def.c`, and “register” the collations for the new character set.

      Add these lines to the “declaration” section:

      ```
      #ifdef HAVE_CHARSET_MYSET
      extern CHARSET_INFO my_charset_MYSET_general_ci;
      extern CHARSET_INFO my_charset_MYSET_bin;
      #endif
      ```

      Add these lines to the “registration” section:

      ```
      #ifdef HAVE_CHARSET_MYSET
      ```

```
  add_compiled_collation(&my_charset_MYSET_general_ci);
  add_compiled_collation(&my_charset_MYSET_bin);
#endif
```

b. If the character set uses `ctype-MYSET.c`, edit `strings/CMakeLists.txt` and add `ctype-MYSET.c` to the definition of the `STRINGS_SOURCES` variable.

c. Edit `cmake/character_sets.cmake`:

i. Add *`MYSET`* to the value of with `CHARSETS_AVAILABLE` in alphabetic order.

ii. Add *`MYSET`* to the value of `CHARSETS_COMPLEX` in alphabetic order. This is needed even for simple character sets, or `CMake` does not recognize `-DDEFAULT_CHARSET=MYSET`.

4. Reconfigure, recompile, and test.

10.13.1 Character Definition Arrays

Each simple character set has a configuration file located in the `sql/share/charsets` directory. For a character set named *`MYSYS`*, the file is named `MYSET.xml`. It uses `<map>` array elements to list character set properties. `<map>` elements appear within these elements:

- `<ctype>` defines attributes for each character.
- `<lower>` and `<upper>` list the lowercase and uppercase characters.
- `<unicode>` maps 8-bit character values to Unicode values.
- `<collation>` elements indicate character ordering for comparison and sorting, one element per collation. Binary collations need no `<map>` element because the character codes themselves provide the ordering.

For a complex character set as implemented in a `ctype-MYSET.c` file in the `strings` directory, there are corresponding arrays: `ctype_MYSET[]`, `to_lower_MYSET[]`, and so forth. Not every complex character set has all of the arrays. See also the existing `ctype-*.c` files for examples. See the `CHARSET_INFO.txt` file in the `strings` directory for additional information.

Most of the arrays are indexed by character value and have 256 elements. The `<ctype>` array is indexed by character value + 1 and has 257 elements. This is a legacy convention for handling `EOF`.

`<ctype>` array elements are bit values. Each element describes the attributes of a single character in the character set. Each attribute is associated with a bitmask, as defined in `include/m_ctype.h`:

```
#define _MY_U   01      /* Upper case */
#define _MY_L   02      /* Lower case */
#define _MY_NMR 04      /* Numeral (digit) */
#define _MY_SPC 010     /* Spacing character */
#define _MY_PNT 020     /* Punctuation */
#define _MY_CTR 040     /* Control character */
#define _MY_B   0100    /* Blank */
#define _MY_X   0200    /* heXadecimal digit */
```

The `<ctype>` value for a given character should be the union of the applicable bitmask values that describe the character. For example, `'A'` is an uppercase character (`_MY_U`) as well as a hexadecimal digit (`_MY_X`), so its `ctype` value should be defined like this:

```
ctype['A'+1] = _MY_U | _MY_X = 01 | 0200 = 0201
```

The bitmask values in `m_ctype.h` are octal values, but the elements of the `<ctype>` array in `MYSET.xml` should be written as hexadecimal values.

The `<lower>` and `<upper>` arrays hold the lowercase and uppercase characters corresponding to each member of the character set. For example:

```
lower['A'] should contain 'a'
upper['a'] should contain 'A'
```

Each <collation> array indicates how characters should be ordered for comparison and sorting purposes. MySQL sorts characters based on the values of this information. In some cases, this is the same as the <upper> array, which means that sorting is case-insensitive. For more complicated sorting rules (for complex character sets), see the discussion of string collating in Section 10.13.2, “String Collating Support for Complex Character Sets”.

10.13.2 String Collating Support for Complex Character Sets

For a simple character set named *MYSET*, sorting rules are specified in the *MYSET*.xml configuration file using <map> array elements within <collation> elements. If the sorting rules for your language are too complex to be handled with simple arrays, you must define string collating functions in the ctype-*MYSET*.c source file in the strings directory.

The existing character sets provide the best documentation and examples to show how these functions are implemented. Look at the ctype-*.c files in the strings directory, such as the files for the big5, czech, gbk, sjis, and tis160 character sets. Take a look at the MY_COLLATION_HANDLER structures to see how they are used. See also the CHARSET_INFO.txt file in the strings directory for additional information.

10.13.3 Multi-Byte Character Support for Complex Character Sets

If you want to add support for a new character set named *MYSET* that includes multibyte characters, you must use multibyte character functions in the ctype-*MYSET*.c source file in the strings directory.

The existing character sets provide the best documentation and examples to show how these functions are implemented. Look at the ctype-*.c files in the strings directory, such as the files for the euc_kr, gb2312, gbk, sjis, and ujis character sets. Take a look at the MY_CHARSET_HANDLER structures to see how they are used. See also the CHARSET_INFO.txt file in the strings directory for additional information.

10.14 Adding a Collation to a Character Set

A collation is a set of rules that defines how to compare and sort character strings. Each collation in MySQL belongs to a single character set. Every character set has at least one collation, and most have two or more collations.

A collation orders characters based on weights. Each character in a character set maps to a weight. Characters with equal weights compare as equal, and characters with unequal weights compare according to the relative magnitude of their weights.

The WEIGHT_STRING() function can be used to see the weights for the characters in a string. The value that it returns to indicate weights is a binary string, so it is convenient to use HEX(WEIGHT_STRING(*str*)) to display the weights in printable form. The following example shows that weights do not differ for lettercase for the letters in 'AaBb' if it is a nonbinary case-insensitive string, but do differ if it is a binary string:

```
mysql> SELECT HEX(WEIGHT_STRING('AaBb' COLLATE latin1_swedish_ci));
+-----------------------------------------------------+
| HEX(WEIGHT_STRING('AaBb' COLLATE latin1_swedish_ci)) |
+-----------------------------------------------------+
| 41414242                                            |
+-----------------------------------------------------+
mysql> SELECT HEX(WEIGHT_STRING(BINARY 'AaBb'));
+----------------------------------+
| HEX(WEIGHT_STRING(BINARY 'AaBb')) |
+----------------------------------+
```

```
| 41614262                          |
+-----------------------------------+
```

MySQL supports several collation implementations, as discussed in Section 10.14.1, “Collation Implementation Types”. Some of these can be added to MySQL without recompiling:

- Simple collations for 8-bit character sets.
- UCA-based collations for Unicode character sets.
- Binary (*xxx*_bin) collations.

The following sections describe how to add user-defined collations of the first two types to existing character sets. All existing character sets already have a binary collation, so there is no need here to describe how to add one.

Summary of the procedure for adding a new user-defined collation:

1. Choose a collation ID.
2. Add configuration information that names the collation and describes the character-ordering rules.
3. Restart the server.
4. Verify that the server recognizes the collation.

The instructions here cover only user-defined collations that can be added without recompiling MySQL. To add a collation that does require recompiling (as implemented by means of functions in a C source file), use the instructions in Section 10.13, “Adding a Character Set”. However, instead of adding all the information required for a complete character set, just modify the appropriate files for an existing character set. That is, based on what is already present for the character set's current collations, add data structures, functions, and configuration information for the new collation.

Note

If you modify an existing user-defined collation, that may affect the ordering of rows for indexes on columns that use the collation. In this case, rebuild any such indexes to avoid problems such as incorrect query results. See Section 2.11.10, “Rebuilding or Repairing Tables or Indexes”.

Additional Resources

- Example showing how to add a collation for full-text searches: Section 12.10.7, “Adding a User-Defined Collation for Full-Text Indexing”
- The Unicode Collation Algorithm (UCA) specification: http://www.unicode.org/reports/tr10/
- The Locale Data Markup Language (LDML) specification: http://www.unicode.org/reports/tr35/

10.14.1 Collation Implementation Types

MySQL implements several types of collations:

Simple collations for 8-bit character sets

This kind of collation is implemented using an array of 256 weights that defines a one-to-one mapping from character codes to weights. `latin1_swedish_ci` is an example. It is a case-insensitive collation, so the uppercase and lowercase versions of a character have the same weights and they compare as equal.

```
mysql> SET NAMES 'latin1' COLLATE 'latin1_swedish_ci';
```

```
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT HEX(WEIGHT_STRING('a')), HEX(WEIGHT_STRING('A'));
+-------------------------+-------------------------+
| HEX(WEIGHT_STRING('a')) | HEX(WEIGHT_STRING('A')) |
+-------------------------+-------------------------+
| 41                      | 41                      |
+-------------------------+-------------------------+
1 row in set (0.01 sec)

mysql> SELECT 'a' = 'A';
+-----------+
| 'a' = 'A' |
+-----------+
|         1 |
+-----------+
1 row in set (0.12 sec)
```

For implementation instructions, see Section 10.14.3, “Adding a Simple Collation to an 8-Bit Character Set”.

Complex collations for 8-bit character sets

This kind of collation is implemented using functions in a C source file that define how to order characters, as described in Section 10.13, “Adding a Character Set”.

Collations for non-Unicode multibyte character sets

For this type of collation, 8-bit (single-byte) and multibyte characters are handled differently. For 8-bit characters, character codes map to weights in case-insensitive fashion. (For example, the single-byte characters `'a'` and `'A'` both have a weight of `0x41`.) For multibyte characters, there are two types of relationship between character codes and weights:

- Weights equal character codes. `sjis_japanese_ci` is an example of this kind of collation. The multibyte character `'ぢ'` has a character code of `0x82C0`, and the weight is also `0x82C0`.

```
mysql> CREATE TABLE t1
       (c1 VARCHAR(2) CHARACTER SET sjis COLLATE sjis_japanese_ci);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t1 VALUES ('a'),('A'),(0x82C0);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT c1, HEX(c1), HEX(WEIGHT_STRING(c1)) FROM t1;
+------+---------+-----------------------+
| c1   | HEX(c1) | HEX(WEIGHT_STRING(c1)) |
+------+---------+-----------------------+
| a    | 61      | 41                    |
| A    | 41      | 41                    |
| ぢ    | 82C0    | 82C0                  |
+------+---------+-----------------------+
3 rows in set (0.00 sec)
```

- Character codes map one-to-one to weights, but a code is not necessarily equal to the weight. `gbk_chinese_ci` is an example of this kind of collation. The multibyte character `'膰'` has a character code of `0x81B0` but a weight of `0xC286`.

```
mysql> CREATE TABLE t1
       (c1 VARCHAR(2) CHARACTER SET gbk COLLATE gbk_chinese_ci);
Query OK, 0 rows affected (0.33 sec)

mysql> INSERT INTO t1 VALUES ('a'),('A'),(0x81B0);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT c1, HEX(c1), HEX(WEIGHT_STRING(c1)) FROM t1;
+------+---------+-----------------------+
```

```
| c1   | HEX(c1) | HEX(WEIGHT_STRING(c1)) |
+------+---------+------------------------+
| a    | 61      | 41                     |
| A    | 41      | 41                     |
| 膰    | 81B0    | C286                    |
+------+---------+------------------------+
3 rows in set (0.00 sec)
```

For implementation instructions, see Section 10.13, “Adding a Character Set”.

Collations for Unicode multibyte character sets

Some of these collations are based on the Unicode Collation Algorithm (UCA), others are not.

Non-UCA collations have a one-to-one mapping from character code to weight. In MySQL, such collations are case-insensitive and accent-insensitive. `utf8_general_ci` is an example: `'a'`, `'A'`, `'À'`, and `'á'` each have different character codes but all have a weight of `0x0041` and compare as equal.

```
mysql> SET NAMES 'utf8' COLLATE 'utf8_general_ci';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t1
       (c1 CHAR(1) CHARACTER SET UTF8 COLLATE utf8_general_ci);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t1 VALUES ('a'),('A'),('À'),('á');
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT c1, HEX(c1), HEX(WEIGHT_STRING(c1)) FROM t1;
+------+---------+------------------------+
| c1   | HEX(c1) | HEX(WEIGHT_STRING(c1)) |
+------+---------+------------------------+
| a    | 61      | 0041                   |
| A    | 41      | 0041                   |
| À    | C380    | 0041                   |
| á    | C3A1    | 0041                   |
+------+---------+------------------------+
4 rows in set (0.00 sec)
```

UCA-based collations in MySQL have these properties:

- If a character has weights, each weight uses 2 bytes (16 bits).
- A character may have zero weights (or an empty weight). In this case, the character is ignorable. Example: "U+0000 NULL" does not have a weight and is ignorable.
- A character may have one weight. Example: `'a'` has a weight of `0x0E33`.

```
mysql> SET NAMES 'utf8' COLLATE 'utf8_unicode_ci';
Query OK, 0 rows affected (0.05 sec)

mysql> SELECT HEX('a'), HEX(WEIGHT_STRING('a'));
+----------+-------------------------+
| HEX('a') | HEX(WEIGHT_STRING('a')) |
+----------+-------------------------+
| 61       | 0E33                    |
+----------+-------------------------+
1 row in set (0.02 sec)
```

- A character may have many weights. This is an expansion. Example: The German letter `'ß'` (SZ ligature, or SHARP S) has a weight of `0x0FEA0FEA`.

```
mysql> SET NAMES 'utf8' COLLATE 'utf8_unicode_ci';
Query OK, 0 rows affected (0.11 sec)

mysql> SELECT HEX('ß'), HEX(WEIGHT_STRING('ß'));
```

```
+-----------+-------------------------+
| HEX('ß')  | HEX(WEIGHT_STRING('ß'))  |
+-----------+-------------------------+
| C39F      | 0FEA0FEA                |
+-----------+-------------------------+
1 row in set (0.00 sec)
```

- Many characters may have one weight. This is a contraction. Example: `'ch'` is a single letter in Czech and has a weight of `0x0EE2`.

```
mysql> SET NAMES 'utf8' COLLATE 'utf8_czech_ci';
Query OK, 0 rows affected (0.09 sec)

mysql> SELECT HEX('ch'), HEX(WEIGHT_STRING('ch'));
+-----------+-------------------------+
| HEX('ch') | HEX(WEIGHT_STRING('ch')) |
+-----------+-------------------------+
| 6368      | 0EE2                    |
+-----------+-------------------------+
1 row in set (0.00 sec)
```

A many-characters-to-many-weights mapping is also possible (this is contraction with expansion), but is not supported by MySQL.

For implementation instructions, for a non-UCA collation, see Section 10.13, “Adding a Character Set”. For a UCA collation, see Section 10.14.4, “Adding a UCA Collation to a Unicode Character Set”.

Miscellaneous collations

There are also a few collations that do not fall into any of the previous categories.

10.14.2 Choosing a Collation ID

Each collation must have a unique ID. To add a collation, you must choose an ID value that is not currently used. MySQL supports two-byte collation IDs. The range of IDs from 1024 to 2047 is reserved for user-defined collations.

The collation ID that you choose appears in these contexts:

- The `ID` column of the `INFORMATION_SCHEMA.COLLATIONS` table.
- The `Id` column of `SHOW COLLATION` output.
- The `charsetnr` member of the `MYSQL_FIELD` C API data structure.
- The `number` member of the `MY_CHARSET_INFO` data structure returned by the `mysql_get_character_set_info()` C API function.

To determine the largest currently used ID, issue the following statement:

```
mysql> SELECT MAX(ID) FROM INFORMATION_SCHEMA.COLLATIONS;
+---------+
| MAX(ID) |
+---------+
|     247 |
+---------+
```

To display a list of all currently used IDs, issue this statement:

```
mysql> SELECT ID FROM INFORMATION_SCHEMA.COLLATIONS ORDER BY ID;
+-----+
| ID  |
+-----+
|   1 |
|   2 |
| ... |
```

```
|  52 |
|  53 |
|  57 |
|  58 |
| ... |
|  98 |
|  99 |
| 128 |
| 129 |
| ... |
| 247 |
+-----+
```

Warning

Before upgrading, you should save the configuration files that you change. If you upgrade in place, the process replaces the modified files.

10.14.3 Adding a Simple Collation to an 8-Bit Character Set

This section describes how to add a simple collation for an 8-bit character set by writing the `<collation>` elements associated with a `<charset>` character set description in the MySQL `Index.xml` file. The procedure described here does not require recompiling MySQL. The example adds a collation named `latin1_test_ci` to the `latin1` character set.

1. Choose a collation ID, as shown in Section 10.14.2, “Choosing a Collation ID”. The following steps use an ID of 1024.

2. Modify the `Index.xml` and `latin1.xml` configuration files. These files are located in the directory named by the `character_sets_dir` system variable. You can check the variable value as follows, although the path name might be different on your system:

```
mysql> SHOW VARIABLES LIKE 'character_sets_dir';
+--------------------+----------------------------------------+
| Variable_name      | Value                                  |
+--------------------+----------------------------------------+
| character_sets_dir | /user/local/mysql/share/mysql/charsets/ |
+--------------------+----------------------------------------+
```

3. Choose a name for the collation and list it in the `Index.xml` file. Find the `<charset>` element for the character set to which the collation is being added, and add a `<collation>` element that indicates the collation name and ID, to associate the name with the ID. For example:

```
<charset name="latin1">
  ...
  <collation name="latin1_test_ci" id="1024"/>
  ...
</charset>
```

4. In the `latin1.xml` configuration file, add a `<collation>` element that names the collation and that contains a `<map>` element that defines a character code-to-weight mapping table for character codes 0 to 255. Each value within the `<map>` element must be a number in hexadecimal format.

```
<collation name="latin1_test_ci">
<map>
 00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
 10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
 20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
 30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
 40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
 50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
 60 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
 50 51 52 53 54 55 56 57 58 59 5A 7B 7C 7D 7E 7F
 80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
 90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
 A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
 B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
```

```
 41 41 41 41 5B 5D 5B 43 45 45 45 45 49 49 49 49
 44 4E 4F 4F 4F 4F 5C D7 5C 55 55 55 59 59 DE DF
 41 41 41 41 5B 5D 5B 43 45 45 45 45 49 49 49 49
 44 4E 4F 4F 4F 4F 5C F7 5C 55 55 55 59 59 DE FF
</map>
</collation>
```

5. Restart the server and use this statement to verify that the collation is present:

```
mysql> SHOW COLLATION WHERE Collation = 'latin1_test_ci';
+----------------+---------+------+---------+----------+---------+
| Collation      | Charset | Id   | Default | Compiled | Sortlen |
+----------------+---------+------+---------+----------+---------+
| latin1_test_ci | latin1  | 1024 |         |          |       1 |
+----------------+---------+------+---------+----------+---------+
```

10.14.4 Adding a UCA Collation to a Unicode Character Set

This section describes how to add a UCA collation for a Unicode character set by writing the `<collation>` element within a `<charset>` character set description in the MySQL `Index.xml` file. The procedure described here does not require recompiling MySQL. It uses a subset of the Locale Data Markup Language (LDML) specification, which is available at http://www.unicode.org/reports/tr35/. With this method, you need not define the entire collation. Instead, you begin with an existing “base” collation and describe the new collation in terms of how it differs from the base collation. The following table lists the base collations of the Unicode character sets for which UCA collations can be defined. It is not possible to create user-defined UCA collations for `utf16le`; there is no `utf16le_unicode_ci` collation that would serve as the basis for such collations.

Table 10.4 MySQL Character Sets Available for User-Defined UCA Collations

Character Set	Base Collation
`utf8`	`utf8_unicode_ci`
`ucs2`	`ucs2_unicode_ci`
`utf16`	`utf16_unicode_ci`
`utf32`	`utf32_unicode_ci`

The following sections show how to add a collation that is defined using LDML syntax, and provide a summary of LDML rules supported in MySQL.

10.14.4.1 Defining a UCA Collation Using LDML Syntax

To add a UCA collation for a Unicode character set without recompiling MySQL, use the following procedure. If you are unfamiliar with the LDML rules used to describe the collation's sort characteristics, see Section 10.14.4.2, “LDML Syntax Supported in MySQL”.

The example adds a collation named `utf8_phone_ci` to the `utf8` character set. The collation is designed for a scenario involving a Web application for which users post their names and phone numbers. Phone numbers can be given in very different formats:

```
+7-12345-67
+7-12-345-67
+7 12 345 67
+7 (12) 345 67
+71234567
```

The problem raised by dealing with these kinds of values is that the varying permissible formats make searching for a specific phone number very difficult. The solution is to define a new collation that reorders punctuation characters, making them ignorable.

1. Choose a collation ID, as shown in Section 10.14.2, “Choosing a Collation ID”. The following steps use an ID of 1029.

2. To modify the `Index.xml` configuration file. This file is located in the directory named by the `character_sets_dir` system variable. You can check the variable value as follows, although the path name might be different on your system:

```
mysql> SHOW VARIABLES LIKE 'character_sets_dir';
+--------------------+----------------------------------------+
| Variable_name      | Value                                  |
+--------------------+----------------------------------------+
| character_sets_dir | /user/local/mysql/share/mysql/charsets/ |
+--------------------+----------------------------------------+
```

3. Choose a name for the collation and list it in the `Index.xml` file. In addition, you'll need to provide the collation ordering rules. Find the `<charset>` element for the character set to which the collation is being added, and add a `<collation>` element that indicates the collation name and ID, to associate the name with the ID. Within the `<collation>` element, provide a `<rules>` element containing the ordering rules:

```
<charset name="utf8">
  ...
  <collation name="utf8_phone_ci" id="1029">
    <rules>
      <reset>\u0000</reset>
      <i>\u0020</i> <!-- space -->
      <i>\u0028</i> <!-- left parenthesis -->
      <i>\u0029</i> <!-- right parenthesis -->
      <i>\u002B</i> <!-- plus -->
      <i>\u002D</i> <!-- hyphen -->
    </rules>
  </collation>
  ...
</charset>
```

4. If you want a similar collation for other Unicode character sets, add other `<collation>` elements. For example, to define `ucs2_phone_ci`, add a `<collation>` element to the `<charset name="ucs2">` element. Remember that each collation must have its own unique ID.

5. Restart the server and use this statement to verify that the collation is present:

```
mysql> SHOW COLLATION WHERE Collation = 'utf8_phone_ci';
+---------------+---------+------+---------+----------+---------+
| Collation     | Charset | Id   | Default | Compiled | Sortlen |
+---------------+---------+------+---------+----------+---------+
| utf8_phone_ci | utf8    | 1029 |         |          |       8 |
+---------------+---------+------+---------+----------+---------+
```

Now test the collation to make sure that it has the desired properties.

Create a table containing some sample phone numbers using the new collation:

```
mysql> CREATE TABLE phonebook (
         name VARCHAR(64),
         phone VARCHAR(64) CHARACTER SET utf8 COLLATE utf8_phone_ci
       );
Query OK, 0 rows affected (0.09 sec)

mysql> INSERT INTO phonebook VALUES ('Svoj','+7 912 800 80 02');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Hf','+7 (912) 800 80 04');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Bar','+7-912-800-80-01');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Ramil','(7912) 800 80 03');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO phonebook VALUES ('Sanja','+380 (912) 8008005');
Query OK, 1 row affected (0.00 sec)
```

Run some queries to see whether the ignored punctuation characters are in fact ignored for comparison and sorting:

```
mysql> SELECT * FROM phonebook ORDER BY phone;
+-------+--------------------+
| name  | phone              |
+-------+--------------------+
| Sanja | +380 (912) 8008005 |
| Bar   | +7-912-800-80-01   |
| Svoj  | +7 912 800 80 02   |
| Ramil | (7912) 800 80 03   |
| Hf    | +7 (912) 800 80 04 |
+-------+--------------------+
5 rows in set (0.00 sec)

mysql> SELECT * FROM phonebook WHERE phone='+7(912)800-80-01';
+------+------------------+
| name | phone            |
+------+------------------+
| Bar  | +7-912-800-80-01 |
+------+------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM phonebook WHERE phone='79128008001';
+------+------------------+
| name | phone            |
+------+------------------+
| Bar  | +7-912-800-80-01 |
+------+------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM phonebook WHERE phone='7 9 1 2 8 0 0 8 0 0 1';
+------+------------------+
| name | phone            |
+------+------------------+
| Bar  | +7-912-800-80-01 |
+------+------------------+
1 row in set (0.00 sec)
```

10.14.4.2 LDML Syntax Supported in MySQL

This section describes the LDML syntax that MySQL recognizes. This is a subset of the syntax described in the LDML specification available at http://www.unicode.org/reports/tr35/, which should be consulted for further information. MySQL recognizes a large enough subset of the syntax that, in many cases, it is possible to download a collation definition from the Unicode Common Locale Data Repository and paste the relevant part (that is, the part between the `<rules>` and `</rules>` tags) into the MySQL `Index.xml` file. The rules described here are all supported except that character sorting occurs only at the primary level. Rules that specify differences at secondary or higher sort levels are recognized (and thus can be included in collation definitions) but are treated as equality at the primary level.

The MySQL server generates diagnostics when it finds problems while parsing the `Index.xml` file. See Section 10.14.4.3, “Diagnostics During Index.xml Parsing”.

Character Representation

Characters named in LDML rules can be written literally or in `\u`*nnnn* format, where *nnnn* is the hexadecimal Unicode code point value. For example, `A` and `á` can be written literally or as `\u0041` and `\u00E1`. Within hexadecimal values, the digits `A` through `F` are not case-sensitive; `\u00E1` and `\u00e1` are equivalent. For UCA 4.0.0 collations, hexadecimal notation can be used only for characters in the Basic Multilingual Plane, not for characters outside the BMP range of `0000` to `FFFF`. For UCA 5.2.0 collations, hexadecimal notation can be used for any character.

The `Index.xml` file itself should be written using UTF-8 encoding.

Syntax Rules

LDML has reset rules and shift rules to specify character ordering. Orderings are given as a set of rules that begin with a reset rule that establishes an anchor point, followed by shift rules that indicate how characters sort relative to the anchor point.

- A `<reset>` rule does not specify any ordering in and of itself. Instead, it “resets” the ordering for subsequent shift rules to cause them to be taken in relation to a given character. Either of the following rules resets subsequent shift rules to be taken in relation to the letter `'A'`:

```
<reset>A</reset>

<reset>\u0041</reset>
```

- The `<p>`, `<s>`, and `<t>` shift rules define primary, secondary, and tertiary differences of a character from another character:

  - Use primary differences to distinguish separate letters.
  - Use secondary differences to distinguish accent variations.
  - Use tertiary differences to distinguish lettercase variations.

  Either of these rules specifies a primary shift rule for the `'G'` character:

```
<p>G</p>

<p>\u0047</p>
```

- The `<i>` shift rule indicates that one character sorts identically to another. The following rules cause `'b'` to sort the same as `'a'`:

```
<reset>a</reset>
<i>b</i>
```

- Abbreviated shift syntax specifies multiple shift rules using a single pair of tags. The following table shows the correspondence between abbreviated syntax rules and the equivalent nonabbreviated rules.

  Table 10.5 Abbreviated Shift Syntax

Abbreviated Syntax	Nonabbreviated Syntax
`<pc>xyz</pc>`	`<p>x</p><p>y</p><p>z</p>`
`<sc>xyz</sc>`	`<s>x</s><s>y</s><s>z</s>`
`<tc>xyz</tc>`	`<t>x</t><t>y</t><t>z</t>`
`<ic>xyz</ic>`	`<i>x</i><i>y</i><i>z</i>`

- An expansion is a reset rule that establishes an anchor point for a multiple-character sequence. MySQL supports expansions 2 to 6 characters long. The following rules put `'z'` greater at the primary level than the sequence of three characters `'abc'`:

```
<reset>abc</reset>
<p>z</p>
```

- A contraction is a shift rule that sorts a multiple-character sequence. MySQL supports contractions 2 to 6 characters long. The following rules put the sequence of three characters `'xyz'` greater at the primary level than `'a'`:

```
<reset>a</reset>
<p>xyz</p>
```

- Long expansions and long contractions can be used together. These rules put the sequence of three characters `'xyz'` greater at the primary level than the sequence of three characters `'abc'`:

```
<reset>abc</reset>
```

```
<p>xyz</p>
```

- Normal expansion syntax uses `<x>` plus `<extend>` elements to specify an expansion. The following rules put the character `'k'` greater at the secondary level than the sequence `'ch'`. That is, `'k'` behaves as if it expands to a character after `'c'` followed by `'h'`:

  ```
  <reset>c</reset>
  <x><s>k</s><extend>h</extend></x>
  ```

  This syntax permits long sequences. These rules sort the sequence `'ccs'` greater at the tertiary level than the sequence `'cscs'`:

  ```
  <reset>cs</reset>
  <x><t>ccs</t><extend>cs</extend></x>
  ```

  The LDML specification describes normal expansion syntax as “tricky.” See that specification for details.

- Previous context syntax uses `<x>` plus `<context>` elements to specify that the context before a character affects how it sorts. The following rules put `'-'` greater at the secondary level than `'a'`, but only when `'-'` occurs after `'b'`:

  ```
  <reset>a</reset>
  <x><context>b</context><s>-</s></x>
  ```

- Previous context syntax can include the `<extend>` element. These rules put `'def'` greater at the primary level than `'aghi'`, but only when `'def'` comes after `'abc'`:

  ```
  <reset>a</reset>
  <x><context>abc</context><p>def</p><extend>ghi</extend></x>
  ```

- Reset rules permit a `before` attribute. Normally, shift rules after a reset rule indicate characters that sort after the reset character. Shift rules after a reset rule that has the `before` attribute indicate characters that sort before the reset character. The following rules put the character `'b'` immediately before `'a'` at the primary level:

  ```
  <reset before="primary">a</reset>
  <p>b</p>
  ```

  Permissible `before` attribute values specify the sort level by name or the equivalent numeric value:

  ```
  <reset before="primary">
  <reset before="1">

  <reset before="secondary">
  <reset before="2">

  <reset before="tertiary">
  <reset before="3">
  ```

- A reset rule can name a logical reset position rather than a literal character:

  ```
  <first_tertiary_ignorable/>
  <last_tertiary_ignorable/>
  <first_secondary_ignorable/>
  <last_secondary_ignorable/>
  <first_primary_ignorable/>
  <last_primary_ignorable/>
  <first_variable/>
  <last_variable/>
  <first_non_ignorable/>
  <last_non_ignorable/>
  <first_trailing/>
  <last_trailing/>
  ```

  These rules put `'z'` greater at the primary level than nonignorable characters that have a Default Unicode Collation Element Table (DUCET) entry and that are not CJK:

```
<reset><last_non_ignorable/></reset>
<p>z</p>
```

Logical positions have the code points shown in the following table.

Table 10.6 Logical Reset Position Code Points

Logical Position	Unicode 4.0.0 Code Point	Unicode 5.2.0 Code Point
`<first_non_ignorable/>`	U+02D0	U+02D0
`<last_non_ignorable/>`	U+A48C	U+1342E
`<first_primary_ignorable/>`	U+0332	U+0332
`<last_primary_ignorable/>`	U+20EA	U+101FD
`<first_secondary_ignorable/>`	U+0000	U+0000
`<last_secondary_ignorable/>`	U+FE73	U+FE73
`<first_tertiary_ignorable/>`	U+0000	U+0000
`<last_tertiary_ignorable/>`	U+FE73	U+FE73
`<first_trailing/>`	U+0000	U+0000
`<last_trailing/>`	U+0000	U+0000
`<first_variable/>`	U+0009	U+0009
`<last_variable/>`	U+2183	U+1D371

- The `<collation>` element permits a `shift-after-method` attribute that affects character weight calculation for shift rules. The attribute has these permitted values:

  - `simple`: Calculate character weights as for reset rules that do not have a `before` attribute. This is the default if the attribute is not given.

  - `expand`: Use expansions for shifts after reset rules.

  Suppose that `'0'` and `'1'` have weights of `0E29` and `0E2A` and we want to put all basic Latin letters between `'0'` and `'1'`:

```
<reset>0</reset>
<pc>abcdefghijklmnopqrstuvwxyz</pc>
```

For simple shift mode, weights are calculated as follows:

```
'a' has weight 0E29+1
'b' has weight 0E29+2
'c' has weight 0E29+3
...
```

However, there are not enough vacant positions to put 26 characters between `'0'` and `'1'`. The result is that digits and letters are intermixed.

To solve this, use `shift-after-method="expand"`. Then weights are calculated like this:

```
'a' has weight [0E29][233D+1]
'b' has weight [0E29][233D+2]
'c' has weight [0E29][233D+3]
...
```

`233D` is the UCA 4.0.0 weight for character `0xA48C`, which is the last nonignorable character (a sort of the greatest character in the collation, excluding CJK). UCA 5.2.0 is similar but uses `3ACA`, for character `0x1342E`.

MySQL-Specific LDML Extensions

An extension to LDML rules permits the `<collation>` element to include an optional `version` attribute in `<collation>` tags to indicate the UCA version on which the collation is based. If the `version` attribute is omitted, its default value is `4.0.0`. For example, this specification indicates a collation that is based on UCA 5.2.0:

```
<collation id="nnn" name="utf8_xxx_ci" version="5.2.0">
...
</collation>
```

10.14.4.3 Diagnostics During Index.xml Parsing

The MySQL server generates diagnostics when it finds problems while parsing the `Index.xml` file:

- Unknown tags are written to the error log. For example, the following message results if a collation definition contains a `<aaa>` tag:

  ```
  [Warning] Buffered warning: Unknown LDML tag:
  'charsets/charset/collation/rules/aaa'
  ```

- If collation initialization is not possible, the server reports an “Unknown collation” error, and also generates warnings explaining the problems, such as in the previous example. In other cases, when a collation description is generally correct but contains some unknown tags, the collation is initialized and is available for use. The unknown parts are ignored, but a warning is generated in the error log.

- Problems with collations generate warnings that clients can display with `SHOW WARNINGS`. Suppose that a reset rule contains an expansion longer than the maximum supported length of 6 characters:

  ```
  <reset>abcdefghi</reset>
  <i>x</i>
  ```

  An attempt to use the collation produces warnings:

  ```
  mysql> SELECT _utf8'test' COLLATE utf8_test_ci;
  ERROR 1273 (HY000): Unknown collation: 'utf8_test_ci'
  mysql> SHOW WARNINGS;
  +---------+------+----------------------------------------+
  | Level   | Code | Message                                |
  +---------+------+----------------------------------------+
  | Error   | 1273 | Unknown collation: 'utf8_test_ci'      |
  | Warning | 1273 | Expansion is too long at 'abcdefghi=x' |
  +---------+------+----------------------------------------+
  ```

10.15 Character Set Configuration

The MySQL server has a compiled-in default character set and collation. To change these defaults, use the `--character-set-server` and `--collation-server` options when you start the server. See Section 5.1.6, “Server Command Options”. The collation must be a legal collation for the default character set. To determine which collations are available for each character set, use the `SHOW COLLATION` statement or query the `INFORMATION_SCHEMA COLLATIONS` table.

If you try to use a character set that is not compiled into your binary, you might run into the following problems:

- If your program uses an incorrect path to determine where the character sets are stored (which is typically the `share/mysql/charsets` or `share/charsets` directory under the MySQL installation directory), this can be fixed by using the `--character-sets-dir` option when you run the program. For example, to specify a directory to be used by MySQL client programs, list it in the `[client]` group of your option file. The examples given here show what the setting might look like for Unix or Windows, respectively:

  ```
  [client]
  character-sets-dir=/usr/local/mysql/share/mysql/charsets

  [client]
  character-sets-dir="C:/Program Files/MySQL/MySQL Server 5.6/share/charsets"
  ```

- If the character set is a complex character set that cannot be loaded dynamically, you must recompile the program with support for the character set.

  For Unicode character sets, you can define collations without recompiling by using LDML notation. See Section 10.14.4, “Adding a UCA Collation to a Unicode Character Set”.

- If the character set is a dynamic character set, but you do not have a configuration file for it, you should install the configuration file for the character set from a new MySQL distribution.

- If your character set index file (`Index.xml`) does not contain the name for the character set, your program displays an error message:

  ```
  Character set 'charset_name' is not a compiled character set and is not
  specified in the '/usr/share/mysql/charsets/Index.xml' file
  ```

  To solve this problem, you should either get a new index file or manually add the name of any missing character sets to the current file.

You can force client programs to use specific character set as follows:

```
[client]
default-character-set=charset_name
```

This is normally unnecessary. However, when `character_set_system` differs from `character_set_server` or `character_set_client`, and you input characters manually (as database object identifiers, column values, or both), these may be displayed incorrectly in output from the client or the output itself may be formatted incorrectly. In such cases, starting the mysql client with `--default-character-set=system_character_set`—that is, setting the client character set to match the system character set—should fix the problem.

10.16 MySQL Server Locale Support

The locale indicated by the `lc_time_names` system variable controls the language used to display day and month names and abbreviations. This variable affects the output from the `DATE_FORMAT()`, `DAYNAME()`, and `MONTHNAME()` functions.

`lc_time_names` does not affect the `STR_TO_DATE()` or `GET_FORMAT()` function.

The `lc_time_names` value does not affect the result from `FORMAT()`, but this function takes an optional third parameter that enables a locale to be specified to be used for the result number's decimal point, thousands separator, and grouping between separators. Permissible locale values are the same as the legal values for the `lc_time_names` system variable.

Locale names have language and region subtags listed by IANA (http://www.iana.org/assignments/language-subtag-registry) such as `'ja_JP'` or `'pt_BR'`. The default value is `'en_US'` regardless of your system's locale setting, but you can set the value at server startup, or set the `GLOBAL` value at runtime if you have privileges sufficient to set global system variables; see Section 5.1.8.1, “System Variable Privileges”. Any client can examine the value of `lc_time_names` or set its `SESSION` value to affect the locale for its own connection.

```
mysql> SET NAMES 'utf8';
Query OK, 0 rows affected (0.09 sec)

mysql> SELECT @@lc_time_names;
+-----------------+
| @@lc_time_names |
+-----------------+
| en_US           |
+-----------------+
1 row in set (0.00 sec)

mysql> SELECT DAYNAME('2010-01-01'), MONTHNAME('2010-01-01');
+----------------------+------------------------+
| DAYNAME('2010-01-01') | MONTHNAME('2010-01-01') |
```

```
+----------------------+------------------------+
| Friday               | January                |
+----------------------+------------------------+
1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT('2010-01-01','%W %a %M %b');
+----------------------------------------+
| DATE_FORMAT('2010-01-01','%W %a %M %b') |
+----------------------------------------+
| Friday Fri January Jan                 |
+----------------------------------------+
1 row in set (0.00 sec)

mysql> SET lc_time_names = 'es_MX';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@lc_time_names;
+-----------------+
| @@lc_time_names |
+-----------------+
| es_MX           |
+-----------------+
1 row in set (0.00 sec)

mysql> SELECT DAYNAME('2010-01-01'), MONTHNAME('2010-01-01');
+----------------------+------------------------+
| DAYNAME('2010-01-01') | MONTHNAME('2010-01-01') |
+----------------------+------------------------+
| viernes              | enero                  |
+----------------------+------------------------+
1 row in set (0.00 sec)

mysql> SELECT DATE_FORMAT('2010-01-01','%W %a %M %b');
+----------------------------------------+
| DATE_FORMAT('2010-01-01','%W %a %M %b') |
+----------------------------------------+
| viernes vie enero ene                  |
+----------------------------------------+
1 row in set (0.00 sec)
```

The day or month name for each of the affected functions is converted from `utf8` to the character set indicated by the `character_set_connection` system variable.

`lc_time_names` may be set to any of the following locale values. The set of locales supported by MySQL may differ from those supported by your operating system.

Locale Value	Meaning
`ar_AE`	Arabic - United Arab Emirates
`ar_BH`	Arabic - Bahrain
`ar_DZ`	Arabic - Algeria
`ar_EG`	Arabic - Egypt
`ar_IN`	Arabic - India
`ar_IQ`	Arabic - Iraq
`ar_JO`	Arabic - Jordan
`ar_KW`	Arabic - Kuwait
`ar_LB`	Arabic - Lebanon
`ar_LY`	Arabic - Libya
`ar_MA`	Arabic - Morocco
`ar_OM`	Arabic - Oman
`ar_QA`	Arabic - Qatar
`ar_SA`	Arabic - Saudi Arabia

Locale Value	Meaning
ar_SD	Arabic - Sudan
ar_SY	Arabic - Syria
ar_TN	Arabic - Tunisia
ar_YE	Arabic - Yemen
be_BY	Belarusian - Belarus
bg_BG	Bulgarian - Bulgaria
ca_ES	Catalan - Spain
cs_CZ	Czech - Czech Republic
da_DK	Danish - Denmark
de_AT	German - Austria
de_BE	German - Belgium
de_CH	German - Switzerland
de_DE	German - Germany
de_LU	German - Luxembourg
el_GR	Greek - Greece
en_AU	English - Australia
en_CA	English - Canada
en_GB	English - United Kingdom
en_IN	English - India
en_NZ	English - New Zealand
en_PH	English - Philippines
en_US	English - United States
en_ZA	English - South Africa
en_ZW	English - Zimbabwe
es_AR	Spanish - Argentina
es_BO	Spanish - Bolivia
es_CL	Spanish - Chile
es_CO	Spanish - Colombia
es_CR	Spanish - Costa Rica
es_DO	Spanish - Dominican Republic
es_EC	Spanish - Ecuador
es_ES	Spanish - Spain
es_GT	Spanish - Guatemala
es_HN	Spanish - Honduras
es_MX	Spanish - Mexico
es_NI	Spanish - Nicaragua
es_PA	Spanish - Panama
es_PE	Spanish - Peru
es_PR	Spanish - Puerto Rico
es_PY	Spanish - Paraguay
es_SV	Spanish - El Salvador

Locale Value	Meaning
es_US	Spanish - United States
es_UY	Spanish - Uruguay
es_VE	Spanish - Venezuela
et_EE	Estonian - Estonia
eu_ES	Basque - Spain
fi_FI	Finnish - Finland
fo_FO	Faroese - Faroe Islands
fr_BE	French - Belgium
fr_CA	French - Canada
fr_CH	French - Switzerland
fr_FR	French - France
fr_LU	French - Luxembourg
gl_ES	Galician - Spain
gu_IN	Gujarati - India
he_IL	Hebrew - Israel
hi_IN	Hindi - India
hr_HR	Croatian - Croatia
hu_HU	Hungarian - Hungary
id_ID	Indonesian - Indonesia
is_IS	Icelandic - Iceland
it_CH	Italian - Switzerland
it_IT	Italian - Italy
ja_JP	Japanese - Japan
ko_KR	Korean - Republic of Korea
lt_LT	Lithuanian - Lithuania
lv_LV	Latvian - Latvia
mk_MK	Macedonian - North Macedonia
mn_MN	Mongolia - Mongolian
ms_MY	Malay - Malaysia
nb_NO	Norwegian(Bokmål) - Norway
nl_BE	Dutch - Belgium
nl_NL	Dutch - The Netherlands
no_NO	Norwegian - Norway
pl_PL	Polish - Poland
pt_BR	Portugese - Brazil
pt_PT	Portugese - Portugal
rm_CH	Romansh - Switzerland
ro_RO	Romanian - Romania
ru_RU	Russian - Russia
ru_UA	Russian - Ukraine
sk_SK	Slovak - Slovakia

Locale Value	Meaning
sl_SI	Slovenian - Slovenia
sq_AL	Albanian - Albania
sr_RS	Serbian - Serbia
sv_FI	Swedish - Finland
sv_SE	Swedish - Sweden
ta_IN	Tamil - India
te_IN	Telugu - India
th_TH	Thai - Thailand
tr_TR	Turkish - Turkey
uk_UA	Ukrainian - Ukraine
ur_PK	Urdu - Pakistan
vi_VN	Vietnamese - Vietnam
zh_CN	Chinese - China
zh_HK	Chinese - Hong Kong
zh_TW	Chinese - Taiwan

Chapter 11 Data Types

Table of Contents

11.1 Numeric Data Types 1252
11.1.1 Numeric Data Type Syntax 1252
11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT 1255
11.1.3 Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC 1256
11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE 1256
11.1.5 Bit-Value Type - BIT 1257
11.1.6 Numeric Type Attributes 1257
11.1.7 Out-of-Range and Overflow Handling 1258
11.2 Date and Time Data Types 1259
11.2.1 Date and Time Data Type Syntax 1261
11.2.2 The DATE, DATETIME, and TIMESTAMP Types 1263
11.2.3 The TIME Type 1264
11.2.4 The YEAR Type 1265
11.2.5 2-Digit YEAR(2) Limitations and Migrating to 4-Digit YEAR 1266
11.2.6 Automatic Initialization and Updating for TIMESTAMP and DATETIME 1268
11.2.7 Fractional Seconds in Time Values 1272
11.2.8 Conversion Between Date and Time Types 1273
11.2.9 2-Digit Years in Dates 1274
11.3 String Data Types 1274
11.3.1 String Data Type Syntax 1274
11.3.2 The CHAR and VARCHAR Types 1278
11.3.3 The BINARY and VARBINARY Types 1279
11.3.4 The BLOB and TEXT Types 1280
11.3.5 The ENUM Type 1282
11.3.6 The SET Type 1285
11.4 Spatial Data Types 1287
11.4.1 Spatial Data Types 1289
11.4.2 The OpenGIS Geometry Model 1289
11.4.3 Supported Spatial Data Formats 1295
11.4.4 Creating Spatial Columns 1297
11.4.5 Populating Spatial Columns 1298
11.4.6 Fetching Spatial Data 1299
11.4.7 Optimizing Spatial Analysis 1299
11.4.8 Creating Spatial Indexes 1299
11.4.9 Using Spatial Indexes 1300
11.5 Data Type Default Values 1302
11.6 Data Type Storage Requirements 1303
11.7 Choosing the Right Type for a Column 1307
11.8 Using Data Types from Other Database Engines 1308

MySQL supports SQL data types in several categories: numeric types, date and time types, string (character and byte) types, and spatial types. This chapter provides an overview and more detailed description of the properties of the types in each category, and a summary of the data type storage requirements. The initial overviews are intentionally brief. Consult the more detailed descriptions for additional information about particular data types, such as the permissible formats in which you can specify values.

Data type descriptions use these conventions:

- For integer types, *M* indicates the maximum display width. For floating-point and fixed-point types, *M* is the total number of digits that can be stored (the precision). For string types, *M* is the maximum length. The maximum permissible value of *M* depends on the data type.

- *D* applies to floating-point and fixed-point types and indicates the number of digits following the decimal point (the scale). The maximum possible value is 30, but should be no greater than *M*−2.
- *fsp* applies to the TIME, DATETIME, and TIMESTAMP types and represents fractional seconds precision; that is, the number of digits following the decimal point for fractional parts of seconds. The *fsp* value, if given, must be in the range 0 to 6. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0. (This differs from the standard SQL default of 6, for compatibility with previous MySQL versions.)
- Square brackets ([and]) indicate optional parts of type definitions.

11.1 Numeric Data Types

MySQL supports all standard SQL numeric data types. These types include the exact numeric data types (INTEGER, SMALLINT, DECIMAL, and NUMERIC), as well as the approximate numeric data types (FLOAT, REAL, and DOUBLE PRECISION). The keyword INT is a synonym for INTEGER, and the keywords DEC and FIXED are synonyms for DECIMAL. MySQL treats DOUBLE as a synonym for DOUBLE PRECISION (a nonstandard extension). MySQL also treats REAL as a synonym for DOUBLE PRECISION (a nonstandard variation), unless the REAL_AS_FLOAT SQL mode is enabled.

The BIT data type stores bit values and is supported for MyISAM, MEMORY, InnoDB, and NDB tables.

For information about how MySQL handles assignment of out-of-range values to columns and overflow during expression evaluation, see Section 11.1.7, “Out-of-Range and Overflow Handling”.

For information about storage requirements of the numeric data types, see Section 11.6, “Data Type Storage Requirements”.

For descriptions of functions that operate on numeric values, see Section 12.6, “Numeric Functions and Operators”. The data type used for the result of a calculation on numeric operands depends on the types of the operands and the operations performed on them. For more information, see Section 12.6.1, “Arithmetic Operators”.

11.1.1 Numeric Data Type Syntax

For integer data types, *M* indicates the maximum display width. The maximum display width is 255. Display width is unrelated to the range of values a type can store, as described in Section 11.1.6, “Numeric Type Attributes”.

For floating-point and fixed-point data types, *M* is the total number of digits that can be stored.

If you specify ZEROFILL for a numeric column, MySQL automatically adds the UNSIGNED attribute to the column.

Numeric data types that permit the UNSIGNED attribute also permit SIGNED. However, these data types are signed by default, so the SIGNED attribute has no effect.

SERIAL is an alias for BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE.

SERIAL DEFAULT VALUE in the definition of an integer column is an alias for NOT NULL AUTO_INCREMENT UNIQUE.

Warning

When you use subtraction between integer values where one is of type UNSIGNED, the result is unsigned unless the NO_UNSIGNED_SUBTRACTION SQL mode is enabled. See Section 12.11, “Cast Functions and Operators”.

- BIT[(*M*)]

A bit-value type. *M* indicates the number of bits per value, from 1 to 64. The default is 1 if *M* is omitted.

- `TINYINT[(M)] [UNSIGNED] [ZEROFILL]`

  A very small integer. The signed range is `-128` to `127`. The unsigned range is `0` to `255`.

- `BOOL`, `BOOLEAN`

  These types are synonyms for `TINYINT(1)`. A value of zero is considered false. Nonzero values are considered true:

```
mysql> SELECT IF(0, 'true', 'false');
+-----------------------+
| IF(0, 'true', 'false') |
+-----------------------+
| false                 |
+-----------------------+

mysql> SELECT IF(1, 'true', 'false');
+-----------------------+
| IF(1, 'true', 'false') |
+-----------------------+
| true                  |
+-----------------------+

mysql> SELECT IF(2, 'true', 'false');
+-----------------------+
| IF(2, 'true', 'false') |
+-----------------------+
| true                  |
+-----------------------+
```

  However, the values `TRUE` and `FALSE` are merely aliases for `1` and `0`, respectively, as shown here:

```
mysql> SELECT IF(0 = FALSE, 'true', 'false');
+--------------------------------+
| IF(0 = FALSE, 'true', 'false') |
+--------------------------------+
| true                           |
+--------------------------------+

mysql> SELECT IF(1 = TRUE, 'true', 'false');
+-------------------------------+
| IF(1 = TRUE, 'true', 'false') |
+-------------------------------+
| true                          |
+-------------------------------+

mysql> SELECT IF(2 = TRUE, 'true', 'false');
+-------------------------------+
| IF(2 = TRUE, 'true', 'false') |
+-------------------------------+
| false                         |
+-------------------------------+

mysql> SELECT IF(2 = FALSE, 'true', 'false');
+--------------------------------+
| IF(2 = FALSE, 'true', 'false') |
+--------------------------------+
| false                          |
+--------------------------------+
```

  The last two statements display the results shown because `2` is equal to neither `1` nor `0`.

- `SMALLINT[(M)] [UNSIGNED] [ZEROFILL]`

  A small integer. The signed range is `-32768` to `32767`. The unsigned range is `0` to `65535`.

- `MEDIUMINT[(M)] [UNSIGNED] [ZEROFILL]`

A medium-sized integer. The signed range is -8388608 to 8388607. The unsigned range is 0 to 16777215.

- INT[(*M*)] [UNSIGNED] [ZEROFILL]

  A normal-size integer. The signed range is -2147483648 to 2147483647. The unsigned range is 0 to 4294967295.

- INTEGER[(*M*)] [UNSIGNED] [ZEROFILL]

  This type is a synonym for INT.

- BIGINT[(*M*)] [UNSIGNED] [ZEROFILL]

  A large integer. The signed range is -9223372036854775808 to 9223372036854775807. The unsigned range is 0 to 18446744073709551615.

  SERIAL is an alias for BIGINT UNSIGNED NOT NULL AUTO_INCREMENT UNIQUE.

  Some things you should be aware of with respect to BIGINT columns:

  - All arithmetic is done using signed BIGINT or DOUBLE values, so you should not use unsigned big integers larger than 9223372036854775807 (63 bits) except with bit functions! If you do that, some of the last digits in the result may be wrong because of rounding errors when converting a BIGINT value to a DOUBLE.

    MySQL can handle BIGINT in the following cases:

    - When using integers to store large unsigned values in a BIGINT column.
    - In MIN(*col_name*) or MAX(*col_name*), where *col_name* refers to a BIGINT column.
    - When using operators (+, -, *, and so on) where both operands are integers.

  - You can always store an exact integer value in a BIGINT column by storing it using a string. In this case, MySQL performs a string-to-number conversion that involves no intermediate double-precision representation.
  - The -, +, and * operators use BIGINT arithmetic when both operands are integer values. This means that if you multiply two big integers (or results from functions that return integers), you may get unexpected results when the result is larger than 9223372036854775807.

- DECIMAL[(*M*[,*D*])] [UNSIGNED] [ZEROFILL]

  A packed “exact” fixed-point number. *M* is the total number of digits (the precision) and *D* is the number of digits after the decimal point (the scale). The decimal point and (for negative numbers) the - sign are not counted in *M*. If *D* is 0, values have no decimal point or fractional part. The maximum number of digits (*M*) for DECIMAL is 65. The maximum number of supported decimals (*D*) is 30. If *D* is omitted, the default is 0. If *M* is omitted, the default is 10. (There is also a limit on how long the text of DECIMAL literals can be; see Section 12.21.3, “Expression Handling”.)

  UNSIGNED, if specified, disallows negative values.

  All basic calculations (+, -, *, /) with DECIMAL columns are done with a precision of 65 digits.

- DEC[(*M*[,*D*])] [UNSIGNED] [ZEROFILL], NUMERIC[(*M*[,*D*])] [UNSIGNED] [ZEROFILL], FIXED[(*M*[,*D*])] [UNSIGNED] [ZEROFILL]

  These types are synonyms for DECIMAL. The FIXED synonym is available for compatibility with other database systems.

- `FLOAT[(M,D)] [UNSIGNED] [ZEROFILL]`

  A small (single-precision) floating-point number. Permissible values are `-3.402823466E+38` to `-1.175494351E-38`, `0`, and `1.175494351E-38` to `3.402823466E+38`. These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.

  M is the total number of digits and *D* is the number of digits following the decimal point. If *M* and *D* are omitted, values are stored to the limits permitted by the hardware. A single-precision floating-point number is accurate to approximately 7 decimal places.

  `FLOAT(M,D)` is a nonstandard MySQL extension.

  `UNSIGNED`, if specified, disallows negative values.

  Using `FLOAT` might give you some unexpected problems because all calculations in MySQL are done with double precision. See Section B.3.4.7, “Solving Problems with No Matching Rows”.

- `FLOAT(p) [UNSIGNED] [ZEROFILL]`

  A floating-point number. *p* represents the precision in bits, but MySQL uses this value only to determine whether to use `FLOAT` or `DOUBLE` for the resulting data type. If *p* is from 0 to 24, the data type becomes `FLOAT` with no *M* or *D* values. If *p* is from 25 to 53, the data type becomes `DOUBLE` with no *M* or *D* values. The range of the resulting column is the same as for the single-precision `FLOAT` or double-precision `DOUBLE` data types described earlier in this section.

  `FLOAT(p)` syntax is provided for ODBC compatibility.

- `DOUBLE[(M,D)] [UNSIGNED] [ZEROFILL]`

  A normal-size (double-precision) floating-point number. Permissible values are `-1.7976931348623157E+308` to `-2.2250738585072014E-308`, `0`, and `2.2250738585072014E-308` to `1.7976931348623157E+308`. These are the theoretical limits, based on the IEEE standard. The actual range might be slightly smaller depending on your hardware or operating system.

  M is the total number of digits and *D* is the number of digits following the decimal point. If *M* and *D* are omitted, values are stored to the limits permitted by the hardware. A double-precision floating-point number is accurate to approximately 15 decimal places.

  `DOUBLE(M,D)` is a nonstandard MySQL extension.

  `UNSIGNED`, if specified, disallows negative values.

- `DOUBLE PRECISION[(M,D)] [UNSIGNED] [ZEROFILL]`, `REAL[(M,D)] [UNSIGNED] [ZEROFILL]`

  These types are synonyms for `DOUBLE`. Exception: If the `REAL_AS_FLOAT` SQL mode is enabled, `REAL` is a synonym for `FLOAT` rather than `DOUBLE`.

11.1.2 Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT

MySQL supports the SQL standard integer types `INTEGER` (or `INT`) and `SMALLINT`. As an extension to the standard, MySQL also supports the integer types `TINYINT`, `MEDIUMINT`, and `BIGINT`. The following table shows the required storage and range for each integer type.

Table 11.1 Required Storage and Range for Integer Types Supported by MySQL

Type	Storage (Bytes)	Minimum Value Signed	Minimum Value Unsigned	Maximum Value Signed	Maximum Value Unsigned
TINYINT	1	-128	0	127	255
SMALLINT	2	-32768	0	32767	65535
MEDIUMINT	3	-8388608	0	8388607	16777215
INT	4	-2147483648	0	2147483647	4294967295
BIGINT	8	-2^{63}	0	$2^{63}-1$	$2^{64}-1$

11.1.3 Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC

The DECIMAL and NUMERIC types store exact numeric data values. These types are used when it is important to preserve exact precision, for example with monetary data. In MySQL, NUMERIC is implemented as DECIMAL, so the following remarks about DECIMAL apply equally to NUMERIC.

MySQL stores DECIMAL values in binary format. See Section 12.21, “Precision Math”.

In a DECIMAL column declaration, the precision and scale can be (and usually is) specified. For example:

```
salary DECIMAL(5,2)
```

In this example, 5 is the precision and 2 is the scale. The precision represents the number of significant digits that are stored for values, and the scale represents the number of digits that can be stored following the decimal point.

Standard SQL requires that DECIMAL(5,2) be able to store any value with five digits and two decimals, so values that can be stored in the salary column range from -999.99 to 999.99.

In standard SQL, the syntax DECIMAL(*M*) is equivalent to DECIMAL(*M*,0). Similarly, the syntax DECIMAL is equivalent to DECIMAL(*M*,0), where the implementation is permitted to decide the value of *M*. MySQL supports both of these variant forms of DECIMAL syntax. The default value of *M* is 10.

If the scale is 0, DECIMAL values contain no decimal point or fractional part.

The maximum number of digits for DECIMAL is 65, but the actual range for a given DECIMAL column can be constrained by the precision or scale for a given column. When such a column is assigned a value with more digits following the decimal point than are permitted by the specified scale, the value is converted to that scale. (The precise behavior is operating system-specific, but generally the effect is truncation to the permissible number of digits.)

11.1.4 Floating-Point Types (Approximate Value) - FLOAT, DOUBLE

The FLOAT and DOUBLE types represent approximate numeric data values. MySQL uses four bytes for single-precision values and eight bytes for double-precision values.

For FLOAT, the SQL standard permits an optional specification of the precision (but not the range of the exponent) in bits following the keyword FLOAT in parentheses; ; that is, FLOAT(*p*). MySQL also supports this optional precision specification, but the precision value in FLOAT(*p*) is used only to determine storage size. A precision from 0 to 23 results in a 4-byte single-precision FLOAT column. A precision from 24 to 53 results in an 8-byte double-precision DOUBLE column.

MySQL permits a nonstandard syntax: FLOAT(*M*,*D*) or REAL(*M*,*D*) or DOUBLE PRECISION(*M*,*D*). Here, (*M*,*D*) means than values can be stored with up to *M* digits in total, of which *D* digits may be after the decimal point. For example, a column defined as FLOAT(7,4) looks like -999.9999 when displayed. MySQL performs rounding when storing values, so if you insert 999.00009 into a FLOAT(7,4) column, the approximate result is 999.0001.

Because floating-point values are approximate and not stored as exact values, attempts to treat them as exact in comparisons may lead to problems. They are also subject to platform or implementation dependencies. For more information, see Section B.3.4.8, “Problems with Floating-Point Values”

For maximum portability, code requiring storage of approximate numeric data values should use `FLOAT` or `DOUBLE PRECISION` with no specification of precision or number of digits.

11.1.5 Bit-Value Type - BIT

The `BIT` data type is used to store bit values. A type of `BIT(M)` enables storage of *`M`*-bit values. *`M`* can range from 1 to 64.

To specify bit values, `b'value'` notation can be used. *`value`* is a binary value written using zeros and ones. For example, `b'111'` and `b'10000000'` represent 7 and 128, respectively. See Section 9.1.5, “Bit-Value Literals”.

If you assign a value to a `BIT(M)` column that is less than *`M`* bits long, the value is padded on the left with zeros. For example, assigning a value of `b'101'` to a `BIT(6)` column is, in effect, the same as assigning `b'000101'`.

NDB Cluster. The maximum combined size of all `BIT` columns used in a given `NDB` table must not exceed 4096 bits.

11.1.6 Numeric Type Attributes

MySQL supports an extension for optionally specifying the display width of integer data types in parentheses following the base keyword for the type. For example, `INT(4)` specifies an `INT` with a display width of four digits. This optional display width may be used by applications to display integer values having a width less than the width specified for the column by left-padding them with spaces. (That is, this width is present in the metadata returned with result sets. Whether it is used is up to the application.)

The display width does *not* constrain the range of values that can be stored in the column. Nor does it prevent values wider than the column display width from being displayed correctly. For example, a column specified as `SMALLINT(3)` has the usual `SMALLINT` range of `-32768` to `32767`, and values outside the range permitted by three digits are displayed in full using more than three digits.

When used in conjunction with the optional (nonstandard) `ZEROFILL` attribute, the default padding of spaces is replaced with zeros. For example, for a column declared as `INT(4) ZEROFILL`, a value of `5` is retrieved as `0005`.

Note

The `ZEROFILL` attribute is ignored for columns involved in expressions or `UNION` queries.

If you store values larger than the display width in an integer column that has the `ZEROFILL` attribute, you may experience problems when MySQL generates temporary tables for some complicated joins. In these cases, MySQL assumes that the data values fit within the column display width.

All integer types can have an optional (nonstandard) `UNSIGNED` attribute. An unsigned type can be used to permit only nonnegative numbers in a column or when you need a larger upper numeric range for the column. For example, if an `INT` column is `UNSIGNED`, the size of the column's range is the same but its endpoints shift up, from `-2147483648` and `2147483647` to `0` and `4294967295`.

Floating-point and fixed-point types also can be `UNSIGNED`. As with integer types, this attribute prevents negative values from being stored in the column. Unlike the integer types, the upper range of column values remains the same.

If you specify `ZEROFILL` for a numeric column, MySQL automatically adds the `UNSIGNED` attribute.

Integer or floating-point data types can have the AUTO_INCREMENT attribute. When you insert a value of NULL into an indexed AUTO_INCREMENT column, the column is set to the next sequence value. Typically this is *value*+1, where *value* is the largest value for the column currently in the table. (AUTO_INCREMENT sequences begin with 1.)

Storing 0 into an AUTO_INCREMENT column has the same effect as storing NULL, unless the NO_AUTO_VALUE_ON_ZERO SQL mode is enabled.

Inserting NULL to generate AUTO_INCREMENT values requires that the column be declared NOT NULL. If the column is declared NULL, inserting NULL stores a NULL. When you insert any other value into an AUTO_INCREMENT column, the column is set to that value and the sequence is reset so that the next automatically generated value follows sequentially from the inserted value.

Negative values for AUTO_INCREMENT columns are not supported.

11.1.7 Out-of-Range and Overflow Handling

When MySQL stores a value in a numeric column that is outside the permissible range of the column data type, the result depends on the SQL mode in effect at the time:

- If strict SQL mode is enabled, MySQL rejects the out-of-range value with an error, and the insert fails, in accordance with the SQL standard.
- If no restrictive modes are enabled, MySQL clips the value to the appropriate endpoint of the column data type range and stores the resulting value instead.

  When an out-of-range value is assigned to an integer column, MySQL stores the value representing the corresponding endpoint of the column data type range.

  When a floating-point or fixed-point column is assigned a value that exceeds the range implied by the specified (or default) precision and scale, MySQL stores the value representing the corresponding endpoint of that range.

Suppose that a table t1 has this definition:

```
CREATE TABLE t1 (i1 TINYINT, i2 TINYINT UNSIGNED);
```

With strict SQL mode enabled, an out of range error occurs:

```
mysql> SET sql_mode = 'TRADITIONAL';
mysql> INSERT INTO t1 (i1, i2) VALUES(256, 256);
ERROR 1264 (22003): Out of range value for column 'i1' at row 1
mysql> SELECT * FROM t1;
Empty set (0.00 sec)
```

With strict SQL mode not enabled, clipping with warnings occurs:

```
mysql> SET sql_mode = '';
mysql> INSERT INTO t1 (i1, i2) VALUES(256, 256);
mysql> SHOW WARNINGS;
+---------+------+-------------------------------------------+
| Level   | Code | Message                                   |
+---------+------+-------------------------------------------+
| Warning | 1264 | Out of range value for column 'i1' at row 1 |
| Warning | 1264 | Out of range value for column 'i2' at row 1 |
+---------+------+-------------------------------------------+
mysql> SELECT * FROM t1;
+------+------+
| i1   | i2   |
+------+------+
|  127 |  255 |
+------+------+
```

When strict SQL mode is not enabled, column-assignment conversions that occur due to clipping are reported as warnings for ALTER TABLE, LOAD DATA, UPDATE, and multiple-row INSERT statements. In strict mode, these statements fail, and some or all the values are not inserted or changed, depending

on whether the table is a transactional table and other factors. For details, see Section 5.1.10, “Server SQL Modes”.

Overflow during numeric expression evaluation results in an error. For example, the largest signed `BIGINT` value is 9223372036854775807, so the following expression produces an error:

```
mysql> SELECT 9223372036854775807 + 1;
ERROR 1690 (22003): BIGINT value is out of range in '(9223372036854775807 + 1)'
```

To enable the operation to succeed in this case, convert the value to unsigned;

```
mysql> SELECT CAST(9223372036854775807 AS UNSIGNED) + 1;
+-----------------------------------------+
| CAST(9223372036854775807 AS UNSIGNED) + 1 |
+-----------------------------------------+
|                     9223372036854775808 |
+-----------------------------------------+
```

Whether overflow occurs depends on the range of the operands, so another way to handle the preceding expression is to use exact-value arithmetic because `DECIMAL` values have a larger range than integers:

```
mysql> SELECT 9223372036854775807.0 + 1;
+--------------------------+
| 9223372036854775807.0 + 1 |
+--------------------------+
|     9223372036854775808.0 |
+--------------------------+
```

Subtraction between integer values, where one is of type `UNSIGNED`, produces an unsigned result by default. If the result would otherwise have been negative, an error results:

```
mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT CAST(0 AS UNSIGNED) - 1;
ERROR 1690 (22003): BIGINT UNSIGNED value is out of range in '(cast(0 as unsigned) - 1)'
```

If the `NO_UNSIGNED_SUBTRACTION` SQL mode is enabled, the result is negative:

```
mysql> SET sql_mode = 'NO_UNSIGNED_SUBTRACTION';
mysql> SELECT CAST(0 AS UNSIGNED) - 1;
+-------------------------+
| CAST(0 AS UNSIGNED) - 1 |
+-------------------------+
|                      -1 |
+-------------------------+
```

If the result of such an operation is used to update an `UNSIGNED` integer column, the result is clipped to the maximum value for the column type, or clipped to 0 if `NO_UNSIGNED_SUBTRACTION` is enabled. If strict SQL mode is enabled, an error occurs and the column remains unchanged.

11.2 Date and Time Data Types

The date and time data types for representing temporal values are `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`, and `YEAR`. Each temporal type has a range of valid values, as well as a “zero” value that may be used when you specify an invalid value that MySQL cannot represent. The `TIMESTAMP` and `DATETIME` types have special automatic updating behavior, described in Section 11.2.6, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”.

For information about storage requirements of the temporal data types, see Section 11.6, “Data Type Storage Requirements”.

For descriptions of functions that operate on temporal values, see Section 12.7, “Date and Time Functions”.

Keep in mind these general considerations when working with date and time types:

- MySQL retrieves values for a given date or time type in a standard output format, but it attempts to interpret a variety of formats for input values that you supply (for example, when you specify a value to be assigned to or compared to a date or time type). For a description of the permitted formats for date and time types, see Section 9.1.3, “Date and Time Literals”. It is expected that you supply valid values. Unpredictable results may occur if you use values in other formats.

- Although MySQL tries to interpret values in several formats, date parts must always be given in year-month-day order (for example, `'98-09-04'`), rather than in the month-day-year or day-month-year orders commonly used elsewhere (for example, `'09-04-98'`, `'04-09-98'`). To convert strings in other orders to year-month-day order, the `STR_TO_DATE()` function may be useful.

- Dates containing 2-digit year values are ambiguous because the century is unknown. MySQL interprets 2-digit year values using these rules:

  - Year values in the range `70-99` become `1970-1999`.

  - Year values in the range `00-69` become `2000-2069`.

  See also Section 11.2.9, “2-Digit Years in Dates”.

- Conversion of values from one temporal type to another occurs according to the rules in Section 11.2.8, “Conversion Between Date and Time Types”.

- MySQL automatically converts a date or time value to a number if the value is used in numeric context and vice versa.

- By default, when MySQL encounters a value for a date or time type that is out of range or otherwise invalid for the type, it converts the value to the “zero” value for that type. The exception is that out-of-range `TIME` values are clipped to the appropriate endpoint of the `TIME` range.

- By setting the SQL mode to the appropriate value, you can specify more exactly what kind of dates you want MySQL to support. (See Section 5.1.10, “Server SQL Modes”.) You can get MySQL to accept certain dates, such as `'2009-11-31'`, by enabling the `ALLOW_INVALID_DATES` SQL mode. This is useful when you want to store a “possibly wrong” value which the user has specified (for example, in a web form) in the database for future processing. Under this mode, MySQL verifies only that the month is in the range from 1 to 12 and that the day is in the range from 1 to 31.

- MySQL permits you to store dates where the day or month and day are zero in a `DATE` or `DATETIME` column. This is useful for applications that need to store birthdates for which you may not know the exact date. In this case, you simply store the date as `'2009-00-00'` or `'2009-01-00'`. However, with dates such as these, you should not expect to get correct results for functions such as `DATE_SUB()` or `DATE_ADD()` that require complete dates. To disallow zero month or day parts in dates, enable the `NO_ZERO_IN_DATE` SQL mode.

- MySQL permits you to store a “zero” value of `'0000-00-00'` as a “dummy date.” In some cases, this is more convenient than using `NULL` values, and uses less data and index space. To disallow `'0000-00-00'`, enable the `NO_ZERO_DATE` SQL mode.

- “Zero” date or time values used through Connector/ODBC are converted automatically to `NULL` because ODBC cannot handle such values.

The following table shows the format of the “zero” value for each type. The “zero” values are special, but you can store or refer to them explicitly using the values shown in the table. You can also do this using the values `'0'` or `0`, which are easier to write. For temporal types that include a date part (`DATE`, `DATETIME`, and `TIMESTAMP`), use of these values may produce warning or errors. The precise behavior depends on which, if any, of the strict and `NO_ZERO_DATE` SQL modes are enabled; see Section 5.1.10, “Server SQL Modes”.

Data Type	“Zero” Value
`DATE`	`'0000-00-00'`

Data Type	“Zero” Value
TIME	'00:00:00'
DATETIME	'0000-00-00 00:00:00'
TIMESTAMP	'0000-00-00 00:00:00'
YEAR	0000

11.2.1 Date and Time Data Type Syntax

The date and time data types for representing temporal values are DATE, TIME, DATETIME, TIMESTAMP, and YEAR.

For the DATE and DATETIME range descriptions, “supported” means that although earlier values might work, there is no guarantee.

MySQL permits fractional seconds for TIME, DATETIME, and TIMESTAMP values, with up to microseconds (6 digits) precision. To define a column that includes a fractional seconds part, use the syntax *type_name*(*fsp*), where *type_name* is TIME, DATETIME, or TIMESTAMP, and *fsp* is the fractional seconds precision. For example:

```
CREATE TABLE t1 (t TIME(3), dt DATETIME(6), ts TIMESTAMP(0));
```

The *fsp* value, if given, must be in the range 0 to 6. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0. (This differs from the standard SQL default of 6, for compatibility with previous MySQL versions.)

Any TIMESTAMP or DATETIME column in a table can have automatic initialization and updating properties; see Section 11.2.6, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”.

- DATE

  A date. The supported range is '1000-01-01' to '9999-12-31'. MySQL displays DATE values in '*YYYY-MM-DD*' format, but permits assignment of values to DATE columns using either strings or numbers.

- DATETIME[(*fsp*)]

  A date and time combination. The supported range is '1000-01-01 00:00:00.000000' to '9999-12-31 23:59:59.999999'. MySQL displays DATETIME values in '*YYYY-MM-DD hh:mm:ss*[.*fraction*]' format, but permits assignment of values to DATETIME columns using either strings or numbers.

  An optional *fsp* value in the range from 0 to 6 may be given to specify fractional seconds precision. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0.

  Automatic initialization and updating to the current date and time for DATETIME columns can be specified using DEFAULT and ON UPDATE column definition clauses, as described in Section 11.2.6, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”.

- TIMESTAMP[(*fsp*)]

  A timestamp. The range is '1970-01-01 00:00:01.000000' UTC to '2038-01-19 03:14:07.999999' UTC. TIMESTAMP values are stored as the number of seconds since the epoch ('1970-01-01 00:00:00' UTC). A TIMESTAMP cannot represent the value '1970-01-01 00:00:00' because that is equivalent to 0 seconds from the epoch and the value 0 is reserved for representing '0000-00-00 00:00:00', the “zero” TIMESTAMP value.

  An optional *fsp* value in the range from 0 to 6 may be given to specify fractional seconds precision. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0.

The way the server handles `TIMESTAMP` definitions depends on the value of the `explicit_defaults_for_timestamp` system variable (see Section 5.1.7, “Server System Variables”).

If `explicit_defaults_for_timestamp` is enabled, there is no automatic assignment of the `DEFAULT CURRENT_TIMESTAMP` or `ON UPDATE CURRENT_TIMESTAMP` attributes to any `TIMESTAMP` column. They must be included explicitly in the column definition. Also, any `TIMESTAMP` not explicitly declared as `NOT NULL` permits `NULL` values.

If `explicit_defaults_for_timestamp` is disabled, the server handles `TIMESTAMP` as follows:

Unless specified otherwise, the first `TIMESTAMP` column in a table is defined to be automatically set to the date and time of the most recent modification if not explicitly assigned a value. This makes `TIMESTAMP` useful for recording the timestamp of an `INSERT` or `UPDATE` operation. You can also set any `TIMESTAMP` column to the current date and time by assigning it a `NULL` value, unless it has been defined with the `NULL` attribute to permit `NULL` values.

Automatic initialization and updating to the current date and time can be specified using `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` column definition clauses. By default, the first `TIMESTAMP` column has these properties, as previously noted. However, any `TIMESTAMP` column in a table can be defined to have these properties.

`explicit_defaults_for_timestamp` is available as of MySQL 5.6.6. Before 5.6.6, the server handles `TIMESTAMP` as discussed for `explicit_defaults_for_timestamp` disabled. Those behaviors, while they remain the default, are nonstandard and are deprecated as of 5.6.6. For discussion regarding upgrading to an installation with `explicit_defaults_for_timestamp` enabled, see Section 2.11.3, “Changes in MySQL 5.6”.

- `TIME[(fsp)]`

  A time. The range is `'-838:59:59.000000'` to `'838:59:59.000000'`. MySQL displays `TIME` values in `'hh:mm:ss[.fraction]'` format, but permits assignment of values to `TIME` columns using either strings or numbers.

  An optional *`fsp`* value in the range from 0 to 6 may be given to specify fractional seconds precision. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0.

- `YEAR[(2|4)]`

  A year in 2-digit or 4-digit format. The default is 4-digit format. `YEAR(2)` or `YEAR(4)` differ in display format, but have the same range of values. In 4-digit format, values display as `1901` to `2155`, or `0000`. In 2-digit format, values display as `70` to `69`, representing years from 1970 to 2069. MySQL displays `YEAR` values in *`YYYY`* or *`YY`* format, but permits assignment of values to `YEAR` columns using either strings or numbers.

  Note

  The 2-digit `YEAR(2)` data type has certain issues that you should consider before choosing to use it. As of MySQL 5.6.6, `YEAR(2)` is deprecated: 2-digit `YEAR(2)` columns in existing tables are treated as before, but 2-digit `YEAR(2)` columns in new or altered tables are converted to 4-digit `YEAR` columns. For more information, see Section 11.2.5, “2-Digit YEAR(2) Limitations and Migrating to 4-Digit YEAR”.

  For additional information about `YEAR` display format and interpretation of input values, see Section 11.2.4, “The YEAR Type”.

The `SUM()` and `AVG()` aggregate functions do not work with temporal values. (They convert the values to numbers, losing everything after the first nonnumeric character.) To work around this problem,

convert to numeric units, perform the aggregate operation, and convert back to a temporal value. Examples:

```
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(time_col))) FROM tbl_name;
SELECT FROM_DAYS(SUM(TO_DAYS(date_col))) FROM tbl_name;
```

Note

The MySQL server can be run with the `MAXDB` SQL mode enabled. In this case, `TIMESTAMP` is identical with `DATETIME`. If this mode is enabled at the time that a table is created, `TIMESTAMP` columns are created as `DATETIME` columns. As a result, such columns use `DATETIME` display format, have the same range of values, and there is no automatic initialization or updating to the current date and time. See Section 5.1.10, “Server SQL Modes”.

11.2.2 The DATE, DATETIME, and TIMESTAMP Types

The `DATE`, `DATETIME`, and `TIMESTAMP` types are related. This section describes their characteristics, how they are similar, and how they differ. MySQL recognizes `DATE`, `DATETIME`, and `TIMESTAMP` values in several formats, described in Section 9.1.3, “Date and Time Literals”. For the `DATE` and `DATETIME` range descriptions, “supported” means that although earlier values might work, there is no guarantee.

The `DATE` type is used for values with a date part but no time part. MySQL retrieves and displays `DATE` values in `'YYYY-MM-DD'` format. The supported range is `'1000-01-01'` to `'9999-12-31'`.

The `DATETIME` type is used for values that contain both date and time parts. MySQL retrieves and displays `DATETIME` values in `'YYYY-MM-DD hh:mm:ss'` format. The supported range is `'1000-01-01 00:00:00'` to `'9999-12-31 23:59:59'`.

The `TIMESTAMP` data type is used for values that contain both date and time parts. `TIMESTAMP` has a range of `'1970-01-01 00:00:01'` UTC to `'2038-01-19 03:14:07'` UTC.

A `DATETIME` or `TIMESTAMP` value can include a trailing fractional seconds part in up to microseconds (6 digits) precision. In particular, as of MySQL 5.6.4, any fractional part in a value inserted into a `DATETIME` or `TIMESTAMP` column is stored rather than discarded. With the fractional part included, the format for these values is `'YYYY-MM-DD hh:mm:ss[.fraction]'`, the range for `DATETIME` values is `'1000-01-01 00:00:00.000000'` to `'9999-12-31 23:59:59.999999'`, and the range for `TIMESTAMP` values is `'1970-01-01 00:00:01.000000'` to `'2038-01-19 03:14:07.999999'`. The fractional part should always be separated from the rest of the time by a decimal point; no other fractional seconds delimiter is recognized. For information about fractional seconds support in MySQL, see Section 11.2.7, “Fractional Seconds in Time Values”.

The `TIMESTAMP` and `DATETIME` data types offer automatic initialization and updating to the current date and time. For more information, see Section 11.2.6, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”.

MySQL converts `TIMESTAMP` values from the current time zone to UTC for storage, and back from UTC to the current time zone for retrieval. (This does not occur for other types such as `DATETIME`.) By default, the current time zone for each connection is the server's time. The time zone can be set on a per-connection basis. As long as the time zone setting remains constant, you get back the same value you store. If you store a `TIMESTAMP` value, and then change the time zone and retrieve the value, the retrieved value is different from the value you stored. This occurs because the same time zone was not used for conversion in both directions. The current time zone is available as the value of the `time_zone` system variable. For more information, see Section 5.1.13, “MySQL Server Time Zone Support”.

Invalid `DATE`, `DATETIME`, or `TIMESTAMP` values are converted to the “zero” value of the appropriate type (`'0000-00-00'` or `'0000-00-00 00:00:00'`), if the SQL mode permits this conversion. The precise behavior depends on which if any of strict SQL mode and the `NO_ZERO_DATE` SQL mode are enabled; see Section 5.1.10, “Server SQL Modes”.

Be aware of certain properties of date value interpretation in MySQL:

- MySQL permits a “relaxed” format for values specified as strings, in which any punctuation character may be used as the delimiter between date parts or time parts. In some cases, this syntax can be deceiving. For example, a value such as `'10:11:12'` might look like a time value because of the `:`, but is interpreted as the year `'2010-11-12'` if used in date context. The value `'10:45:15'` is converted to `'0000-00-00'` because `'45'` is not a valid month.

  The only delimiter recognized between a date and time part and a fractional seconds part is the decimal point.

- The server requires that month and day values be valid, and not merely in the range 1 to 12 and 1 to 31, respectively. With strict mode disabled, invalid dates such as `'2004-04-31'` are converted to `'0000-00-00'` and a warning is generated. With strict mode enabled, invalid dates generate an error. To permit such dates, enable `ALLOW_INVALID_DATES`. See Section 5.1.10, “Server SQL Modes”, for more information.

- MySQL does not accept `TIMESTAMP` values that include a zero in the day or month column or values that are not a valid date. The sole exception to this rule is the special “zero” value `'0000-00-00 00:00:00'`, if the SQL mode permits this value. The precise behavior depends on which if any of strict SQL mode and the `NO_ZERO_DATE` SQL mode are enabled; see Section 5.1.10, “Server SQL Modes”.

- Dates containing 2-digit year values are ambiguous because the century is unknown. MySQL interprets 2-digit year values using these rules:

  - Year values in the range `00-69` become `2000-2069`.

  - Year values in the range `70-99` become `1970-1999`.

  See also Section 11.2.9, “2-Digit Years in Dates”.

Note

The MySQL server can be run with the `MAXDB` SQL mode enabled. In this case, `TIMESTAMP` is identical with `DATETIME`. If this mode is enabled at the time that a table is created, `TIMESTAMP` columns are created as `DATETIME` columns. As a result, such columns use `DATETIME` display format, have the same range of values, and there is no automatic initialization or updating to the current date and time. See Section 5.1.10, “Server SQL Modes”.

11.2.3 The TIME Type

MySQL retrieves and displays `TIME` values in *`'hh:mm:ss'`* format (or *`'hhh:mm:ss'`* format for large hours values). `TIME` values may range from `'-838:59:59'` to `'838:59:59'`. The hours part may be so large because the `TIME` type can be used not only to represent a time of day (which must be less than 24 hours), but also elapsed time or a time interval between two events (which may be much greater than 24 hours, or even negative).

MySQL recognizes `TIME` values in several formats, some of which can include a trailing fractional seconds part in up to microseconds (6 digits) precision. See Section 9.1.3, “Date and Time Literals”. For information about fractional seconds support in MySQL, see Section 11.2.7, “Fractional Seconds in Time Values”. In particular, as of MySQL 5.6.4, any fractional part in a value inserted into a `TIME` column is stored rather than discarded. With the fractional part included, the range for `TIME` values is `'-838:59:59.000000'` to `'838:59:59.000000'`.

Be careful about assigning abbreviated values to a `TIME` column. MySQL interprets abbreviated `TIME` values with colons as time of the day. That is, `'11:12'` means `'11:12:00'`, not `'00:11:12'`. MySQL interprets abbreviated values without colons using the assumption that the two rightmost digits

represent seconds (that is, as elapsed time rather than as time of day). For example, you might think of `'1112'` and `1112` as meaning `'11:12:00'` (12 minutes after 11 o'clock), but MySQL interprets them as `'00:11:12'` (11 minutes, 12 seconds). Similarly, `'12'` and `12` are interpreted as `'00:00:12'`.

The only delimiter recognized between a time part and a fractional seconds part is the decimal point.

By default, values that lie outside the `TIME` range but are otherwise valid are clipped to the closest endpoint of the range. For example, `'-850:00:00'` and `'850:00:00'` are converted to `'-838:59:59'` and `'838:59:59'`. Invalid `TIME` values are converted to `'00:00:00'`. Note that because `'00:00:00'` is itself a valid `TIME` value, there is no way to tell, from a value of `'00:00:00'` stored in a table, whether the original value was specified as `'00:00:00'` or whether it was invalid.

For more restrictive treatment of invalid `TIME` values, enable strict SQL mode to cause errors to occur. See Section 5.1.10, “Server SQL Modes”.

11.2.4 The YEAR Type

The `YEAR` type is a 1-byte type used to represent year values. It can be declared as `YEAR` with an implicit display width of 4 characters, or as `YEAR(4)` or `YEAR(2)` with an explicit display width of 4 or 2 characters.

Note

The 2-digit `YEAR(2)` data type has certain issues that you should consider before choosing to use it. As of MySQL 5.6.6, `YEAR(2)` is deprecated: 2-digit `YEAR(2)` columns in existing tables are treated as before, but 2-digit `YEAR(2)` columns in new or altered tables are converted to 4-digit `YEAR` columns. For more information, see Section 11.2.5, “2-Digit YEAR(2) Limitations and Migrating to 4-Digit YEAR”.

`YEAR`/`YEAR(4)` and `YEAR(2)` differ in display format, but have the same range of values. For 4-digit format, MySQL displays `YEAR` values in *`YYYY`* format, with a range of `1901` to *`2155`*, and `0000`. For 2-digit format, MySQL displays only the last (least significant) 2 digits; for example, `70` (1970 or 2070) or `69` (2069).

`YEAR` accepts input values in a variety of formats:

- As 4-digit strings in the range `'1901'` to `'2155'`.
- As 4-digit numbers in the range `1901` to `2155`.
- As 1- or 2-digit strings in the range `'0'` to `'99'`. MySQL converts values in the ranges `'0'` to `'69'` and `'70'` to `'99'` to `YEAR` values in the ranges `2000` to `2069` and `1970` to `1999`.
- As 1- or 2-digit numbers in the range `0` to `99`. MySQL converts values in the ranges `1` to `69` and `70` to `99` to `YEAR` values in the ranges `2001` to `2069` and `1970` to `1999`.

  Inserting a numeric `0` has different effects for `YEAR`/`YEAR(4)` and `YEAR(2)`:

  - For `YEAR`/`YEAR(4)`, the result has a display value of `0000` and an internal value of `0000`. To specify zero and have it be interpreted as `2000`, specify it as a string `'0'` or `'00'`.
  - For `YEAR(2)`, the result has a display value of `00` and an internal value of `2000`.
- As the result of functions that return a value that is acceptable in `YEAR` context, such as `NOW()`.

If strict SQL mode is not enabled, MySQL converts invalid `YEAR` values to `0000`. In strict SQL mode, attempting to insert an invalid `YEAR` value produces an error.

See also Section 11.2.9, “2-Digit Years in Dates”.

11.2.5 2-Digit YEAR(2) Limitations and Migrating to 4-Digit YEAR

This section describes problems that can occur when using the 2-digit `YEAR(2)` data type and provides information about converting existing `YEAR(2)` columns to 4-digit year-valued columns, which can be declared as `YEAR` with an implicit display width of 4 characters, or equivalently as `YEAR(4)` with an explicit display width.

Although the internal range of values for `YEAR`/`YEAR(4)` and the deprecated `YEAR(2)` type is the same (`1901` to `2155`, and `0000`), the display width for `YEAR(2)` makes that type inherently ambiguous because displayed values indicate only the last two digits of the internal values and omit the century digits. The result can be a loss of information under certain circumstances. For this reason, avoid using `YEAR(2)` in your applications and use `YEAR`/`YEAR(4)` wherever you need a year-valued data type. Note that conversion is required at some point because support for `YEAR` data types with display values other than 4, most notably `YEAR(2)`, is reduced as of MySQL 5.6.6 and is removed entirely in MySQL 5.7.

YEAR(2) Limitations

Issues with the `YEAR(2)` data type include ambiguity of displayed values, and possible loss of information when values are dumped and reloaded or converted to strings.

- Displayed `YEAR(2)` values can be ambiguous. It is possible for up to three `YEAR(2)` values that have different internal values to have the same displayed value, as the following example demonstrates:

```
mysql> CREATE TABLE t (y2 YEAR(2), y4 YEAR);
Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql> INSERT INTO t (y2) VALUES(1912),(2012),(2112);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> UPDATE t SET y4 = y2;
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> SELECT * FROM t;
+------+------+
| y2   | y4   |
+------+------+
|   12 | 1912 |
|   12 | 2012 |
|   12 | 2112 |
+------+------+
3 rows in set (0.00 sec)
```

- If you use `mysqldump` to dump the table created in the preceding example, the dump file represents all `y2` values using the same 2-digit representation (`12`). If you reload the table from the dump file, all resulting rows have internal value `2012` and display value `12`, thus losing the distinctions between them.

- Conversion of a 2-digit or 4-digit `YEAR` data value to string form uses the data type display width. Suppose that a `YEAR(2)` column and a `YEAR`/`YEAR(4)` column both contain the value `1970`. Assigning each column to a string results in a value of `'70'` or `'1970'`, respectively. That is, loss of information occurs for conversion from `YEAR(2)` to string.

- Values outside the range from `1970` to `2069` are stored incorrectly when inserted into a `YEAR(2)` column in a `CSV` table. For example, inserting `2211` results in a display value of `11` but an internal value of `2011`.

To avoid these problems, use the 4-digit `YEAR` or `YEAR(4)` data type rather than the 2-digit `YEAR(2)` data type. Suggestions regarding migration strategies appear later in this section.

Reduced YEAR(2) Support in MySQL 5.6

As of MySQL 5.6.6, support for `YEAR(2)` is diminished:

- `YEAR(2)` column definitions for new tables are converted (with an `ER_INVALID_YEAR_COLUMN_LENGTH` warning) to 4-digit `YEAR` columns:

```
mysql> CREATE TABLE t1 (y YEAR(2));
Query OK, 0 rows affected, 1 warning (0.04 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1818
Message: YEAR(2) column type is deprecated. Creating YEAR(4) column instead.
1 row in set (0.00 sec)

mysql> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `y` year(4) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

- `YEAR(2)` columns in existing tables remain as `YEAR(2)` and are processed in queries as in older versions of MySQL. However, several programs or statements convert `YEAR(2)` columns to 4-digit `YEAR` columns automatically:
  - `ALTER TABLE` statements that result in a table rebuild.
  - `REPAIR TABLE` (which `CHECK TABLE` recommends you use, if it finds a table that contains `YEAR(2)` columns).
  - `mysql_upgrade` (which uses `REPAIR TABLE`).
  - Dumping with `mysqldump` and reloading the dump file. Unlike the conversions performed by the preceding three items, a dump and reload has the potential to change data values.

  A MySQL upgrade usually involves at least one of the last two items. However, with respect to `YEAR(2)`, `mysql_upgrade` is preferable to `mysqldump`, which, as noted, can change data values.

Migrating from YEAR(2) to 4-Digit YEAR

To convert 2-digit `YEAR(2)` columns to 4-digit `YEAR` columns, you can do so manually at any time without upgrading. Alternatively, you can upgrade to a version of MySQL with reduced support for `YEAR(2)` (MySQL 5.6.6 or later), then have MySQL convert `YEAR(2)` columns automatically. In the latter case, avoid upgrading by dumping and reloading your data because that can change data values. In addition, if you use replication, there are upgrade considerations you must take into account.

To convert 2-digit `YEAR(2)` columns to 4-digit `YEAR` manually, use `ALTER TABLE` or `REPAIR TABLE`. Suppose that a table `t1` has this definition:

```
CREATE TABLE t1 (ycol YEAR(2) NOT NULL DEFAULT '70');
```

Modify the column using `ALTER TABLE` as follows:

```
ALTER TABLE t1 FORCE;
```

The `ALTER TABLE` statement converts the table without changing `YEAR(2)` values. If the server is a replication source, the `ALTER TABLE` statement replicates to replicas and makes the corresponding table change on each one.

Another migration method is to perform a binary upgrade: Upgrade MySQL 5.6.6 or later in place without dumping and reloading your data. Then run `mysql_upgrade`, which uses `REPAIR TABLE` to convert 2-digit `YEAR(2)` columns to 4-digit `YEAR` columns without changing data values. If the server is

a replication source, the `REPAIR TABLE` statements replicate to replicas and make the corresponding table changes on each one, unless you invoke `mysql_upgrade` with the `--skip-write-binlog` option.

Upgrades to replication servers usually involve upgrading replicas to a newer version of MySQL, then upgrading the source. For example, if a source and replica both run MySQL 5.5, a typical upgrade sequence involves upgrading the replica to 5.6, then upgrading the source to 5.6. With regard to the different treatment of `YEAR(2)` as of MySQL 5.6.6, that upgrade sequence results in a problem: Suppose that the replica has been upgraded but not yet the source. Then creating a table containing a 2-digit `YEAR(2)` column on the source results in a table containing a 4-digit `YEAR` column on the replica. Consequently, the following operations have a different result on the source and replica, if you use statement-based replication:

- Inserting numeric `0`. The resulting value has an internal value of `2000` on the source but `0000` on the replica.
- Converting `YEAR(2)` to string. This operation uses the display value of `YEAR(2)` on the source but `YEAR(4)` on the replica.

To avoid such problems, use one of these strategies:

- Use row-based replication instead of statement-based replication.
- Modify all 2-digit `YEAR(2)` columns on the source to 4-digit `YEAR` columns before upgrading. (Use `ALTER TABLE`, as described previously.) That makes it possible to upgrade normally (replica first, then source) without introducing any `YEAR(2)` to `YEAR(4)` differences between the source and replica.

One migration method should be avoided: Do not dump your data with `mysqldump` and reload the dump file after upgrading. That has the potential to change `YEAR(2)` values, as described previously.

A migration from 2-digit `YEAR(2)` columns to 4-digit `YEAR` columns should also involve examining application code for the possibility of changed behavior under conditions such as these:

- Code that expects selecting a `YEAR` column to produce exactly two digits.
- Code that does not account for different handling for inserts of numeric `0`: Inserting `0` into `YEAR(2)` or `YEAR(4)` results in an internal value of `2000` or `0000`, respectively.

11.2.6 Automatic Initialization and Updating for TIMESTAMP and DATETIME

As of MySQL 5.6.5, `TIMESTAMP` and `DATETIME` columns can be automatically initializated and updated to the current date and time (that is, the current timestamp). Before 5.6.5, this is true only for `TIMESTAMP`, and for at most one `TIMESTAMP` column per table. The following notes first describe automatic initialization and updating for MySQL 5.6.5 and up, then the differences for versions preceding 5.6.5.

For any `TIMESTAMP` or `DATETIME` column in a table, you can assign the current timestamp as the default value, the auto-update value, or both:

- An auto-initialized column is set to the current timestamp for inserted rows that specify no value for the column.
- An auto-updated column is automatically updated to the current timestamp when the value of any other column in the row is changed from its current value. An auto-updated column remains unchanged if all other columns are set to their current values. To prevent an auto-updated column from updating when other columns change, explicitly set it to its current value. To update an auto-updated column even when other columns do not change, explicitly set it to the value it should have (for example, set it to `CURRENT_TIMESTAMP`).

In addition, if the `explicit_defaults_for_timestamp` system variable is disabled, you can initialize or update any `TIMESTAMP` (but not `DATETIME`) column to the current date and time by assigning it a `NULL` value, unless it has been defined with the `NULL` attribute to permit `NULL` values.

To specify automatic properties, use the `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` clauses in column definitions. The order of the clauses does not matter. If both are present in a column definition, either can occur first. Any of the synonyms for `CURRENT_TIMESTAMP` have the same meaning as `CURRENT_TIMESTAMP`. These are `CURRENT_TIMESTAMP()`, `NOW()`, `LOCALTIME`, `LOCALTIME()`, `LOCALTIMESTAMP`, and `LOCALTIMESTAMP()`.

Use of `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` is specific to `TIMESTAMP` and `DATETIME`. The `DEFAULT` clause also can be used to specify a constant (nonautomatic) default value (for example, `DEFAULT 0` or `DEFAULT '2000-01-01 00:00:00'`).

Note

The following examples use `DEFAULT 0`, a default that can produce warnings or errors depending on whether strict SQL mode or the `NO_ZERO_DATE` SQL mode is enabled. Be aware that the `TRADITIONAL` SQL mode includes strict mode and `NO_ZERO_DATE`. See Section 5.1.10, “Server SQL Modes”.

`TIMESTAMP` or `DATETIME` column definitions can specify the current timestamp for both the default and auto-update values, for one but not the other, or for neither. Different columns can have different combinations of automatic properties. The following rules describe the possibilities:

- With both `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP`, the column has the current timestamp for its default value and is automatically updated to the current timestamp.

```
CREATE TABLE t1 (
  ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  dt DATETIME DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
);
```

- With a `DEFAULT` clause but no `ON UPDATE CURRENT_TIMESTAMP` clause, the column has the given default value and is not automatically updated to the current timestamp.

  The default depends on whether the `DEFAULT` clause specifies `CURRENT_TIMESTAMP` or a constant value. With `CURRENT_TIMESTAMP`, the default is the current timestamp.

```
CREATE TABLE t1 (
  ts TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
  dt DATETIME DEFAULT CURRENT_TIMESTAMP
);
```

  With a constant, the default is the given value. In this case, the column has no automatic properties at all.

```
CREATE TABLE t1 (
  ts TIMESTAMP DEFAULT 0,
  dt DATETIME DEFAULT 0
);
```

- With an `ON UPDATE CURRENT_TIMESTAMP` clause and a constant `DEFAULT` clause, the column is automatically updated to the current timestamp and has the given constant default value.

```
CREATE TABLE t1 (
  ts TIMESTAMP DEFAULT 0 ON UPDATE CURRENT_TIMESTAMP,
  dt DATETIME DEFAULT 0 ON UPDATE CURRENT_TIMESTAMP
);
```

- With an `ON UPDATE CURRENT_TIMESTAMP` clause but no `DEFAULT` clause, the column is automatically updated to the current timestamp but does not have the current timestamp for its default value.

  The default in this case is type dependent. `TIMESTAMP` has a default of 0 unless defined with the `NULL` attribute, in which case the default is `NULL`.

```
CREATE TABLE t1 (
```

```
  ts1 TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,     -- default 0
  ts2 TIMESTAMP NULL ON UPDATE CURRENT_TIMESTAMP -- default NULL
);
```

`DATETIME` has a default of `NULL` unless defined with the `NOT NULL` attribute, in which case the default is 0.

```
CREATE TABLE t1 (
  dt1 DATETIME ON UPDATE CURRENT_TIMESTAMP,         -- default NULL
  dt2 DATETIME NOT NULL ON UPDATE CURRENT_TIMESTAMP -- default 0
);
```

`TIMESTAMP` and `DATETIME` columns have no automatic properties unless they are specified explicitly, with this exception: If the `explicit_defaults_for_timestamp` system variable is disabled, the *first* `TIMESTAMP` column has both `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` if neither is specified explicitly. To suppress automatic properties for the first `TIMESTAMP` column, use one of these strategies:

- Enable the `explicit_defaults_for_timestamp` system variable. In this case, the `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` clauses that specify automatic initialization and updating are available, but are not assigned to any `TIMESTAMP` column unless explicitly included in the column definition.
- Alternatively, if `explicit_defaults_for_timestamp` is disabled, do either of the following:
  - Define the column with a `DEFAULT` clause that specifies a constant default value.
  - Specify the `NULL` attribute. This also causes the column to permit `NULL` values, which means that you cannot assign the current timestamp by setting the column to `NULL`. Assigning `NULL` sets the column to `NULL`, not the current timestamp. To assign the current timestamp, set the column to `CURRENT_TIMESTAMP` or a synonym such as `NOW()`.

Consider these table definitions:

```
CREATE TABLE t1 (
  ts1 TIMESTAMP DEFAULT 0,
  ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                ON UPDATE CURRENT_TIMESTAMP);
CREATE TABLE t2 (
  ts1 TIMESTAMP NULL,
  ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                ON UPDATE CURRENT_TIMESTAMP);
CREATE TABLE t3 (
  ts1 TIMESTAMP NULL DEFAULT 0,
  ts2 TIMESTAMP DEFAULT CURRENT_TIMESTAMP
                ON UPDATE CURRENT_TIMESTAMP);
```

The tables have these properties:

- In each table definition, the first `TIMESTAMP` column has no automatic initialization or updating.
- The tables differ in how the `ts1` column handles `NULL` values. For `t1`, `ts1` is `NOT NULL` and assigning it a value of `NULL` sets it to the current timestamp. For `t2` and `t3`, `ts1` permits `NULL` and assigning it a value of `NULL` sets it to `NULL`.
- `t2` and `t3` differ in the default value for `ts1`. For `t2`, `ts1` is defined to permit `NULL`, so the default is also `NULL` in the absence of an explicit `DEFAULT` clause. For `t3`, `ts1` permits `NULL` but has an explicit default of 0.

If a `TIMESTAMP` or `DATETIME` column definition includes an explicit fractional seconds precision value anywhere, the same value must be used throughout the column definition. This is permitted:

```
CREATE TABLE t1 (
  ts TIMESTAMP(6) DEFAULT CURRENT_TIMESTAMP(6) ON UPDATE CURRENT_TIMESTAMP(6)
);
```

This is not permitted:

```
CREATE TABLE t1 (
  ts TIMESTAMP(6) DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP(3)
);
```

Automatic Timestamp Properties Before MySQL 5.6.5

Before MySQL 5.6.5, support for automatic initialization and updating is more limited:

- `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` cannot be used with `DATETIME` columns.
- `DEFAULT CURRENT_TIMESTAMP` and `ON UPDATE CURRENT_TIMESTAMP` can be used with at most one `TIMESTAMP` column per table. It is not possible to have the current timestamp be the default value for one column and the auto-update value for another column.

You can choose whether to use these properties and which `TIMESTAMP` column should have them. It need not be the first one in a table that is automatically initialized or updated to the current timestamp. To specify automatic initialization or updating for a different `TIMESTAMP` column, you must suppress the automatic properties for the first one, as previously described. Then, for the other `TIMESTAMP` column, the rules for the `DEFAULT` and `ON UPDATE` clauses are the same as for the first `TIMESTAMP` column, except that if you omit both clauses, no automatic initialization or updating occurs.

TIMESTAMP Initialization and the NULL Attribute

If the `explicit_defaults_for_timestamp` system variable is disabled, `TIMESTAMP` columns by default are `NOT NULL`, cannot contain `NULL` values, and assigning `NULL` assigns the current timestamp. To permit a `TIMESTAMP` column to contain `NULL`, explicitly declare it with the `NULL` attribute. In this case, the default value also becomes `NULL` unless overridden with a `DEFAULT` clause that specifies a different default value. `DEFAULT NULL` can be used to explicitly specify `NULL` as the default value. (For a `TIMESTAMP` column not declared with the `NULL` attribute, `DEFAULT NULL` is invalid.) If a `TIMESTAMP` column permits `NULL` values, assigning `NULL` sets it to `NULL`, not to the current timestamp.

The following table contains several `TIMESTAMP` columns that permit `NULL` values:

```
CREATE TABLE t
(
  ts1 TIMESTAMP NULL DEFAULT NULL,
  ts2 TIMESTAMP NULL DEFAULT 0,
  ts3 TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP
);
```

A `TIMESTAMP` column that permits `NULL` values does *not* take on the current timestamp at insert time except under one of the following conditions:

- Its default value is defined as `CURRENT_TIMESTAMP` and no value is specified for the column
- `CURRENT_TIMESTAMP` or any of its synonyms such as `NOW()` is explicitly inserted into the column

In other words, a `TIMESTAMP` column defined to permit `NULL` values auto-initializes only if its definition includes `DEFAULT CURRENT_TIMESTAMP`:

```
CREATE TABLE t (ts TIMESTAMP NULL DEFAULT CURRENT_TIMESTAMP);
```

If the `TIMESTAMP` column permits `NULL` values but its definition does not include `DEFAULT CURRENT_TIMESTAMP`, you must explicitly insert a value corresponding to the current date and time. Suppose that tables `t1` and `t2` have these definitions:

```
CREATE TABLE t1 (ts TIMESTAMP NULL DEFAULT '0000-00-00 00:00:00');
CREATE TABLE t2 (ts TIMESTAMP NULL DEFAULT NULL);
```

To set the `TIMESTAMP` column in either table to the current timestamp at insert time, explicitly assign it that value. For example:

```
INSERT INTO t2 VALUES (CURRENT_TIMESTAMP);
INSERT INTO t1 VALUES (NOW());
```

If the `explicit_defaults_for_timestamp` system variable is enabled, `TIMESTAMP` columns permit `NULL` values only if declared with the `NULL` attribute. Also, `TIMESTAMP` columns do not permit assigning `NULL` to assign the current timestamp, whether declared with the `NULL` or `NOT NULL` attribute. To assign the current timestamp, set the column to `CURRENT_TIMESTAMP` or a synonym such as `NOW()`.

11.2.7 Fractional Seconds in Time Values

MySQL 5.6 has fractional seconds support for `TIME`, `DATETIME`, and `TIMESTAMP` values, with up to microseconds (6 digits) precision:

- To define a column that includes a fractional seconds part, use the syntax *`type_name(fsp)`*, where *`type_name`* is `TIME`, `DATETIME`, or `TIMESTAMP`, and *`fsp`* is the fractional seconds precision. For example:

  ```
  CREATE TABLE t1 (t TIME(3), dt DATETIME(6));
  ```

  The *`fsp`* value, if given, must be in the range 0 to 6. A value of 0 signifies that there is no fractional part. If omitted, the default precision is 0. (This differs from the standard SQL default of 6, for compatibility with previous MySQL versions.)

- Inserting a `TIME`, `DATE`, or `TIMESTAMP` value with a fractional seconds part into a column of the same type but having fewer fractional digits results in rounding. Consider a table created and populated as follows:

  ```
  CREATE TABLE fractest( c1 TIME(2), c2 DATETIME(2), c3 TIMESTAMP(2) );
  INSERT INTO fractest VALUES
  ('17:51:04.777', '2018-09-08 17:51:04.777', '2018-09-08 17:51:04.777');
  ```

  The temporal values are inserted into the table with rounding:

  ```
  mysql> SELECT * FROM fractest;
  +-------------+-------------------------+-------------------------+
  | c1          | c2                      | c3                      |
  +-------------+-------------------------+-------------------------+
  | 17:51:04.78 | 2018-09-08 17:51:04.78 | 2018-09-08 17:51:04.78 |
  +-------------+-------------------------+-------------------------+
  ```

  No warning or error is given when such rounding occurs. This behavior follows the SQL standard, and is not affected by the server `sql_mode` setting.

- Functions that take temporal arguments accept values with fractional seconds. Return values from temporal functions include fractional seconds as appropriate. For example, `NOW()` with no argument returns the current date and time with no fractional part, but takes an optional argument from 0 to 6 to specify that the return value includes a fractional seconds part of that many digits.

- Syntax for temporal literals produces temporal values: `DATE '`*`str`*`'`, `TIME '`*`str`*`'`, and `TIMESTAMP '`*`str`*`'`, and the ODBC-syntax equivalents. The resulting value includes a trailing fractional seconds part if specified. Previously, the temporal type keyword was ignored and these constructs produced the string value. See Standard SQL and ODBC Date and Time Literals

In some cases, previously accepted syntax may produce different results. The following items indicate where existing code may need to be changed to avoid problems:

- Some expressions produce results that differ from previous results. Examples: The `timestamp` system variable returns a value that includes a microseconds fractional part rather than an integer. Functions that return a result that includes the current time (such as `CURTIME()`, `SYSDATE()`, or `UTC_TIMESTAMP()`) interpret an argument as an *`fsp`* value and the return value includes a fractional seconds part of that many digits. Previously, these functions permitted an argument but ignored it.

- `TIME` values are converted to `DATETIME` by adding the time to the current date. (This means that the date part of the result differs from the current date if the time value is outside the range from `'00:00:00'` to `'23:59:59'`.) Previously, conversion of `TIME` values to `DATETIME` was unreliable. See Section 11.2.8, “Conversion Between Date and Time Types”.
- `TIMESTAMP(N)` was permitted in old MySQL versions, but *`N`* was a display width rather than fractional seconds precision. Support for this behavior was removed in MySQL 5.5.3, so applications that are reasonably up to date should not be subject to this issue. Otherwise, code must be rewritten.

11.2.8 Conversion Between Date and Time Types

To some extent, you can convert a value from one temporal type to another. However, there may be some alteration of the value or loss of information. In all cases, conversion between temporal types is subject to the range of valid values for the resulting type. For example, although `DATE`, `DATETIME`, and `TIMESTAMP` values all can be specified using the same set of formats, the types do not all have the same range of values. `TIMESTAMP` values cannot be earlier than `1970` UTC or later than `'2038-01-19 03:14:07'` UTC. This means that a date such as `'1968-01-01'`, while valid as a `DATE` or `DATETIME` value, is not valid as a `TIMESTAMP` value and is converted to `0`.

Conversion of `DATE` values:

- Conversion to a `DATETIME` or `TIMESTAMP` value adds a time part of `'00:00:00'` because the `DATE` value contains no time information.
- Conversion to a `TIME` value is not useful; the result is `'00:00:00'`.

Conversion of `DATETIME` and `TIMESTAMP` values:

- Conversion to a `DATE` value takes fractional seconds into account and rounds the time part. For example, `'1999-12-31 23:59:59.499'` becomes `'1999-12-31'`, whereas `'1999-12-31 23:59:59.500'` becomes `'2000-01-01'`.
- Conversion to a `TIME` value discards the date part because the `TIME` type contains no date information.

For conversion of `TIME` values to other temporal types, the value of `CURRENT_DATE()` is used for the date part. The `TIME` is interpreted as elapsed time (not time of day) and added to the date. This means that the date part of the result differs from the current date if the time value is outside the range from `'00:00:00'` to `'23:59:59'`.

Suppose that the current date is `'2012-01-01'`. `TIME` values of `'12:00:00'`, `'24:00:00'`, and `'-12:00:00'`, when converted to `DATETIME` or `TIMESTAMP` values, result in `'2012-01-01 12:00:00'`, `'2012-01-02 00:00:00'`, and `'2011-12-31 12:00:00'`, respectively.

Conversion of `TIME` to `DATE` is similar but discards the time part from the result: `'2012-01-01'`, `'2012-01-02'`, and `'2011-12-31'`, respectively.

Explicit conversion can be used to override implicit conversion. For example, in comparison of `DATE` and `DATETIME` values, the `DATE` value is coerced to the `DATETIME` type by adding a time part of `'00:00:00'`. To perform the comparison by ignoring the time part of the `DATETIME` value instead, use the `CAST()` function in the following way:

```
date_col = CAST(datetime_col AS DATE)
```

Conversion of `TIME` and `DATETIME` values to numeric form (for example, by adding `+0`) depends on whether the value contains a fractional seconds part. `TIME(N)` or `DATETIME(N)` is converted to integer when *`N`* is 0 (or omitted) and to a `DECIMAL` value with *`N`* decimal digits when *`N`* is greater than 0:

```
mysql> SELECT CURTIME(), CURTIME()+0, CURTIME(3)+0;
+-----------+-------------+--------------+
| CURTIME() | CURTIME()+0 | CURTIME(3)+0 |
+-----------+-------------+--------------+
| 09:28:00  |       92800 |    92800.887 |
```

```
+-----------+------------+-------------+
mysql> SELECT NOW(), NOW()+0, NOW(3)+0;
+---------------------+----------------+--------------------+
| NOW()               | NOW()+0        | NOW(3)+0           |
+---------------------+----------------+--------------------+
| 2012-08-15 09:28:00 | 20120815092800 | 20120815092800.889 |
+---------------------+----------------+--------------------+
```

11.2.9 2-Digit Years in Dates

Date values with 2-digit years are ambiguous because the century is unknown. Such values must be interpreted into 4-digit form because MySQL stores years internally using 4 digits.

For `DATETIME`, `DATE`, and `TIMESTAMP` types, MySQL interprets dates specified with ambiguous year values using these rules:

- Year values in the range `00-69` become `2000-2069`.
- Year values in the range `70-99` become `1970-1999`.

For `YEAR`, the rules are the same, with this exception: A numeric `00` inserted into `YEAR(4)` results in `0000` rather than `2000`. To specify zero for `YEAR(4)` and have it be interpreted as `2000`, specify it as a string `'0'` or `'00'`.

Remember that these rules are only heuristics that provide reasonable guesses as to what your data values mean. If the rules used by MySQL do not produce the values you require, you must provide unambiguous input containing 4-digit year values.

`ORDER BY` properly sorts `YEAR` values that have 2-digit years.

Some functions like `MIN()` and `MAX()` convert a `YEAR` to a number. This means that a value with a 2-digit year does not work properly with these functions. The fix in this case is to convert the `YEAR` to 4-digit year format.

11.3 String Data Types

The string data types are `CHAR`, `VARCHAR`, `BINARY`, `VARBINARY`, `BLOB`, `TEXT`, `ENUM`, and `SET`.

For information about storage requirements of the string data types, see Section 11.6, “Data Type Storage Requirements”.

For descriptions of functions that operate on string values, see Section 12.8, “String Functions and Operators”.

11.3.1 String Data Type Syntax

The string data types are `CHAR`, `VARCHAR`, `BINARY`, `VARBINARY`, `BLOB`, `TEXT`, `ENUM`, and `SET`.

In some cases, MySQL may change a string column to a type different from that given in a `CREATE TABLE` or `ALTER TABLE` statement. See Section 13.1.17.6, “Silent Column Specification Changes”.

For definitions of character string columns (`CHAR`, `VARCHAR`, and the `TEXT` types), MySQL interprets length specifications in character units. For definitions of binary string columns (`BINARY`, `VARBINARY`, and the `BLOB` types), MySQL interprets length specifications in byte units.

Column definitions for character string data types `CHAR`, `VARCHAR`, the `TEXT` types, `ENUM`, `SET`, and any synonyms) can specify the column character set and collation:

- `CHARACTER SET` specifies the character set. If desired, a collation for the character set can be specified with the `COLLATE` attribute, along with any other attributes. For example:

```
CREATE TABLE t
(
    c1 VARCHAR(20) CHARACTER SET utf8,
```

```
    c2 TEXT CHARACTER SET latin1 COLLATE latin1_general_cs
);
```

This table definition creates a column named `c1` that has a character set of `utf8` with the default collation for that character set, and a column named `c2` that has a character set of `latin1` and a case-sensitive (`_cs`) collation.

The rules for assigning the character set and collation when either or both of `CHARACTER SET` and the `COLLATE` attribute are missing are described in Section 10.3.5, “Column Character Set and Collation”.

`CHARSET` is a synonym for `CHARACTER SET`.

- Specifying the `CHARACTER SET binary` attribute for a character string data type causes the column to be created as the corresponding binary string data type: `CHAR` becomes `BINARY`, `VARCHAR` becomes `VARBINARY`, and `TEXT` becomes `BLOB`. For the `ENUM` and `SET` data types, this does not occur; they are created as declared. Suppose that you specify a table using this definition:

```
CREATE TABLE t
(
  c1 VARCHAR(10) CHARACTER SET binary,
  c2 TEXT CHARACTER SET binary,
  c3 ENUM('a','b','c') CHARACTER SET binary
);
```

The resulting table has this definition:

```
CREATE TABLE t
(
  c1 VARBINARY(10),
  c2 BLOB,
  c3 ENUM('a','b','c') CHARACTER SET binary
);
```

- The `BINARY` attribute is a nonstandard MySQL extension that is shorthand for specifying the binary (`_bin`) collation of the column character set (or of the table default character set if no column character set is specified). In this case, comparison and sorting are based on numeric character code values. Suppose that you specify a table using this definition:

```
CREATE TABLE t
(
  c1 VARCHAR(10) CHARACTER SET latin1 BINARY,
  c2 TEXT BINARY
) CHARACTER SET utf8mb4;
```

The resulting table has this definition:

```
CREATE TABLE t (
  c1 VARCHAR(10) CHARACTER SET latin1 COLLATE latin1_bin,
  c2 TEXT CHARACTER SET utf8mb4 COLLATE utf8mb4_bin
) CHARACTER SET utf8mb4;
```

- The `ASCII` attribute is shorthand for `CHARACTER SET latin1`.
- The `UNICODE` attribute is shorthand for `CHARACTER SET ucs2`.

Character column comparison and sorting are based on the collation assigned to the column. For the `CHAR`, `VARCHAR`, `TEXT`, `ENUM`, and `SET` data types, you can declare a column with a binary (`_bin`) collation or the `BINARY` attribute to cause comparison and sorting to use the underlying character code values rather than a lexical ordering.

For additional information about use of character sets in MySQL, see Chapter 10, *Character Sets, Collations, Unicode*.

- `[NATIONAL] CHAR[(M)] [CHARACTER SET charset_name] [COLLATE collation_name]`

A fixed-length string that is always right-padded with spaces to the specified length when stored. *M* represents the column length in characters. The range of *M* is 0 to 255. If *M* is omitted, the length is 1.

Note

Trailing spaces are removed when CHAR values are retrieved unless the PAD_CHAR_TO_FULL_LENGTH SQL mode is enabled.

CHAR is shorthand for CHARACTER. NATIONAL CHAR (or its equivalent short form, NCHAR) is the standard SQL way to define that a CHAR column should use some predefined character set. MySQL uses utf8 as this predefined character set. Section 10.3.7, “The National Character Set”.

The CHAR BYTE data type is an alias for the BINARY data type. This is a compatibility feature.

MySQL permits you to create a column of type CHAR(0). This is useful primarily when you must be compliant with old applications that depend on the existence of a column but that do not actually use its value. CHAR(0) is also quite nice when you need a column that can take only two values: A column that is defined as CHAR(0) NULL occupies only one bit and can take only the values NULL and '' (the empty string).

- [NATIONAL] VARCHAR(*M*) [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

  A variable-length string. *M* represents the maximum column length in characters. The range of *M* is 0 to 65,535. The effective maximum length of a VARCHAR is subject to the maximum row size (65,535 bytes, which is shared among all columns) and the character set used. For example, utf8 characters can require up to three bytes per character, so a VARCHAR column that uses the utf8 character set can be declared to be a maximum of 21,844 characters. See Section 8.4.7, “Limits on Table Column Count and Row Size”.

  MySQL stores VARCHAR values as a 1-byte or 2-byte length prefix plus data. The length prefix indicates the number of bytes in the value. A VARCHAR column uses one length byte if values require no more than 255 bytes, two length bytes if values may require more than 255 bytes.

  Note

  MySQL follows the standard SQL specification, and does *not* remove trailing spaces from VARCHAR values.

  VARCHAR is shorthand for CHARACTER VARYING. NATIONAL VARCHAR is the standard SQL way to define that a VARCHAR column should use some predefined character set. MySQL uses utf8 as this predefined character set. Section 10.3.7, “The National Character Set”. NVARCHAR is shorthand for NATIONAL VARCHAR.

- BINARY[(*M*)]

  The BINARY type is similar to the CHAR type, but stores binary byte strings rather than nonbinary character strings. An optional length *M* represents the column length in bytes. If omitted, *M* defaults to 1.

- VARBINARY(*M*)

  The VARBINARY type is similar to the VARCHAR type, but stores binary byte strings rather than nonbinary character strings. *M* represents the maximum column length in bytes.

- TINYBLOB

  A BLOB column with a maximum length of 255 (2^8 − 1) bytes. Each TINYBLOB value is stored using a 1-byte length prefix that indicates the number of bytes in the value.

- TINYTEXT [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

A TEXT column with a maximum length of 255 (2^8 − 1) characters. The effective maximum length is less if the value contains multibyte characters. Each TINYTEXT value is stored using a 1-byte length prefix that indicates the number of bytes in the value.

- BLOB[(*M*)]

  A BLOB column with a maximum length of 65,535 (2^{16} − 1) bytes. Each BLOB value is stored using a 2-byte length prefix that indicates the number of bytes in the value.

  An optional length *M* can be given for this type. If this is done, MySQL creates the column as the smallest BLOB type large enough to hold values *M* bytes long.

- TEXT[(*M*)] [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

  A TEXT column with a maximum length of 65,535 (2^{16} − 1) characters. The effective maximum length is less if the value contains multibyte characters. Each TEXT value is stored using a 2-byte length prefix that indicates the number of bytes in the value.

  An optional length *M* can be given for this type. If this is done, MySQL creates the column as the smallest TEXT type large enough to hold values *M* characters long.

- MEDIUMBLOB

  A BLOB column with a maximum length of 16,777,215 (2^{24} − 1) bytes. Each MEDIUMBLOB value is stored using a 3-byte length prefix that indicates the number of bytes in the value.

- MEDIUMTEXT [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

  A TEXT column with a maximum length of 16,777,215 (2^{24} − 1) characters. The effective maximum length is less if the value contains multibyte characters. Each MEDIUMTEXT value is stored using a 3-byte length prefix that indicates the number of bytes in the value.

- LONGBLOB

  A BLOB column with a maximum length of 4,294,967,295 or 4GB (2^{32} − 1) bytes. The effective maximum length of LONGBLOB columns depends on the configured maximum packet size in the client/server protocol and available memory. Each LONGBLOB value is stored using a 4-byte length prefix that indicates the number of bytes in the value.

- LONGTEXT [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

  A TEXT column with a maximum length of 4,294,967,295 or 4GB (2^{32} − 1) characters. The effective maximum length is less if the value contains multibyte characters. The effective maximum length of LONGTEXT columns also depends on the configured maximum packet size in the client/server protocol and available memory. Each LONGTEXT value is stored using a 4-byte length prefix that indicates the number of bytes in the value.

- ENUM('*value1*','*value2*',...) [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

  An enumeration. A string object that can have only one value, chosen from the list of values '*value1*', '*value2*', ..., NULL or the special '' error value. ENUM values are represented internally as integers.

  An ENUM column can have a maximum of 65,535 distinct elements. (The practical limit is less than 3000.) A table can have no more than 255 unique element list definitions among its ENUM and SET columns considered as a group. For more information on these limits, see Limits Imposed by .frm File Structure.

- SET('*value1*','*value2*',...) [CHARACTER SET *charset_name*] [COLLATE *collation_name*]

A set. A string object that can have zero or more values, each of which must be chosen from the list of values `'value1'`, `'value2'`, `...` `SET` values are represented internally as integers.

A `SET` column can have a maximum of 64 distinct members. A table can have no more than 255 unique element list definitions among its `ENUM` and `SET` columns considered as a group. For more information on this limit, see Limits Imposed by .frm File Structure.

11.3.2 The CHAR and VARCHAR Types

The `CHAR` and `VARCHAR` types are similar, but differ in the way they are stored and retrieved. They also differ in maximum length and in whether trailing spaces are retained.

The `CHAR` and `VARCHAR` types are declared with a length that indicates the maximum number of characters you want to store. For example, `CHAR(30)` can hold up to 30 characters.

The length of a `CHAR` column is fixed to the length that you declare when you create the table. The length can be any value from 0 to 255. When `CHAR` values are stored, they are right-padded with spaces to the specified length. When `CHAR` values are retrieved, trailing spaces are removed unless the `PAD_CHAR_TO_FULL_LENGTH` SQL mode is enabled.

Values in `VARCHAR` columns are variable-length strings. The length can be specified as a value from 0 to 65,535. The effective maximum length of a `VARCHAR` is subject to the maximum row size (65,535 bytes, which is shared among all columns) and the character set used. See Section 8.4.7, “Limits on Table Column Count and Row Size”.

In contrast to `CHAR`, `VARCHAR` values are stored as a 1-byte or 2-byte length prefix plus data. The length prefix indicates the number of bytes in the value. A column uses one length byte if values require no more than 255 bytes, two length bytes if values may require more than 255 bytes.

If strict SQL mode is not enabled and you assign a value to a `CHAR` or `VARCHAR` column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For truncation of nonspace characters, you can cause an error to occur (rather than a warning) and suppress insertion of the value by using strict SQL mode. See Section 5.1.10, “Server SQL Modes”.

For `VARCHAR` columns, trailing spaces in excess of the column length are truncated prior to insertion and a warning is generated, regardless of the SQL mode in use. For `CHAR` columns, truncation of excess trailing spaces from inserted values is performed silently regardless of the SQL mode.

`VARCHAR` values are not padded when they are stored. Trailing spaces are retained when values are stored and retrieved, in conformance with standard SQL.

The following table illustrates the differences between `CHAR` and `VARCHAR` by showing the result of storing various string values into `CHAR(4)` and `VARCHAR(4)` columns (assuming that the column uses a single-byte character set such as `latin1`).

Value	`CHAR(4)`	Storage Required	`VARCHAR(4)`	Storage Required
`''`	`'    '`	4 bytes	`''`	1 byte
`'ab'`	`'ab  '`	4 bytes	`'ab'`	3 bytes
`'abcd'`	`'abcd'`	4 bytes	`'abcd'`	5 bytes
`'abcdefgh'`	`'abcd'`	4 bytes	`'abcd'`	5 bytes

The values shown as stored in the last row of the table apply *only when not using strict SQL mode*; if strict mode is enabled, values that exceed the column length are *not stored*, and an error results.

`InnoDB` encodes fixed-length fields greater than or equal to 768 bytes in length as variable-length fields, which can be stored off-page. For example, a `CHAR(255)` column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with `utf8mb4`.

If a given value is stored into the `CHAR(4)` and `VARCHAR(4)` columns, the values retrieved from the columns are not always the same because trailing spaces are removed from `CHAR` columns upon retrieval. The following example illustrates this difference:

```
mysql> CREATE TABLE vc (v VARCHAR(4), c CHAR(4));
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO vc VALUES ('ab  ', 'ab  ');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT CONCAT('(', v, ')'), CONCAT('(', c, ')') FROM vc;
+--------------------+--------------------+
| CONCAT('(', v, ')') | CONCAT('(', c, ')') |
+--------------------+--------------------+
| (ab  )             | (ab)               |
+--------------------+--------------------+
1 row in set (0.06 sec)
```

Values in `CHAR`, `VARCHAR`, and `TEXT` columns are sorted and compared according to the character set collation assigned to the column.

All MySQL collations are of type `PAD SPACE`. This means that all `CHAR`, `VARCHAR`, and `TEXT` values are compared without regard to any trailing spaces. “Comparison” in this context does not include the `LIKE` pattern-matching operator, for which trailing spaces are significant. For example:

```
mysql> CREATE TABLE names (myname CHAR(10));
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO names VALUES ('Jones');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT myname = 'Jones', myname = 'Jones  ' FROM names;
+-----------------+-------------------+
| myname = 'Jones' | myname = 'Jones  ' |
+-----------------+-------------------+
|               1 |                 1 |
+-----------------+-------------------+
1 row in set (0.00 sec)

mysql> SELECT myname LIKE 'Jones', myname LIKE 'Jones  ' FROM names;
+--------------------+----------------------+
| myname LIKE 'Jones' | myname LIKE 'Jones  ' |
+--------------------+----------------------+
|                  1 |                    0 |
+--------------------+----------------------+
1 row in set (0.00 sec)
```

This is not affected by the server SQL mode.

Note

For more information about MySQL character sets and collations, see Chapter 10, *Character Sets, Collations, Unicode*. For additional information about storage requirements, see Section 11.6, “Data Type Storage Requirements”.

For those cases where trailing pad characters are stripped or comparisons ignore them, if a column has an index that requires unique values, inserting into the column values that differ only in number of trailing pad characters results in a duplicate-key error. For example, if a table contains `'a'`, an attempt to store `'a '` causes a duplicate-key error.

11.3.3 The BINARY and VARBINARY Types

The `BINARY` and `VARBINARY` types are similar to `CHAR` and `VARCHAR`, except that they store binary strings rather than nonbinary strings. That is, they store byte strings rather than character strings. This means they have the `binary` character set and collation, and comparison and sorting are based on the numeric values of the bytes in the values.

The permissible maximum length is the same for BINARY and VARBINARY as it is for CHAR and VARCHAR, except that the length for BINARY and VARBINARY is measured in bytes rather than characters.

The BINARY and VARBINARY data types are distinct from the CHAR BINARY and VARCHAR BINARY data types. For the latter types, the BINARY attribute does not cause the column to be treated as a binary string column. Instead, it causes the binary (_bin) collation for the column character set (or the table default character set if no column character set is specified) to be used, and the column itself stores nonbinary character strings rather than binary byte strings. For example, if the default character set is latin1, CHAR(5) BINARY is treated as CHAR(5) CHARACTER SET latin1 COLLATE latin1_bin. This differs from BINARY(5), which stores 5-byte binary strings that have the binary character set and collation. For information about the differences between the binary collation of the binary character set and the _bin collations of nonbinary character sets, see Section 10.8.5, “The binary Collation Compared to _bin Collations”.

If strict SQL mode is not enabled and you assign a value to a BINARY or VARBINARY column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For cases of truncation, to cause an error to occur (rather than a warning) and suppress insertion of the value, use strict SQL mode. See Section 5.1.10, “Server SQL Modes”.

When BINARY values are stored, they are right-padded with the pad value to the specified length. The pad value is 0x00 (the zero byte). Values are right-padded with 0x00 for inserts, and no trailing bytes are removed for retrievals. All bytes are significant in comparisons, including ORDER BY and DISTINCT operations. 0x00 and space differ in comparisons, with 0x00 sorting before space.

Example: For a BINARY(3) column, 'a ' becomes 'a \0' when inserted. 'a\0' becomes 'a \0\0' when inserted. Both inserted values remain unchanged for retrievals.

For VARBINARY, there is no padding for inserts and no bytes are stripped for retrievals. All bytes are significant in comparisons, including ORDER BY and DISTINCT operations. 0x00 and space differ in comparisons, with 0x00 sorting before space.

For those cases where trailing pad bytes are stripped or comparisons ignore them, if a column has an index that requires unique values, inserting values into the column that differ only in number of trailing pad bytes results in a duplicate-key error. For example, if a table contains 'a', an attempt to store 'a \0' causes a duplicate-key error.

You should consider the preceding padding and stripping characteristics carefully if you plan to use the BINARY data type for storing binary data and you require that the value retrieved be exactly the same as the value stored. The following example illustrates how 0x00-padding of BINARY values affects column value comparisons:

```
mysql> CREATE TABLE t (c BINARY(3));
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t SET c = 'a';
Query OK, 1 row affected (0.01 sec)

mysql> SELECT HEX(c), c = 'a', c = 'a\0\0' from t;
+--------+---------+-------------+
| HEX(c) | c = 'a' | c = 'a\0\0' |
+--------+---------+-------------+
| 610000 |       0 |           1 |
+--------+---------+-------------+
1 row in set (0.09 sec)
```

If the value retrieved must be the same as the value specified for storage with no padding, it might be preferable to use VARBINARY or one of the BLOB data types instead.

11.3.4 The BLOB and TEXT Types

A BLOB is a binary large object that can hold a variable amount of data. The four BLOB types are TINYBLOB, BLOB, MEDIUMBLOB, and LONGBLOB. These differ only in the maximum length of the

values they can hold. The four TEXT types are TINYTEXT, TEXT, MEDIUMTEXT, and LONGTEXT. These correspond to the four BLOB types and have the same maximum lengths and storage requirements. See Section 11.6, “Data Type Storage Requirements”.

BLOB values are treated as binary strings (byte strings). They have the binary character set and collation, and comparison and sorting are based on the numeric values of the bytes in column values. TEXT values are treated as nonbinary strings (character strings). They have a character set other than binary, and values are sorted and compared based on the collation of the character set.

If strict SQL mode is not enabled and you assign a value to a BLOB or TEXT column that exceeds the column's maximum length, the value is truncated to fit and a warning is generated. For truncation of nonspace characters, you can cause an error to occur (rather than a warning) and suppress insertion of the value by using strict SQL mode. See Section 5.1.10, “Server SQL Modes”.

Truncation of excess trailing spaces from values to be inserted into TEXT columns always generates a warning, regardless of the SQL mode.

For TEXT and BLOB columns, there is no padding on insert and no bytes are stripped on select.

If a TEXT column is indexed, index entry comparisons are space-padded at the end. This means that, if the index requires unique values, duplicate-key errors occur for values that differ only in the number of trailing spaces. For example, if a table contains 'a', an attempt to store 'a ' causes a duplicate-key error. This is not true for BLOB columns.

In most respects, you can regard a BLOB column as a VARBINARY column that can be as large as you like. Similarly, you can regard a TEXT column as a VARCHAR column. BLOB and TEXT differ from VARBINARY and VARCHAR in the following ways:

- For indexes on BLOB and TEXT columns, you must specify an index prefix length. For CHAR and VARCHAR, a prefix length is optional. See Section 8.3.4, “Column Indexes”.
- BLOB and TEXT columns cannot have DEFAULT values.

If you use the BINARY attribute with a TEXT data type, the column is assigned the binary (_bin) collation of the column character set.

LONG and LONG VARCHAR map to the MEDIUMTEXT data type. This is a compatibility feature.

MySQL Connector/ODBC defines BLOB values as LONGVARBINARY and TEXT values as LONGVARCHAR.

Because BLOB and TEXT values can be extremely long, you might encounter some constraints in using them:

- Only the first max_sort_length bytes of the column are used when sorting. The default value of max_sort_length is 1024. You can make more bytes significant in sorting or grouping by increasing the value of max_sort_length at server startup or runtime. Any client can change the value of its session max_sort_length variable:

```
mysql> SET max_sort_length = 2000;
mysql> SELECT id, comment FROM t
    -> ORDER BY comment;
```

- Instances of BLOB or TEXT columns in the result of a query that is processed using a temporary table causes the server to use a table on disk rather than in memory because the MEMORY storage engine does not support those data types (see Section 8.4.4, “Internal Temporary Table Use in MySQL”). Use of disk incurs a performance penalty, so include BLOB or TEXT columns in the query result only if they are really needed. For example, avoid using SELECT *, which selects all columns.
- The maximum size of a BLOB or TEXT object is determined by its type, but the largest value you actually can transmit between the client and server is determined by the amount of available memory

and the size of the communications buffers. You can change the message buffer size by changing the value of the `max_allowed_packet` variable, but you must do so for both the server and your client program. For example, both `mysql` and `mysqldump` enable you to change the client-side `max_allowed_packet` value. See Section 5.1.1, "Configuring the Server", Section 4.5.1, "`mysql` — The MySQL Command-Line Client", and Section 4.5.4, "`mysqldump` — A Database Backup Program". You may also want to compare the packet sizes and the size of the data objects you are storing with the storage requirements, see Section 11.6, "Data Type Storage Requirements"

Each `BLOB` or `TEXT` value is represented internally by a separately allocated object. This is in contrast to all other data types, for which storage is allocated once per column when the table is opened.

In some cases, it may be desirable to store binary data such as media files in `BLOB` or `TEXT` columns. You may find MySQL's string handling functions useful for working with such data. See Section 12.8, "String Functions and Operators". For security and other reasons, it is usually preferable to do so using application code rather than giving application users the `FILE` privilege. You can discuss specifics for various languages and platforms in the MySQL Forums (http://forums.mysql.com/).

11.3.5 The ENUM Type

An `ENUM` is a string object with a value chosen from a list of permitted values that are enumerated explicitly in the column specification at table creation time.

See Section 11.3.1, "String Data Type Syntax" for `ENUM` type syntax and length limits.

The `ENUM` type has these advantages:

- Compact data storage in situations where a column has a limited set of possible values. The strings you specify as input values are automatically encoded as numbers. See Section 11.6, "Data Type Storage Requirements" for storage requirements for the `ENUM` type.
- Readable queries and output. The numbers are translated back to the corresponding strings in query results.

and these potential issues to consider:

- If you make enumeration values that look like numbers, it is easy to mix up the literal values with their internal index numbers, as explained in Enumeration Limitations.
- Using `ENUM` columns in `ORDER BY` clauses requires extra care, as explained in Enumeration Sorting.
- Creating and Using ENUM Columns
- Index Values for Enumeration Literals
- Handling of Enumeration Literals
- Empty or NULL Enumeration Values
- Enumeration Sorting
- Enumeration Limitations

Creating and Using ENUM Columns

An enumeration value must be a quoted string literal. For example, you can create a table with an `ENUM` column like this:

```
CREATE TABLE shirts (
    name VARCHAR(40),
    size ENUM('x-small', 'small', 'medium', 'large', 'x-large')
);
```

```
INSERT INTO shirts (name, size) VALUES ('dress shirt','large'), ('t-shirt','medium'),
  ('polo shirt','small');
SELECT name, size FROM shirts WHERE size = 'medium';
+---------+--------+
| name    | size   |
+---------+--------+
| t-shirt | medium |
+---------+--------+
UPDATE shirts SET size = 'small' WHERE size = 'large';
COMMIT;
```

Inserting 1 million rows into this table with a value of `'medium'` would require 1 million bytes of storage, as opposed to 6 million bytes if you stored the actual string `'medium'` in a `VARCHAR` column.

Index Values for Enumeration Literals

Each enumeration value has an index:

- The elements listed in the column specification are assigned index numbers, beginning with 1.
- The index value of the empty string error value is 0. This means that you can use the following `SELECT` statement to find rows into which invalid `ENUM` values were assigned:

  ```
  mysql> SELECT * FROM tbl_name WHERE enum_col=0;
  ```

- The index of the `NULL` value is `NULL`.
- The term “index” here refers to a position within the list of enumeration values. It has nothing to do with table indexes.

For example, a column specified as `ENUM('Mercury', 'Venus', 'Earth')` can have any of the values shown here. The index of each value is also shown.

Value	Index
`NULL`	`NULL`
`''`	0
`'Mercury'`	1
`'Venus'`	2
`'Earth'`	3

An `ENUM` column can have a maximum of 65,535 distinct elements. (The practical limit is less than 3000.) A table can have no more than 255 unique element list definitions among its `ENUM` and `SET` columns considered as a group. For more information on these limits, see Limits Imposed by .frm File Structure.

If you retrieve an `ENUM` value in a numeric context, the column value's index is returned. For example, you can retrieve numeric values from an `ENUM` column like this:

```
mysql> SELECT enum_col+0 FROM tbl_name;
```

Functions such as `SUM()` or `AVG()` that expect a numeric argument cast the argument to a number if necessary. For `ENUM` values, the index number is used in the calculation.

Handling of Enumeration Literals

Trailing spaces are automatically deleted from `ENUM` member values in the table definition when a table is created.

When retrieved, values stored into an `ENUM` column are displayed using the lettercase that was used in the column definition. Note that `ENUM` columns can be assigned a character set and collation. For

binary or case-sensitive collations, lettercase is taken into account when assigning values to the column.

If you store a number into an `ENUM` column, the number is treated as the index into the possible values, and the value stored is the enumeration member with that index. (However, this does *not* work with `LOAD DATA`, which treats all input as strings.) If the numeric value is quoted, it is still interpreted as an index if there is no matching string in the list of enumeration values. For these reasons, it is not advisable to define an `ENUM` column with enumeration values that look like numbers, because this can easily become confusing. For example, the following column has enumeration members with string values of `'0'`, `'1'`, and `'2'`, but numeric index values of `1`, `2`, and `3`:

```
numbers ENUM('0','1','2')
```

If you store `2`, it is interpreted as an index value, and becomes `'1'` (the value with index 2). If you store `'2'`, it matches an enumeration value, so it is stored as `'2'`. If you store `'3'`, it does not match any enumeration value, so it is treated as an index and becomes `'2'` (the value with index 3).

```
mysql> INSERT INTO t (numbers) VALUES(2),('2'),('3');
mysql> SELECT * FROM t;
+---------+
| numbers |
+---------+
| 1       |
| 2       |
| 2       |
+---------+
```

To determine all possible values for an `ENUM` column, use `SHOW COLUMNS FROM tbl_name LIKE 'enum_col'` and parse the `ENUM` definition in the `Type` column of the output.

In the C API, `ENUM` values are returned as strings. For information about using result set metadata to distinguish them from other strings, see C API Data Structures.

Empty or NULL Enumeration Values

An enumeration value can also be the empty string (`''`) or `NULL` under certain circumstances:

- If you insert an invalid value into an `ENUM` (that is, a string not present in the list of permitted values), the empty string is inserted instead as a special error value. This string can be distinguished from a “normal” empty string by the fact that this string has the numeric value 0. See Index Values for Enumeration Literals for details about the numeric indexes for the enumeration values.

  If strict SQL mode is enabled, attempts to insert invalid `ENUM` values result in an error.

- If an `ENUM` column is declared to permit `NULL`, the `NULL` value is a valid value for the column, and the default value is `NULL`. If an `ENUM` column is declared `NOT NULL`, its default value is the first element of the list of permitted values.

Enumeration Sorting

`ENUM` values are sorted based on their index numbers, which depend on the order in which the enumeration members were listed in the column specification. For example, `'b'` sorts before `'a'` for `ENUM('b', 'a')`. The empty string sorts before nonempty strings, and `NULL` values sort before all other enumeration values.

To prevent unexpected results when using the `ORDER BY` clause on an `ENUM` column, use one of these techniques:

- Specify the `ENUM` list in alphabetic order.

- Make sure that the column is sorted lexically rather than by index number by coding `ORDER BY CAST(col AS CHAR)` or `ORDER BY CONCAT(col)`.

Enumeration Limitations

An enumeration value cannot be an expression, even one that evaluates to a string value.

For example, this `CREATE TABLE` statement does *not* work because the `CONCAT` function cannot be used to construct an enumeration value:

```
CREATE TABLE sizes (
    size ENUM('small', CONCAT('med','ium'), 'large')
);
```

You also cannot employ a user variable as an enumeration value. This pair of statements do *not* work:

```
SET @mysize = 'medium';

CREATE TABLE sizes (
    size ENUM('small', @mysize, 'large')
);
```

We strongly recommend that you do *not* use numbers as enumeration values, because it does not save on storage over the appropriate `TINYINT` or `SMALLINT` type, and it is easy to mix up the strings and the underlying number values (which might not be the same) if you quote the `ENUM` values incorrectly. If you do use a number as an enumeration value, always enclose it in quotation marks. If the quotation marks are omitted, the number is regarded as an index. See Handling of Enumeration Literals to see how even a quoted number could be mistakenly used as a numeric index value.

Duplicate values in the definition cause a warning, or an error if strict SQL mode is enabled.

11.3.6 The SET Type

A `SET` is a string object that can have zero or more values, each of which must be chosen from a list of permitted values specified when the table is created. `SET` column values that consist of multiple set members are specified with members separated by commas (`,`). A consequence of this is that `SET` member values should not themselves contain commas.

For example, a column specified as `SET('one', 'two') NOT NULL` can have any of these values:

```
''
'one'
'two'
'one,two'
```

A `SET` column can have a maximum of 64 distinct members. A table can have no more than 255 unique element list definitions among its `ENUM` and `SET` columns considered as a group. For more information on this limit, see Limits Imposed by .frm File Structure.

Duplicate values in the definition cause a warning, or an error if strict SQL mode is enabled.

Trailing spaces are automatically deleted from `SET` member values in the table definition when a table is created.

See String Type Storage Requirements for storage requirements for the `SET` type.

See Section 11.3.1, “String Data Type Syntax” for `SET` type syntax and length limits.

When retrieved, values stored in a `SET` column are displayed using the lettercase that was used in the column definition. Note that `SET` columns can be assigned a character set and collation. For binary or case-sensitive collations, lettercase is taken into account when assigning values to the column.

MySQL stores `SET` values numerically, with the low-order bit of the stored value corresponding to the first set member. If you retrieve a `SET` value in a numeric context, the value retrieved has bits set corresponding to the set members that make up the column value. For example, you can retrieve numeric values from a `SET` column like this:

```
mysql> SELECT set_col+0 FROM tbl_name;
```

If a number is stored into a SET column, the bits that are set in the binary representation of the number determine the set members in the column value. For a column specified as SET('a','b','c','d'), the members have the following decimal and binary values.

SET Member	Decimal Value	Binary Value
'a'	1	0001
'b'	2	0010
'c'	4	0100
'd'	8	1000

If you assign a value of 9 to this column, that is 1001 in binary, so the first and fourth SET value members 'a' and 'd' are selected and the resulting value is 'a,d'.

For a value containing more than one SET element, it does not matter what order the elements are listed in when you insert the value. It also does not matter how many times a given element is listed in the value. When the value is retrieved later, each element in the value appears once, with elements listed according to the order in which they were specified at table creation time. Suppose that a column is specified as SET('a','b','c','d'):

```
mysql> CREATE TABLE myset (col SET('a', 'b', 'c', 'd'));
```

If you insert the values 'a,d', 'd,a', 'a,d,d', 'a,d,a', and 'd,a,d':

```
mysql> INSERT INTO myset (col) VALUES
-> ('a,d'), ('d,a'), ('a,d,a'), ('a,d,d'), ('d,a,d');
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Then all these values appear as 'a,d' when retrieved:

```
mysql> SELECT col FROM myset;
+------+
| col  |
+------+
| a,d  |
| a,d  |
| a,d  |
| a,d  |
| a,d  |
+------+
5 rows in set (0.04 sec)
```

If you set a SET column to an unsupported value, the value is ignored and a warning is issued:

```
mysql> INSERT INTO myset (col) VALUES ('a,d,d,s');
Query OK, 1 row affected, 1 warning (0.03 sec)

mysql> SHOW WARNINGS;
+---------+------+-----------------------------------------+
| Level   | Code | Message                                 |
+---------+------+-----------------------------------------+
| Warning | 1265 | Data truncated for column 'col' at row 1 |
+---------+------+-----------------------------------------+
1 row in set (0.04 sec)

mysql> SELECT col FROM myset;
+------+
| col  |
+------+
| a,d  |
| a,d  |
| a,d  |
| a,d  |
| a,d  |
```

```
| a,d  |
+------+
6 rows in set (0.01 sec)
```

If strict SQL mode is enabled, attempts to insert invalid `SET` values result in an error.

`SET` values are sorted numerically. `NULL` values sort before non-`NULL` `SET` values.

Functions such as `SUM()` or `AVG()` that expect a numeric argument cast the argument to a number if necessary. For `SET` values, the cast operation causes the numeric value to be used.

Normally, you search for `SET` values using the `FIND_IN_SET()` function or the `LIKE` operator:

```
mysql> SELECT * FROM tbl_name WHERE FIND_IN_SET('value',set_col)>0;
mysql> SELECT * FROM tbl_name WHERE set_col LIKE '%value%';
```

The first statement finds rows where *`set_col`* contains the *`value`* set member. The second is similar, but not the same: It finds rows where *`set_col`* contains *`value`* anywhere, even as a substring of another set member.

The following statements also are permitted:

```
mysql> SELECT * FROM tbl_name WHERE set_col & 1;
mysql> SELECT * FROM tbl_name WHERE set_col = 'val1,val2';
```

The first of these statements looks for values containing the first set member. The second looks for an exact match. Be careful with comparisons of the second type. Comparing set values to *`'val1,val2'`* returns different results than comparing values to *`'val2,val1'`*. You should specify the values in the same order they are listed in the column definition.

To determine all possible values for a `SET` column, use `SHOW COLUMNS FROM` *`tbl_name`* `LIKE` *`set_col`* and parse the `SET` definition in the `Type` column of the output.

In the C API, `SET` values are returned as strings. For information about using result set metadata to distinguish them from other strings, see C API Data Structures.

11.4 Spatial Data Types

The Open Geospatial Consortium (OGC) is an international consortium of more than 250 companies, agencies, and universities participating in the development of publicly available conceptual solutions that can be useful with all kinds of applications that manage spatial data.

The Open Geospatial Consortium publishes the *OpenGIS® Implementation Standard for Geographic information - Simple Feature Access - Part 2: SQL Option*, a document that proposes several conceptual ways for extending an SQL RDBMS to support spatial data. This specification is available from the OGC website at http://www.opengeospatial.org/standards/sfs.

Following the OGC specification, MySQL implements spatial extensions as a subset of the **SQL with Geometry Types** environment. This term refers to an SQL environment that has been extended with a set of geometry types. A geometry-valued SQL column is implemented as a column that has a geometry type. The specification describes a set of SQL geometry types, as well as functions on those types to create and analyze geometry values.

MySQL spatial extensions enable the generation, storage, and analysis of geographic features:

- Data types for representing spatial values
- Functions for manipulating spatial values
- Spatial indexing for improved access times to spatial columns

The spatial data types and functions are available for `MyISAM`, `InnoDB`, `NDB`, and `ARCHIVE` tables. For indexing spatial columns, `MyISAM` supports both `SPATIAL` and non-`SPATIAL` indexes. The other

storage engines support non-SPATIAL indexes, as described in Section 13.1.13, “CREATE INDEX Statement”.

A **geographic feature** is anything in the world that has a location. A feature can be:

- An entity. For example, a mountain, a pond, a city.
- A space. For example, town district, the tropics.
- A definable location. For example, a crossroad, as a particular place where two streets intersect.

Some documents use the term **geospatial feature** to refer to geographic features.

Geometry is another word that denotes a geographic feature. Originally the word **geometry** meant measurement of the earth. Another meaning comes from cartography, referring to the geometric features that cartographers use to map the world.

The discussion here considers these terms synonymous: **geographic feature**, **geospatial feature**, **feature**, or **geometry**. The term most commonly used is **geometry**, defined as *a point or an aggregate of points representing anything in the world that has a location*.

The following material covers these topics:

- The spatial data types implemented in MySQL model
- The basis of the spatial extensions in the OpenGIS geometry model
- Data formats for representing spatial data
- How to use spatial data in MySQL
- Use of indexing for spatial data
- MySQL differences from the OpenGIS specification

For information about functions that operate on spatial data, see Section 12.17, “Spatial Analysis Functions”.

MySQL GIS Conformance and Compatibility

MySQL does not implement the following GIS features:

- Additional Metadata Views

  OpenGIS specifications propose several additional metadata views. For example, a system view named GEOMETRY_COLUMNS contains a description of geometry columns, one row for each geometry column in the database.

- The OpenGIS function Length() on LineString and MultiLineString should be called in MySQL as GLength()

  The problem is that there is an existing SQL function Length() that calculates the length of string values, and sometimes it is not possible to distinguish whether the function is called in a textual or spatial context.

Additional Resources

The Open Geospatial Consortium publishes the *OpenGIS® Implementation Standard for Geographic information - Simple feature access - Part 2: SQL option*, a document that proposes several conceptual ways for extending an SQL RDBMS to support spatial data. The Open Geospatial Consortium (OGC) maintains a website at http://www.opengeospatial.org/. The specification is available there at http://www.opengeospatial.org/standards/sfs. It contains additional information relevant to the material here.

If you have questions or concerns about the use of the spatial extensions to MySQL, you can discuss them in the GIS forum: https://forums.mysql.com/list.php?23.

11.4.1 Spatial Data Types

MySQL has spatial data types that correspond to OpenGIS classes. The basis for these types is described in Section 11.4.2, “The OpenGIS Geometry Model”.

Some spatial data types hold single geometry values:

- `GEOMETRY`
- `POINT`
- `LINESTRING`
- `POLYGON`

`GEOMETRY` can store geometry values of any type. The other single-value types (`POINT`, `LINESTRING`, and `POLYGON`) restrict their values to a particular geometry type.

The other spatial data types hold collections of values:

- `MULTIPOINT`
- `MULTILINESTRING`
- `MULTIPOLYGON`
- `GEOMETRYCOLLECTION`

`GEOMETRYCOLLECTION` can store a collection of objects of any type. The other collection types (`MULTIPOINT`, `MULTILINESTRING`, and `MULTIPOLYGON`) restrict collection members to those having a particular geometry type.

Example: To create a table named `geom` that has a column named `g` that can store values of any geometry type, use this statement:

```
CREATE TABLE geom (g GEOMETRY);
```

`SPATIAL` indexes can be created on `NOT NULL` spatial columns, so if you plan to index the column, declare it `NOT NULL`:

```
CREATE TABLE geom (g GEOMETRY NOT NULL);
```

For other examples showing how to use spatial data types in MySQL, see Section 11.4.4, “Creating Spatial Columns”.

11.4.2 The OpenGIS Geometry Model

The set of geometry types proposed by OGC's **SQL with Geometry Types** environment is based on the **OpenGIS Geometry Model**. In this model, each geometric object has the following general properties:

- It is associated with a spatial reference system, which describes the coordinate space in which the object is defined.
- It belongs to some geometry class.

11.4.2.1 The Geometry Class Hierarchy

The geometry classes define a hierarchy as follows:

- `Geometry` (noninstantiable)

- `Point` (instantiable)
- `Curve` (noninstantiable)
  - `LineString` (instantiable)
    - `Line`
    - `LinearRing`
- `Surface` (noninstantiable)
  - `Polygon` (instantiable)
- `GeometryCollection` (instantiable)
  - `MultiPoint` (instantiable)
  - `MultiCurve` (noninstantiable)
    - `MultiLineString` (instantiable)
  - `MultiSurface` (noninstantiable)
    - `MultiPolygon` (instantiable)

It is not possible to create objects in noninstantiable classes. It is possible to create objects in instantiable classes. All classes have properties, and instantiable classes may also have assertions (rules that define valid class instances).

`Geometry` is the base class. It is an abstract class. The instantiable subclasses of `Geometry` are restricted to zero-, one-, and two-dimensional geometric objects that exist in two-dimensional coordinate space. All instantiable geometry classes are defined so that valid instances of a geometry class are topologically closed (that is, all defined geometries include their boundary).

The base `Geometry` class has subclasses for `Point`, `Curve`, `Surface`, and `GeometryCollection`:

- `Point` represents zero-dimensional objects.
- `Curve` represents one-dimensional objects, and has subclass `LineString`, with sub-subclasses `Line` and `LinearRing`.
- `Surface` is designed for two-dimensional objects and has subclass `Polygon`.
- `GeometryCollection` has specialized zero-, one-, and two-dimensional collection classes named `MultiPoint`, `MultiLineString`, and `MultiPolygon` for modeling geometries corresponding to collections of `Points`, `LineStrings`, and `Polygons`, respectively. `MultiCurve` and `MultiSurface` are introduced as abstract superclasses that generalize the collection interfaces to handle `Curves` and `Surfaces`.

`Geometry`, `Curve`, `Surface`, `MultiCurve`, and `MultiSurface` are defined as noninstantiable classes. They define a common set of methods for their subclasses and are included for extensibility.

`Point`, `LineString`, `Polygon`, `GeometryCollection`, `MultiPoint`, `MultiLineString`, and `MultiPolygon` are instantiable classes.

11.4.2.2 Geometry Class

`Geometry` is the root class of the hierarchy. It is a noninstantiable class but has a number of properties, described in the following list, that are common to all geometry values created from any of the `Geometry` subclasses. Particular subclasses have their own specific properties, described later.

Geometry Properties

A geometry value has the following properties:

- Its **type**. Each geometry belongs to one of the instantiable classes in the hierarchy.
- Its **SRID**, or spatial reference identifier. This value identifies the geometry's associated spatial reference system that describes the coordinate space in which the geometry object is defined.

  In MySQL, the SRID value is an integer associated with the geometry value. The maximum usable SRID value is $2^{32}-1$. If a larger value is given, only the lower 32 bits are used. All computations are done assuming SRID 0, regardless of the actual SRID value. SRID 0 represents an infinite flat Cartesian plane with no units assigned to its axes.
- Its **coordinates** in its spatial reference system, represented as double-precision (8-byte) numbers. All nonempty geometries include at least one pair of (X,Y) coordinates. Empty geometries contain no coordinates.

  Coordinates are related to the SRID. For example, in different coordinate systems, the distance between two objects may differ even when objects have the same coordinates, because the distance on the **planar** coordinate system and the distance on the **geodetic** system (coordinates on the Earth's surface) are different things.
- Its **interior**, **boundary**, and **exterior**.

  Every geometry occupies some position in space. The exterior of a geometry is all space not occupied by the geometry. The interior is the space occupied by the geometry. The boundary is the interface between the geometry's interior and exterior.
- Its **MBR** (minimum bounding rectangle), or envelope. This is the bounding geometry, formed by the minimum and maximum (X,Y) coordinates:

  ```
  ((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
  ```
- Whether the value is **simple** or **nonsimple**. Geometry values of types (`LineString`, `MultiPoint`, `MultiLineString`) are either simple or nonsimple. Each type determines its own assertions for being simple or nonsimple.
- Whether the value is **closed** or **not closed**. Geometry values of types (`LineString`, `MultiString`) are either closed or not closed. Each type determines its own assertions for being closed or not closed.
- Whether the value is **empty** or **nonempty** A geometry is empty if it does not have any points. Exterior, interior, and boundary of an empty geometry are not defined (that is, they are represented by a `NULL` value). An empty geometry is defined to be always simple and has an area of 0.
- Its **dimension**. A geometry can have a dimension of −1, 0, 1, or 2:
  - −1 for an empty geometry.
  - 0 for a geometry with no length and no area.
  - 1 for a geometry with nonzero length and zero area.
  - 2 for a geometry with nonzero area.

  `Point` objects have a dimension of zero. `LineString` objects have a dimension of 1. `Polygon` objects have a dimension of 2. The dimensions of `MultiPoint`, `MultiLineString`, and `MultiPolygon` objects are the same as the dimensions of the elements they consist of.

11.4.2.3 Point Class

A `Point` is a geometry that represents a single location in coordinate space.

`Point` Examples

- Imagine a large-scale map of the world with many cities. A `Point` object could represent each city.
- On a city map, a `Point` object could represent a bus stop.

`Point` Properties

- X-coordinate value.
- Y-coordinate value.
- `Point` is defined as a zero-dimensional geometry.
- The boundary of a `Point` is the empty set.

11.4.2.4 Curve Class

A `Curve` is a one-dimensional geometry, usually represented by a sequence of points. Particular subclasses of `Curve` define the type of interpolation between points. `Curve` is a noninstantiable class.

`Curve` Properties

- A `Curve` has the coordinates of its points.
- A `Curve` is defined as a one-dimensional geometry.
- A `Curve` is simple if it does not pass through the same point twice, with the exception that a curve can still be simple if the start and end points are the same.
- A `Curve` is closed if its start point is equal to its endpoint.
- The boundary of a closed `Curve` is empty.
- The boundary of a nonclosed `Curve` consists of its two endpoints.
- A `Curve` that is simple and closed is a `LinearRing`.

11.4.2.5 LineString Class

A `LineString` is a `Curve` with linear interpolation between points.

`LineString` Examples

- On a world map, `LineString` objects could represent rivers.
- In a city map, `LineString` objects could represent streets.

`LineString` Properties

- A `LineString` has coordinates of segments, defined by each consecutive pair of points.
- A `LineString` is a `Line` if it consists of exactly two points.
- A `LineString` is a `LinearRing` if it is both closed and simple.

11.4.2.6 Surface Class

A `Surface` is a two-dimensional geometry. It is a noninstantiable class. Its only instantiable subclass is `Polygon`.

`Surface` Properties

- A `Surface` is defined as a two-dimensional geometry.
- The OpenGIS specification defines a simple `Surface` as a geometry that consists of a single “patch” that is associated with a single exterior boundary and zero or more interior boundaries.
- The boundary of a simple `Surface` is the set of closed curves corresponding to its exterior and interior boundaries.

11.4.2.7 Polygon Class

A `Polygon` is a planar `Surface` representing a multisided geometry. It is defined by a single exterior boundary and zero or more interior boundaries, where each interior boundary defines a hole in the `Polygon`.

`Polygon` Examples

- On a region map, `Polygon` objects could represent forests, districts, and so on.

`Polygon` Assertions

- The boundary of a `Polygon` consists of a set of `LinearRing` objects (that is, `LineString` objects that are both simple and closed) that make up its exterior and interior boundaries.
- A `Polygon` has no rings that cross. The rings in the boundary of a `Polygon` may intersect at a `Point`, but only as a tangent.
- A `Polygon` has no lines, spikes, or punctures.
- A `Polygon` has an interior that is a connected point set.
- A `Polygon` may have holes. The exterior of a `Polygon` with holes is not connected. Each hole defines a connected component of the exterior.

The preceding assertions make a `Polygon` a simple geometry.

11.4.2.8 GeometryCollection Class

A `GeometryCollection` is a geometry that is a collection of zero or more geometries of any class.

All the elements in a geometry collection must be in the same spatial reference system (that is, in the same coordinate system). There are no other constraints on the elements of a geometry collection, although the subclasses of `GeometryCollection` described in the following sections may restrict membership. Restrictions may be based on:

- Element type (for example, a `MultiPoint` may contain only `Point` elements)
- Dimension
- Constraints on the degree of spatial overlap between elements

11.4.2.9 MultiPoint Class

A `MultiPoint` is a geometry collection composed of `Point` elements. The points are not connected or ordered in any way.

`MultiPoint` Examples

- On a world map, a `MultiPoint` could represent a chain of small islands.
- On a city map, a `MultiPoint` could represent the outlets for a ticket office.

`MultiPoint` Properties

- A `MultiPoint` is a zero-dimensional geometry.
- A `MultiPoint` is simple if no two of its `Point` values are equal (have identical coordinate values).
- The boundary of a `MultiPoint` is the empty set.

11.4.2.10 MultiCurve Class

A `MultiCurve` is a geometry collection composed of `Curve` elements. `MultiCurve` is a noninstantiable class.

`MultiCurve` Properties

- A `MultiCurve` is a one-dimensional geometry.
- A `MultiCurve` is simple if and only if all of its elements are simple; the only intersections between any two elements occur at points that are on the boundaries of both elements.
- A `MultiCurve` boundary is obtained by applying the “mod 2 union rule” (also known as the “odd-even rule”): A point is in the boundary of a `MultiCurve` if it is in the boundaries of an odd number of `Curve` elements.
- A `MultiCurve` is closed if all of its elements are closed.
- The boundary of a closed `MultiCurve` is always empty.

11.4.2.11 MultiLineString Class

A `MultiLineString` is a `MultiCurve` geometry collection composed of `LineString` elements.

`MultiLineString` Examples

- On a region map, a `MultiLineString` could represent a river system or a highway system.

11.4.2.12 MultiSurface Class

A `MultiSurface` is a geometry collection composed of surface elements. `MultiSurface` is a noninstantiable class. Its only instantiable subclass is `MultiPolygon`.

`MultiSurface` Assertions

- Surfaces within a `MultiSurface` have no interiors that intersect.
- Surfaces within a `MultiSurface` have boundaries that intersect at most at a finite number of points.

11.4.2.13 MultiPolygon Class

A `MultiPolygon` is a `MultiSurface` object composed of `Polygon` elements.

`MultiPolygon` Examples

- On a region map, a `MultiPolygon` could represent a system of lakes.

`MultiPolygon` Assertions

- A `MultiPolygon` has no two `Polygon` elements with interiors that intersect.
- A `MultiPolygon` has no two `Polygon` elements that cross (crossing is also forbidden by the previous assertion), or that touch at an infinite number of points.
- A `MultiPolygon` may not have cut lines, spikes, or punctures. A `MultiPolygon` is a regular, closed point set.

- A `MultiPolygon` that has more than one `Polygon` has an interior that is not connected. The number of connected components of the interior of a `MultiPolygon` is equal to the number of `Polygon` values in the `MultiPolygon`.

`MultiPolygon` Properties

- A `MultiPolygon` is a two-dimensional geometry.
- A `MultiPolygon` boundary is a set of closed curves (`LineString` values) corresponding to the boundaries of its `Polygon` elements.
- Each `Curve` in the boundary of the `MultiPolygon` is in the boundary of exactly one `Polygon` element.
- Every `Curve` in the boundary of an `Polygon` element is in the boundary of the `MultiPolygon`.

11.4.3 Supported Spatial Data Formats

Two standard spatial data formats are used to represent geometry objects in queries:

- Well-Known Text (WKT) format
- Well-Known Binary (WKB) format

Internally, MySQL stores geometry values in a format that is not identical to either WKT or WKB format. (Internal format is like WKB but with an initial 4 bytes to indicate the SRID.)

There are functions available to convert between different data formats; see Section 12.17.6, “Geometry Format Conversion Functions”.

The following sections describe the spatial data formats MySQL uses:

- Well-Known Text (WKT) Format
- Well-Known Binary (WKB) Format
- Internal Geometry Storage Format

Well-Known Text (WKT) Format

The Well-Known Text (WKT) representation of geometry values is designed for exchanging geometry data in ASCII form. The OpenGIS specification provides a Backus-Naur grammar that specifies the formal production rules for writing WKT values (see Section 11.4, “Spatial Data Types”).

Examples of WKT representations of geometry objects:

- A `Point`:

```
POINT(15 20)
```

The point coordinates are specified with no separating comma. This differs from the syntax for the SQL `Point()` function, which requires a comma between the coordinates. Take care to use the syntax appropriate to the context of a given spatial operation. For example, the following statements both use `ST_X()` to extract the X-coordinate from a `Point` object. The first produces the object directly using the `Point()` function. The second uses a WKT representation converted to a `Point` with `ST_GeomFromText()`.

```
mysql> SELECT X(Point(15, 20));
+------------------+
| X(POINT(15, 20)) |
+------------------+
|               15 |
+------------------+
```

```
mysql> SELECT X(GeomFromText('POINT(15 20)'));
+---------------------------------+
| X(GeomFromText('POINT(15 20)')) |
+---------------------------------+
|                              15 |
+---------------------------------+
```

- A `LineString` with four points:

```
LINESTRING(0 0, 10 10, 20 25, 50 60)
```

  The point coordinate pairs are separated by commas.

- A `Polygon` with one exterior ring and one interior ring:

```
POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))
```

- A `MultiPoint` with three `Point` values:

```
MULTIPOINT(0 0, 20 20, 60 60)
```

- A `MultiLineString` with two `LineString` values:

```
MULTILINESTRING((10 10, 20 20), (15 15, 30 15))
```

- A `MultiPolygon` with two `Polygon` values:

```
MULTIPOLYGON(((0 0,10 0,10 10,0 10,0 0)),((5 5,7 5,7 7,5 7, 5 5)))
```

- A `GeometryCollection` consisting of two `Point` values and one `LineString`:

```
GEOMETRYCOLLECTION(POINT(10 10), POINT(30 30), LINESTRING(15 15, 20 20))
```

Well-Known Binary (WKB) Format

The Well-Known Binary (WKB) representation of geometric values is used for exchanging geometry data as binary streams represented by `BLOB` values containing geometric WKB information. This format is defined by the OpenGIS specification (see Section 11.4, “Spatial Data Types”). It is also defined in the ISO *SQL/MM Part 3: Spatial* standard.

WKB uses 1-byte unsigned integers, 4-byte unsigned integers, and 8-byte double-precision numbers (IEEE 754 format). A byte is eight bits.

For example, a WKB value that corresponds to `POINT(1 -1)` consists of this sequence of 21 bytes, each represented by two hexadecimal digits:

```
0101000000000000000000F03F000000000000F0BF
```

The sequence consists of the components shown in the following table.

Table 11.2 WKB Components Example

Component	Size	Value
Byte order	1 byte	`01`
WKB type	4 bytes	`01000000`
X coordinate	8 bytes	`000000000000F03F`
Y coordinate	8 bytes	`000000000000F0BF`

Component representation is as follows:

- The byte order indicator is either 1 or 0 to signify little-endian or big-endian storage. The little-endian and big-endian byte orders are also known as Network Data Representation (NDR) and External Data Representation (XDR), respectively.

- The WKB type is a code that indicates the geometry type. MySQL uses values from 1 through 7 to indicate `Point`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString`, `MultiPolygon`, and `GeometryCollection`.
- A `Point` value has X and Y coordinates, each represented as a double-precision value.

WKB values for more complex geometry values have more complex data structures, as detailed in the OpenGIS specification.

Internal Geometry Storage Format

MySQL stores geometry values using 4 bytes to indicate the SRID followed by the WKB representation of the value. For a description of WKB format, see Well-Known Binary (WKB) Format.

For the WKB part, these MySQL-specific considerations apply:

- The byte-order indicator byte is 1 because MySQL stores geometries as little-endian values.
- MySQL supports geometry types of `Point`, `LineString`, `Polygon`, `MultiPoint`, `MultiLineString`, `MultiPolygon`, and `GeometryCollection`. Other geometry types are not supported.

The `LENGTH()` function returns the space in bytes required for value storage. Example:

```
mysql> SET @g = ST_GeomFromText('POINT(1 -1)');
mysql> SELECT LENGTH(@g);
+------------+
| LENGTH(@g) |
+------------+
|         25 |
+------------+
mysql> SELECT HEX(@g);
+----------------------------------------------------+
| HEX(@g)                                            |
+----------------------------------------------------+
| 000000000101000000000000000000F03F000000000000F0BF |
+----------------------------------------------------+
```

The value length is 25 bytes, made up of these components (as can be seen from the hexadecimal value):

- 4 bytes for integer SRID (0)
- 1 byte for integer byte order (1 = little-endian)
- 4 bytes for integer type information (1 = `Point`)
- 8 bytes for double-precision X coordinate (1)
- 8 bytes for double-precision Y coordinate (−1)

11.4.4 Creating Spatial Columns

MySQL provides a standard way of creating spatial columns for geometry types, for example, with `CREATE TABLE` or `ALTER TABLE`. Spatial columns are supported for `MyISAM`, `InnoDB`, `NDB`, and `ARCHIVE` tables. See also the notes about spatial indexes under Section 11.4.8, “Creating Spatial Indexes”.

- Use the `CREATE TABLE` statement to create a table with a spatial column:

```
CREATE TABLE geom (g GEOMETRY);
```

- Use the `ALTER TABLE` statement to add or drop a spatial column to or from an existing table:

```
ALTER TABLE geom ADD pt POINT;
ALTER TABLE geom DROP pt;
```

11.4.5 Populating Spatial Columns

After you have created spatial columns, you can populate them with spatial data.

Values should be stored in internal geometry format, but you can convert them to that format from either Well-Known Text (WKT) or Well-Known Binary (WKB) format. The following examples demonstrate how to insert geometry values into a table by converting WKT values to internal geometry format:

- Perform the conversion directly in the `INSERT` statement:

```
INSERT INTO geom VALUES (GeomFromText('POINT(1 1)'));

SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (GeomFromText(@g));
```

- Perform the conversion prior to the `INSERT`:

```
SET @g = GeomFromText('POINT(1 1)');
INSERT INTO geom VALUES (@g);
```

The following examples insert more complex geometries into the table:

```
SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (GeomFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (GeomFromText(@g));

SET @g =
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
INSERT INTO geom VALUES (GeomFromText(@g));
```

The preceding examples use `GeomFromText()` to create geometry values. You can also use type-specific functions:

```
SET @g = 'POINT(1 1)';
INSERT INTO geom VALUES (PointFromText(@g));

SET @g = 'LINESTRING(0 0,1 1,2 2)';
INSERT INTO geom VALUES (LineStringFromText(@g));

SET @g = 'POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7, 5 5))';
INSERT INTO geom VALUES (PolygonFromText(@g));

SET @g =
'GEOMETRYCOLLECTION(POINT(1 1),LINESTRING(0 0,1 1,2 2,3 3,4 4))';
INSERT INTO geom VALUES (GeomCollFromText(@g));
```

A client application program that wants to use WKB representations of geometry values is responsible for sending correctly formed WKB in queries to the server. There are several ways to satisfy this requirement. For example:

- Inserting a `POINT(1 1)` value with hex literal syntax:

```
INSERT INTO geom VALUES
(GeomFromWKB(X'0101000000000000000000F03F000000000000F03F'));
```

- An ODBC application can send a WKB representation, binding it to a placeholder using an argument of `BLOB` type:

```
INSERT INTO geom VALUES (GeomFromWKB(?))
```

  Other programming interfaces may support a similar placeholder mechanism.

- In a C program, you can escape a binary value using `mysql_real_escape_string()` and include the result in a query string that is sent to the server. See mysql_real_escape_string().

11.4.6 Fetching Spatial Data

Geometry values stored in a table can be fetched in internal format. You can also convert them to WKT or WKB format.

- Fetching spatial data in internal format:

  Fetching geometry values using internal format can be useful in table-to-table transfers:

  ```
  CREATE TABLE geom2 (g GEOMETRY) SELECT g FROM geom;
  ```

- Fetching spatial data in WKT format:

  The `AsText()` function converts a geometry from internal format to a WKT string.

  ```
  SELECT AsText(g) FROM geom;
  ```

- Fetching spatial data in WKB format:

  The `AsBinary()` function converts a geometry from internal format to a `BLOB` containing the WKB value.

  ```
  SELECT AsBinary(g) FROM geom;
  ```

11.4.7 Optimizing Spatial Analysis

For `MyISAM` tables, search operations in columns containing spatial data can be optimized using `SPATIAL` indexes. The most typical operations are:

- Point queries that search for all objects that contain a given point
- Region queries that search for all objects that overlap a given region

MySQL uses **R-Trees with quadratic splitting** for `SPATIAL` indexes on spatial columns. A `SPATIAL` index is built using the minimum bounding rectangle (MBR) of a geometry. For most geometries, the MBR is a minimum rectangle that surrounds the geometries. For a horizontal or a vertical linestring, the MBR is a rectangle degenerated into the linestring. For a point, the MBR is a rectangle degenerated into the point.

It is also possible to create normal indexes on spatial columns. In a non-`SPATIAL` index, you must declare a prefix for any spatial column except for `POINT` columns.

`MyISAM` supports both `SPATIAL` and non-`SPATIAL` indexes. Other storage engines support non-`SPATIAL` indexes, as described in Section 13.1.13, “CREATE INDEX Statement”.

11.4.8 Creating Spatial Indexes

For `MyISAM` tables, MySQL can create spatial indexes using syntax similar to that for creating regular indexes, but using the `SPATIAL` keyword. Columns in spatial indexes must be declared `NOT NULL`. The following examples demonstrate how to create spatial indexes:

- With `CREATE TABLE`:

  ```
  CREATE TABLE geom (g GEOMETRY NOT NULL, SPATIAL INDEX(g)) ENGINE=MyISAM;
  ```

- With `ALTER TABLE`:

  ```
  CREATE TABLE geom (g GEOMETRY NOT NULL) ENGINE=MyISAM;
  ALTER TABLE geom ADD SPATIAL INDEX(g);
  ```

- With `CREATE INDEX`:

  ```
  CREATE TABLE geom (g GEOMETRY NOT NULL) ENGINE=MyISAM;
  ```

```
CREATE SPATIAL INDEX g ON geom (g);
```

`SPATIAL INDEX` creates an R-tree index. For storage engines that support nonspatial indexing of spatial columns, the engine creates a B-tree index. A B-tree index on spatial values is useful for exact-value lookups, but not for range scans.

For more information on indexing spatial columns, see Section 13.1.13, “CREATE INDEX Statement”.

To drop spatial indexes, use `ALTER TABLE` or `DROP INDEX`:

- With `ALTER TABLE`:

```
ALTER TABLE geom DROP INDEX g;
```

- With `DROP INDEX`:

```
DROP INDEX g ON geom;
```

Example: Suppose that a table `geom` contains more than 32,000 geometries, which are stored in the column `g` of type `GEOMETRY`. The table also has an `AUTO_INCREMENT` column `fid` for storing object ID values.

```
mysql> DESCRIBE geom;
+-------+----------+------+-----+---------+----------------+
| Field | Type     | Null | Key | Default | Extra          |
+-------+----------+------+-----+---------+----------------+
| fid   | int(11)  |      | PRI | NULL    | auto_increment |
| g     | geometry |      |     |         |                |
+-------+----------+------+-----+---------+----------------+
2 rows in set (0.00 sec)

mysql> SELECT COUNT(*) FROM geom;
+----------+
| count(*) |
+----------+
|    32376 |
+----------+
1 row in set (0.00 sec)
```

To add a spatial index on the column `g`, use this statement:

```
mysql> ALTER TABLE geom ADD SPATIAL INDEX(g) ENGINE=MyISAM;
Query OK, 32376 rows affected (4.05 sec)
Records: 32376  Duplicates: 0  Warnings: 0
```

11.4.9 Using Spatial Indexes

The optimizer investigates whether available spatial indexes can be involved in the search for queries that use a function such as `MBRContains()` or `MBRWithin()` in the `WHERE` clause. The following query finds all objects that are in the given rectangle:

```
mysql> SET @poly =
    -> 'Polygon((30000 15000,
                 31000 15000,
                 31000 16000,
                 30000 16000,
                 30000 15000))';
mysql> SELECT fid,AsText(g) FROM geom WHERE
    -> MBRContains(GeomFromText(@poly),g);
+-----+-------------------------------------------------------------+
| fid | AsText(g)                                                   |
+-----+-------------------------------------------------------------+
|  21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30 ... |
|  22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8, ... |
|  23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4, ... |
|  24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4, ... |
|  25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882. ... |
|  26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4, ... |
```

```
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946. ... |
|   1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136. ... |
|   2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136, ... |
|   3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,3016 ... |
|   4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30 ... |
|   5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4, ... |
|   6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,3024 ... |
|   7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8, ... |
|  10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6, ... |
|  11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2, ... |
|  13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,3011 ... |
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30 ... |
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30 ... |
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4, ... |
+-----+-----------------------------------------------------------+
20 rows in set (0.00 sec)
```

Use `EXPLAIN` to check the way this query is executed:

```
mysql> SET @poly =
    -> 'Polygon((30000 15000,
                 31000 15000,
                 31000 16000,
                 30000 16000,
                 30000 15000))';
mysql> EXPLAIN SELECT fid,AsText(g) FROM geom WHERE
    -> MBRContains(GeomFromText(@poly),g)\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: geom
         type: range
possible_keys: g
          key: g
      key_len: 32
          ref: NULL
         rows: 50
        Extra: Using where
1 row in set (0.00 sec)
```

Check what would happen without a spatial index:

```
mysql> SET @poly =
    -> 'Polygon((30000 15000,
                 31000 15000,
                 31000 16000,
                 30000 16000,
                 30000 15000))';
mysql> EXPLAIN SELECT fid,AsText(g) FROM g IGNORE INDEX (g) WHERE
    -> MBRContains(GeomFromText(@poly),g)\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: geom
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 32376
        Extra: Using where
1 row in set (0.00 sec)
```

Executing the `SELECT` statement without the spatial index yields the same result but causes the execution time to rise from 0.00 seconds to 0.46 seconds:

```
mysql> SET @poly =
    -> 'Polygon((30000 15000,
                 31000 15000,
                 31000 16000,
                 30000 16000,
```

```
                30000 15000))';
mysql> SELECT fid,AsText(g) FROM geom IGNORE INDEX (g) WHERE
    -> MBRContains(GeomFromText(@poly),g);
+-----+-------------------------------------------------------------+
| fid | AsText(g)                                                   |
+-----+-------------------------------------------------------------+
|   1 | LINESTRING(30250.4 15129.2,30248.8 15138.4,30238.2 15136. ... |
|   2 | LINESTRING(30220.2 15122.8,30217.2 15137.8,30207.6 15136, ... |
|   3 | LINESTRING(30179 15114.4,30176.6 15129.4,30167 15128,3016 ... |
|   4 | LINESTRING(30155.2 15121.4,30140.4 15118.6,30142 15109,30 ... |
|   5 | LINESTRING(30192.4 15085,30177.6 15082.2,30179.2 15072.4, ... |
|   6 | LINESTRING(30244 15087,30229 15086.2,30229.4 15076.4,3024 ... |
|   7 | LINESTRING(30200.6 15059.4,30185.6 15058.6,30186 15048.8, ... |
|  10 | LINESTRING(30179.6 15017.8,30181 15002.8,30190.8 15003.6, ... |
|  11 | LINESTRING(30154.2 15000.4,30168.6 15004.8,30166 15014.2, ... |
|  13 | LINESTRING(30105 15065.8,30108.4 15050.8,30118 15053,3011 ... |
|  21 | LINESTRING(30350.4 15828.8,30350.6 15845,30333.8 15845,30 ... |
|  22 | LINESTRING(30350.6 15871.4,30350.6 15887.8,30334 15887.8, ... |
|  23 | LINESTRING(30350.6 15914.2,30350.6 15930.4,30334 15930.4, ... |
|  24 | LINESTRING(30290.2 15823,30290.2 15839.4,30273.4 15839.4, ... |
|  25 | LINESTRING(30291.4 15866.2,30291.6 15882.4,30274.8 15882. ... |
|  26 | LINESTRING(30291.6 15918.2,30291.6 15934.4,30275 15934.4, ... |
| 154 | LINESTRING(30276.2 15143.8,30261.4 15141,30263 15131.4,30 ... |
| 155 | LINESTRING(30269.8 15084,30269.4 15093.4,30258.6 15093,30 ... |
| 157 | LINESTRING(30128.2 15011,30113.2 15010.2,30113.6 15000.4, ... |
| 249 | LINESTRING(30337.8 15938.6,30337.8 15946.8,30320.4 15946. ... |
+-----+-------------------------------------------------------------+
20 rows in set (0.46 sec)
```

11.5 Data Type Default Values

Data type specifications can have explicit or implicit default values.

- Explicit Default Handling
- Implicit Default Handling

Explicit Default Handling

A `DEFAULT value` clause in a data type specification explicitly indicates a default value for a column. Examples:

```
CREATE TABLE t1 (
  i     INT DEFAULT -1,
  c     VARCHAR(10) DEFAULT '',
  price DOUBLE(16,2) DEFAULT '0.00'
);
```

`SERIAL DEFAULT VALUE` is a special case. In the definition of an integer column, it is an alias for `NOT NULL AUTO_INCREMENT UNIQUE`.

With one exception, the default value specified in a `DEFAULT` clause must be a literal constant; it cannot be a function or an expression. This means, for example, that you cannot set the default for a date column to be the value of a function such as `NOW()` or `CURRENT_DATE`. The exception is that, for `TIMESTAMP` and `DATETIME` columns, you can specify `CURRENT_TIMESTAMP` as the default. See Section 11.2.6, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”.

The `BLOB` and `TEXT` data types cannot be assigned a default value.

Implicit Default Handling

If a data type specification includes no explicit `DEFAULT` value, MySQL determines the default value as follows:

If the column can take `NULL` as a value, the column is defined with an explicit `DEFAULT NULL` clause.

If the column cannot take `NULL` as a value, MySQL defines the column with no explicit `DEFAULT` clause. Exception: If the column is defined as part of a `PRIMARY KEY` but not explicitly as `NOT NULL`, MySQL creates it as a `NOT NULL` column (because `PRIMARY KEY` columns must be `NOT NULL`), but also assigns it a `DEFAULT` clause using the implicit default value. To prevent this, include an explicit `NOT NULL` in the definition of any `PRIMARY KEY` column.

For data entry into a `NOT NULL` column that has no explicit `DEFAULT` clause, if an `INSERT` or `REPLACE` statement includes no value for the column, or an `UPDATE` statement sets the column to `NULL`, MySQL handles the column according to the SQL mode in effect at the time:

- If strict SQL mode is enabled, an error occurs for transactional tables and the statement is rolled back. For nontransactional tables, an error occurs, but if this happens for the second or subsequent row of a multiple-row statement, the preceding rows have already been inserted.
- If strict mode is not enabled, MySQL sets the column to the implicit default value for the column data type.

Suppose that a table `t` is defined as follows:

```
CREATE TABLE t (i INT NOT NULL);
```

In this case, `i` has no explicit default, so in strict mode each of the following statements produce an error and no row is inserted. When not using strict mode, only the third statement produces an error; the implicit default is inserted for the first two statements, but the third fails because `DEFAULT(i)` cannot produce a value:

```
INSERT INTO t VALUES();
INSERT INTO t VALUES(DEFAULT);
INSERT INTO t VALUES(DEFAULT(i));
```

See Section 5.1.10, “Server SQL Modes”.

For a given table, the `SHOW CREATE TABLE` statement displays which columns have an explicit `DEFAULT` clause.

Implicit defaults are defined as follows:

- For numeric types, the default is `0`, with the exception that for integer or floating-point types declared with the `AUTO_INCREMENT` attribute, the default is the next value in the sequence.
- For date and time types other than `TIMESTAMP`, the default is the appropriate “zero” value for the type. This is also true for `TIMESTAMP` if the `explicit_defaults_for_timestamp` system variable is enabled (see Section 5.1.7, “Server System Variables”). Otherwise, for the first `TIMESTAMP` column in a table, the default value is the current date and time. See Section 11.2, “Date and Time Data Types”.
- For string types other than `ENUM`, the default value is the empty string. For `ENUM`, the default is the first enumeration value.

11.6 Data Type Storage Requirements

- InnoDB Table Storage Requirements
- NDB Table Storage Requirements
- Numeric Type Storage Requirements
- Date and Time Type Storage Requirements
- String Type Storage Requirements
- Spatial Type Storage Requirements

The storage requirements for table data on disk depend on several factors. Different storage engines represent data types and store raw data differently. Table data might be compressed, either for a column or an entire row, complicating the calculation of storage requirements for a table or column.

Despite differences in storage layout on disk, the internal MySQL APIs that communicate and exchange information about table rows use a consistent data structure that applies across all storage engines.

This section includes guidelines and information for the storage requirements for each data type supported by MySQL, including the internal format and size for storage engines that use a fixed-size representation for data types. Information is listed by category or storage engine.

The internal representation of a table has a maximum row size of 65,535 bytes, even if the storage engine is capable of supporting larger rows. This figure excludes `BLOB` or `TEXT` columns, which contribute only 9 to 12 bytes toward this size. For `BLOB` and `TEXT` data, the information is stored internally in a different area of memory than the row buffer. Different storage engines handle the allocation and storage of this data in different ways, according to the method they use for handling the corresponding types. For more information, see Chapter 15, *Alternative Storage Engines*, and Section 8.4.7, “Limits on Table Column Count and Row Size”.

InnoDB Table Storage Requirements

See Section 14.11, “InnoDB Row Formats” for information about storage requirements for `InnoDB` tables.

NDB Table Storage Requirements

Important

`NDB` tables use *4-byte alignment*; all `NDB` data storage is done in multiples of 4 bytes. Thus, a column value that would typically take 15 bytes requires 16 bytes in an `NDB` table. For example, in `NDB` tables, the `TINYINT`, `SMALLINT`, `MEDIUMINT`, and `INTEGER` (`INT`) column types each require 4 bytes storage per record due to the alignment factor.

Each `BIT(M)` column takes *`M`* bits of storage space. Although an individual `BIT` column is *not* 4-byte aligned, `NDB` reserves 4 bytes (32 bits) per row for the first 1-32 bits needed for `BIT` columns, then another 4 bytes for bits 33-64, and so on.

While a `NULL` itself does not require any storage space, `NDB` reserves 4 bytes per row if the table definition contains any columns defined as `NULL`, up to 32 `NULL` columns. (If an NDB Cluster table is defined with more than 32 `NULL` columns up to 64 `NULL` columns, then 8 bytes per row are reserved.)

Every table using the `NDB` storage engine requires a primary key; if you do not define a primary key, a “hidden” primary key is created by `NDB`. This hidden primary key consumes 31-35 bytes per table record.

You can use the `ndb_size.pl` Perl script to estimate `NDB` storage requirements. It connects to a current MySQL (not NDB Cluster) database and creates a report on how much space that database would require if it used the `NDB` storage engine. See Section 18.4.27, “`ndb_size.pl` — NDBCLUSTER Size Requirement Estimator” for more information.

Numeric Type Storage Requirements

Data Type	Storage Required
`TINYINT`	1 byte

Data Type	Storage Required
SMALLINT	2 bytes
MEDIUMINT	3 bytes
INT, INTEGER	4 bytes
BIGINT	8 bytes
FLOAT(*p*)	4 bytes if 0 <= *p* <= 24, 8 bytes if 25 <= *p* <= 53
FLOAT	4 bytes
DOUBLE [PRECISION], REAL	8 bytes
DECIMAL(*M*,*D*), NUMERIC(*M*,*D*)	Varies; see following discussion
BIT(*M*)	approximately (*M*+7)/8 bytes

Values for DECIMAL (and NUMERIC) columns are represented using a binary format that packs nine decimal (base 10) digits into four bytes. Storage for the integer and fractional parts of each value are determined separately. Each multiple of nine digits requires four bytes, and the “leftover” digits require some fraction of four bytes. The storage required for excess digits is given by the following table.

Leftover Digits	Number of Bytes
0	0
1	1
2	1
3	2
4	2
5	3
6	3
7	4
8	4

Date and Time Type Storage Requirements

For TIME, DATETIME, and TIMESTAMP columns, the storage required for tables created before MySQL 5.6.4 differs from tables created from 5.6.4 on. This is due to a change in 5.6.4 that permits these types to have a fractional part, which requires from 0 to 3 bytes.

Data Type	Storage Required Before MySQL 5.6.4	Storage Required as of MySQL 5.6.4
YEAR	1 byte	1 byte
DATE	3 bytes	3 bytes
TIME	3 bytes	3 bytes + fractional seconds storage
DATETIME	8 bytes	5 bytes + fractional seconds storage
TIMESTAMP	4 bytes	4 bytes + fractional seconds storage

As of MySQL 5.6.4, storage for YEAR and DATE remains unchanged. However, TIME, DATETIME, and TIMESTAMP are represented differently. DATETIME is packed more efficiently, requiring 5 rather than 8 bytes for the nonfractional part, and all three parts have a fractional part that requires from 0 to 3 bytes, depending on the fractional seconds precision of stored values.

Fractional Seconds Precision	Storage Required
0	0 bytes

Fractional Seconds Precision	Storage Required
1, 2	1 byte
3, 4	2 bytes
5, 6	3 bytes

For example, `TIME(0)`, `TIME(2)`, `TIME(4)`, and `TIME(6)` use 3, 4, 5, and 6 bytes, respectively. `TIME` and `TIME(0)` are equivalent and require the same storage.

For details about internal representation of temporal values, see MySQL Internals: Important Algorithms and Structures.

String Type Storage Requirements

In the following table, *M* represents the declared column length in characters for nonbinary string types and bytes for binary string types. *L* represents the actual length in bytes of a given string value.

Data Type	Storage Required
`CHAR(M)`	The compact family of InnoDB row formats optimize storage for variable-length character sets. See COMPACT Row Format Storage Characteristics. Otherwise, *M* × *w* bytes, <= *M* <= 255, where *w* is the number of bytes required for the maximum-length character in the character set.
`BINARY(M)`	*M* bytes, 0 <= *M* <= 255
`VARCHAR(M)`, `VARBINARY(M)`	*L* + 1 bytes if column values require 0 − 255 bytes, *L* + 2 bytes if values may require more than 255 bytes
`TINYBLOB`, `TINYTEXT`	*L* + 1 bytes, where *L* < 2^8
`BLOB`, `TEXT`	*L* + 2 bytes, where *L* < 2^{16}
`MEDIUMBLOB`, `MEDIUMTEXT`	*L* + 3 bytes, where *L* < 2^{24}
`LONGBLOB`, `LONGTEXT`	*L* + 4 bytes, where *L* < 2^{32}
`ENUM('value1','value2',...)`	1 or 2 bytes, depending on the number of enumeration values (65,535 values maximum)
`SET('value1','value2',...)`	1, 2, 3, 4, or 8 bytes, depending on the number of set members (64 members maximum)

Variable-length string types are stored using a length prefix plus data. The length prefix requires from one to four bytes depending on the data type, and the value of the prefix is *L* (the byte length of the string). For example, storage for a `MEDIUMTEXT` value requires *L* bytes to store the value plus three bytes to store the length of the value.

To calculate the number of bytes used to store a particular `CHAR`, `VARCHAR`, or `TEXT` column value, you must take into account the character set used for that column and whether the value contains multibyte characters. In particular, when using a `utf8` Unicode character set, you must keep in mind that not all characters use the same number of bytes. `utf8mb3` and `utf8mb4` character sets can require up to three and four bytes per character, respectively. For a breakdown of the storage used for different categories of `utf8mb3` or `utf8mb4` characters, see Section 10.9, “Unicode Support”.

`VARCHAR`, `VARBINARY`, and the `BLOB` and `TEXT` types are variable-length types. For each, the storage requirements depend on these factors:

- The actual length of the column value
- The column's maximum possible length
- The character set used for the column, because some character sets contain multibyte characters

For example, a VARCHAR(255) column can hold a string with a maximum length of 255 characters. Assuming that the column uses the latin1 character set (one byte per character), the actual storage required is the length of the string (*L*), plus one byte to record the length of the string. For the string 'abcd', *L* is 4 and the storage requirement is five bytes. If the same column is instead declared to use the ucs2 double-byte character set, the storage requirement is 10 bytes: The length of 'abcd' is eight bytes and the column requires two bytes to store lengths because the maximum length is greater than 255 (up to 510 bytes).

The effective maximum number of *bytes* that can be stored in a VARCHAR or VARBINARY column is subject to the maximum row size of 65,535 bytes, which is shared among all columns. For a VARCHAR column that stores multibyte characters, the effective maximum number of *characters* is less. For example, utf8mb3 characters can require up to three bytes per character, so a VARCHAR column that uses the utf8mb3 character set can be declared to be a maximum of 21,844 characters. See Section 8.4.7, “Limits on Table Column Count and Row Size”.

InnoDB encodes fixed-length fields greater than or equal to 768 bytes in length as variable-length fields, which can be stored off-page. For example, a CHAR(255) column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with utf8mb4.

The NDB storage engine supports variable-width columns. This means that a VARCHAR column in an NDB Cluster table requires the same amount of storage as would any other storage engine, with the exception that such values are 4-byte aligned. Thus, the string 'abcd' stored in a VARCHAR(50) column using the latin1 character set requires 8 bytes (rather than 5 bytes for the same column value in a MyISAM table).

TEXT and BLOB columns are implemented differently in the NDB storage engine, wherein each row in a TEXT column is made up of two separate parts. One of these is of fixed size (256 bytes), and is actually stored in the original table. The other consists of any data in excess of 256 bytes, which is stored in a hidden table. The rows in this second table are always 2000 bytes long. This means that the size of a TEXT column is 256 if *size* <= 256 (where *size* represents the size of the row); otherwise, the size is 256 + *size* + (2000 × (*size* − 256) % 2000).

The size of an ENUM object is determined by the number of different enumeration values. One byte is used for enumerations with up to 255 possible values. Two bytes are used for enumerations having between 256 and 65,535 possible values. See Section 11.3.5, “The ENUM Type”.

The size of a SET object is determined by the number of different set members. If the set size is *N*, the object occupies (*N*+7)/8 bytes, rounded up to 1, 2, 3, 4, or 8 bytes. A SET can have a maximum of 64 members. See Section 11.3.6, “The SET Type”.

Spatial Type Storage Requirements

MySQL stores geometry values using 4 bytes to indicate the SRID followed by the WKB representation of the value. The LENGTH() function returns the space in bytes required for value storage.

For descriptions of WKB and internal storage formats for spatial values, see Section 11.4.3, “Supported Spatial Data Formats”.

11.7 Choosing the Right Type for a Column

For optimum storage, you should try to use the most precise type in all cases. For example, if an integer column is used for values in the range from 1 to 99999, MEDIUMINT UNSIGNED is the best type. Of the types that represent all the required values, this type uses the least amount of storage.

All basic calculations (+, -, *, and /) with DECIMAL columns are done with precision of 65 decimal (base 10) digits. See Section 11.1.1, “Numeric Data Type Syntax”.

If accuracy is not too important or if speed is the highest priority, the DOUBLE type may be good enough. For high precision, you can always convert to a fixed-point type stored in a BIGINT. This

enables you to do all calculations with 64-bit integers and then convert results back to floating-point values as necessary.

`PROCEDURE ANALYSE` can be used to obtain suggestions for optimal column data types. For more information, see Section 8.4.2.4, “Using PROCEDURE ANALYSE”.

11.8 Using Data Types from Other Database Engines

To facilitate the use of code written for SQL implementations from other vendors, MySQL maps data types as shown in the following table. These mappings make it easier to import table definitions from other database systems into MySQL.

Other Vendor Type	MySQL Type
`BOOL`	`TINYINT`
`BOOLEAN`	`TINYINT`
`CHARACTER VARYING(M)`	`VARCHAR(M)`
`FIXED`	`DECIMAL`
`FLOAT4`	`FLOAT`
`FLOAT8`	`DOUBLE`
`INT1`	`TINYINT`
`INT2`	`SMALLINT`
`INT3`	`MEDIUMINT`
`INT4`	`INT`
`INT8`	`BIGINT`
`LONG VARBINARY`	`MEDIUMBLOB`
`LONG VARCHAR`	`MEDIUMTEXT`
`LONG`	`MEDIUMTEXT`
`MIDDLEINT`	`MEDIUMINT`
`NUMERIC`	`DECIMAL`

Data type mapping occurs at table creation time, after which the original type specifications are discarded. If you create a table with types used by other vendors and then issue a `DESCRIBE tbl_name` statement, MySQL reports the table structure using the equivalent MySQL types. For example:

```
mysql> CREATE TABLE t (a BOOL, b FLOAT8, c LONG VARCHAR, d NUMERIC);
Query OK, 0 rows affected (0.00 sec)

mysql> DESCRIBE t;
+-------+---------------+------+-----+---------+-------+
| Field | Type          | Null | Key | Default | Extra |
+-------+---------------+------+-----+---------+-------+
| a     | tinyint(1)    | YES  |     | NULL    |       |
| b     | double        | YES  |     | NULL    |       |
| c     | mediumtext    | YES  |     | NULL    |       |
| d     | decimal(10,0) | YES  |     | NULL    |       |
+-------+---------------+------+-----+---------+-------+
4 rows in set (0.01 sec)
```

Chapter 12 Functions and Operators

Table of Contents

12.1 SQL Function and Operator Reference 1310
12.2 User-Defined Function Reference 1320
12.3 Type Conversion in Expression Evaluation 1321
12.4 Operators 1324
12.4.1 Operator Precedence 1325
12.4.2 Comparison Functions and Operators 1326
12.4.3 Logical Operators 1332
12.4.4 Assignment Operators 1333
12.5 Flow Control Functions 1334
12.6 Numeric Functions and Operators 1336
12.6.1 Arithmetic Operators 1337
12.6.2 Mathematical Functions 1339
12.7 Date and Time Functions 1348
12.8 String Functions and Operators 1368
12.8.1 String Comparison Functions and Operators 1383
12.8.2 Regular Expressions 1386
12.8.3 Character Set and Collation of Function Results 1392
12.9 What Calendar Is Used By MySQL? 1392
12.10 Full-Text Search Functions 1393
12.10.1 Natural Language Full-Text Searches 1394
12.10.2 Boolean Full-Text Searches 1397
12.10.3 Full-Text Searches with Query Expansion 1402
12.10.4 Full-Text Stopwords 1403
12.10.5 Full-Text Restrictions 1407
12.10.6 Fine-Tuning MySQL Full-Text Search 1408
12.10.7 Adding a User-Defined Collation for Full-Text Indexing 1411
12.11 Cast Functions and Operators 1412
12.12 XML Functions 1417
12.13 Bit Functions and Operators 1428
12.14 Encryption and Compression Functions 1429
12.15 Locking Functions 1438
12.16 Information Functions 1439
12.17 Spatial Analysis Functions 1449
12.17.1 Spatial Function Reference 1449
12.17.2 Argument Handling by Spatial Functions 1452
12.17.3 Functions That Create Geometry Values from WKT Values 1453
12.17.4 Functions That Create Geometry Values from WKB Values 1454
12.17.5 MySQL-Specific Functions That Create Geometry Values 1456
12.17.6 Geometry Format Conversion Functions 1457
12.17.7 Geometry Property Functions 1457
12.17.8 Spatial Operator Functions 1465
12.17.9 Functions That Test Spatial Relations Between Geometry Objects 1466
12.18 Functions Used with Global Transaction IDs 1470
12.19 Aggregate Functions 1471
12.19.1 Aggregate Function Descriptions 1471
12.19.2 GROUP BY Modifiers 1476
12.19.3 MySQL Handling of GROUP BY 1479
12.20 Miscellaneous Functions 1480
12.21 Precision Math 1486
12.21.1 Types of Numeric Values 1487
12.21.2 DECIMAL Data Type Characteristics 1487
12.21.3 Expression Handling 1488

12.21.4 Rounding Behavior 1490
12.21.5 Precision Math Examples 1491

Expressions can be used at several points in SQL statements, such as in the `ORDER BY` or `HAVING` clauses of `SELECT` statements, in the `WHERE` clause of a `SELECT`, `DELETE`, or `UPDATE` statement, or in `SET` statements. Expressions can be written using literal values, column values, `NULL`, built-in functions, stored functions, user-defined functions, and operators. This chapter describes the SQL functions and operators that are permitted for writing expressions in MySQL. Instructions for writing stored functions and user-defined functions are given in Section 20.2, “Using Stored Routines”, and Adding Functions to MySQL. See Section 9.2.5, “Function Name Parsing and Resolution”, for the rules describing how the server interprets references to different kinds of functions.

An expression that contains `NULL` always produces a `NULL` value unless otherwise indicated in the documentation for a particular function or operator.

Note

By default, there must be no whitespace between a function name and the parenthesis following it. This helps the MySQL parser distinguish between function calls and references to tables or columns that happen to have the same name as a function. However, spaces around function arguments are permitted.

You can tell the MySQL server to accept spaces after function names by starting it with the `--sql-mode=IGNORE_SPACE` option. (See Section 5.1.10, “Server SQL Modes”.) Individual client programs can request this behavior by using the `CLIENT_IGNORE_SPACE` option for `mysql_real_connect()`. In either case, all function names become reserved words.

For the sake of brevity, most examples in this chapter display the output from the `mysql` program in abbreviated form. Rather than showing examples in this format:

```
mysql> SELECT MOD(29,9);
+-----------+
| mod(29,9) |
+-----------+
|         2 |
+-----------+
1 rows in set (0.00 sec)
```

This format is used instead:

```
mysql> SELECT MOD(29,9);
        -> 2
```

12.1 SQL Function and Operator Reference

The following table lists each SQL function and operator and provides a short description of each one. For a table listing user-defined functions, see Section 12.2, “User-Defined Function Reference”.

Table 12.1 SQL Functions and Operators

Name	Description
`&`	Bitwise AND
`>`	Greater than operator
`>>`	Right shift
`>=`	Greater than or equal operator
`<`	Less than operator
`<>, !=`	Not equal operator
`<<`	Left shift
`<=`	Less than or equal operator

Name	Description
<=>	NULL-safe equal to operator
%, MOD	Modulo operator
*	Multiplication operator
+	Addition operator
-	Minus operator
-	Change the sign of the argument
/	Division operator
:=	Assign a value
=	Assign a value (as part of a SET statement, or as part of the SET clause in an UPDATE statement)
=	Equal operator
^	Bitwise XOR
ABS()	Return the absolute value
ACOS()	Return the arc cosine
ADDDATE()	Add time values (intervals) to a date value
ADDTIME()	Add time
AES_DECRYPT()	Decrypt using AES
AES_ENCRYPT()	Encrypt using AES
AND, &&	Logical AND
Area()	Return Polygon or MultiPolygon area
AsBinary(), AsWKB()	Convert from internal geometry format to WKB
ASCII()	Return numeric value of left-most character
ASIN()	Return the arc sine
AsText(), AsWKT()	Convert from internal geometry format to WKT
ATAN()	Return the arc tangent
ATAN2(), ATAN()	Return the arc tangent of the two arguments
AVG()	Return the average value of the argument
BENCHMARK()	Repeatedly execute an expression
BETWEEN ... AND ...	Whether a value is within a range of values
BIN()	Return a string containing binary representation of a number
BINARY	Cast a string to a binary string
BIT_AND()	Return bitwise AND
BIT_COUNT()	Return the number of bits that are set
BIT_LENGTH()	Return length of argument in bits
BIT_OR()	Return bitwise OR
BIT_XOR()	Return bitwise XOR
Buffer()	Return geometry of points within given distance from geometry
CASE	Case operator
CAST()	Cast a value as a certain type
CEIL()	Return the smallest integer value not less than the argument

Name	Description
CEILING()	Return the smallest integer value not less than the argument
Centroid()	Return centroid as a point
CHAR()	Return the character for each integer passed
CHAR_LENGTH()	Return number of characters in argument
CHARACTER_LENGTH()	Synonym for CHAR_LENGTH()
CHARSET()	Return the character set of the argument
COALESCE()	Return the first non-NULL argument
COERCIBILITY()	Return the collation coercibility value of the string argument
COLLATION()	Return the collation of the string argument
COMPRESS()	Return result as a binary string
CONCAT()	Return concatenated string
CONCAT_WS()	Return concatenate with separator
CONNECTION_ID()	Return the connection ID (thread ID) for the connection
Contains()	Whether MBR of one geometry contains MBR of another
CONV()	Convert numbers between different number bases
CONVERT()	Cast a value as a certain type
CONVERT_TZ()	Convert from one time zone to another
COS()	Return the cosine
COT()	Return the cotangent
COUNT()	Return a count of the number of rows returned
COUNT(DISTINCT)	Return the count of a number of different values
CRC32()	Compute a cyclic redundancy check value
Crosses()	Whether one geometry crosses another
CURDATE()	Return the current date
CURRENT_DATE(), CURRENT_DATE	Synonyms for CURDATE()
CURRENT_TIME(), CURRENT_TIME	Synonyms for CURTIME()
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP	Synonyms for NOW()
CURRENT_USER(), CURRENT_USER	The authenticated user name and host name
CURTIME()	Return the current time
DATABASE()	Return the default (current) database name
DATE()	Extract the date part of a date or datetime expression
DATE_ADD()	Add time values (intervals) to a date value
DATE_FORMAT()	Format date as specified
DATE_SUB()	Subtract a time value (interval) from a date
DATEDIFF()	Subtract two dates
DAY()	Synonym for DAYOFMONTH()
DAYNAME()	Return the name of the weekday
DAYOFMONTH()	Return the day of the month (0-31)
DAYOFWEEK()	Return the weekday index of the argument
DAYOFYEAR()	Return the day of the year (1-366)

Name	Description
DECODE()	Decode a string encrypted using ENCODE()
DEFAULT()	Return the default value for a table column
DEGREES()	Convert radians to degrees
DES_DECRYPT()	Decrypt a string
DES_ENCRYPT()	Encrypt a string
Dimension()	Dimension of geometry
Disjoint()	Whether MBRs of two geometries are disjoint
DIV	Integer division
ELT()	Return string at index number
ENCODE()	Encode a string
ENCRYPT()	Encrypt a string
EndPoint()	End Point of LineString
Envelope()	Return MBR of geometry
Equals()	Whether MBRs of two geometries are equal
EXP()	Raise to the power of
EXPORT_SET()	Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
ExteriorRing()	Return exterior ring of Polygon
EXTRACT()	Extract part of a date
ExtractValue()	Extract a value from an XML string using XPath notation
FIELD()	Index (position) of first argument in subsequent arguments
FIND_IN_SET()	Index (position) of first argument within second argument
FLOOR()	Return the largest integer value not greater than the argument
FORMAT()	Return a number formatted to specified number of decimal places
FOUND_ROWS()	For a SELECT with a LIMIT clause, the number of rows that would be returned were there no LIMIT clause
FROM_BASE64()	Decode base64 encoded string and return result
FROM_DAYS()	Convert a day number to a date
FROM_UNIXTIME()	Format Unix timestamp as a date
GeomCollFromText(), GeometryCollectionFromText()	Return geometry collection from WKT
GeomCollFromWKB(), GeometryCollectionFromWKB()	Return geometry collection from WKB
GeometryCollection()	Construct geometry collection from geometries
GeometryN()	Return N-th geometry from geometry collection
GeometryType()	Return name of geometry type
GeomFromText(), GeometryFromText()	Return geometry from WKT
GeomFromWKB(), GeometryFromWKB()	Return geometry from WKB

Name	Description
GET_FORMAT()	Return a date format string
GET_LOCK()	Get a named lock
GLength()	Return length of LineString
GREATEST()	Return the largest argument
GROUP_CONCAT()	Return a concatenated string
GTID_SUBSET()	Return true if all GTIDs in subset are also in set; otherwise false.
GTID_SUBTRACT()	Return all GTIDs in set that are not in subset.
HEX()	Hexadecimal representation of decimal or string value
HOUR()	Extract the hour
IF()	If/else construct
IFNULL()	Null if/else construct
IN()	Whether a value is within a set of values
INET_ATON()	Return the numeric value of an IP address
INET_NTOA()	Return the IP address from a numeric value
INET6_ATON()	Return the numeric value of an IPv6 address
INET6_NTOA()	Return the IPv6 address from a numeric value
INSERT()	Insert substring at specified position up to specified number of characters
INSTR()	Return the index of the first occurrence of substring
InteriorRingN()	Return N-th interior ring of Polygon
Intersects()	Whether MBRs of two geometries intersect
INTERVAL()	Return the index of the argument that is less than the first argument
IS	Test a value against a boolean
IS_FREE_LOCK()	Whether the named lock is free
IS_IPV4()	Whether argument is an IPv4 address
IS_IPV4_COMPAT()	Whether argument is an IPv4-compatible address
IS_IPV4_MAPPED()	Whether argument is an IPv4-mapped address
IS_IPV6()	Whether argument is an IPv6 address
IS NOT	Test a value against a boolean
IS NOT NULL	NOT NULL value test
IS NULL	NULL value test
IS_USED_LOCK()	Whether the named lock is in use; return connection identifier if true
IsClosed()	Whether a geometry is closed and simple
IsEmpty()	Whether a geometry is empty
ISNULL()	Test whether the argument is NULL
IsSimple()	Whether a geometry is simple
LAST_DAY	Return the last day of the month for the argument
LAST_INSERT_ID()	Value of the AUTOINCREMENT column for the last INSERT
LCASE()	Synonym for LOWER()

Name	Description
LEAST()	Return the smallest argument
LEFT()	Return the leftmost number of characters as specified
LENGTH()	Return the length of a string in bytes
LIKE	Simple pattern matching
LineFromText(), LineStringFromText()	Construct LineString from WKT
LineFromWKB(), LineStringFromWKB()	Construct LineString from WKB
LineString()	Construct LineString from Point values
LN()	Return the natural logarithm of the argument
LOAD_FILE()	Load the named file
LOCALTIME(), LOCALTIME	Synonym for NOW()
LOCALTIMESTAMP, LOCALTIMESTAMP()	Synonym for NOW()
LOCATE()	Return the position of the first occurrence of substring
LOG()	Return the natural logarithm of the first argument
LOG10()	Return the base-10 logarithm of the argument
LOG2()	Return the base-2 logarithm of the argument
LOWER()	Return the argument in lowercase
LPAD()	Return the string argument, left-padded with the specified string
LTRIM()	Remove leading spaces
MAKE_SET()	Return a set of comma-separated strings that have the corresponding bit in bits set
MAKEDATE()	Create a date from the year and day of year
MAKETIME()	Create time from hour, minute, second
MASTER_POS_WAIT()	Block until the replica has read and applied all updates up to the specified position
MATCH	Perform full-text search
MAX()	Return the maximum value
MBRContains()	Whether MBR of one geometry contains MBR of another
MBRDisjoint()	Whether MBRs of two geometries are disjoint
MBREqual()	Whether MBRs of two geometries are equal
MBRIntersects()	Whether MBRs of two geometries intersect
MBROverlaps()	Whether MBRs of two geometries overlap
MBRTouches()	Whether MBRs of two geometries touch
MBRWithin()	Whether MBR of one geometry is within MBR of another
MD5()	Calculate MD5 checksum
MICROSECOND()	Return the microseconds from argument
MID()	Return a substring starting from the specified position
MIN()	Return the minimum value
MINUTE()	Return the minute from the argument

Name	Description
MLineFromText(), MultiLineStringFromText()	Construct MultiLineString from WKT
MLineFromWKB(), MultiLineStringFromWKB()	Construct MultiLineString from WKB
MOD()	Return the remainder
MONTH()	Return the month from the date passed
MONTHNAME()	Return the name of the month
MPointFromText(), MultiPointFromText()	Construct MultiPoint from WKT
MPointFromWKB(), MultiPointFromWKB()	Construct MultiPoint from WKB
MPolyFromText(), MultiPolygonFromText()	Construct MultiPolygon from WKT
MPolyFromWKB(), MultiPolygonFromWKB()	Construct MultiPolygon from WKB
MultiLineString()	Contruct MultiLineString from LineString values
MultiPoint()	Construct MultiPoint from Point values
MultiPolygon()	Construct MultiPolygon from Polygon values
NAME_CONST()	Cause the column to have the given name
NOT, !	Negates value
NOT BETWEEN ... AND ...	Whether a value is not within a range of values
NOT IN()	Whether a value is not within a set of values
NOT LIKE	Negation of simple pattern matching
NOT REGEXP	Negation of REGEXP
NOW()	Return the current date and time
NULLIF()	Return NULL if expr1 = expr2
NumGeometries()	Return number of geometries in geometry collection
NumInteriorRings()	Return number of interior rings in Polygon
NumPoints()	Return number of points in LineString
OCT()	Return a string containing octal representation of a number
OCTET_LENGTH()	Synonym for LENGTH()
OLD_PASSWORD() (deprecated)	Return the value of the pre-4.1 implementation of PASSWORD
OR, ||	Logical OR
ORD()	Return character code for leftmost character of the argument
Overlaps()	Whether MBRs of two geometries overlap
PASSWORD()	Calculate and return a password string
PERIOD_ADD()	Add a period to a year-month
PERIOD_DIFF()	Return the number of months between periods
PI()	Return the value of pi
Point()	Construct Point from coordinates
PointFromText()	Construct Point from WKT
PointFromWKB()	Construct Point from WKB

Name	Description
PointN()	Return N-th point from LineString
PolyFromText(), PolygonFromText()	Construct Polygon from WKT
PolyFromWKB(), PolygonFromWKB()	Construct Polygon from WKB
Polygon()	Construct Polygon from LineString arguments
POSITION()	Synonym for LOCATE()
POW()	Return the argument raised to the specified power
POWER()	Return the argument raised to the specified power
PROCEDURE ANALYSE()	Analyze the results of a query
QUARTER()	Return the quarter from a date argument
QUOTE()	Escape the argument for use in an SQL statement
RADIANS()	Return argument converted to radians
RAND()	Return a random floating-point value
RANDOM_BYTES() (introduced 5.6.17)	Return a random byte vector
REGEXP	Whether string matches regular expression
RELEASE_LOCK()	Release the named lock
REPEAT()	Repeat a string the specified number of times
REPLACE()	Replace occurrences of a specified string
REVERSE()	Reverse the characters in a string
RIGHT()	Return the specified rightmost number of characters
RLIKE	Whether string matches regular expression
ROUND()	Round the argument
ROW_COUNT()	The number of rows updated
RPAD()	Append string the specified number of times
RTRIM()	Remove trailing spaces
SCHEMA()	Synonym for DATABASE()
SEC_TO_TIME()	Converts seconds to 'hh:mm:ss' format
SECOND()	Return the second (0-59)
SESSION_USER()	Synonym for USER()
SHA1(), SHA()	Calculate an SHA-1 160-bit checksum
SHA2()	Calculate an SHA-2 checksum
SIGN()	Return the sign of the argument
SIN()	Return the sine of the argument
SLEEP()	Sleep for a number of seconds
SOUNDEX()	Return a soundex string
SOUNDS LIKE	Compare sounds
SPACE()	Return a string of the specified number of spaces
SQL_THREAD_WAIT_AFTER_GTIDS() (deprecated)	Obsolete. Use WAIT_FOR_EXECUTED_GTID_SET().
SQRT()	Return the square root of the argument
SRID()	Return spatial reference system ID for geometry

Name	Description
ST_Area()	Return Polygon or MultiPolygon area
ST_AsBinary(), ST_AsWKB()	Convert from internal geometry format to WKB
ST_AsText(), ST_AsWKT()	Convert from internal geometry format to WKT
ST_Buffer()	Return geometry of points within given distance from geometry
ST_Centroid()	Return centroid as a point
ST_Contains()	Whether one geometry contains another
ST_Crosses()	Whether one geometry crosses another
ST_Difference()	Return point set difference of two geometries
ST_Dimension()	Dimension of geometry
ST_Disjoint()	Whether one geometry is disjoint from another
ST_Distance()	The distance of one geometry from another
ST_EndPoint()	End Point of LineString
ST_Envelope()	Return MBR of geometry
ST_Equals()	Whether one geometry is equal to another
ST_ExteriorRing()	Return exterior ring of Polygon
ST_GeomCollFromText(), ST_GeometryCollectionFromText(), ST_GeomCollFromTxt()	Return geometry collection from WKT
ST_GeomCollFromWKB(), ST_GeometryCollectionFromWKB()	Return geometry collection from WKB
ST_GeometryN()	Return N-th geometry from geometry collection
ST_GeometryType()	Return name of geometry type
ST_GeomFromText(), ST_GeometryFromText()	Return geometry from WKT
ST_GeomFromWKB(), ST_GeometryFromWKB()	Return geometry from WKB
ST_InteriorRingN()	Return N-th interior ring of Polygon
ST_Intersection()	Return point set intersection of two geometries
ST_Intersects()	Whether one geometry intersects another
ST_IsClosed()	Whether a geometry is closed and simple
ST_IsEmpty()	Whether a geometry is empty
ST_IsSimple()	Whether a geometry is simple
ST_LineFromText(), ST_LineStringFromText()	Construct LineString from WKT
ST_LineFromWKB(), ST_LineStringFromWKB()	Construct LineString from WKB
ST_NumGeometries()	Return number of geometries in geometry collection
ST_NumInteriorRing(), ST_NumInteriorRings()	Return number of interior rings in Polygon
ST_NumPoints()	Return number of points in LineString
ST_Overlaps()	Whether one geometry overlaps another
ST_PointFromText()	Construct Point from WKT

Name	Description
ST_PointFromWKB()	Construct Point from WKB
ST_PointN()	Return N-th point from LineString
ST_PolyFromText(), ST_PolygonFromText()	Construct Polygon from WKT
ST_PolyFromWKB(), ST_PolygonFromWKB()	Construct Polygon from WKB
ST_SRID()	Return spatial reference system ID for geometry
ST_StartPoint()	Start Point of LineString
ST_SymDifference()	Return point set symmetric difference of two geometries
ST_Touches()	Whether one geometry touches another
ST_Union()	Return point set union of two geometries
ST_Within()	Whether one geometry is within another
ST_X()	Return X coordinate of Point
ST_Y()	Return Y coordinate of Point
StartPoint()	Start Point of LineString
STD()	Return the population standard deviation
STDDEV()	Return the population standard deviation
STDDEV_POP()	Return the population standard deviation
STDDEV_SAMP()	Return the sample standard deviation
STR_TO_DATE()	Convert a string to a date
STRCMP()	Compare two strings
SUBDATE()	Synonym for DATE_SUB() when invoked with three arguments
SUBSTR()	Return the substring as specified
SUBSTRING()	Return the substring as specified
SUBSTRING_INDEX()	Return a substring from a string before the specified number of occurrences of the delimiter
SUBTIME()	Subtract times
SUM()	Return the sum
SYSDATE()	Return the time at which the function executes
SYSTEM_USER()	Synonym for USER()
TAN()	Return the tangent of the argument
TIME()	Extract the time portion of the expression passed
TIME_FORMAT()	Format as time
TIME_TO_SEC()	Return the argument converted to seconds
TIMEDIFF()	Subtract time
TIMESTAMP()	With a single argument, this function returns the date or datetime expression; with two arguments, the sum of the arguments
TIMESTAMPADD()	Add an interval to a datetime expression
TIMESTAMPDIFF()	Subtract an interval from a datetime expression
TO_BASE64()	Return the argument converted to a base-64 string

Name	Description
TO_DAYS()	Return the date argument converted to days
TO_SECONDS()	Return the date or datetime argument converted to seconds since Year 0
Touches()	Whether one geometry touches another
TRIM()	Remove leading and trailing spaces
TRUNCATE()	Truncate to specified number of decimal places
UCASE()	Synonym for UPPER()
UNCOMPRESS()	Uncompress a string compressed
UNCOMPRESSED_LENGTH()	Return the length of a string before compression
UNHEX()	Return a string containing hex representation of a number
UNIX_TIMESTAMP()	Return a Unix timestamp
UpdateXML()	Return replaced XML fragment
UPPER()	Convert to uppercase
USER()	The user name and host name provided by the client
UTC_DATE()	Return the current UTC date
UTC_TIME()	Return the current UTC time
UTC_TIMESTAMP()	Return the current UTC date and time
UUID()	Return a Universal Unique Identifier (UUID)
UUID_SHORT()	Return an integer-valued universal identifier
VALIDATE_PASSWORD_STRENGTH()	Determine strength of password
VALUES()	Define the values to be used during an INSERT
VAR_POP()	Return the population standard variance
VAR_SAMP()	Return the sample variance
VARIANCE()	Return the population standard variance
VERSION()	Return a string that indicates the MySQL server version
WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS()	Use WAIT_FOR_EXECUTED_GTID_SET().
WEEK()	Return the week number
WEEKDAY()	Return the weekday index
WEEKOFYEAR()	Return the calendar week of the date (1-53)
WEIGHT_STRING()	Return the weight string for a string
Within()	Whether MBR of one geometry is within MBR of another
X()	Return X coordinate of Point
XOR	Logical XOR
Y()	Return Y coordinate of Point
YEAR()	Return the year
YEARWEEK()	Return the year and week
|	Bitwise OR
~	Bitwise inversion

12.2 User-Defined Function Reference

The following table lists each user-defined function and provides a short description of each one. For a table listing SQL functions and operators, see Section 12.1, “SQL Function and Operator Reference”

For general information about user-defined functions, see Section 5.6, “MySQL Server User-Defined Functions”.

Table 12.2 User-Defined Functions

Name	Description
`asymmetric_decrypt()` (introduced 5.6.21)	Decrypt ciphertext using private or public key
`asymmetric_derive()` (introduced 5.6.21)	Derive symmetric key from asymmetric keys
`asymmetric_encrypt()` (introduced 5.6.21)	Encrypt cleartext using private or public key
`asymmetric_sign()` (introduced 5.6.21)	Generate signature from digest
`asymmetric_verify()` (introduced 5.6.21)	Verify that signature matches digest
`create_asymmetric_priv_key()` (introduced 5.6.21)	Create private key
`create_asymmetric_pub_key()` (introduced 5.6.21)	Create public key
`create_dh_parameters()` (introduced 5.6.21)	Generate shared DH secret
`create_digest()` (introduced 5.6.21)	Generate digest from string
`mysql_firewall_flush_status()`	Reset firewall status variables
`normalize_statement()`	Normalize SQL statement to digest form
`read_firewall_users()`	Update firewall account profile cache
`read_firewall_whitelist()`	Update firewall account profile recorded-statement cache
`set_firewall_mode()`	Establish firewall account profile operational mode

12.3 Type Conversion in Expression Evaluation

When an operator is used with operands of different types, type conversion occurs to make the operands compatible. Some conversions occur implicitly. For example, MySQL automatically converts strings to numbers as necessary, and vice versa.

```
mysql> SELECT 1+'1';
        -> 2
mysql> SELECT CONCAT(2,' test');
        -> '2 test'
```

It is also possible to convert a number to a string explicitly using the `CAST()` function. Conversion occurs implicitly with the `CONCAT()` function because it expects string arguments.

```
mysql> SELECT 38.8, CAST(38.8 AS CHAR);
        -> 38.8, '38.8'
mysql> SELECT 38.8, CONCAT(38.8);
        -> 38.8, '38.8'
```

See later in this section for information about the character set of implicit number-to-string conversions, and for modified rules that apply to `CREATE TABLE ... SELECT` statements.

The following rules describe how conversion occurs for comparison operations:

- If one or both arguments are `NULL`, the result of the comparison is `NULL`, except for the `NULL`-safe `<=>` equality comparison operator. For `NULL <=> NULL`, the result is true. No conversion is needed.
- If both arguments in a comparison operation are strings, they are compared as strings.
- If both arguments are integers, they are compared as integers.
- Hexadecimal values are treated as binary strings if not compared to a number.
- If one of the arguments is a `TIMESTAMP` or `DATETIME` column and the other argument is a constant, the constant is converted to a timestamp before the comparison is performed. This is done to be more ODBC-friendly. This is not done for the arguments to `IN()`. To be safe, always use complete datetime, date, or time strings when doing comparisons. For example, to achieve best results when using `BETWEEN` with date or time values, use `CAST()` to explicitly convert the values to the desired data type.

  A single-row subquery from a table or tables is not considered a constant. For example, if a subquery returns an integer to be compared to a `DATETIME` value, the comparison is done as two integers. The integer is not converted to a temporal value. To compare the operands as `DATETIME` values, use `CAST()` to explicitly convert the subquery value to `DATETIME`.
- If one of the arguments is a decimal value, comparison depends on the other argument. The arguments are compared as decimal values if the other argument is a decimal or integer value, or as floating-point values if the other argument is a floating-point value.
- In all other cases, the arguments are compared as floating-point (real) numbers. For example, a comparison of string and numeric operands takes place as a comparison of floating-point numbers.

For information about conversion of values from one temporal type to another, see Section 11.2.8, “Conversion Between Date and Time Types”.

The following examples illustrate conversion of strings to numbers for comparison operations:

```
mysql> SELECT 1 > '6x';
        -> 0
mysql> SELECT 7 > '6x';
        -> 1
mysql> SELECT 0 > 'x6';
        -> 0
mysql> SELECT 0 = 'x6';
        -> 1
```

For comparisons of a string column with a number, MySQL cannot use an index on the column to look up the value quickly. If *`str_col`* is an indexed string column, the index cannot be used when performing the lookup in the following statement:

```
SELECT * FROM tbl_name WHERE str_col=1;
```

The reason for this is that there are many different strings that may convert to the value `1`, such as `'1'`, `' 1'`, or `'1a'`.

Comparisons between floating-point numbers and large values of `INTEGER` type are approximate because the integer is converted to double-precision floating point before comparison, which is not capable of representing all 64-bit integers exactly. For example, the integer value 2^{53} + 1 is not representable as a float, and is rounded to 2^{53} or 2^{53} + 2 before a float comparison, depending on the platform.

To illustrate, only the first of the following comparisons compares equal values, but both comparisons return true (1):

```
mysql> SELECT '9223372036854775807' = 9223372036854775807;
        -> 1
mysql> SELECT '9223372036854775807' = 9223372036854775806;
        -> 1
```

When conversions from string to floating-point and from integer to floating-point occur, they do not necessarily occur the same way. The integer may be converted to floating-point by the CPU, whereas the string is converted digit by digit in an operation that involves floating-point multiplications. Also, results can be affected by factors such as computer architecture or the compiler version or optimization level. One way to avoid such problems is to use `CAST()` so that a value is not converted implicitly to a float-point number:

```
mysql> SELECT CAST('9223372036854775807' AS UNSIGNED) = 9223372036854775806;
        -> 0
```

For more information about floating-point comparisons, see Section B.3.4.8, “Problems with Floating-Point Values”.

The server includes `dtoa`, a conversion library that provides the basis for improved conversion between string or `DECIMAL` values and approximate-value (`FLOAT`/`DOUBLE`) numbers:

- Consistent conversion results across platforms, which eliminates, for example, Unix versus Windows conversion differences.
- Accurate representation of values in cases where results previously did not provide sufficient precision, such as for values close to IEEE limits.
- Conversion of numbers to string format with the best possible precision. The precision of `dtoa` is always the same or better than that of the standard C library functions.

Because the conversions produced by this library differ in some cases from non-`dtoa` results, the potential exists for incompatibilities in applications that rely on previous results. For example, applications that depend on a specific exact result from previous conversions might need adjustment to accommodate additional precision.

The `dtoa` library provides conversions with the following properties. *`D`* represents a value with a `DECIMAL` or string representation, and *`F`* represents a floating-point number in native binary (IEEE) format.

- *`F`* -> *`D`* conversion is done with the best possible precision, returning *`D`* as the shortest string that yields *`F`* when read back in and rounded to the nearest value in native binary format as specified by IEEE.
- *`D`* -> *`F`* conversion is done such that *`F`* is the nearest native binary number to the input decimal string *`D`*.

These properties imply that *`F`* -> *`D`* -> *`F`* conversions are lossless unless *`F`* is `-inf`, `+inf`, or `NaN`. The latter values are not supported because the SQL standard defines them as invalid values for `FLOAT` or `DOUBLE`.

For *`D`* -> *`F`* -> *`D`* conversions, a sufficient condition for losslessness is that *`D`* uses 15 or fewer digits of precision, is not a denormal value, `-inf`, `+inf`, or `NaN`. In some cases, the conversion is lossless even if *`D`* has more than 15 digits of precision, but this is not always the case.

Implicit conversion of a numeric or temporal value to string produces a value that has a character set and collation determined by the `character_set_connection` and `collation_connection` system variables. (These variables commonly are set with `SET NAMES`. For information about connection character sets, see Section 10.4, “Connection Character Sets and Collations”.)

This means that such a conversion results in a character (nonbinary) string (a `CHAR`, `VARCHAR`, or `LONGTEXT` value), except in the case that the connection character set is set to `binary`. In that case, the conversion result is a binary string (a `BINARY`, `VARBINARY`, or `LONGBLOB` value).

For integer expressions, the preceding remarks about expression *evaluation* apply somewhat differently for expression *assignment*; for example, in a statement such as this:

```
CREATE TABLE t SELECT integer_expr;
```

In this case, the table in the column resulting from the expression has type `INT` or `BIGINT` depending on the length of the integer expression. If the maximum length of the expression does not fit in an `INT`, `BIGINT` is used instead. The length is taken from the `max_length` value of the `SELECT` result set metadata (see C API Data Structures). This means that you can force a `BIGINT` rather than `INT` by use of a sufficiently long expression:

```
CREATE TABLE t SELECT 00000000000000000000;
```

12.4 Operators

Table 12.3 Operators

Name	Description
`&`	Bitwise AND
`>`	Greater than operator
`>>`	Right shift
`>=`	Greater than or equal operator
`<`	Less than operator
`<>`, `!=`	Not equal operator
`<<`	Left shift
`<=`	Less than or equal operator
`<=>`	NULL-safe equal to operator
`%`, `MOD`	Modulo operator
`*`	Multiplication operator
`+`	Addition operator
`-`	Minus operator
`-`	Change the sign of the argument
`/`	Division operator
`:=`	Assign a value
`=`	Assign a value (as part of a `SET` statement, or as part of the `SET` clause in an `UPDATE` statement)
`=`	Equal operator
`^`	Bitwise XOR
`AND`, `&&`	Logical AND
`BETWEEN ... AND ...`	Whether a value is within a range of values
`BINARY`	Cast a string to a binary string
`CASE`	Case operator
`DIV`	Integer division
`IN()`	Whether a value is within a set of values
`IS`	Test a value against a boolean
`IS NOT`	Test a value against a boolean
`IS NOT NULL`	NOT NULL value test
`IS NULL`	NULL value test
`LIKE`	Simple pattern matching
`NOT`, `!`	Negates value
`NOT BETWEEN ... AND ...`	Whether a value is not within a range of values

Name	Description
NOT IN()	Whether a value is not within a set of values
NOT LIKE	Negation of simple pattern matching
NOT REGEXP	Negation of REGEXP
OR, ||	Logical OR
REGEXP	Whether string matches regular expression
RLIKE	Whether string matches regular expression
SOUNDS LIKE	Compare sounds
XOR	Logical XOR
|	Bitwise OR
~	Bitwise inversion

12.4.1 Operator Precedence

Operator precedences are shown in the following list, from highest precedence to the lowest. Operators that are shown together on a line have the same precedence.

```
INTERVAL
BINARY, COLLATE
!
- (unary minus), ~ (unary bit inversion)
^
*, /, DIV, %, MOD
-, +
<<, >>
&
|
= (comparison), <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, IN
BETWEEN, CASE, WHEN, THEN, ELSE
NOT
AND, &&
XOR
OR, ||
= (assignment), :=
```

The precedence of = depends on whether it is used as a comparison operator (=) or as an assignment operator (=). When used as a comparison operator, it has the same precedence as <=>, >=, >, <=, <, <>, !=, IS, LIKE, REGEXP, and IN(). When used as an assignment operator, it has the same precedence as :=. Section 13.7.4.1, “SET Syntax for Variable Assignment”, and Section 9.4, “User-Defined Variables”, explain how MySQL determines which interpretation of = should apply.

For operators that occur at the same precedence level within an expression, evaluation proceeds left to right, with the exception that assignments evaluate right to left.

The precedence and meaning of some operators depends on the SQL mode:

- By default, || is a logical OR operator. With PIPES_AS_CONCAT enabled, || is string concatenation, with a precedence between ^ and the unary operators.
- By default, ! has a higher precedence than NOT. With HIGH_NOT_PRECEDENCE enabled, ! and NOT have the same precedence.

See Section 5.1.10, “Server SQL Modes”.

The precedence of operators determines the order of evaluation of terms in an expression. To override this order and group terms explicitly, use parentheses. For example:

```
mysql> SELECT 1+2*3;
        -> 7
```

```
mysql> SELECT (1+2)*3;
        -> 9
```

12.4.2 Comparison Functions and Operators

Table 12.4 Comparison Operators

Name	Description
`>`	Greater than operator
`>=`	Greater than or equal operator
`<`	Less than operator
`<>, !=`	Not equal operator
`<=`	Less than or equal operator
`<=>`	NULL-safe equal to operator
`=`	Equal operator
`BETWEEN ... AND ...`	Whether a value is within a range of values
`COALESCE()`	Return the first non-NULL argument
`GREATEST()`	Return the largest argument
`IN()`	Whether a value is within a set of values
`INTERVAL()`	Return the index of the argument that is less than the first argument
`IS`	Test a value against a boolean
`IS NOT`	Test a value against a boolean
`IS NOT NULL`	NOT NULL value test
`IS NULL`	NULL value test
`ISNULL()`	Test whether the argument is NULL
`LEAST()`	Return the smallest argument
`LIKE`	Simple pattern matching
`NOT BETWEEN ... AND ...`	Whether a value is not within a range of values
`NOT IN()`	Whether a value is not within a set of values
`NOT LIKE`	Negation of simple pattern matching
`STRCMP()`	Compare two strings

Comparison operations result in a value of `1` (`TRUE`), `0` (`FALSE`), or `NULL`. These operations work for both numbers and strings. Strings are automatically converted to numbers and numbers to strings as necessary.

The following relational comparison operators can be used to compare not only scalar operands, but row operands:

```
=  >  <  >=  <=  <>  !=
```

The descriptions for those operators later in this section detail how they work with row operands. For additional examples of row comparisons in the context of row subqueries, see Section 13.2.10.5, “Row Subqueries”.

Some of the functions in this section return values other than `1` (`TRUE`), `0` (`FALSE`), or `NULL`. `LEAST()` and `GREATEST()` are examples of such functions; Section 12.3, “Type Conversion in Expression Evaluation”, describes the rules for comparison operations performed by these and similar functions for determining their return values.

To convert a value to a specific type for comparison purposes, you can use the `CAST()` function. String values can be converted to a different character set using `CONVERT()`. See Section 12.11, “Cast Functions and Operators”.

By default, string comparisons are not case-sensitive and use the current character set. The default is `latin1` (cp1252 West European), which also works well for English.

- `=`

  Equal:

```
mysql> SELECT 1 = 0;
        -> 0
mysql> SELECT '0' = 0;
        -> 1
mysql> SELECT '0.0' = 0;
        -> 1
mysql> SELECT '0.01' = 0;
        -> 0
mysql> SELECT '.01' = 0.01;
        -> 1
```

  For row comparisons, `(a, b) = (x, y)` is equivalent to:

```
(a = x) AND (b = y)
```

- `<=>`

  `NULL`-safe equal. This operator performs an equality comparison like the `=` operator, but returns `1` rather than `NULL` if both operands are `NULL`, and `0` rather than `NULL` if one operand is `NULL`.

  The `<=>` operator is equivalent to the standard SQL `IS NOT DISTINCT FROM` operator.

```
mysql> SELECT 1 <=> 1, NULL <=> NULL, 1 <=> NULL;
        -> 1, 1, 0
mysql> SELECT 1 = 1, NULL = NULL, 1 = NULL;
        -> 1, NULL, NULL
```

  For row comparisons, `(a, b) <=> (x, y)` is equivalent to:

```
(a <=> x) AND (b <=> y)
```

- `<>`, `!=`

  Not equal:

```
mysql> SELECT '.01' <> '0.01';
        -> 1
mysql> SELECT .01 <> '0.01';
        -> 0
mysql> SELECT 'zapp' <> 'zappp';
        -> 1
```

  For row comparisons, `(a, b) <> (x, y)` and `(a, b) != (x, y)` are equivalent to:

```
(a <> x) OR (b <> y)
```

- `<=`

  Less than or equal:

```
mysql> SELECT 0.1 <= 2;
        -> 1
```

  For row comparisons, `(a, b) <= (x, y)` is equivalent to:

```
(a < x) OR ((a = x) AND (b <= y))
```

- `<`

  Less than:

```
mysql> SELECT 2 < 2;
        -> 0
```

  For row comparisons, `(a, b) < (x, y)` is equivalent to:

```
(a < x) OR ((a = x) AND (b < y))
```

- `>=`

  Greater than or equal:

```
mysql> SELECT 2 >= 2;
        -> 1
```

  For row comparisons, `(a, b) >= (x, y)` is equivalent to:

```
(a > x) OR ((a = x) AND (b >= y))
```

- `>`

  Greater than:

```
mysql> SELECT 2 > 2;
        -> 0
```

  For row comparisons, `(a, b) > (x, y)` is equivalent to:

```
(a > x) OR ((a = x) AND (b > y))
```

- *`expr`* `BETWEEN` *`min`* `AND` *`max`*

  If *`expr`* is greater than or equal to *`min`* and *`expr`* is less than or equal to *`max`*, `BETWEEN` returns `1`, otherwise it returns `0`. This is equivalent to the expression `(`*`min`* `<=` *`expr`* `AND` *`expr`* `<=` *`max`*`)` if all the arguments are of the same type. Otherwise type conversion takes place according to the rules described in Section 12.3, “Type Conversion in Expression Evaluation”, but applied to all the three arguments.

```
mysql> SELECT 2 BETWEEN 1 AND 3, 2 BETWEEN 3 and 1;
        -> 1, 0
mysql> SELECT 1 BETWEEN 2 AND 3;
        -> 0
mysql> SELECT 'b' BETWEEN 'a' AND 'c';
        -> 1
mysql> SELECT 2 BETWEEN 2 AND '3';
        -> 1
mysql> SELECT 2 BETWEEN 2 AND 'x-3';
        -> 0
```

  For best results when using `BETWEEN` with date or time values, use `CAST()` to explicitly convert the values to the desired data type. Examples: If you compare a `DATETIME` to two `DATE` values, convert the `DATE` values to `DATETIME` values. If you use a string constant such as `'2001-1-1'` in a comparison to a `DATE`, cast the string to a `DATE`.

- *`expr`* `NOT BETWEEN` *`min`* `AND` *`max`*

  This is the same as `NOT (`*`expr`* `BETWEEN` *`min`* `AND` *`max`*`)`.

- `COALESCE(`*`value`*`,...)`

  Returns the first non-`NULL` value in the list, or `NULL` if there are no non-`NULL` values.

  The return type of `COALESCE()` is the aggregated type of the argument types.

```
mysql> SELECT COALESCE(NULL,1);
        -> 1
mysql> SELECT COALESCE(NULL,NULL,NULL);
        -> NULL
```

- `GREATEST(value1,value2,...)`

  With two or more arguments, returns the largest (maximum-valued) argument. The arguments are compared using the same rules as for `LEAST()`.

```
mysql> SELECT GREATEST(2,0);
        -> 2
mysql> SELECT GREATEST(34.0,3.0,5.0,767.0);
        -> 767.0
mysql> SELECT GREATEST('B','A','C');
        -> 'C'
```

  `GREATEST()` returns `NULL` if any argument is `NULL`.

- *`expr`* `IN (value,...)`

  Returns `1` (true) if *`expr`* is equal to any of the values in the `IN()` list, else returns `0` (false).

  Type conversion takes place according to the rules described in Section 12.3, “Type Conversion in Expression Evaluation”, applied to all the arguments. If no type conversion is needed for the values in the `IN()` list, they are all constants of the same type, and *`expr`* can be compared to each of them as a value of the same type (possibly after type conversion), an optimization takes place. The values the list are sorted and the search for *`expr`* is done using a binary search, which makes the `IN()` operation very quick.

```
mysql> SELECT 2 IN (0,3,5,7);
        -> 0
mysql> SELECT 'wefwf' IN ('wee','wefwf','weg');
        -> 1
```

  `IN()` can be used to compare row constructors:

```
mysql> SELECT (3,4) IN ((1,2), (3,4));
        -> 1
mysql> SELECT (3,4) IN ((1,2), (3,5));
        -> 0
```

  You should never mix quoted and unquoted values in an `IN()` list because the comparison rules for quoted values (such as strings) and unquoted values (such as numbers) differ. Mixing types may therefore lead to inconsistent results. For example, do not write an `IN()` expression like this:

```
SELECT val1 FROM tbl1 WHERE val1 IN (1,2,'a');
```

  Instead, write it like this:

```
SELECT val1 FROM tbl1 WHERE val1 IN ('1','2','a');
```

  Implicit type conversion may produce nonintuitive results:

```
mysql> SELECT 'a' IN (0), 0 IN ('b');
```

```
        -> 1, 1
```

In both cases, the comparison values are converted to floating-point values, yielding 0.0 in each case, and a comparison result of 1 (true).

The number of values in the `IN()` list is only limited by the `max_allowed_packet` value.

To comply with the SQL standard, `IN()` returns `NULL` not only if the expression on the left hand side is `NULL`, but also if no match is found in the list and one of the expressions in the list is `NULL`.

`IN()` syntax can also be used to write certain types of subqueries. See Section 13.2.10.3, “Subqueries with ANY, IN, or SOME”.

- *`expr`* `NOT IN (`*`value`*`,...)`

  This is the same as `NOT (`*`expr`* `IN (`*`value`*`,...))`.

- `INTERVAL(`*`N`*`,`*`N1`*`,`*`N2`*`,`*`N3`*`,...)`

  Returns `0` if *`N`* < *`N1`*, `1` if *`N`* < *`N2`* and so on or `-1` if *`N`* is `NULL`. All arguments are treated as integers. It is required that *`N1`* < *`N2`* < *`N3`* < `...` < *`Nn`* for this function to work correctly. This is because a binary search is used (very fast).

```
mysql> SELECT INTERVAL(23, 1, 15, 17, 30, 44, 200);
        -> 3
mysql> SELECT INTERVAL(10, 1, 10, 100, 1000);
        -> 2
mysql> SELECT INTERVAL(22, 23, 30, 44, 200);
        -> 0
```

- `IS` *`boolean_value`*

  Tests a value against a boolean value, where *`boolean_value`* can be `TRUE`, `FALSE`, or `UNKNOWN`.

```
mysql> SELECT 1 IS TRUE, 0 IS FALSE, NULL IS UNKNOWN;
        -> 1, 1, 1
```

- `IS NOT` *`boolean_value`*

  Tests a value against a boolean value, where *`boolean_value`* can be `TRUE`, `FALSE`, or `UNKNOWN`.

```
mysql> SELECT 1 IS NOT UNKNOWN, 0 IS NOT UNKNOWN, NULL IS NOT UNKNOWN;
        -> 1, 1, 0
```

- `IS NULL`

  Tests whether a value is `NULL`.

```
mysql> SELECT 1 IS NULL, 0 IS NULL, NULL IS NULL;
        -> 0, 0, 1
```

To work well with ODBC programs, MySQL supports the following extra features when using `IS NULL`:

- If `sql_auto_is_null` variable is set to 1, then after a statement that successfully inserts an automatically generated `AUTO_INCREMENT` value, you can find that value by issuing a statement of the following form:

  ```
  SELECT * FROM tbl_name WHERE auto_col IS NULL
  ```

  If the statement returns a row, the value returned is the same as if you invoked the `LAST_INSERT_ID()` function. For details, including the return value after a multiple-row insert,

see Section 12.16, “Information Functions”. If no AUTO_INCREMENT value was successfully inserted, the SELECT statement returns no row.

The behavior of retrieving an AUTO_INCREMENT value by using an IS NULL comparison can be disabled by setting sql_auto_is_null = 0. See Section 5.1.7, “Server System Variables”.

The default value of sql_auto_is_null is 0.

- For DATE and DATETIME columns that are declared as NOT NULL, you can find the special date '0000-00-00' by using a statement like this:

```
SELECT * FROM tbl_name WHERE date_column IS NULL
```

This is needed to get some ODBC applications to work because ODBC does not support a '0000-00-00' date value.

See Obtaining Auto-Increment Values, and the description for the FLAG_AUTO_IS_NULL option at Connector/ODBC Connection Parameters.

- IS NOT NULL

Tests whether a value is not NULL.

```
mysql> SELECT 1 IS NOT NULL, 0 IS NOT NULL, NULL IS NOT NULL;
        -> 1, 1, 0
```

- ISNULL(*expr*)

If *expr* is NULL, ISNULL() returns 1, otherwise it returns 0.

```
mysql> SELECT ISNULL(1+1);
        -> 0
mysql> SELECT ISNULL(1/0);
        -> 1
```

ISNULL() can be used instead of = to test whether a value is NULL. (Comparing a value to NULL using = always yields NULL.)

The ISNULL() function shares some special behaviors with the IS NULL comparison operator. See the description of IS NULL.

- LEAST(*value1,value2,...*)

With two or more arguments, returns the smallest (minimum-valued) argument. The arguments are compared using the following rules:

- If any argument is NULL, the result is NULL. No comparison is needed.
- If all arguments are integer-valued, they are compared as integers.
- If at least one argument is double precision, they are compared as double-precision values. Otherwise, if at least one argument is a DECIMAL value, they are compared as DECIMAL values.
- If the arguments comprise a mix of numbers and strings, they are compared as numbers.
- If any argument is a nonbinary (character) string, the arguments are compared as nonbinary strings.
- In all other cases, the arguments are compared as binary strings.

The return type of LEAST() is the aggregated type of the comparison argument types.

```
mysql> SELECT LEAST(2,0);
        -> 0
```

```
mysql> SELECT LEAST(34.0,3.0,5.0,767.0);
        -> 3.0
mysql> SELECT LEAST('B','A','C');
        -> 'A'
```

12.4.3 Logical Operators

Table 12.5 Logical Operators

Name	Description
AND, &&	Logical AND
NOT, !	Negates value
OR, ||	Logical OR
XOR	Logical XOR

In SQL, all logical operators evaluate to TRUE, FALSE, or NULL (UNKNOWN). In MySQL, these are implemented as 1 (TRUE), 0 (FALSE), and NULL. Most of this is common to different SQL database servers, although some servers may return any nonzero value for TRUE.

MySQL evaluates any nonzero, non-NULL value to TRUE. For example, the following statements all assess to TRUE:

```
mysql> SELECT 10 IS TRUE;
-> 1
mysql> SELECT -10 IS TRUE;
-> 1
mysql> SELECT 'string' IS NOT NULL;
-> 1
```

- NOT, !

  Logical NOT. Evaluates to 1 if the operand is 0, to 0 if the operand is nonzero, and NOT NULL returns NULL.

  ```
  mysql> SELECT NOT 10;
          -> 0
  mysql> SELECT NOT 0;
          -> 1
  mysql> SELECT NOT NULL;
          -> NULL
  mysql> SELECT ! (1+1);
          -> 0
  mysql> SELECT ! 1+1;
          -> 1
  ```

  The last example produces 1 because the expression evaluates the same way as (!1)+1.

- AND, &&

  Logical AND. Evaluates to 1 if all operands are nonzero and not NULL, to 0 if one or more operands are 0, otherwise NULL is returned.

  ```
  mysql> SELECT 1 AND 1;
          -> 1
  mysql> SELECT 1 AND 0;
          -> 0
  mysql> SELECT 1 AND NULL;
          -> NULL
  mysql> SELECT 0 AND NULL;
          -> 0
  mysql> SELECT NULL AND 0;
          -> 0
  ```

- OR, ||

Logical OR. When both operands are non-`NULL`, the result is `1` if any operand is nonzero, and `0` otherwise. With a `NULL` operand, the result is `1` if the other operand is nonzero, and `NULL` otherwise. If both operands are `NULL`, the result is `NULL`.

```
mysql> SELECT 1 OR 1;
        -> 1
mysql> SELECT 1 OR 0;
        -> 1
mysql> SELECT 0 OR 0;
        -> 0
mysql> SELECT 0 OR NULL;
        -> NULL
mysql> SELECT 1 OR NULL;
        -> 1
```

Note

If the `PIPES_AS_CONCAT` SQL mode is enabled, `||` signifies the SQL-standard string concatenation operator (like `CONCAT()`).

- `XOR`

Logical XOR. Returns `NULL` if either operand is `NULL`. For non-`NULL` operands, evaluates to `1` if an odd number of operands is nonzero, otherwise `0` is returned.

```
mysql> SELECT 1 XOR 1;
        -> 0
mysql> SELECT 1 XOR 0;
        -> 1
mysql> SELECT 1 XOR NULL;
        -> NULL
mysql> SELECT 1 XOR 1 XOR 1;
        -> 1
```

`a XOR b` is mathematically equal to `(a AND (NOT b)) OR ((NOT a) and b)`.

12.4.4 Assignment Operators

Table 12.6 Assignment Operators

Name	Description
`:=`	Assign a value
`=`	Assign a value (as part of a `SET` statement, or as part of the `SET` clause in an `UPDATE` statement)

- `:=`

Assignment operator. Causes the user variable on the left hand side of the operator to take on the value to its right. The value on the right hand side may be a literal value, another variable storing a value, or any legal expression that yields a scalar value, including the result of a query (provided that this value is a scalar value). You can perform multiple assignments in the same `SET` statement. You can perform multiple assignments in the same statement.

Unlike `=`, the `:=` operator is never interpreted as a comparison operator. This means you can use `:=` in any valid SQL statement (not just in `SET` statements) to assign a value to a variable.

```
mysql> SELECT @var1, @var2;
        -> NULL, NULL
mysql> SELECT @var1 := 1, @var2;
        -> 1, NULL
mysql> SELECT @var1, @var2;
        -> 1, NULL
mysql> SELECT @var1, @var2 := @var1;
```

```
        -> 1, 1
mysql> SELECT @var1, @var2;
        -> 1, 1

mysql> SELECT @var1:=COUNT(*) FROM t1;
        -> 4
mysql> SELECT @var1;
        -> 4
```

You can make value assignments using `:=` in other statements besides `SELECT`, such as `UPDATE`, as shown here:

```
mysql> SELECT @var1;
        -> 4
mysql> SELECT * FROM t1;
        -> 1, 3, 5, 7

mysql> UPDATE t1 SET c1 = 2 WHERE c1 = @var1:= 1;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT @var1;
        -> 1
mysql> SELECT * FROM t1;
        -> 2, 3, 5, 7
```

While it is also possible both to set and to read the value of the same variable in a single SQL statement using the `:=` operator, this is not recommended. Section 9.4, “User-Defined Variables”, explains why you should avoid doing this.

- `=`

  This operator is used to perform value assignments in two cases, described in the next two paragraphs.

  Within a `SET` statement, `=` is treated as an assignment operator that causes the user variable on the left hand side of the operator to take on the value to its right. (In other words, when used in a `SET` statement, `=` is treated identically to `:=`.) The value on the right hand side may be a literal value, another variable storing a value, or any legal expression that yields a scalar value, including the result of a query (provided that this value is a scalar value). You can perform multiple assignments in the same `SET` statement.

  In the `SET` clause of an `UPDATE` statement, `=` also acts as an assignment operator; in this case, however, it causes the column named on the left hand side of the operator to assume the value given to the right, provided any `WHERE` conditions that are part of the `UPDATE` are met. You can make multiple assignments in the same `SET` clause of an `UPDATE` statement.

  In any other context, `=` is treated as a comparison operator.

```
mysql> SELECT @var1, @var2;
        -> NULL, NULL
mysql> SELECT @var1 := 1, @var2;
        -> 1, NULL
mysql> SELECT @var1, @var2;
        -> 1, NULL
mysql> SELECT @var1, @var2 := @var1;
        -> 1, 1
mysql> SELECT @var1, @var2;
        -> 1, 1
```

For more information, see Section 13.7.4.1, “SET Syntax for Variable Assignment”, Section 13.2.11, “UPDATE Statement”, and Section 13.2.10, “Subqueries”.

12.5 Flow Control Functions

Table 12.7 Flow Control Operators

Name	Description
CASE	Case operator
IF()	If/else construct
IFNULL()	Null if/else construct
NULLIF()	Return NULL if expr1 = expr2

- `CASE value WHEN compare_value THEN result [WHEN compare_value THEN result ...] [ELSE result] END`

  `CASE WHEN condition THEN result [WHEN condition THEN result ...] [ELSE result] END`

  The first `CASE` syntax returns the *`result`* for the first *`value=compare_value`* comparison that is true. The second syntax returns the result for the first condition that is true. If no comparison or condition is true, the result after `ELSE` is returned, or `NULL` if there is no `ELSE` part.

  Note

  The syntax of the `CASE` *operator* described here differs slightly from that of the SQL `CASE` *statement* described in Section 13.6.5.1, “CASE Statement”, for use inside stored programs. The `CASE` statement cannot have an `ELSE NULL` clause, and it is terminated with `END CASE` instead of `END`.

  The return type of a `CASE` expression result is the aggregated type of all result values.

```
mysql> SELECT CASE 1 WHEN 1 THEN 'one'
    ->     WHEN 2 THEN 'two' ELSE 'more' END;
        -> 'one'
mysql> SELECT CASE WHEN 1>0 THEN 'true' ELSE 'false' END;
        -> 'true'
mysql> SELECT CASE BINARY 'B'
    ->     WHEN 'a' THEN 1 WHEN 'b' THEN 2 END;
        -> NULL
```

- `IF(expr1,expr2,expr3)`

  If *`expr1`* is `TRUE` (*`expr1 <> 0`* and *`expr1 <> NULL`*), `IF()` returns *`expr2`*. Otherwise, it returns *`expr3`*.

  Note

  There is also an `IF` *statement*, which differs from the `IF()` *function* described here. See Section 13.6.5.2, “IF Statement”.

  If only one of *`expr2`* or *`expr3`* is explicitly `NULL`, the result type of the `IF()` function is the type of the non-`NULL` expression.

  The default return type of `IF()` (which may matter when it is stored into a temporary table) is calculated as follows:

  - If *`expr2`* or *`expr3`* produce a string, the result is a string.

    If *`expr2`* and *`expr3`* are both strings, the result is case-sensitive if either string is case-sensitive.

  - If *`expr2`* or *`expr3`* produce a floating-point value, the result is a floating-point value.

  - If *`expr2`* or *`expr3`* produce an integer, the result is an integer.

```
mysql> SELECT IF(1>2,2,3);
        -> 3
```

```
mysql> SELECT IF(1<2,'yes','no');
        -> 'yes'
mysql> SELECT IF(STRCMP('test','test1'),'no','yes');
        -> 'no'
```

- `IFNULL(expr1,expr2)`

  If *`expr1`* is not `NULL`, `IFNULL()` returns *`expr1`*; otherwise it returns *`expr2`*.

```
mysql> SELECT IFNULL(1,0);
        -> 1
mysql> SELECT IFNULL(NULL,10);
        -> 10
mysql> SELECT IFNULL(1/0,10);
        -> 10
mysql> SELECT IFNULL(1/0,'yes');
        -> 'yes'
```

  The default return type of `IFNULL(expr1,expr2)` is the more “general” of the two expressions, in the order `STRING`, `REAL`, or `INTEGER`. Consider the case of a table based on expressions or where MySQL must internally store a value returned by `IFNULL()` in a temporary table:

```
mysql> CREATE TABLE tmp SELECT IFNULL(1,'test') AS test;
mysql> DESCRIBE tmp;
+-------+--------------+------+-----+---------+-------+
| Field | Type         | Null | Key | Default | Extra |
+-------+--------------+------+-----+---------+-------+
| test  | varbinary(4) | NO   |     |         |       |
+-------+--------------+------+-----+---------+-------+
```

  In this example, the type of the `test` column is `VARBINARY(4)` (a string type).

- `NULLIF(expr1,expr2)`

  Returns `NULL` if *`expr1 = expr2`* is true, otherwise returns *`expr1`*. This is the same as `CASE WHEN` *`expr1 = expr2`* `THEN NULL ELSE` *`expr1`* `END`.

  The return value has the same type as the first argument.

```
mysql> SELECT NULLIF(1,1);
        -> NULL
mysql> SELECT NULLIF(1,2);
        -> 1
```

Note

MySQL evaluates *`expr1`* twice if the arguments are not equal.

12.6 Numeric Functions and Operators

Table 12.8 Numeric Functions and Operators

Name	Description
`%`, `MOD`	Modulo operator
`*`	Multiplication operator
`+`	Addition operator
`-`	Minus operator
`-`	Change the sign of the argument
`/`	Division operator
`ABS()`	Return the absolute value

Name	Description
ACOS()	Return the arc cosine
ASIN()	Return the arc sine
ATAN()	Return the arc tangent
ATAN2(), ATAN()	Return the arc tangent of the two arguments
CEIL()	Return the smallest integer value not less than the argument
CEILING()	Return the smallest integer value not less than the argument
CONV()	Convert numbers between different number bases
COS()	Return the cosine
COT()	Return the cotangent
CRC32()	Compute a cyclic redundancy check value
DEGREES()	Convert radians to degrees
DIV	Integer division
EXP()	Raise to the power of
FLOOR()	Return the largest integer value not greater than the argument
LN()	Return the natural logarithm of the argument
LOG()	Return the natural logarithm of the first argument
LOG10()	Return the base-10 logarithm of the argument
LOG2()	Return the base-2 logarithm of the argument
MOD()	Return the remainder
PI()	Return the value of pi
POW()	Return the argument raised to the specified power
POWER()	Return the argument raised to the specified power
RADIANS()	Return argument converted to radians
RAND()	Return a random floating-point value
ROUND()	Round the argument
SIGN()	Return the sign of the argument
SIN()	Return the sine of the argument
SQRT()	Return the square root of the argument
TAN()	Return the tangent of the argument
TRUNCATE()	Truncate to specified number of decimal places

12.6.1 Arithmetic Operators

Table 12.9 Arithmetic Operators

Name	Description
%, MOD	Modulo operator
*	Multiplication operator
+	Addition operator
-	Minus operator
-	Change the sign of the argument
/	Division operator

Name	Description
DIV	Integer division

The usual arithmetic operators are available. The result is determined according to the following rules:

- In the case of `-`, `+`, and `*`, the result is calculated with `BIGINT` (64-bit) precision if both operands are integers.
- If both operands are integers and any of them are unsigned, the result is an unsigned integer. For subtraction, if the `NO_UNSIGNED_SUBTRACTION` SQL mode is enabled, the result is signed even if any operand is unsigned.
- If any of the operands of a `+`, `-`, `/`, `*`, `%` is a real or string value, the precision of the result is the precision of the operand with the maximum precision.
- In division performed with `/`, the scale of the result when using two exact-value operands is the scale of the first operand plus the value of the `div_precision_increment` system variable (which is 4 by default). For example, the result of the expression `5.05 / 0.014` has a scale of six decimal places (`360.714286`).

These rules are applied for each operation, such that nested calculations imply the precision of each component. Hence, `(14620 / 9432456) / (24250 / 9432456)`, resolves first to `(0.0014) / (0.0026)`, with the final result having 8 decimal places (`0.60288653`).

Because of these rules and the way they are applied, care should be taken to ensure that components and subcomponents of a calculation use the appropriate level of precision. See Section 12.11, “Cast Functions and Operators”.

For information about handling of overflow in numeric expression evaluation, see Section 11.1.7, “Out-of-Range and Overflow Handling”.

Arithmetic operators apply to numbers. For other types of values, alternative operations may be available. For example, to add date values, use `DATE_ADD()`; see Section 12.7, “Date and Time Functions”.

- `+`

  Addition:

```
mysql> SELECT 3+5;
        -> 8
```

- `-`

  Subtraction:

```
mysql> SELECT 3-5;
        -> -2
```

- `-`

  Unary minus. This operator changes the sign of the operand.

```
mysql> SELECT - 2;
        -> -2
```

Note

If this operator is used with a `BIGINT`, the return value is also a `BIGINT`. This means that you should avoid using `-` on integers that may have the value of -2^{63}.

- `*`

Multiplication:

```
mysql> SELECT 3*5;
        -> 15
mysql> SELECT 18014398509481984*18014398509481984.0;
        -> 324518553658426726783156020576256.0
mysql> SELECT 18014398509481984*18014398509481984;
        -> out-of-range error
```

The last expression produces an error because the result of the integer multiplication exceeds the 64-bit range of `BIGINT` calculations. (See Section 11.1, “Numeric Data Types”.)

- `/`

  Division:

  ```
  mysql> SELECT 3/5;
          -> 0.60
  ```

  Division by zero produces a `NULL` result:

  ```
  mysql> SELECT 102/(1-1);
          -> NULL
  ```

  A division is calculated with `BIGINT` arithmetic only if performed in a context where its result is converted to an integer.

- `DIV`

  Integer division. Discards from the division result any fractional part to the right of the decimal point.

  If either operand has a noninteger type, the operands are converted to `DECIMAL` and divided using `DECIMAL` arithmetic before converting the result to `BIGINT`. If the result exceeds `BIGINT` range, an error occurs.

  ```
  mysql> SELECT 5 DIV 2, -5 DIV 2, 5 DIV -2, -5 DIV -2;
          -> 2, -2, -2, 2
  ```

- *`N`* `%` *`M`*, *`N`* `MOD` *`M`*

  Modulo operation. Returns the remainder of *`N`* divided by *`M`*. For more information, see the description for the `MOD()` function in Section 12.6.2, “Mathematical Functions”.

12.6.2 Mathematical Functions

Table 12.10 Mathematical Functions

Name	Description
`ABS()`	Return the absolute value
`ACOS()`	Return the arc cosine
`ASIN()`	Return the arc sine
`ATAN()`	Return the arc tangent
`ATAN2()`, `ATAN()`	Return the arc tangent of the two arguments
`CEIL()`	Return the smallest integer value not less than the argument
`CEILING()`	Return the smallest integer value not less than the argument
`CONV()`	Convert numbers between different number bases
`COS()`	Return the cosine
`COT()`	Return the cotangent
`CRC32()`	Compute a cyclic redundancy check value

Name	Description
DEGREES()	Convert radians to degrees
EXP()	Raise to the power of
FLOOR()	Return the largest integer value not greater than the argument
LN()	Return the natural logarithm of the argument
LOG()	Return the natural logarithm of the first argument
LOG10()	Return the base-10 logarithm of the argument
LOG2()	Return the base-2 logarithm of the argument
MOD()	Return the remainder
PI()	Return the value of pi
POW()	Return the argument raised to the specified power
POWER()	Return the argument raised to the specified power
RADIANS()	Return argument converted to radians
RAND()	Return a random floating-point value
ROUND()	Round the argument
SIGN()	Return the sign of the argument
SIN()	Return the sine of the argument
SQRT()	Return the square root of the argument
TAN()	Return the tangent of the argument
TRUNCATE()	Truncate to specified number of decimal places

All mathematical functions return NULL in the event of an error.

- ABS(*X*)

  Returns the absolute value of *X*, or NULL if *X* is NULL.

  The result type is derived from the argument type. An implication of this is that ABS(-9223372036854775808) produces an error because the result cannot be stored in a signed BIGINT value.

```
mysql> SELECT ABS(2);
        -> 2
mysql> SELECT ABS(-32);
        -> 32
```

  This function is safe to use with BIGINT values.

- ACOS(*X*)

  Returns the arc cosine of *X*, that is, the value whose cosine is *X*. Returns NULL if *X* is not in the range -1 to 1.

```
mysql> SELECT ACOS(1);
        -> 0
mysql> SELECT ACOS(1.0001);
        -> NULL
mysql> SELECT ACOS(0);
        -> 1.5707963267949
```

- ASIN(*X*)

  Returns the arc sine of *X*, that is, the value whose sine is *X*. Returns NULL if *X* is not in the range -1 to 1.

```
mysql> SELECT ASIN(0.2);
        -> 0.20135792079033
mysql> SELECT ASIN('foo');

+-------------+
| ASIN('foo') |
+-------------+
|           0 |
+-------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1292 | Truncated incorrect DOUBLE value: 'foo' |
+---------+------+----------------------------------------+
```

- `ATAN(X)`

  Returns the arc tangent of *X*, that is, the value whose tangent is *X*.

```
mysql> SELECT ATAN(2);
        -> 1.1071487177941
mysql> SELECT ATAN(-2);
        -> -1.1071487177941
```

- `ATAN(Y,X)`, `ATAN2(Y,X)`

  Returns the arc tangent of the two variables *X* and *Y*. It is similar to calculating the arc tangent of *Y* / *X*, except that the signs of both arguments are used to determine the quadrant of the result.

```
mysql> SELECT ATAN(-2,2);
        -> -0.78539816339745
mysql> SELECT ATAN2(PI(),0);
        -> 1.5707963267949
```

- `CEIL(X)`

  `CEIL()` is a synonym for `CEILING()`.

- `CEILING(X)`

  Returns the smallest integer value not less than *X*.

```
mysql> SELECT CEILING(1.23);
        -> 2
mysql> SELECT CEILING(-1.23);
        -> -1
```

  For exact-value numeric arguments, the return value has an exact-value numeric type. For string or floating-point arguments, the return value has a floating-point type.

- `CONV(N,from_base,to_base)`

  Converts numbers between different number bases. Returns a string representation of the number *N*, converted from base *from_base* to base *to_base*. Returns `NULL` if any argument is `NULL`. The argument *N* is interpreted as an integer, but may be specified as an integer or a string. The minimum base is `2` and the maximum base is `36`. If *from_base* is a negative number, *N* is regarded as a signed number. Otherwise, *N* is treated as unsigned. `CONV()` works with 64-bit precision.

```
mysql> SELECT CONV('a',16,2);
        -> '1010'
mysql> SELECT CONV('6E',18,8);
        -> '172'
mysql> SELECT CONV(-17,10,-18);
        -> '-H'
```

```
mysql> SELECT CONV(10+'10'+'10'+X'0a',10,10);
        -> '40'
```

- `COS(X)`

  Returns the cosine of *X*, where *X* is given in radians.

```
mysql> SELECT COS(PI());
        -> -1
```

- `COT(X)`

  Returns the cotangent of *X*.

```
mysql> SELECT COT(12);
        -> -1.5726734063977
mysql> SELECT COT(0);
        -> out-of-range error
```

- `CRC32(expr)`

  Computes a cyclic redundancy check value and returns a 32-bit unsigned value. The result is `NULL` if the argument is `NULL`. The argument is expected to be a string and (if possible) is treated as one if it is not.

```
mysql> SELECT CRC32('MySQL');
        -> 3259397556
mysql> SELECT CRC32('mysql');
        -> 2501908538
```

- `DEGREES(X)`

  Returns the argument *X*, converted from radians to degrees.

```
mysql> SELECT DEGREES(PI());
        -> 180
mysql> SELECT DEGREES(PI() / 2);
        -> 90
```

- `EXP(X)`

  Returns the value of *e* (the base of natural logarithms) raised to the power of *X*. The inverse of this function is `LOG()` (using a single argument only) or `LN()`.

```
mysql> SELECT EXP(2);
        -> 7.3890560989307
mysql> SELECT EXP(-2);
        -> 0.13533528323661
mysql> SELECT EXP(0);
        -> 1
```

- `FLOOR(X)`

  Returns the largest integer value not greater than *X*.

```
mysql> SELECT FLOOR(1.23), FLOOR(-1.23);
        -> 1, -2
```

  For exact-value numeric arguments, the return value has an exact-value numeric type. For string or floating-point arguments, the return value has a floating-point type.

- `FORMAT(X,D)`

  Formats the number *X* to a format like `'#,###,###.##'`, rounded to *D* decimal places, and returns the result as a string. For details, see Section 12.8, “String Functions and Operators”.

- `HEX(N_or_S)`

This function can be used to obtain a hexadecimal representation of a decimal number or a string; the manner in which it does so varies according to the argument's type. See this function's description in Section 12.8, “String Functions and Operators”, for details.

- `LN(X)`

  Returns the natural logarithm of *X*; that is, the base-*e* logarithm of *X*. If *X* is less than or equal to 0, then `NULL` is returned.

```
mysql> SELECT LN(2);
        -> 0.69314718055995
mysql> SELECT LN(-2);
        -> NULL
```

  This function is synonymous with `LOG(X)`. The inverse of this function is the `EXP()` function.

- `LOG(X)`, `LOG(B,X)`

  If called with one parameter, this function returns the natural logarithm of *X*. If *X* is less than or equal to 0, then `NULL` is returned.

  The inverse of this function (when called with a single argument) is the `EXP()` function.

```
mysql> SELECT LOG(2);
        -> 0.69314718055995
mysql> SELECT LOG(-2);
        -> NULL
```

  If called with two parameters, this function returns the logarithm of *X* to the base *B*. If *X* is less than or equal to 0, or if *B* is less than or equal to 1, then `NULL` is returned.

```
mysql> SELECT LOG(2,65536);
        -> 16
mysql> SELECT LOG(10,100);
        -> 2
mysql> SELECT LOG(1,100);
        -> NULL
```

  `LOG(B,X)` is equivalent to `LOG(X) / LOG(B)`.

- `LOG2(X)`

  Returns the base-2 logarithm of *X*.

```
mysql> SELECT LOG2(65536);
        -> 16
mysql> SELECT LOG2(-100);
        -> NULL
```

  `LOG2()` is useful for finding out how many bits a number requires for storage. This function is equivalent to the expression `LOG(X) / LOG(2)`.

- `LOG10(X)`

  Returns the base-10 logarithm of *X*.

```
mysql> SELECT LOG10(2);
        -> 0.30102999566398
mysql> SELECT LOG10(100);
        -> 2
mysql> SELECT LOG10(-100);
        -> NULL
```

  `LOG10(X)` is equivalent to `LOG(10,X)`.

- `MOD(N,M)`, `N % M`, `N MOD M`

Modulo operation. Returns the remainder of *N* divided by *M*.

```
mysql> SELECT MOD(234, 10);
        -> 4
mysql> SELECT 253 % 7;
        -> 1
mysql> SELECT MOD(29,9);
        -> 2
mysql> SELECT 29 MOD 9;
        -> 2
```

This function is safe to use with `BIGINT` values.

`MOD()` also works on values that have a fractional part and returns the exact remainder after division:

```
mysql> SELECT MOD(34.5,3);
        -> 1.5
```

`MOD(N,0)` returns `NULL`.

- `PI()`

  Returns the value of π (pi). The default number of decimal places displayed is seven, but MySQL uses the full double-precision value internally.

  ```
  mysql> SELECT PI();
          -> 3.141593
  mysql> SELECT PI()+0.00000000000000000;
          -> 3.141592653589793116
  ```

- `POW(X,Y)`

  Returns the value of *X* raised to the power of *Y*.

  ```
  mysql> SELECT POW(2,2);
          -> 4
  mysql> SELECT POW(2,-2);
          -> 0.25
  ```

- `POWER(X,Y)`

  This is a synonym for `POW()`.

- `RADIANS(X)`

  Returns the argument *X*, converted from degrees to radians.

  Note

  π radians equals 180 degrees.

  ```
  mysql> SELECT RADIANS(90);
          -> 1.5707963267949
  ```

- `RAND([N])`

  Returns a random floating-point value *v* in the range `0` <= *v* < `1.0`. To obtain a random integer *R* in the range *i* <= *R* < *j*, use the expression `FLOOR(i + RAND() * (j - i))`. For example, to obtain a random integer in the range the range `7` <= *R* < `12`, use the following statement:

  ```
  SELECT FLOOR(7 + (RAND() * 5));
  ```

  If an integer argument *N* is specified, it is used as the seed value:

  - With a constant initializer argument, the seed is initialized once when the statement is prepared, prior to execution.
  - With a nonconstant initializer argument (such as a column name), the seed is initialized with the value for each invocation of `RAND()`.

  One implication of this behavior is that for equal argument values, `RAND(N)` returns the same value each time, and thus produces a repeatable sequence of column values. In the following example, the sequence of values produced by `RAND(3)` is the same both places it occurs.

  ```
  mysql> CREATE TABLE t (i INT);
  Query OK, 0 rows affected (0.42 sec)

  mysql> INSERT INTO t VALUES(1),(2),(3);
  Query OK, 3 rows affected (0.00 sec)
  Records: 3  Duplicates: 0  Warnings: 0

  mysql> SELECT i, RAND() FROM t;
  +------+------------------+
  | i    | RAND()           |
  +------+------------------+
  |    1 | 0.61914388706828 |
  |    2 | 0.93845168309142 |
  |    3 | 0.83482678498591 |
  +------+------------------+
  3 rows in set (0.00 sec)

  mysql> SELECT i, RAND(3) FROM t;
  +------+------------------+
  | i    | RAND(3)          |
  +------+------------------+
  |    1 | 0.90576975597606 |
  |    2 | 0.37307905813035 |
  |    3 | 0.14808605345719 |
  +------+------------------+
  3 rows in set (0.00 sec)

  mysql> SELECT i, RAND() FROM t;
  +------+------------------+
  | i    | RAND()           |
  +------+------------------+
  |    1 | 0.35877890638893 |
  |    2 | 0.28941420772058 |
  |    3 | 0.37073435016976 |
  +------+------------------+
  3 rows in set (0.00 sec)

  mysql> SELECT i, RAND(3) FROM t;
  +------+------------------+
  | i    | RAND(3)          |
  +------+------------------+
  |    1 | 0.90576975597606 |
  |    2 | 0.37307905813035 |
  |    3 | 0.14808605345719 |
  +------+------------------+
  ```

```
3 rows in set (0.01 sec)
```

`RAND()` in a `WHERE` clause is evaluated for every row (when selecting from one table) or combination of rows (when selecting from a multiple-table join). Thus, for optimizer purposes, `RAND()` is not a constant value and cannot be used for index optimizations. For more information, see Section 8.2.1.17, “Function Call Optimization”.

Use of a column with `RAND()` values in an `ORDER BY` or `GROUP BY` clause may yield unexpected results because for either clause a `RAND()` expression can be evaluated multiple times for the same row, each time returning a different result. If the goal is to retrieve rows in random order, you can use a statement like this:

```
SELECT * FROM tbl_name ORDER BY RAND();
```

To select a random sample from a set of rows, combine `ORDER BY RAND()` with `LIMIT`:

```
SELECT * FROM table1, table2 WHERE a=b AND c<d ORDER BY RAND() LIMIT 1000;
```

`RAND()` is not meant to be a perfect random generator. It is a fast way to generate random numbers on demand that is portable between platforms for the same MySQL version.

This function is unsafe for statement-based replication. A warning is logged if you use this function when `binlog_format` is set to `STATEMENT`.

- `ROUND(X)`, `ROUND(X,D)`

  Rounds the argument *X* to *D* decimal places. The rounding algorithm depends on the data type of *X*. *D* defaults to 0 if not specified. *D* can be negative to cause *D* digits left of the decimal point of the value *X* to become zero. The maximum absolute value for *D* is 30; any digits in excess of 30 (or -30) are truncated.

```
mysql> SELECT ROUND(-1.23);
        -> -1
mysql> SELECT ROUND(-1.58);
        -> -2
mysql> SELECT ROUND(1.58);
        -> 2
mysql> SELECT ROUND(1.298, 1);
        -> 1.3
mysql> SELECT ROUND(1.298, 0);
        -> 1
mysql> SELECT ROUND(23.298, -1);
        -> 20
mysql> SELECT ROUND(.12345678901234567890123456789012345, 35);
        -> 0.12345678901234567890123456789
```

The return value has the same type as the first argument (assuming that it is integer, double, or decimal). This means that for an integer argument, the result is an integer (no decimal places):

```
mysql> SELECT ROUND(150.000,2), ROUND(150,2);
+------------------+--------------+
| ROUND(150.000,2) | ROUND(150,2) |
+------------------+--------------+
|           150.00 |          150 |
+------------------+--------------+
```

`ROUND()` uses the following rules depending on the type of the first argument:

- For exact-value numbers, `ROUND()` uses the “round half away from zero” or “round toward nearest” rule: A value with a fractional part of .5 or greater is rounded up to the next integer if positive or down to the next integer if negative. (In other words, it is rounded away from zero.) A value with a fractional part less than .5 is rounded down to the next integer if positive or up to the next integer if negative.

- For approximate-value numbers, the result depends on the C library. On many systems, this means that `ROUND()` uses the “round to nearest even” rule: A value with a fractional part exactly half way between two integers is rounded to the nearest even integer.

The following example shows how rounding differs for exact and approximate values:

```
mysql> SELECT ROUND(2.5), ROUND(25E-1);
+------------+--------------+
| ROUND(2.5) | ROUND(25E-1) |
+------------+--------------+
| 3          |            2 |
+------------+--------------+
```

For more information, see Section 12.21, “Precision Math”.

- `SIGN(X)`

  Returns the sign of the argument as `-1`, `0`, or `1`, depending on whether *X* is negative, zero, or positive.

```
mysql> SELECT SIGN(-32);
        -> -1
mysql> SELECT SIGN(0);
        -> 0
mysql> SELECT SIGN(234);
        -> 1
```

- `SIN(X)`

  Returns the sine of *X*, where *X* is given in radians.

```
mysql> SELECT SIN(PI());
        -> 1.2246063538224e-16
mysql> SELECT ROUND(SIN(PI()));
        -> 0
```

- `SQRT(X)`

  Returns the square root of a nonnegative number *X*.

```
mysql> SELECT SQRT(4);
        -> 2
mysql> SELECT SQRT(20);
        -> 4.4721359549996
mysql> SELECT SQRT(-16);
        -> NULL
```

- `TAN(X)`

  Returns the tangent of *X*, where *X* is given in radians.

```
mysql> SELECT TAN(PI());
        -> -1.2246063538224e-16
mysql> SELECT TAN(PI()+1);
        -> 1.5574077246549
```

- `TRUNCATE(X,D)`

  Returns the number *X*, truncated to *D* decimal places. If *D* is `0`, the result has no decimal point or fractional part. *D* can be negative to cause *D* digits left of the decimal point of the value *X* to become zero.

```
mysql> SELECT TRUNCATE(1.223,1);
        -> 1.2
mysql> SELECT TRUNCATE(1.999,1);
        -> 1.9
mysql> SELECT TRUNCATE(1.999,0);
```

```
        -> 1
mysql> SELECT TRUNCATE(-1.999,1);
        -> -1.9
mysql> SELECT TRUNCATE(122,-2);
       -> 100
mysql> SELECT TRUNCATE(10.28*100,0);
       -> 1028
```

All numbers are rounded toward zero.

12.7 Date and Time Functions

This section describes the functions that can be used to manipulate temporal values. See Section 11.2, “Date and Time Data Types”, for a description of the range of values each date and time type has and the valid formats in which values may be specified.

Table 12.11 Date and Time Functions

Name	Description
ADDDATE()	Add time values (intervals) to a date value
ADDTIME()	Add time
CONVERT_TZ()	Convert from one time zone to another
CURDATE()	Return the current date
CURRENT_DATE(), CURRENT_DATE	Synonyms for CURDATE()
CURRENT_TIME(), CURRENT_TIME	Synonyms for CURTIME()
CURRENT_TIMESTAMP(), CURRENT_TIMESTAMP	Synonyms for NOW()
CURTIME()	Return the current time
DATE()	Extract the date part of a date or datetime expression
DATE_ADD()	Add time values (intervals) to a date value
DATE_FORMAT()	Format date as specified
DATE_SUB()	Subtract a time value (interval) from a date
DATEDIFF()	Subtract two dates
DAY()	Synonym for DAYOFMONTH()
DAYNAME()	Return the name of the weekday
DAYOFMONTH()	Return the day of the month (0-31)
DAYOFWEEK()	Return the weekday index of the argument
DAYOFYEAR()	Return the day of the year (1-366)
EXTRACT()	Extract part of a date
FROM_DAYS()	Convert a day number to a date
FROM_UNIXTIME()	Format Unix timestamp as a date
GET_FORMAT()	Return a date format string
HOUR()	Extract the hour
LAST_DAY	Return the last day of the month for the argument
LOCALTIME(), LOCALTIME	Synonym for NOW()
LOCALTIMESTAMP, LOCALTIMESTAMP()	Synonym for NOW()
MAKEDATE()	Create a date from the year and day of year
MAKETIME()	Create time from hour, minute, second

Name	Description
MICROSECOND()	Return the microseconds from argument
MINUTE()	Return the minute from the argument
MONTH()	Return the month from the date passed
MONTHNAME()	Return the name of the month
NOW()	Return the current date and time
PERIOD_ADD()	Add a period to a year-month
PERIOD_DIFF()	Return the number of months between periods
QUARTER()	Return the quarter from a date argument
SEC_TO_TIME()	Converts seconds to 'hh:mm:ss' format
SECOND()	Return the second (0-59)
STR_TO_DATE()	Convert a string to a date
SUBDATE()	Synonym for DATE_SUB() when invoked with three arguments
SUBTIME()	Subtract times
SYSDATE()	Return the time at which the function executes
TIME()	Extract the time portion of the expression passed
TIME_FORMAT()	Format as time
TIME_TO_SEC()	Return the argument converted to seconds
TIMEDIFF()	Subtract time
TIMESTAMP()	With a single argument, this function returns the date or datetime expression; with two arguments, the sum of the arguments
TIMESTAMPADD()	Add an interval to a datetime expression
TIMESTAMPDIFF()	Subtract an interval from a datetime expression
TO_DAYS()	Return the date argument converted to days
TO_SECONDS()	Return the date or datetime argument converted to seconds since Year 0
UNIX_TIMESTAMP()	Return a Unix timestamp
UTC_DATE()	Return the current UTC date
UTC_TIME()	Return the current UTC time
UTC_TIMESTAMP()	Return the current UTC date and time
WEEK()	Return the week number
WEEKDAY()	Return the weekday index
WEEKOFYEAR()	Return the calendar week of the date (1-53)
YEAR()	Return the year
YEARWEEK()	Return the year and week

Here is an example that uses date functions. The following query selects all rows with a *`date_col`* value from within the last 30 days:

```
mysql> SELECT something FROM tbl_name
    -> WHERE DATE_SUB(CURDATE(),INTERVAL 30 DAY) <= date_col;
```

The query also selects rows with dates that lie in the future.

Functions that expect date values usually accept datetime values and ignore the time part. Functions that expect time values usually accept datetime values and ignore the date part.

Functions that return the current date or time each are evaluated only once per query at the start of query execution. This means that multiple references to a function such as `NOW()` within a single query always produce the same result. (For our purposes, a single query also includes a call to a stored program (stored routine, trigger, or event) and all subprograms called by that program.) This principle also applies to `CURDATE()`, `CURTIME()`, `UTC_DATE()`, `UTC_TIME()`, `UTC_TIMESTAMP()`, and to any of their synonyms.

The `CURRENT_TIMESTAMP()`, `CURRENT_TIME()`, `CURRENT_DATE()`, and `FROM_UNIXTIME()` functions return values in the current session time zone, which is available as the session value of the `time_zone` system variable. In addition, `UNIX_TIMESTAMP()` assumes that its argument is a datetime value in the session time zone. See Section 5.1.13, “MySQL Server Time Zone Support”.

Some date functions can be used with “zero” dates or incomplete dates such as `'2001-11-00'`, whereas others cannot. Functions that extract parts of dates typically work with incomplete dates and thus can return 0 when you might otherwise expect a nonzero value. For example:

```
mysql> SELECT DAYOFMONTH('2001-11-00'), MONTH('2005-00-00');
        -> 0, 0
```

Other functions expect complete dates and return `NULL` for incomplete dates. These include functions that perform date arithmetic or that map parts of dates to names. For example:

```
mysql> SELECT DATE_ADD('2006-05-00',INTERVAL 1 DAY);
        -> NULL
mysql> SELECT DAYNAME('2006-05-00');
        -> NULL
```

Several functions are strict when passed a `DATE()` function value as their argument and reject incomplete dates with a day part of zero: `CONVERT_TZ()`, `DATE_ADD()`, `DATE_SUB()`, `DAYOFYEAR()`, `TIMESTAMPDIFF()`, `TO_DAYS()`, `TO_SECONDS()`, `WEEK()`, `WEEKDAY()`, `WEEKOFYEAR()`, `YEARWEEK()`.

Fractional seconds for `TIME`, `DATETIME`, and `TIMESTAMP` values are supported, with up to microsecond precision. Functions that take temporal arguments accept values with fractional seconds. Return values from temporal functions include fractional seconds as appropriate.

- `ADDDATE(`*`date`*`,INTERVAL `*`expr unit`*`)`, `ADDDATE(`*`expr`*`,`*`days`*`)`

  When invoked with the `INTERVAL` form of the second argument, `ADDDATE()` is a synonym for `DATE_ADD()`. The related function `SUBDATE()` is a synonym for `DATE_SUB()`. For information on the `INTERVAL` *`unit`* argument, see Temporal Intervals.

  ```
  mysql> SELECT DATE_ADD('2008-01-02', INTERVAL 31 DAY);
          -> '2008-02-02'
  mysql> SELECT ADDDATE('2008-01-02', INTERVAL 31 DAY);
          -> '2008-02-02'
  ```

  When invoked with the *`days`* form of the second argument, MySQL treats it as an integer number of days to be added to *`expr`*.

  ```
  mysql> SELECT ADDDATE('2008-01-02', 31);
          -> '2008-02-02'
  ```

- `ADDTIME(`*`expr1`*`,`*`expr2`*`)`

  `ADDTIME()` adds *`expr2`* to *`expr1`* and returns the result. *`expr1`* is a time or datetime expression, and *`expr2`* is a time expression.

  ```
  mysql> SELECT ADDTIME('2007-12-31 23:59:59.999999', '1 1:1:1.000002');
          -> '2008-01-02 01:01:01.000001'
  mysql> SELECT ADDTIME('01:00:00.999999', '02:00:00.999998');
          -> '03:00:01.999997'
  ```

- `CONVERT_TZ(dt,from_tz,to_tz)`

  `CONVERT_TZ()` converts a datetime value *`dt`* from the time zone given by *`from_tz`* to the time zone given by *`to_tz`* and returns the resulting value. Time zones are specified as described in Section 5.1.13, “MySQL Server Time Zone Support”. This function returns `NULL` if the arguments are invalid.

  If the value falls out of the supported range of the `TIMESTAMP` type when converted from *`from_tz`* to UTC, no conversion occurs. The `TIMESTAMP` range is described in Section 11.2.1, “Date and Time Data Type Syntax”.

```
mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','GMT','MET');
        -> '2004-01-01 13:00:00'
mysql> SELECT CONVERT_TZ('2004-01-01 12:00:00','+00:00','+10:00');
        -> '2004-01-01 22:00:00'
```

Note

To use named time zones such as `'MET'` or `'Europe/Amsterdam'`, the time zone tables must be properly set up. For instructions, see Section 5.1.13, “MySQL Server Time Zone Support”.

- `CURDATE()`

  Returns the current date as a value in *`'YYYY-MM-DD'`* or *`YYYYMMDD`* format, depending on whether the function is used in string or numeric context.

```
mysql> SELECT CURDATE();
        -> '2008-06-13'
mysql> SELECT CURDATE() + 0;
        -> 20080613
```

- `CURRENT_DATE`, `CURRENT_DATE()`

  `CURRENT_DATE` and `CURRENT_DATE()` are synonyms for `CURDATE()`.

- `CURRENT_TIME`, `CURRENT_TIME([fsp])`

  `CURRENT_TIME` and `CURRENT_TIME()` are synonyms for `CURTIME()`.

- `CURRENT_TIMESTAMP`, `CURRENT_TIMESTAMP([fsp])`

  `CURRENT_TIMESTAMP` and `CURRENT_TIMESTAMP()` are synonyms for `NOW()`.

- `CURTIME([fsp])`

  Returns the current time as a value in *`'hh:mm:ss'`* or *`hhmmss`* format, depending on whether the function is used in string or numeric context. The value is expressed in the session time zone.

  If the *`fsp`* argument is given to specify a fractional seconds precision from 0 to 6, the return value includes a fractional seconds part of that many digits.

```
mysql> SELECT CURTIME();
        -> '23:50:26'
mysql> SELECT CURTIME() + 0;
        -> 235026.000000
```

- `DATE(expr)`

  Extracts the date part of the date or datetime expression *`expr`*.

```
mysql> SELECT DATE('2003-12-31 01:02:03');
        -> '2003-12-31'
```

- `DATEDIFF(expr1,expr2)`

`DATEDIFF()` returns *`expr1`* − *`expr2`* expressed as a value in days from one date to the other. *`expr1`* and *`expr2`* are date or date-and-time expressions. Only the date parts of the values are used in the calculation.

```
mysql> SELECT DATEDIFF('2007-12-31 23:59:59','2007-12-30');
        -> 1
mysql> SELECT DATEDIFF('2010-11-30 23:59:59','2010-12-31');
        -> -31
```

- `DATE_ADD(`*`date`*`,INTERVAL` *`expr unit`*`)`, `DATE_SUB(`*`date`*`,INTERVAL` *`expr unit`*`)`

  These functions perform date arithmetic. The *`date`* argument specifies the starting date or datetime value. *`expr`* is an expression specifying the interval value to be added or subtracted from the starting date. *`expr`* is evaluated as a string; it may start with a `-` for negative intervals. *`unit`* is a keyword indicating the units in which the expression should be interpreted.

  For more information about temporal interval syntax, including a full list of *`unit`* specifiers, the expected form of the *`expr`* argument for each *`unit`* value, and rules for operand interpretation in temporal arithmetic, see Temporal Intervals.

  The return value depends on the arguments:

  - `DATE` if the *`date`* argument is a `DATE` value and your calculations involve only `YEAR`, `MONTH`, and `DAY` parts (that is, no time parts).
  - `DATETIME` if the first argument is a `DATETIME` (or `TIMESTAMP`) value, or if the first argument is a `DATE` and the *`unit`* value uses `HOURS`, `MINUTES`, or `SECONDS`.
  - String otherwise.

  To ensure that the result is `DATETIME`, you can use `CAST()` to convert the first argument to `DATETIME`.

```
mysql> SELECT DATE_ADD('2018-05-01',INTERVAL 1 DAY);
        -> '2018-05-02'
mysql> SELECT DATE_SUB('2018-05-01',INTERVAL 1 YEAR);
        -> '2017-05-01'
mysql> SELECT DATE_ADD('2020-12-31 23:59:59',
    ->                 INTERVAL 1 SECOND);
        -> '2021-01-01 00:00:00'
mysql> SELECT DATE_ADD('2018-12-31 23:59:59',
    ->                 INTERVAL 1 DAY);
        -> '2019-01-01 23:59:59'
mysql> SELECT DATE_ADD('2100-12-31 23:59:59',
    ->                 INTERVAL '1:1' MINUTE_SECOND);
        -> '2101-01-01 00:01:00'
mysql> SELECT DATE_SUB('2025-01-01 00:00:00',
    ->                 INTERVAL '1 1:1:1' DAY_SECOND);
        -> '2024-12-30 22:58:59'
mysql> SELECT DATE_ADD('1900-01-01 00:00:00',
    ->                 INTERVAL '-1 10' DAY_HOUR);
        -> '1899-12-30 14:00:00'
mysql> SELECT DATE_SUB('1998-01-02', INTERVAL 31 DAY);
        -> '1997-12-02'
mysql> SELECT DATE_ADD('1992-12-31 23:59:59.000002',
    ->            INTERVAL '1.999999' SECOND_MICROSECOND);
        -> '1993-01-01 00:00:01.000001'
```

- `DATE_FORMAT(date,format)`

  Formats the *`date`* value according to the *`format`* string.

  The specifiers shown in the following table may be used in the *`format`* string. The `%` character is required before format specifier characters. The specifiers apply to other functions as well: `STR_TO_DATE()`, `TIME_FORMAT()`, `UNIX_TIMESTAMP()`.

Specifier	Description
`%a`	Abbreviated weekday name (`Sun`..`Sat`)
`%b`	Abbreviated month name (`Jan`..`Dec`)
`%c`	Month, numeric (`0`..`12`)
`%D`	Day of the month with English suffix (`0th`, `1st`, `2nd`, `3rd`, …)
`%d`	Day of the month, numeric (`00`..`31`)
`%e`	Day of the month, numeric (`0`..`31`)
`%f`	Microseconds (`000000`..`999999`)
`%H`	Hour (`00`..`23`)
`%h`	Hour (`01`..`12`)
`%I`	Hour (`01`..`12`)
`%i`	Minutes, numeric (`00`..`59`)
`%j`	Day of year (`001`..`366`)
`%k`	Hour (`0`..`23`)
`%l`	Hour (`1`..`12`)
`%M`	Month name (`January`..`December`)
`%m`	Month, numeric (`00`..`12`)
`%p`	`AM` or `PM`
`%r`	Time, 12-hour (*`hh:mm:ss`* followed by `AM` or `PM`)
`%S`	Seconds (`00`..`59`)
`%s`	Seconds (`00`..`59`)
`%T`	Time, 24-hour (*`hh:mm:ss`*)
`%U`	Week (`00`..`53`), where Sunday is the first day of the week; `WEEK()` mode 0
`%u`	Week (`00`..`53`), where Monday is the first day of the week; `WEEK()` mode 1
`%V`	Week (`01`..`53`), where Sunday is the first day of the week; `WEEK()` mode 2; used with `%X`
`%v`	Week (`01`..`53`), where Monday is the first day of the week; `WEEK()` mode 3; used with `%x`
`%W`	Weekday name (`Sunday`..`Saturday`)
`%w`	Day of the week (`0`=Sunday..`6`=Saturday)
`%X`	Year for the week where Sunday is the first day of the week, numeric, four digits; used with `%V`
`%x`	Year for the week, where Monday is the first day of the week, numeric, four digits; used with `%v`
`%Y`	Year, numeric, four digits
`%y`	Year, numeric (two digits)

Specifier	Description
%%	A literal % character
%*x*	*x*, for any "*x*" not listed above

Ranges for the month and day specifiers begin with zero due to the fact that MySQL permits the storing of incomplete dates such as '2014-00-00'.

The language used for day and month names and abbreviations is controlled by the value of the lc_time_names system variable (Section 10.16, "MySQL Server Locale Support").

For the %U, %u, %V, and %v specifiers, see the description of the WEEK() function for information about the mode values. The mode affects how week numbering occurs.

DATE_FORMAT() returns a string with a character set and collation given by character_set_connection and collation_connection so that it can return month and weekday names containing non-ASCII characters.

```
mysql> SELECT DATE_FORMAT('2009-10-04 22:23:00', '%W %M %Y');
        -> 'Sunday October 2009'
mysql> SELECT DATE_FORMAT('2007-10-04 22:23:00', '%H:%i:%s');
        -> '22:23:00'
mysql> SELECT DATE_FORMAT('1900-10-04 22:23:00',
    ->                 '%D %y %a %d %m %b %j');
        -> '4th 00 Thu 04 10 Oct 277'
mysql> SELECT DATE_FORMAT('1997-10-04 22:23:00',
    ->                 '%H %k %I %r %T %S %w');
        -> '22 22 10 10:23:00 PM 22:23:00 00 6'
mysql> SELECT DATE_FORMAT('1999-01-01', '%X %V');
        -> '1998 52'
mysql> SELECT DATE_FORMAT('2006-06-00', '%d');
        -> '00'
```

- DATE_SUB(*date*,INTERVAL *expr unit*)

  See the description for DATE_ADD().

- DAY(*date*)

  DAY() is a synonym for DAYOFMONTH().

- DAYNAME(*date*)

  Returns the name of the weekday for *date*. The language used for the name is controlled by the value of the lc_time_names system variable (Section 10.16, "MySQL Server Locale Support").

```
mysql> SELECT DAYNAME('2007-02-03');
        -> 'Saturday'
```

- DAYOFMONTH(*date*)

  Returns the day of the month for *date*, in the range 1 to 31, or 0 for dates such as '0000-00-00' or '2008-00-00' that have a zero day part.

```
mysql> SELECT DAYOFMONTH('2007-02-03');
        -> 3
```

- DAYOFWEEK(*date*)

  Returns the weekday index for *date* (1 = Sunday, 2 = Monday, …, 7 = Saturday). These index values correspond to the ODBC standard.

```
mysql> SELECT DAYOFWEEK('2007-02-03');
        -> 7
```

- DAYOFYEAR(*date*)

Returns the day of the year for *date*, in the range 1 to 366.

```
mysql> SELECT DAYOFYEAR('2007-02-03');
        -> 34
```

- EXTRACT(*unit* FROM *date*)

  The EXTRACT() function uses the same kinds of *unit* specifiers as DATE_ADD() or DATE_SUB(), but extracts parts from the date rather than performing date arithmetic. For information on the *unit* argument, see Temporal Intervals.

```
mysql> SELECT EXTRACT(YEAR FROM '2019-07-02');
        -> 2019
mysql> SELECT EXTRACT(YEAR_MONTH FROM '2019-07-02 01:02:03');
        -> 201907
mysql> SELECT EXTRACT(DAY_MINUTE FROM '2019-07-02 01:02:03');
        -> 20102
mysql> SELECT EXTRACT(MICROSECOND
    ->                FROM '2003-01-02 10:30:00.000123');
        -> 123
```

- FROM_DAYS(*N*)

  Given a day number *N*, returns a DATE value.

```
mysql> SELECT FROM_DAYS(730669);
        -> '2000-07-03'
```

  Use FROM_DAYS() with caution on old dates. It is not intended for use with values that precede the advent of the Gregorian calendar (1582). See Section 12.9, “What Calendar Is Used By MySQL?”.

- FROM_UNIXTIME(*unix_timestamp*[,*format*])

  Returns a representation of the *unix_timestamp* argument as a value in *'YYYY-MM-DD hh:mm:ss'* or *YYYYMMDDhhmmss* format, depending on whether the function is used in a string or numeric context. *unix_timestamp* is an internal timestamp value representing seconds since '1970-01-01 00:00:00' UTC, such as produced by the UNIX_TIMESTAMP() function.

  The return value is expressed in the session time zone. (Clients can set the session time zone as described in Section 5.1.13, “MySQL Server Time Zone Support”.) The *format* string, if given, is used to format the result the same way as described in the entry for the DATE_FORMAT() function.

```
mysql> SELECT FROM_UNIXTIME(1447430881);
        -> '2015-11-13 10:08:01'
mysql> SELECT FROM_UNIXTIME(1447430881) + 0;
        -> 20151113100801
mysql> SELECT FROM_UNIXTIME(1447430881,
    ->                      '%Y %D %M %h:%i:%s %x');
        -> '2015 13th November 10:08:01 2015'
```

Note

If you use UNIX_TIMESTAMP() and FROM_UNIXTIME() to convert between values in a non-UTC time zone and Unix timestamp values, the conversion is lossy because the mapping is not one-to-one in both directions. For details, see the description of the UNIX_TIMESTAMP() function.

- `GET_FORMAT({DATE|TIME|DATETIME}, {'EUR'|'USA'|'JIS'|'ISO'|'INTERNAL'})`

  Returns a format string. This function is useful in combination with the `DATE_FORMAT()` and the `STR_TO_DATE()` functions.

  The possible values for the first and second arguments result in several possible format strings (for the specifiers used, see the table in the `DATE_FORMAT()` function description). ISO format refers to ISO 9075, not ISO 8601.

Function Call	Result
`GET_FORMAT(DATE,'USA')`	`'%m.%d.%Y'`
`GET_FORMAT(DATE,'JIS')`	`'%Y-%m-%d'`
`GET_FORMAT(DATE,'ISO')`	`'%Y-%m-%d'`
`GET_FORMAT(DATE,'EUR')`	`'%d.%m.%Y'`
`GET_FORMAT(DATE,'INTERNAL')`	`'%Y%m%d'`
`GET_FORMAT(DATETIME,'USA')`	`'%Y-%m-%d %H.%i.%s'`
`GET_FORMAT(DATETIME,'JIS')`	`'%Y-%m-%d %H:%i:%s'`
`GET_FORMAT(DATETIME,'ISO')`	`'%Y-%m-%d %H:%i:%s'`
`GET_FORMAT(DATETIME,'EUR')`	`'%Y-%m-%d %H.%i.%s'`
`GET_FORMAT(DATETIME,'INTERNAL')`	`'%Y%m%d%H%i%s'`
`GET_FORMAT(TIME,'USA')`	`'%h:%i:%s %p'`
`GET_FORMAT(TIME,'JIS')`	`'%H:%i:%s'`
`GET_FORMAT(TIME,'ISO')`	`'%H:%i:%s'`
`GET_FORMAT(TIME,'EUR')`	`'%H.%i.%s'`
`GET_FORMAT(TIME,'INTERNAL')`	`'%H%i%s'`

  `TIMESTAMP` can also be used as the first argument to `GET_FORMAT()`, in which case the function returns the same values as for `DATETIME`.

```
mysql> SELECT DATE_FORMAT('2003-10-03',GET_FORMAT(DATE,'EUR'));
        -> '03.10.2003'
mysql> SELECT STR_TO_DATE('10.31.2003',GET_FORMAT(DATE,'USA'));
        -> '2003-10-31'
```

- `HOUR(time)`

  Returns the hour for *`time`*. The range of the return value is `0` to `23` for time-of-day values. However, the range of `TIME` values actually is much larger, so `HOUR` can return values greater than `23`.

```
mysql> SELECT HOUR('10:05:03');
        -> 10
mysql> SELECT HOUR('272:59:59');
        -> 272
```

- `LAST_DAY(date)`

  Takes a date or datetime value and returns the corresponding value for the last day of the month. Returns `NULL` if the argument is invalid.

```
mysql> SELECT LAST_DAY('2003-02-05');
        -> '2003-02-28'
mysql> SELECT LAST_DAY('2004-02-05');
        -> '2004-02-29'
mysql> SELECT LAST_DAY('2004-01-01 01:01:01');
        -> '2004-01-31'
mysql> SELECT LAST_DAY('2003-03-32');
        -> NULL
```

- `LOCALTIME`, `LOCALTIME([fsp])`

  `LOCALTIME` and `LOCALTIME()` are synonyms for `NOW()`.

- `LOCALTIMESTAMP`, `LOCALTIMESTAMP([fsp])`

  `LOCALTIMESTAMP` and `LOCALTIMESTAMP()` are synonyms for `NOW()`.

- `MAKEDATE(year,dayofyear)`

  Returns a date, given year and day-of-year values. *`dayofyear`* must be greater than 0 or the result is `NULL`.

```
mysql> SELECT MAKEDATE(2011,31), MAKEDATE(2011,32);
        -> '2011-01-31', '2011-02-01'
mysql> SELECT MAKEDATE(2011,365), MAKEDATE(2014,365);
        -> '2011-12-31', '2014-12-31'
mysql> SELECT MAKEDATE(2011,0);
        -> NULL
```

- `MAKETIME(hour,minute,second)`

  Returns a time value calculated from the *`hour`*, *`minute`*, and *`second`* arguments.

  The *`second`* argument can have a fractional part.

```
mysql> SELECT MAKETIME(12,15,30);
        -> '12:15:30'
```

- `MICROSECOND(expr)`

  Returns the microseconds from the time or datetime expression *`expr`* as a number in the range from `0` to `999999`.

```
mysql> SELECT MICROSECOND('12:00:00.123456');
        -> 123456
mysql> SELECT MICROSECOND('2019-12-31 23:59:59.000010');
        -> 10
```

- `MINUTE(time)`

  Returns the minute for *`time`*, in the range `0` to `59`.

```
mysql> SELECT MINUTE('2008-02-03 10:05:03');
        -> 5
```

- `MONTH(date)`

  Returns the month for *`date`*, in the range `1` to `12` for January to December, or `0` for dates such as `'0000-00-00'` or `'2008-00-00'` that have a zero month part.

```
mysql> SELECT MONTH('2008-02-03');
        -> 2
```

- `MONTHNAME(date)`

  Returns the full name of the month for *`date`*. The language used for the name is controlled by the value of the `lc_time_names` system variable (Section 10.16, “MySQL Server Locale Support”).

```
mysql> SELECT MONTHNAME('2008-02-03');
        -> 'February'
```

- `NOW([fsp])`

  Returns the current date and time as a value in *`'YYYY-MM-DD hh:mm:ss'`* or *`YYYYMMDDhhmmss`* format, depending on whether the function is used in string or numeric context. The value is expressed in the session time zone.

If the *fsp* argument is given to specify a fractional seconds precision from 0 to 6, the return value includes a fractional seconds part of that many digits.

```
mysql> SELECT NOW();
        -> '2007-12-15 23:50:26'
mysql> SELECT NOW() + 0;
        -> 20071215235026.000000
```

`NOW()` returns a constant time that indicates the time at which the statement began to execute. (Within a stored function or trigger, `NOW()` returns the time at which the function or triggering statement began to execute.) This differs from the behavior for `SYSDATE()`, which returns the exact time at which it executes.

```
mysql> SELECT NOW(), SLEEP(2), NOW();
+---------------------+----------+---------------------+
| NOW()               | SLEEP(2) | NOW()               |
+---------------------+----------+---------------------+
| 2006-04-12 13:47:36 |        0 | 2006-04-12 13:47:36 |
+---------------------+----------+---------------------+

mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
+---------------------+----------+---------------------+
| SYSDATE()           | SLEEP(2) | SYSDATE()           |
+---------------------+----------+---------------------+
| 2006-04-12 13:47:44 |        0 | 2006-04-12 13:47:46 |
+---------------------+----------+---------------------+
```

In addition, the `SET TIMESTAMP` statement affects the value returned by `NOW()` but not by `SYSDATE()`. This means that timestamp settings in the binary log have no effect on invocations of `SYSDATE()`. Setting the timestamp to a nonzero value causes each subsequent invocation of `NOW()` to return that value. Setting the timestamp to zero cancels this effect so that `NOW()` once again returns the current date and time.

See the description for `SYSDATE()` for additional information about the differences between the two functions.

- `PERIOD_ADD(P,N)`

  Adds *N* months to period *P* (in the format *YYMM* or *YYYYMM*). Returns a value in the format *YYYYMM*.

  Note

  The period argument *P* is *not* a date value.

  ```
  mysql> SELECT PERIOD_ADD(200801,2);
          -> 200803
  ```

- `PERIOD_DIFF(P1,P2)`

  Returns the number of months between periods *P1* and *P2*. *P1* and *P2* should be in the format *YYMM* or *YYYYMM*. Note that the period arguments *P1* and *P2* are *not* date values.

  ```
  mysql> SELECT PERIOD_DIFF(200802,200703);
          -> 11
  ```

- `QUARTER(date)`

  Returns the quarter of the year for *date*, in the range `1` to `4`.

  ```
  mysql> SELECT QUARTER('2008-04-01');
          -> 2
  ```

- `SECOND(time)`

  Returns the second for *`time`*, in the range `0` to `59`.

```
mysql> SELECT SECOND('10:05:03');
        -> 3
```

- `SEC_TO_TIME(seconds)`

  Returns the *`seconds`* argument, converted to hours, minutes, and seconds, as a `TIME` value. The range of the result is constrained to that of the `TIME` data type. A warning occurs if the argument corresponds to a value outside that range.

```
mysql> SELECT SEC_TO_TIME(2378);
        -> '00:39:38'
mysql> SELECT SEC_TO_TIME(2378) + 0;
        -> 3938
```

- `STR_TO_DATE(str,format)`

  This is the inverse of the `DATE_FORMAT()` function. It takes a string *`str`* and a format string *`format`*. `STR_TO_DATE()` returns a `DATETIME` value if the format string contains both date and time parts, or a `DATE` or `TIME` value if the string contains only date or time parts. If the date, time, or datetime value extracted from *`str`* is illegal, `STR_TO_DATE()` returns `NULL` and produces a warning.

  The server scans *`str`* attempting to match *`format`* to it. The format string can contain literal characters and format specifiers beginning with `%`. Literal characters in *`format`* must match literally in *`str`*. Format specifiers in *`format`* must match a date or time part in *`str`*. For the specifiers that can be used in *`format`*, see the `DATE_FORMAT()` function description.

```
mysql> SELECT STR_TO_DATE('01,5,2013','%d,%m,%Y');
        -> '2013-05-01'
mysql> SELECT STR_TO_DATE('May 1, 2013','%M %d,%Y');
        -> '2013-05-01'
```

  Scanning starts at the beginning of *`str`* and fails if *`format`* is found not to match. Extra characters at the end of *`str`* are ignored.

```
mysql> SELECT STR_TO_DATE('a09:30:17','a%h:%i:%s');
        -> '09:30:17'
mysql> SELECT STR_TO_DATE('a09:30:17','%h:%i:%s');
        -> NULL
mysql> SELECT STR_TO_DATE('09:30:17a','%h:%i:%s');
        -> '09:30:17'
```

  Unspecified date or time parts have a value of 0, so incompletely specified values in *`str`* produce a result with some or all parts set to 0:

```
mysql> SELECT STR_TO_DATE('abc','abc');
        -> '0000-00-00'
mysql> SELECT STR_TO_DATE('9','%m');
        -> '0000-09-00'
mysql> SELECT STR_TO_DATE('9','%s');
        -> '00:00:09'
```

  Range checking on the parts of date values is as described in Section 11.2.2, “The DATE, DATETIME, and TIMESTAMP Types”. This means, for example, that “zero” dates or dates with part values of 0 are permitted unless the SQL mode is set to disallow such values.

```
mysql> SELECT STR_TO_DATE('00/00/0000', '%m/%d/%Y');
        -> '0000-00-00'
mysql> SELECT STR_TO_DATE('04/31/2004', '%m/%d/%Y');
```

```
        -> '2004-04-31'
```

If the NO_ZERO_DATE SQL mode is enabled, zero dates are disallowed. In that case, STR_TO_DATE() returns NULL and generates a warning:

```
mysql> SET sql_mode = '';
mysql> SELECT STR_TO_DATE('00/00/0000', '%m/%d/%Y');
+--------------------------------------+
| STR_TO_DATE('00/00/0000', '%m/%d/%Y') |
+--------------------------------------+
| 0000-00-00                           |
+--------------------------------------+
mysql> SET sql_mode = 'NO_ZERO_DATE';
mysql> SELECT STR_TO_DATE('00/00/0000', '%m/%d/%Y');
+--------------------------------------+
| STR_TO_DATE('00/00/0000', '%m/%d/%Y') |
+--------------------------------------+
| NULL                                 |
+--------------------------------------+
mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1411
Message: Incorrect datetime value: '00/00/0000' for function str_to_date
```

Note

You cannot use format "%X%V" to convert a year-week string to a date because the combination of a year and week does not uniquely identify a year and month if the week crosses a month boundary. To convert a year-week to a date, you should also specify the weekday:

```
mysql> SELECT STR_TO_DATE('200442 Monday', '%X%V %W');
        -> '2004-10-18'
```

- SUBDATE(*date*,INTERVAL *expr unit*), SUBDATE(*expr*,*days*)

  When invoked with the INTERVAL form of the second argument, SUBDATE() is a synonym for DATE_SUB(). For information on the INTERVAL *unit* argument, see the discussion for DATE_ADD().

```
mysql> SELECT DATE_SUB('2008-01-02', INTERVAL 31 DAY);
        -> '2007-12-02'
mysql> SELECT SUBDATE('2008-01-02', INTERVAL 31 DAY);
        -> '2007-12-02'
```

  The second form enables the use of an integer value for *days*. In such cases, it is interpreted as the number of days to be subtracted from the date or datetime expression *expr*.

```
mysql> SELECT SUBDATE('2008-01-02 12:00:00', 31);
        -> '2007-12-02 12:00:00'
```

- SUBTIME(*expr1*,*expr2*)

  SUBTIME() returns *expr1* − *expr2* expressed as a value in the same format as *expr1*. *expr1* is a time or datetime expression, and *expr2* is a time expression.

```
mysql> SELECT SUBTIME('2007-12-31 23:59:59.999999','1 1:1:1.000002');
        -> '2007-12-30 22:58:58.999997'
mysql> SELECT SUBTIME('01:00:00.999999', '02:00:00.999998');
        -> '-00:59:59.999999'
```

- `SYSDATE([fsp])`

  Returns the current date and time as a value in `'YYYY-MM-DD hh:mm:ss'` or *YYYYMMDDhhmmss* format, depending on whether the function is used in string or numeric context.

  If the *fsp* argument is given to specify a fractional seconds precision from 0 to 6, the return value includes a fractional seconds part of that many digits.

  `SYSDATE()` returns the time at which it executes. This differs from the behavior for `NOW()`, which returns a constant time that indicates the time at which the statement began to execute. (Within a stored function or trigger, `NOW()` returns the time at which the function or triggering statement began to execute.)

```
mysql> SELECT NOW(), SLEEP(2), NOW();
+---------------------+----------+---------------------+
| NOW()               | SLEEP(2) | NOW()               |
+---------------------+----------+---------------------+
| 2006-04-12 13:47:36 |        0 | 2006-04-12 13:47:36 |
+---------------------+----------+---------------------+

mysql> SELECT SYSDATE(), SLEEP(2), SYSDATE();
+---------------------+----------+---------------------+
| SYSDATE()           | SLEEP(2) | SYSDATE()           |
+---------------------+----------+---------------------+
| 2006-04-12 13:47:44 |        0 | 2006-04-12 13:47:46 |
+---------------------+----------+---------------------+
```

  In addition, the `SET TIMESTAMP` statement affects the value returned by `NOW()` but not by `SYSDATE()`. This means that timestamp settings in the binary log have no effect on invocations of `SYSDATE()`.

  Because `SYSDATE()` can return different values even within the same statement, and is not affected by `SET TIMESTAMP`, it is nondeterministic and therefore unsafe for replication if statement-based binary logging is used. If that is a problem, you can use row-based logging.

  Alternatively, you can use the `--sysdate-is-now` option to cause `SYSDATE()` to be an alias for `NOW()`. This works if the option is used on both the source and the replica.

  The nondeterministic nature of `SYSDATE()` also means that indexes cannot be used for evaluating expressions that refer to it.

- `TIME(expr)`

  Extracts the time part of the time or datetime expression *expr* and returns it as a string.

  This function is unsafe for statement-based replication. A warning is logged if you use this function when `binlog_format` is set to `STATEMENT`.

```
mysql> SELECT TIME('2003-12-31 01:02:03');
        -> '01:02:03'
mysql> SELECT TIME('2003-12-31 01:02:03.000123');
        -> '01:02:03.000123'
```

- `TIMEDIFF(expr1,expr2)`

  `TIMEDIFF()` returns *expr1* − *expr2* expressed as a time value. *expr1* and *expr2* are time or date-and-time expressions, but both must be of the same type.

  The result returned by `TIMEDIFF()` is limited to the range allowed for `TIME` values. Alternatively, you can use either of the functions `TIMESTAMPDIFF()` and `UNIX_TIMESTAMP()`, both of which return integers.

```
mysql> SELECT TIMEDIFF('2000:01:01 00:00:00',
    ->                 '2000:01:01 00:00:00.000001');
        -> '-00:00:00.000001'
```

```
mysql> SELECT TIMEDIFF('2008-12-31 23:59:59.000001',
    ->                 '2008-12-30 01:01:01.000002');
        -> '46:58:57.999999'
```

- `TIMESTAMP(expr)`, `TIMESTAMP(expr1,expr2)`

  With a single argument, this function returns the date or datetime expression *`expr`* as a datetime value. With two arguments, it adds the time expression *`expr2`* to the date or datetime expression *`expr1`* and returns the result as a datetime value.

```
mysql> SELECT TIMESTAMP('2003-12-31');
        -> '2003-12-31 00:00:00'
mysql> SELECT TIMESTAMP('2003-12-31 12:00:00','12:00:00');
        -> '2004-01-01 00:00:00'
```

- `TIMESTAMPADD(unit,interval,datetime_expr)`

  Adds the integer expression *`interval`* to the date or datetime expression *`datetime_expr`*. The unit for *`interval`* is given by the *`unit`* argument, which should be one of the following values: `MICROSECOND` (microseconds), `SECOND`, `MINUTE`, `HOUR`, `DAY`, `WEEK`, `MONTH`, `QUARTER`, or `YEAR`.

  The *`unit`* value may be specified using one of keywords as shown, or with a prefix of `SQL_TSI_`. For example, `DAY` and `SQL_TSI_DAY` both are legal.

```
mysql> SELECT TIMESTAMPADD(MINUTE,1,'2003-01-02');
        -> '2003-01-02 00:01:00'
mysql> SELECT TIMESTAMPADD(WEEK,1,'2003-01-02');
        -> '2003-01-09'
```

- `TIMESTAMPDIFF(unit,datetime_expr1,datetime_expr2)`

  Returns *`datetime_expr2`* − *`datetime_expr1`*, where *`datetime_expr1`* and *`datetime_expr2`* are date or datetime expressions. One expression may be a date and the other a datetime; a date value is treated as a datetime having the time part `'00:00:00'` where necessary. The unit for the result (an integer) is given by the *`unit`* argument. The legal values for *`unit`* are the same as those listed in the description of the `TIMESTAMPADD()` function.

```
mysql> SELECT TIMESTAMPDIFF(MONTH,'2003-02-01','2003-05-01');
        -> 3
mysql> SELECT TIMESTAMPDIFF(YEAR,'2002-05-01','2001-01-01');
        -> -1
mysql> SELECT TIMESTAMPDIFF(MINUTE,'2003-02-01','2003-05-01 12:05:55');
        -> 128885
```

Note

The order of the date or datetime arguments for this function is the opposite of that used with the `TIMESTAMP()` function when invoked with 2 arguments.

- `TIME_FORMAT(time,format)`

  This is used like the `DATE_FORMAT()` function, but the *`format`* string may contain format specifiers only for hours, minutes, seconds, and microseconds. Other specifiers produce a `NULL` value or `0`.

  If the *`time`* value contains an hour part that is greater than `23`, the `%H` and `%k` hour format specifiers produce a value larger than the usual range of `0..23`. The other hour format specifiers produce the hour value modulo 12.

```
mysql> SELECT TIME_FORMAT('100:00:00', '%H %k %h %I %l');
        -> '100 100 04 04 4'
```

- `TIME_TO_SEC(time)`

  Returns the *`time`* argument, converted to seconds.

```
mysql> SELECT TIME_TO_SEC('22:23:00');
```

```
        -> 80580
mysql> SELECT TIME_TO_SEC('00:39:38');
        -> 2378
```

- `TO_DAYS(date)`

  Given a date *`date`*, returns a day number (the number of days since year 0).

```
mysql> SELECT TO_DAYS(950501);
        -> 728779
mysql> SELECT TO_DAYS('2007-10-07');
        -> 733321
```

  `TO_DAYS()` is not intended for use with values that precede the advent of the Gregorian calendar (1582), because it does not take into account the days that were lost when the calendar was changed. For dates before 1582 (and possibly a later year in other locales), results from this function are not reliable. See Section 12.9, “What Calendar Is Used By MySQL?”, for details.

  Remember that MySQL converts two-digit year values in dates to four-digit form using the rules in Section 11.2, “Date and Time Data Types”. For example, `'2008-10-07'` and `'08-10-07'` are seen as identical dates:

```
mysql> SELECT TO_DAYS('2008-10-07'), TO_DAYS('08-10-07');
        -> 733687, 733687
```

  In MySQL, the zero date is defined as `'0000-00-00'`, even though this date is itself considered invalid. This means that, for `'0000-00-00'` and `'0000-01-01'`, `TO_DAYS()` returns the values shown here:

```
mysql> SELECT TO_DAYS('0000-00-00');
+-----------------------+
| to_days('0000-00-00') |
+-----------------------+
|                  NULL |
+-----------------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1292 | Incorrect datetime value: '0000-00-00' |
+---------+------+----------------------------------------+
1 row in set (0.00 sec)


mysql> SELECT TO_DAYS('0000-01-01');
+-----------------------+
| to_days('0000-01-01') |
+-----------------------+
|                     1 |
+-----------------------+
1 row in set (0.00 sec)
```

  This is true whether or not the `ALLOW_INVALID_DATES` SQL server mode is enabled.

- `TO_SECONDS(expr)`

  Given a date or datetime *`expr`*, returns the number of seconds since the year 0. If *`expr`* is not a valid date or datetime value, returns `NULL`.

```
mysql> SELECT TO_SECONDS(950501);
        -> 62966505600
mysql> SELECT TO_SECONDS('2009-11-29');
        -> 63426672000
mysql> SELECT TO_SECONDS('2009-11-29 13:43:32');
        -> 63426721412
mysql> SELECT TO_SECONDS( NOW() );
```

```
        -> 63426721458
```

Like `TO_DAYS()`, `TO_SECONDS()` is not intended for use with values that precede the advent of the Gregorian calendar (1582), because it does not take into account the days that were lost when the calendar was changed. For dates before 1582 (and possibly a later year in other locales), results from this function are not reliable. See Section 12.9, “What Calendar Is Used By MySQL?”, for details.

Like `TO_DAYS()`, `TO_SECONDS()`, converts two-digit year values in dates to four-digit form using the rules in Section 11.2, “Date and Time Data Types”.

In MySQL, the zero date is defined as `'0000-00-00'`, even though this date is itself considered invalid. This means that, for `'0000-00-00'` and `'0000-01-01'`, `TO_SECONDS()` returns the values shown here:

```
mysql> SELECT TO_SECONDS('0000-00-00');
+--------------------------+
| TO_SECONDS('0000-00-00') |
+--------------------------+
|                     NULL |
+--------------------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+---------+------+---------------------------------------+
| Level   | Code | Message                               |
+---------+------+---------------------------------------+
| Warning | 1292 | Incorrect datetime value: '0000-00-00' |
+---------+------+---------------------------------------+
1 row in set (0.00 sec)


mysql> SELECT TO_SECONDS('0000-01-01');
+--------------------------+
| TO_SECONDS('0000-01-01') |
+--------------------------+
|                    86400 |
+--------------------------+
1 row in set (0.00 sec)
```

This is true whether or not the `ALLOW_INVALID_DATES` SQL server mode is enabled.

- `UNIX_TIMESTAMP([`*`date`*`])`

  If `UNIX_TIMESTAMP()` is called with no *`date`* argument, it returns a Unix timestamp representing seconds since `'1970-01-01 00:00:00'` UTC.

  If `UNIX_TIMESTAMP()` is called with a *`date`* argument, it returns the value of the argument as seconds since `'1970-01-01 00:00:00'` UTC. The server interprets *`date`* as a value in the session time zone and converts it to an internal Unix timestamp value in UTC. (Clients can set the session time zone as described in Section 5.1.13, “MySQL Server Time Zone Support”.) The *`date`* argument may be a `DATE`, `DATETIME`, or `TIMESTAMP` string, or a number in *`YYMMDD`*,

YYMMDDhhmmss, *YYYYMMDD*, or *YYYYMMDDhhmmss* format. If the argument includes a time part, it may optionally include a fractional seconds part.

The return value is an integer if no argument is given or the argument does not include a fractional seconds part, or DECIMAL if an argument is given that includes a fractional seconds part.

When the *date* argument is a TIMESTAMP column, UNIX_TIMESTAMP() returns the internal timestamp value directly, with no implicit "string-to-Unix-timestamp" conversion.

The valid range of argument values is the same as for the TIMESTAMP data type: '1970-01-01 00:00:01.000000' UTC to '2038-01-19 03:14:07.999999' UTC. If you pass an out-of-range date to UNIX_TIMESTAMP(), it returns 0.

```
mysql> SELECT UNIX_TIMESTAMP();
        -> 1447431666
mysql> SELECT UNIX_TIMESTAMP('2015-11-13 10:20:19');
        -> 1447431619
mysql> SELECT UNIX_TIMESTAMP('2015-11-13 10:20:19.012');
        -> 1447431619.012
```

If you use UNIX_TIMESTAMP() and FROM_UNIXTIME() to convert between values in a non-UTC time zone and Unix timestamp values, the conversion is lossy because the mapping is not one-to-one in both directions. For example, due to conventions for local time zone changes such as Daylight Saving Time (DST), it is possible for UNIX_TIMESTAMP() to map two values that are distinct in a non-UTC time zone to the same Unix timestamp value. FROM_UNIXTIME() maps that value back to only one of the original values. Here is an example, using values that are distinct in the MET time zone:

```
mysql> SET time_zone = 'MET';
mysql> SELECT UNIX_TIMESTAMP('2005-03-27 03:00:00');
+---------------------------------------+
| UNIX_TIMESTAMP('2005-03-27 03:00:00') |
+---------------------------------------+
|                            1111885200 |
+---------------------------------------+
mysql> SELECT UNIX_TIMESTAMP('2005-03-27 02:00:00');
+---------------------------------------+
| UNIX_TIMESTAMP('2005-03-27 02:00:00') |
+---------------------------------------+
|                            1111885200 |
+---------------------------------------+
mysql> SELECT FROM_UNIXTIME(1111885200);
+---------------------------+
| FROM_UNIXTIME(1111885200) |
+---------------------------+
| 2005-03-27 03:00:00       |
+---------------------------+
```

Note

To use named time zones such as 'MET' or 'Europe/Amsterdam', the time zone tables must be properly set up. For instructions, see Section 5.1.13, "MySQL Server Time Zone Support".

If you want to subtract UNIX_TIMESTAMP() columns, you might want to cast them to signed integers. See Section 12.11, "Cast Functions and Operators".

- UTC_DATE, UTC_DATE()

  Returns the current UTC date as a value in *'YYYY-MM-DD'* or *YYYYMMDD* format, depending on whether the function is used in string or numeric context.

```
mysql> SELECT UTC_DATE(), UTC_DATE() + 0;
        -> '2003-08-14', 20030814
```

- `UTC_TIME`, `UTC_TIME([fsp])`

  Returns the current UTC time as a value in `'hh:mm:ss'` or `hhmmss` format, depending on whether the function is used in string or numeric context.

  If the `fsp` argument is given to specify a fractional seconds precision from 0 to 6, the return value includes a fractional seconds part of that many digits.

```
mysql> SELECT UTC_TIME(), UTC_TIME() + 0;
        -> '18:07:53', 180753.000000
```

- `UTC_TIMESTAMP`, `UTC_TIMESTAMP([fsp])`

  Returns the current UTC date and time as a value in `'YYYY-MM-DD hh:mm:ss'` or `YYYYMMDDhhmmss` format, depending on whether the function is used in string or numeric context.

  If the `fsp` argument is given to specify a fractional seconds precision from 0 to 6, the return value includes a fractional seconds part of that many digits.

```
mysql> SELECT UTC_TIMESTAMP(), UTC_TIMESTAMP() + 0;
        -> '2003-08-14 18:08:04', 20030814180804.000000
```

- `WEEK(date[,mode])`

  This function returns the week number for `date`. The two-argument form of `WEEK()` enables you to specify whether the week starts on Sunday or Monday and whether the return value should be in the range from `0` to `53` or from `1` to `53`. If the `mode` argument is omitted, the value of the `default_week_format` system variable is used. See Section 5.1.7, “Server System Variables”.

  The following table describes how the `mode` argument works.

Mode	First day of week	Range	Week 1 is the first week …
0	Sunday	0-53	with a Sunday in this year
1	Monday	0-53	with 4 or more days this year
2	Sunday	1-53	with a Sunday in this year
3	Monday	1-53	with 4 or more days this year
4	Sunday	0-53	with 4 or more days this year
5	Monday	0-53	with a Monday in this year
6	Sunday	1-53	with 4 or more days this year
7	Monday	1-53	with a Monday in this year

  For `mode` values with a meaning of “with 4 or more days this year,” weeks are numbered according to ISO 8601:1988:

  - If the week containing January 1 has 4 or more days in the new year, it is week 1.
  - Otherwise, it is the last week of the previous year, and the next week is week 1.

```
mysql> SELECT WEEK('2008-02-20');
        -> 7
mysql> SELECT WEEK('2008-02-20',0);
        -> 7
mysql> SELECT WEEK('2008-02-20',1);
        -> 8
mysql> SELECT WEEK('2008-12-31',1);
```

```
        -> 53
```

If a date falls in the last week of the previous year, MySQL returns `0` if you do not use `2`, `3`, `6`, or `7` as the optional *`mode`* argument:

```
mysql> SELECT YEAR('2000-01-01'), WEEK('2000-01-01',0);
        -> 2000, 0
```

One might argue that `WEEK()` should return `52` because the given date actually occurs in the 52nd week of 1999. `WEEK()` returns `0` instead so that the return value is “the week number in the given year.” This makes use of the `WEEK()` function reliable when combined with other functions that extract a date part from a date.

If you prefer a result evaluated with respect to the year that contains the first day of the week for the given date, use `0`, `2`, `5`, or `7` as the optional *`mode`* argument.

```
mysql> SELECT WEEK('2000-01-01',2);
        -> 52
```

Alternatively, use the `YEARWEEK()` function:

```
mysql> SELECT YEARWEEK('2000-01-01');
        -> 199952
mysql> SELECT MID(YEARWEEK('2000-01-01'),5,2);
        -> '52'
```

- `WEEKDAY(`*`date`*`)`

  Returns the weekday index for *`date`* (`0` = Monday, `1` = Tuesday, … `6` = Sunday).

  ```
  mysql> SELECT WEEKDAY('2008-02-03 22:23:00');
          -> 6
  mysql> SELECT WEEKDAY('2007-11-06');
          -> 1
  ```

- `WEEKOFYEAR(`*`date`*`)`

  Returns the calendar week of the date as a number in the range from `1` to `53`. `WEEKOFYEAR()` is a compatibility function that is equivalent to `WEEK(`*`date`*`,3)`.

  ```
  mysql> SELECT WEEKOFYEAR('2008-02-20');
          -> 8
  ```

- `YEAR(`*`date`*`)`

  Returns the year for *`date`*, in the range `1000` to `9999`, or `0` for the “zero” date.

  ```
  mysql> SELECT YEAR('1987-01-01');
          -> 1987
  ```

- `YEARWEEK(`*`date`*`)`, `YEARWEEK(`*`date`*`,`*`mode`*`)`

  Returns year and week for a date. The year in the result may be different from the year in the date argument for the first and the last week of the year.

  The *`mode`* argument works exactly like the *`mode`* argument to `WEEK()`. For the single-argument syntax, a *`mode`* value of 0 is used. Unlike `WEEK()`, the value of `default_week_format` does not influence `YEARWEEK()`.

  ```
  mysql> SELECT YEARWEEK('1987-01-01');
          -> 198652
  ```

  The week number is different from what the `WEEK()` function would return (`0`) for optional arguments `0` or `1`, as `WEEK()` then returns the week in the context of the given year.

12.8 String Functions and Operators

Table 12.12 String Functions and Operators

Name	Description
ASCII()	Return numeric value of left-most character
BIN()	Return a string containing binary representation of a number
BIT_LENGTH()	Return length of argument in bits
CHAR()	Return the character for each integer passed
CHAR_LENGTH()	Return number of characters in argument
CHARACTER_LENGTH()	Synonym for CHAR_LENGTH()
CONCAT()	Return concatenated string
CONCAT_WS()	Return concatenate with separator
ELT()	Return string at index number
EXPORT_SET()	Return a string such that for every bit set in the value bits, you get an on string and for every unset bit, you get an off string
FIELD()	Index (position) of first argument in subsequent arguments
FIND_IN_SET()	Index (position) of first argument within second argument
FORMAT()	Return a number formatted to specified number of decimal places
FROM_BASE64()	Decode base64 encoded string and return result
HEX()	Hexadecimal representation of decimal or string value
INSERT()	Insert substring at specified position up to specified number of characters
INSTR()	Return the index of the first occurrence of substring
LCASE()	Synonym for LOWER()
LEFT()	Return the leftmost number of characters as specified
LENGTH()	Return the length of a string in bytes
LIKE	Simple pattern matching
LOAD_FILE()	Load the named file
LOCATE()	Return the position of the first occurrence of substring
LOWER()	Return the argument in lowercase
LPAD()	Return the string argument, left-padded with the specified string
LTRIM()	Remove leading spaces
MAKE_SET()	Return a set of comma-separated strings that have the corresponding bit in bits set
MATCH	Perform full-text search
MID()	Return a substring starting from the specified position
NOT LIKE	Negation of simple pattern matching
NOT REGEXP	Negation of REGEXP
OCT()	Return a string containing octal representation of a number
OCTET_LENGTH()	Synonym for LENGTH()
ORD()	Return character code for leftmost character of the argument

Name	Description
POSITION()	Synonym for LOCATE()
QUOTE()	Escape the argument for use in an SQL statement
REGEXP	Whether string matches regular expression
REPEAT()	Repeat a string the specified number of times
REPLACE()	Replace occurrences of a specified string
REVERSE()	Reverse the characters in a string
RIGHT()	Return the specified rightmost number of characters
RLIKE	Whether string matches regular expression
RPAD()	Append string the specified number of times
RTRIM()	Remove trailing spaces
SOUNDEX()	Return a soundex string
SOUNDS LIKE	Compare sounds
SPACE()	Return a string of the specified number of spaces
STRCMP()	Compare two strings
SUBSTR()	Return the substring as specified
SUBSTRING()	Return the substring as specified
SUBSTRING_INDEX()	Return a substring from a string before the specified number of occurrences of the delimiter
TO_BASE64()	Return the argument converted to a base-64 string
TRIM()	Remove leading and trailing spaces
UCASE()	Synonym for UPPER()
UNHEX()	Return a string containing hex representation of a number
UPPER()	Convert to uppercase
WEIGHT_STRING()	Return the weight string for a string

String-valued functions return NULL if the length of the result would be greater than the value of the max_allowed_packet system variable. See Section 5.1.1, “Configuring the Server”.

For functions that operate on string positions, the first position is numbered 1.

For functions that take length arguments, noninteger arguments are rounded to the nearest integer.

- ASCII(*str*)

  Returns the numeric value of the leftmost character of the string *str*. Returns 0 if *str* is the empty string. Returns NULL if *str* is NULL. ASCII() works for 8-bit characters.

```
mysql> SELECT ASCII('2');
        -> 50
mysql> SELECT ASCII(2);
        -> 50
mysql> SELECT ASCII('dx');
        -> 100
```

  See also the ORD() function.

- BIN(*N*)

  Returns a string representation of the binary value of *N*, where *N* is a longlong (BIGINT) number. This is equivalent to CONV(*N*,10,2). Returns NULL if *N* is NULL.

```
mysql> SELECT BIN(12);
```

```
        -> '1100'
```

- `BIT_LENGTH(str)`

  Returns the length of the string *`str`* in bits.

```
mysql> SELECT BIT_LENGTH('text');
        -> 32
```

- `CHAR(N,... [USING charset_name])`

  `CHAR()` interprets each argument *`N`* as an integer and returns a string consisting of the characters given by the code values of those integers. `NULL` values are skipped.

```
mysql> SELECT CHAR(77,121,83,81,'76');
        -> 'MySQL'
mysql> SELECT CHAR(77,77.3,'77.3');
        -> 'MMM'
```

  `CHAR()` arguments larger than 255 are converted into multiple result bytes. For example, `CHAR(256)` is equivalent to `CHAR(1,0)`, and `CHAR(256*256)` is equivalent to `CHAR(1,0,0)`:

```
mysql> SELECT HEX(CHAR(1,0)), HEX(CHAR(256));
+----------------+----------------+
| HEX(CHAR(1,0)) | HEX(CHAR(256)) |
+----------------+----------------+
| 0100           | 0100           |
+----------------+----------------+
mysql> SELECT HEX(CHAR(1,0,0)), HEX(CHAR(256*256));
+------------------+--------------------+
| HEX(CHAR(1,0,0)) | HEX(CHAR(256*256)) |
+------------------+--------------------+
| 010000           | 010000             |
+------------------+--------------------+
```

  By default, `CHAR()` returns a binary string. To produce a string in a given character set, use the optional `USING` clause:

```
mysql> SELECT CHARSET(CHAR(X'65')), CHARSET(CHAR(X'65' USING utf8));
+----------------------+---------------------------------+
| CHARSET(CHAR(X'65')) | CHARSET(CHAR(X'65' USING utf8)) |
+----------------------+---------------------------------+
| binary               | utf8                            |
+----------------------+---------------------------------+
```

  If `USING` is given and the result string is illegal for the given character set, a warning is issued. Also, if strict SQL mode is enabled, the result from `CHAR()` becomes `NULL`.

- `CHAR_LENGTH(str)`

  Returns the length of the string *`str`*, measured in characters. A multibyte character counts as a single character. This means that for a string containing five 2-byte characters, `LENGTH()` returns `10`, whereas `CHAR_LENGTH()` returns `5`.

- `CHARACTER_LENGTH(str)`

  `CHARACTER_LENGTH()` is a synonym for `CHAR_LENGTH()`.

- `CONCAT(str1,str2,...)`

  Returns the string that results from concatenating the arguments. May have one or more arguments. If all arguments are nonbinary strings, the result is a nonbinary string. If the arguments include

any binary strings, the result is a binary string. A numeric argument is converted to its equivalent nonbinary string form.

`CONCAT()` returns `NULL` if any argument is `NULL`.

```
mysql> SELECT CONCAT('My', 'S', 'QL');
        -> 'MySQL'
mysql> SELECT CONCAT('My', NULL, 'QL');
        -> NULL
mysql> SELECT CONCAT(14.3);
        -> '14.3'
```

For quoted strings, concatenation can be performed by placing the strings next to each other:

```
mysql> SELECT 'My' 'S' 'QL';
        -> 'MySQL'
```

- `CONCAT_WS(`*`separator`*`,`*`str1`*`,`*`str2`*`,...)`

  `CONCAT_WS()` stands for Concatenate With Separator and is a special form of `CONCAT()`. The first argument is the separator for the rest of the arguments. The separator is added between the strings to be concatenated. The separator can be a string, as can the rest of the arguments. If the separator is `NULL`, the result is `NULL`.

  ```
  mysql> SELECT CONCAT_WS(',','First name','Second name','Last Name');
          -> 'First name,Second name,Last Name'
  mysql> SELECT CONCAT_WS(',','First name',NULL,'Last Name');
          -> 'First name,Last Name'
  ```

  `CONCAT_WS()` does not skip empty strings. However, it does skip any `NULL` values after the separator argument.

- `ELT(`*`N`*`,`*`str1`*`,`*`str2`*`,`*`str3`*`,...)`

  `ELT()` returns the *`N`*th element of the list of strings: *`str1`* if *`N`* = `1`, *`str2`* if *`N`* = `2`, and so on. Returns `NULL` if *`N`* is less than `1` or greater than the number of arguments. `ELT()` is the complement of `FIELD()`.

  ```
  mysql> SELECT ELT(1, 'Aa', 'Bb', 'Cc', 'Dd');
          -> 'Aa'
  mysql> SELECT ELT(4, 'Aa', 'Bb', 'Cc', 'Dd');
          -> 'Dd'
  ```

- `EXPORT_SET(`*`bits`*`,`*`on`*`,`*`off`*`[,`*`separator`*`[,`*`number_of_bits`*`]])`

  Returns a string such that for every bit set in the value *`bits`*, you get an *`on`* string and for every bit not set in the value, you get an *`off`* string. Bits in *`bits`* are examined from right to left (from low-order to high-order bits). Strings are added to the result from left to right, separated by the *`separator`* string (the default being the comma character `,`). The number of bits examined is given by *`number_of_bits`*, which has a default of 64 if not specified. *`number_of_bits`* is silently clipped to 64 if larger than 64. It is treated as an unsigned integer, so a value of −1 is effectively the same as 64.

  ```
  mysql> SELECT EXPORT_SET(5,'Y','N',',',4);
          -> 'Y,N,Y,N'
  mysql> SELECT EXPORT_SET(6,'1','0',',',10);
          -> '0,1,1,0,0,0,0,0,0,0'
  ```

- `FIELD(str,str1,str2,str3,...)`

  Returns the index (position) of *`str`* in the *`str1`*, *`str2`*, *`str3`*, `...` list. Returns `0` if *`str`* is not found.

  If all arguments to `FIELD()` are strings, all arguments are compared as strings. If all arguments are numbers, they are compared as numbers. Otherwise, the arguments are compared as double.

  If *`str`* is `NULL`, the return value is `0` because `NULL` fails equality comparison with any value. `FIELD()` is the complement of `ELT()`.

```
mysql> SELECT FIELD('Bb', 'Aa', 'Bb', 'Cc', 'Dd', 'Ff');
        -> 2
mysql> SELECT FIELD('Gg', 'Aa', 'Bb', 'Cc', 'Dd', 'Ff');
        -> 0
```

- `FIND_IN_SET(str,strlist)`

  Returns a value in the range of 1 to *`N`* if the string *`str`* is in the string list *`strlist`* consisting of *`N`* substrings. A string list is a string composed of substrings separated by `,` characters. If the first argument is a constant string and the second is a column of type `SET`, the `FIND_IN_SET()` function is optimized to use bit arithmetic. Returns `0` if *`str`* is not in *`strlist`* or if *`strlist`* is the empty string. Returns `NULL` if either argument is `NULL`. This function does not work properly if the first argument contains a comma (`,`) character.

```
mysql> SELECT FIND_IN_SET('b','a,b,c,d');
        -> 2
```

- `FORMAT(X,D[,locale])`

  Formats the number *`X`* to a format like `'#,###,###.##'`, rounded to *`D`* decimal places, and returns the result as a string. If *`D`* is `0`, the result has no decimal point or fractional part.

  The optional third parameter enables a locale to be specified to be used for the result number's decimal point, thousands separator, and grouping between separators. Permissible locale values are the same as the legal values for the `lc_time_names` system variable (see Section 10.16, “MySQL Server Locale Support”). If no locale is specified, the default is `'en_US'`.

```
mysql> SELECT FORMAT(12332.123456, 4);
        -> '12,332.1235'
mysql> SELECT FORMAT(12332.1,4);
        -> '12,332.1000'
mysql> SELECT FORMAT(12332.2,0);
        -> '12,332'
mysql> SELECT FORMAT(12332.2,2,'de_DE');
        -> '12.332,20'
```

- `FROM_BASE64(str)`

  Takes a string encoded with the base-64 encoded rules used by `TO_BASE64()` and returns the decoded result as a binary string. The result is `NULL` if the argument is `NULL` or not a valid base-64 string. See the description of `TO_BASE64()` for details about the encoding and decoding rules.

```
mysql> SELECT TO_BASE64('abc'), FROM_BASE64(TO_BASE64('abc'));
        -> 'JWJj', 'abc'
```

- `HEX(str)`, `HEX(N)`

  For a string argument *`str`*, `HEX()` returns a hexadecimal string representation of *`str`* where each byte of each character in *`str`* is converted to two hexadecimal digits. (Multibyte characters therefore become more than two digits.) The inverse of this operation is performed by the `UNHEX()` function.

  For a numeric argument *`N`*, `HEX()` returns a hexadecimal string representation of the value of *`N`* treated as a longlong (`BIGINT`) number. This is equivalent to `CONV(N,10,16)`. The inverse of this operation is performed by `CONV(HEX(N),16,10)`.

  ```
  mysql> SELECT X'616263', HEX('abc'), UNHEX(HEX('abc'));
         -> 'abc', 616263, 'abc'
  mysql> SELECT HEX(255), CONV(HEX(255),16,10);
         -> 'FF', 255
  ```

- `INSERT(str,pos,len,newstr)`

  Returns the string *`str`*, with the substring beginning at position *`pos`* and *`len`* characters long replaced by the string *`newstr`*. Returns the original string if *`pos`* is not within the length of the string. Replaces the rest of the string from position *`pos`* if *`len`* is not within the length of the rest of the string. Returns `NULL` if any argument is `NULL`.

  ```
  mysql> SELECT INSERT('Quadratic', 3, 4, 'What');
         -> 'QuWhattic'
  mysql> SELECT INSERT('Quadratic', -1, 4, 'What');
         -> 'Quadratic'
  mysql> SELECT INSERT('Quadratic', 3, 100, 'What');
         -> 'QuWhat'
  ```

  This function is multibyte safe.

- `INSTR(str,substr)`

  Returns the position of the first occurrence of substring *`substr`* in string *`str`*. This is the same as the two-argument form of `LOCATE()`, except that the order of the arguments is reversed.

  ```
  mysql> SELECT INSTR('foobarbar', 'bar');
         -> 4
  mysql> SELECT INSTR('xbar', 'foobar');
         -> 0
  ```

  This function is multibyte safe, and is case-sensitive only if at least one argument is a binary string.

- `LCASE(str)`

  `LCASE()` is a synonym for `LOWER()`.

- `LEFT(str,len)`

  Returns the leftmost *`len`* characters from the string *`str`*, or `NULL` if any argument is `NULL`.

  ```
  mysql> SELECT LEFT('foobarbar', 5);
         -> 'fooba'
  ```

  This function is multibyte safe.

- `LENGTH(str)`

  Returns the length of the string *`str`*, measured in bytes. A multibyte character counts as multiple bytes. This means that for a string containing five 2-byte characters, `LENGTH()` returns `10`, whereas `CHAR_LENGTH()` returns `5`.

  ```
  mysql> SELECT LENGTH('text');
  ```

```
        -> 4
```

Note

The `Length()` OpenGIS spatial function is named `GLength()` in MySQL.

- `LOAD_FILE(file_name)`

  Reads the file and returns the file contents as a string. To use this function, the file must be located on the server host, you must specify the full path name to the file, and you must have the `FILE` privilege. The file must be readable by all and its size less than `max_allowed_packet` bytes. If the `secure_file_priv` system variable is set to a nonempty directory name, the file to be loaded must be located in that directory.

  If the file does not exist or cannot be read because one of the preceding conditions is not satisfied, the function returns `NULL`.

  The `character_set_filesystem` system variable controls interpretation of file names that are given as literal strings.

```
mysql> UPDATE t
           SET blob_col=LOAD_FILE('/tmp/picture')
           WHERE id=1;
```

- `LOCATE(substr,str)`, `LOCATE(substr,str,pos)`

  The first syntax returns the position of the first occurrence of substring *`substr`* in string *`str`*. The second syntax returns the position of the first occurrence of substring *`substr`* in string *`str`*, starting at position *`pos`*. Returns `0` if *`substr`* is not in *`str`*. Returns `NULL` if *`substr`* or *`str`* is `NULL`.

```
mysql> SELECT LOCATE('bar', 'foobarbar');
        -> 4
mysql> SELECT LOCATE('xbar', 'foobar');
        -> 0
mysql> SELECT LOCATE('bar', 'foobarbar', 5);
        -> 7
```

  This function is multibyte safe, and is case-sensitive only if at least one argument is a binary string.

- `LOWER(str)`

  Returns the string *`str`* with all characters changed to lowercase according to the current character set mapping. The default is `latin1` (cp1252 West European).

```
mysql> SELECT LOWER('QUADRATICALLY');
        -> 'quadratically'
```

  `LOWER()` (and `UPPER()`) are ineffective when applied to binary strings (`BINARY`, `VARBINARY`, `BLOB`). To perform lettercase conversion of a binary string, first convert it to a nonbinary string using a character set appropriate for the data stored in the string:

```
mysql> SET @str = BINARY 'New York';
mysql> SELECT LOWER(@str), LOWER(CONVERT(@str USING latin1));
+-------------+-----------------------------------+
| LOWER(@str) | LOWER(CONVERT(@str USING latin1)) |
+-------------+-----------------------------------+
| New York    | new york                          |
+-------------+-----------------------------------+
```

  For collations of Unicode character sets, `LOWER()` and `UPPER()` work according to the Unicode Collation Algorithm (UCA) version in the collation name, if there is one, and UCA 4.0.0 if no version

is specified. For example, `utf8_unicode_520_ci` works according to UCA 5.2.0, whereas `utf8_unicode_ci` works according to UCA 4.0.0. See Section 10.10.1, “Unicode Character Sets”.

This function is multibyte safe.

- `LPAD(str,len,padstr)`

  Returns the string *`str`*, left-padded with the string *`padstr`* to a length of *`len`* characters. If *`str`* is longer than *`len`*, the return value is shortened to *`len`* characters.

```
mysql> SELECT LPAD('hi',4,'??');
        -> '??hi'
mysql> SELECT LPAD('hi',1,'??');
        -> 'h'
```

- `LTRIM(str)`

  Returns the string *`str`* with leading space characters removed.

```
mysql> SELECT LTRIM('  barbar');
        -> 'barbar'
```

  This function is multibyte safe.

- `MAKE_SET(bits,str1,str2,...)`

  Returns a set value (a string containing substrings separated by `,` characters) consisting of the strings that have the corresponding bit in *`bits`* set. *`str1`* corresponds to bit 0, *`str2`* to bit 1, and so on. `NULL` values in *`str1`*, *`str2`*, `...` are not appended to the result.

```
mysql> SELECT MAKE_SET(1,'a','b','c');
        -> 'a'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice','world');
        -> 'hello,world'
mysql> SELECT MAKE_SET(1 | 4,'hello','nice',NULL,'world');
        -> 'hello'
mysql> SELECT MAKE_SET(0,'a','b','c');
        -> ''
```

- `MID(str,pos,len)`

  `MID(str,pos,len)` is a synonym for `SUBSTRING(str,pos,len)`.

- `OCT(N)`

  Returns a string representation of the octal value of *`N`*, where *`N`* is a longlong (`BIGINT`) number. This is equivalent to `CONV(N,10,8)`. Returns `NULL` if *`N`* is `NULL`.

```
mysql> SELECT OCT(12);
        -> '14'
```

- `OCTET_LENGTH(str)`

  `OCTET_LENGTH()` is a synonym for `LENGTH()`.

- `ORD(str)`

  If the leftmost character of the string *`str`* is a multibyte character, returns the code for that character, calculated from the numeric values of its constituent bytes using this formula:

```
  (1st byte code)
+ (2nd byte code * 256)
+ (3rd byte code * 256^2) ...
```

  If the leftmost character is not a multibyte character, `ORD()` returns the same value as the `ASCII()` function.

```
mysql> SELECT ORD('2');
        -> 50
```

- `POSITION(substr IN str)`

  `POSITION(substr IN str)` is a synonym for `LOCATE(substr,str)`.

- `QUOTE(str)`

  Quotes a string to produce a result that can be used as a properly escaped data value in an SQL statement. The string is returned enclosed by single quotation marks and with each instance of backslash (\), single quote ('), ASCII `NUL`, and Control+Z preceded by a backslash. If the argument is `NULL`, the return value is the word "NULL" without enclosing single quotation marks.

  ```
  mysql> SELECT QUOTE('Don\'t!');
          -> 'Don\'t!'
  mysql> SELECT QUOTE(NULL);
          -> NULL
  ```

  For comparison, see the quoting rules for literal strings and within the C API in Section 9.1.1, "String Literals", and mysql_real_escape_string().

- `REPEAT(str,count)`

  Returns a string consisting of the string *`str`* repeated *`count`* times. If *`count`* is less than 1, returns an empty string. Returns `NULL` if *`str`* or *`count`* are `NULL`.

  ```
  mysql> SELECT REPEAT('MySQL', 3);
          -> 'MySQLMySQLMySQL'
  ```

- `REPLACE(str,from_str,to_str)`

  Returns the string *`str`* with all occurrences of the string *`from_str`* replaced by the string *`to_str`*. `REPLACE()` performs a case-sensitive match when searching for *`from_str`*.

  ```
  mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
          -> 'WwWwWw.mysql.com'
  ```

  This function is multibyte safe.

- `REVERSE(str)`

  Returns the string *`str`* with the order of the characters reversed.

  ```
  mysql> SELECT REVERSE('abc');
          -> 'cba'
  ```

  This function is multibyte safe.

- `RIGHT(str,len)`

  Returns the rightmost *`len`* characters from the string *`str`*, or `NULL` if any argument is `NULL`.

  ```
  mysql> SELECT RIGHT('foobarbar', 4);
          -> 'rbar'
  ```

  This function is multibyte safe.

- `RPAD(str,len,padstr)`

  Returns the string *`str`*, right-padded with the string *`padstr`* to a length of *`len`* characters. If *`str`* is longer than *`len`*, the return value is shortened to *`len`* characters.

  ```
  mysql> SELECT RPAD('hi',5,'?');
          -> 'hi???'
  ```

```
mysql> SELECT RPAD('hi',1,'?');
        -> 'h'
```

This function is multibyte safe.

- `RTRIM(str)`

  Returns the string *`str`* with trailing space characters removed.

```
mysql> SELECT RTRIM('barbar   ');
        -> 'barbar'
```

  This function is multibyte safe.

- `SOUNDEX(str)`

  Returns a soundex string from *`str`*. Two strings that sound almost the same should have identical soundex strings. A standard soundex string is four characters long, but the `SOUNDEX()` function returns an arbitrarily long string. You can use `SUBSTRING()` on the result to get a standard soundex string. All nonalphabetic characters in *`str`* are ignored. All international alphabetic characters outside the A-Z range are treated as vowels.

Important

When using `SOUNDEX()`, you should be aware of the following limitations:

  - This function, as currently implemented, is intended to work well with strings that are in the English language only. Strings in other languages may not produce reliable results.
  - This function is not guaranteed to provide consistent results with strings that use multibyte character sets, including `utf-8`. See Bug #22638 for more information.

```
mysql> SELECT SOUNDEX('Hello');
        -> 'H400'
mysql> SELECT SOUNDEX('Quadratically');
        -> 'Q36324'
```

Note

This function implements the original Soundex algorithm, not the more popular enhanced version (also described by D. Knuth). The difference is that original version discards vowels first and duplicates second, whereas the enhanced version discards duplicates first and vowels second.

- *`expr1`* `SOUNDS LIKE` *`expr2`*

  This is the same as `SOUNDEX(`*`expr1`*`) = SOUNDEX(`*`expr2`*`)`.

- `SPACE(N)`

  Returns a string consisting of *`N`* space characters.

```
mysql> SELECT SPACE(6);
        -> '      '
```

- `SUBSTR(str,pos)`, `SUBSTR(str FROM pos)`, `SUBSTR(str,pos,len)`, `SUBSTR(str FROM pos FOR len)`

  `SUBSTR()` is a synonym for `SUBSTRING()`.

- `SUBSTRING(str,pos)`, `SUBSTRING(str FROM pos)`, `SUBSTRING(str,pos,len)`, `SUBSTRING(str FROM pos FOR len)`

  The forms without a *`len`* argument return a substring from string *`str`* starting at position *`pos`*. The forms with a *`len`* argument return a substring *`len`* characters long from string *`str`*, starting at position *`pos`*. The forms that use `FROM` are standard SQL syntax. It is also possible to use a negative value for *`pos`*. In this case, the beginning of the substring is *`pos`* characters from the end of the string, rather than the beginning. A negative value may be used for *`pos`* in any of the forms of this function. A value of 0 for *`pos`* returns an empty string.

  For all forms of `SUBSTRING()`, the position of the first character in the string from which the substring is to be extracted is reckoned as `1`.

```
mysql> SELECT SUBSTRING('Quadratically',5);
        -> 'ratically'
mysql> SELECT SUBSTRING('foobarbar' FROM 4);
        -> 'barbar'
mysql> SELECT SUBSTRING('Quadratically',5,6);
        -> 'ratica'
mysql> SELECT SUBSTRING('Sakila', -3);
        -> 'ila'
mysql> SELECT SUBSTRING('Sakila', -5, 3);
        -> 'aki'
mysql> SELECT SUBSTRING('Sakila' FROM -4 FOR 2);
        -> 'ki'
```

  This function is multibyte safe.

  If *`len`* is less than 1, the result is the empty string.

- `SUBSTRING_INDEX(str,delim,count)`

  Returns the substring from string *`str`* before *`count`* occurrences of the delimiter *`delim`*. If *`count`* is positive, everything to the left of the final delimiter (counting from the left) is returned. If *`count`* is negative, everything to the right of the final delimiter (counting from the right) is returned. `SUBSTRING_INDEX()` performs a case-sensitive match when searching for *`delim`*.

```
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', 2);
        -> 'www.mysql'
mysql> SELECT SUBSTRING_INDEX('www.mysql.com', '.', -2);
        -> 'mysql.com'
```

  This function is multibyte safe.

- `TO_BASE64(str)`

  Converts the string argument to base-64 encoded form and returns the result as a character string with the connection character set and collation. If the argument is not a string, it is converted to a string before conversion takes place. The result is `NULL` if the argument is `NULL`. Base-64 encoded strings can be decoded using the `FROM_BASE64()` function.

```
mysql> SELECT TO_BASE64('abc'), FROM_BASE64(TO_BASE64('abc'));
```

```
        -> 'JWJj', 'abc'
```

Different base-64 encoding schemes exist. These are the encoding and decoding rules used by `TO_BASE64()` and `FROM_BASE64()`:

- The encoding for alphabet value 62 is `'+'`.
- The encoding for alphabet value 63 is `'/'`.
- Encoded output consists of groups of 4 printable characters. Each 3 bytes of the input data are encoded using 4 characters. If the last group is incomplete, it is padded with `'='` characters to a length of 4.
- A newline is added after each 76 characters of encoded output to divide long output into multiple lines.
- Decoding recognizes and ignores newline, carriage return, tab, and space.

- `TRIM([{BOTH | LEADING | TRAILING} [`*`remstr`*`] FROM] `*`str`*`)`, `TRIM([`*`remstr`*` FROM] `*`str`*`)`

  Returns the string *`str`* with all *`remstr`* prefixes or suffixes removed. If none of the specifiers `BOTH`, `LEADING`, or `TRAILING` is given, `BOTH` is assumed. *`remstr`* is optional and, if not specified, spaces are removed.

```
mysql> SELECT TRIM('  bar   ');
        -> 'bar'
mysql> SELECT TRIM(LEADING 'x' FROM 'xxxbarxxx');
        -> 'barxxx'
mysql> SELECT TRIM(BOTH 'x' FROM 'xxxbarxxx');
        -> 'bar'
mysql> SELECT TRIM(TRAILING 'xyz' FROM 'barxxyz');
        -> 'barx'
```

  This function is multibyte safe.

- `UCASE(`*`str`*`)`

  `UCASE()` is a synonym for `UPPER()`.

- `UNHEX(`*`str`*`)`

  For a string argument *`str`*, `UNHEX(`*`str`*`)` interprets each pair of characters in the argument as a hexadecimal number and converts it to the byte represented by the number. The return value is a binary string.

```
mysql> SELECT UNHEX('4D7953514C');
        -> 'MySQL'
mysql> SELECT X'4D7953514C';
        -> 'MySQL'
mysql> SELECT UNHEX(HEX('string'));
        -> 'string'
mysql> SELECT HEX(UNHEX('1267'));
        -> '1267'
```

  The characters in the argument string must be legal hexadecimal digits: `'0'` .. `'9'`, `'A'` .. `'F'`, `'a'` .. `'f'`. If the argument contains any nonhexadecimal digits, the result is `NULL`:

```
mysql> SELECT UNHEX('GG');
+-------------+
| UNHEX('GG') |
+-------------+
| NULL        |
```

```
+-------------+
```

A `NULL` result can occur if the argument to `UNHEX()` is a `BINARY` column, because values are padded with `0x00` bytes when stored but those bytes are not stripped on retrieval. For example, `'41'` is stored into a `CHAR(3)` column as `'41 '` and retrieved as `'41'` (with the trailing pad space stripped), so `UNHEX()` for the column value returns `X'41'`. By contrast, `'41'` is stored into a `BINARY(3)` column as `'41\0'` and retrieved as `'41\0'` (with the trailing pad `0x00` byte not stripped). `'\0'` is not a legal hexadecimal digit, so `UNHEX()` for the column value returns `NULL`.

For a numeric argument *`N`*, the inverse of `HEX(N)` is not performed by `UNHEX()`. Use `CONV(HEX(N),16,10)` instead. See the description of `HEX()`.

- `UPPER(str)`

  Returns the string *`str`* with all characters changed to uppercase according to the current character set mapping. The default is `latin1` (cp1252 West European).

```
mysql> SELECT UPPER('Hej');
        -> 'HEJ'
```

  See the description of `LOWER()` for information that also applies to `UPPER()`. This included information about how to perform lettercase conversion of binary strings (`BINARY`, `VARBINARY`, `BLOB`) for which these functions are ineffective, and information about case folding for Unicode character sets.

  This function is multibyte safe.

- `WEIGHT_STRING(str [AS {CHAR|BINARY}(N)] [LEVEL levels] [flags])`

  `levels: N [ASC|DESC|REVERSE] [, N [ASC|DESC|REVERSE]] ...`

  This function returns the weight string for the input string. The return value is a binary string that represents the comparison and sorting value of the string. It has these properties:

  - If `WEIGHT_STRING(str1)` = `WEIGHT_STRING(str2)`, then *`str1 = str2`* (*`str1`* and *`str2`* are considered equal)
  - If `WEIGHT_STRING(str1)` < `WEIGHT_STRING(str2)`, then *`str1 < str2`* (*`str1`* sorts before *`str2`*)

  `WEIGHT_STRING()` is a debugging function intended for internal use. Its behavior can change without notice between MySQL versions. It can be used for testing and debugging of collations, especially if you are adding a new collation. See Section 10.14, “Adding a Collation to a Character Set”.

  This list briefly summarizes the arguments. More details are given in the discussion following the list.

  - *`str`*: The input string expression.
  - `AS` clause: Optional; cast the input string to a given type and length.
  - `LEVEL` clause: Optional; specify weight levels for the return value.
  - *`flags`*: Optional; unused.

  The input string, *`str`*, is a string expression. If the input is a nonbinary (character) string such as a `CHAR`, `VARCHAR`, or `TEXT` value, the return value contains the collation weights for the string. If the input is a binary (byte) string such as a `BINARY`, `VARBINARY`, or `BLOB` value, the return value is the

same as the input (the weight for each byte in a binary string is the byte value). If the input is `NULL`, `WEIGHT_STRING()` returns `NULL`.

Examples:

```
mysql> SET @s = _latin1 'AB' COLLATE latin1_swedish_ci;
mysql> SELECT @s, HEX(@s), HEX(WEIGHT_STRING(@s));
+------+---------+-----------------------+
| @s   | HEX(@s) | HEX(WEIGHT_STRING(@s)) |
+------+---------+-----------------------+
| AB   | 4142    | 4142                  |
+------+---------+-----------------------+
```

```
mysql> SET @s = _latin1 'ab' COLLATE latin1_swedish_ci;
mysql> SELECT @s, HEX(@s), HEX(WEIGHT_STRING(@s));
+------+---------+-----------------------+
| @s   | HEX(@s) | HEX(WEIGHT_STRING(@s)) |
+------+---------+-----------------------+
| ab   | 6162    | 4142                  |
+------+---------+-----------------------+
```

```
mysql> SET @s = CAST('AB' AS BINARY);
mysql> SELECT @s, HEX(@s), HEX(WEIGHT_STRING(@s));
+------+---------+-----------------------+
| @s   | HEX(@s) | HEX(WEIGHT_STRING(@s)) |
+------+---------+-----------------------+
| AB   | 4142    | 4142                  |
+------+---------+-----------------------+
```

```
mysql> SET @s = CAST('ab' AS BINARY);
mysql> SELECT @s, HEX(@s), HEX(WEIGHT_STRING(@s));
+------+---------+-----------------------+
| @s   | HEX(@s) | HEX(WEIGHT_STRING(@s)) |
+------+---------+-----------------------+
| ab   | 6162    | 6162                  |
+------+---------+-----------------------+
```

The preceding examples use `HEX()` to display the `WEIGHT_STRING()` result. Because the result is a binary value, `HEX()` can be especially useful when the result contains nonprinting values, to display it in printable form:

```
mysql> SET @s = CONVERT(X'C39F' USING utf8) COLLATE utf8_czech_ci;
mysql> SELECT HEX(WEIGHT_STRING(@s));
+-----------------------+
| HEX(WEIGHT_STRING(@s)) |
+-----------------------+
| 0FEA0FEA              |
+-----------------------+
```

For non-`NULL` return values, the data type of the value is `VARBINARY` if its length is within the maximum length for `VARBINARY`, otherwise the data type is `BLOB`.

The `AS` clause may be given to cast the input string to a nonbinary or binary string and to force it to a given length:

- `AS CHAR(N)` casts the string to a nonbinary string and pads it on the right with spaces to a length of *`N`* characters. *`N`* must be at least 1. If *`N`* is less than the length of the input string, the string is truncated to *`N`* characters. No warning occurs for truncation.
- `AS BINARY(N)` is similar but casts the string to a binary string, *`N`* is measured in bytes (not characters), and padding uses `0x00` bytes (not spaces).

```
mysql> SET NAMES 'latin1';
mysql> SELECT HEX(WEIGHT_STRING('ab' AS CHAR(4)));
+-------------------------------------+
| HEX(WEIGHT_STRING('ab' AS CHAR(4))) |
+-------------------------------------+
| 41422020                            |
```

```
+-----------------------------------+
mysql> SET NAMES 'utf8';
mysql> SELECT HEX(WEIGHT_STRING('ab' AS CHAR(4)));
+-----------------------------------+
| HEX(WEIGHT_STRING('ab' AS CHAR(4))) |
+-----------------------------------+
| 0041004200200020                  |
+-----------------------------------+
```

```
mysql> SELECT HEX(WEIGHT_STRING('ab' AS BINARY(4)));
+--------------------------------------+
| HEX(WEIGHT_STRING('ab' AS BINARY(4))) |
+--------------------------------------+
| 61620000                             |
+--------------------------------------+
```

The LEVEL clause may be given to specify that the return value should contain weights for specific collation levels.

The *levels* specifier following the LEVEL keyword may be given either as a list of one or more integers separated by commas, or as a range of two integers separated by a dash. Whitespace around the punctuation characters does not matter.

Examples:

```
LEVEL 1
LEVEL 2, 3, 5
LEVEL 1-3
```

Any level less than 1 is treated as 1. Any level greater than the maximum for the input string collation is treated as maximum for the collation. The maximum varies per collation, but is never greater than 6.

In a list of levels, levels must be given in increasing order. In a range of levels, if the second number is less than the first, it is treated as the first number (for example, 4-2 is the same as 4-4).

If the LEVEL clause is omitted, MySQL assumes LEVEL 1 - *max*, where *max* is the maximum level for the collation.

If LEVEL is specified using list syntax (not range syntax), any level number can be followed by these modifiers:

- ASC: Return the weights without modification. This is the default.
- DESC: Return bitwise-inverted weights (for example, 0x78f0 DESC = 0x870f).
- REVERSE: Return the weights in reverse order (that is,the weights for the reversed string, with the first character last and the last first).

Examples:

```
mysql> SELECT HEX(WEIGHT_STRING(0x007fff LEVEL 1));
+--------------------------------------+
| HEX(WEIGHT_STRING(0x007fff LEVEL 1)) |
+--------------------------------------+
| 007FFF                               |
+--------------------------------------+
```

```
mysql> SELECT HEX(WEIGHT_STRING(0x007fff LEVEL 1 DESC));
+-----------------------------------------+
| HEX(WEIGHT_STRING(0x007fff LEVEL 1 DESC)) |
+-----------------------------------------+
| FF8000                                  |
+-----------------------------------------+
```

```
mysql> SELECT HEX(WEIGHT_STRING(0x007fff LEVEL 1 REVERSE));
+-------------------------------------------+
```

```
| HEX(WEIGHT_STRING(0x007fff LEVEL 1 REVERSE)) |
+----------------------------------------------+
| FF7F00                                       |
+----------------------------------------------+
```

```
mysql> SELECT HEX(WEIGHT_STRING(0x007fff LEVEL 1 DESC REVERSE));
+-------------------------------------------------+
| HEX(WEIGHT_STRING(0x007fff LEVEL 1 DESC REVERSE)) |
+-------------------------------------------------+
| 0080FF                                          |
+-------------------------------------------------+
```

The *`flags`* clause currently is unused.

12.8.1 String Comparison Functions and Operators

Table 12.13 String Comparison Functions and Operators

Name	Description
`LIKE`	Simple pattern matching
`NOT LIKE`	Negation of simple pattern matching
`STRCMP()`	Compare two strings

If a string function is given a binary string as an argument, the resulting string is also a binary string. A number converted to a string is treated as a binary string. This affects only comparisons.

Normally, if any expression in a string comparison is case-sensitive, the comparison is performed in case-sensitive fashion.

- *`expr`* `LIKE` *`pat`* `[ESCAPE '`*`escape_char`*`']`

  Pattern matching using an SQL pattern. Returns `1` (`TRUE`) or `0` (`FALSE`). If either *`expr`* or *`pat`* is `NULL`, the result is `NULL`.

  The pattern need not be a literal string. For example, it can be specified as a string expression or table column. In the latter case, the column must be defined as one of the MySQL string types (see Section 11.3, “String Data Types”).

  Per the SQL standard, `LIKE` performs matching on a per-character basis, thus it can produce results different from the `=` comparison operator:

```
mysql> SELECT 'ä' LIKE 'ae' COLLATE latin1_german2_ci;
+-----------------------------------------+
| 'ä' LIKE 'ae' COLLATE latin1_german2_ci |
+-----------------------------------------+
|                                       0 |
+-----------------------------------------+
mysql> SELECT 'ä' = 'ae' COLLATE latin1_german2_ci;
+-------------------------------------+
| 'ä' = 'ae' COLLATE latin1_german2_ci |
+-------------------------------------+
|                                   1 |
+-------------------------------------+
```

  In particular, trailing spaces are significant, which is not true for comparisons of nonbinary strings (`CHAR`, `VARCHAR`, and `TEXT` values) performed with the `=` operator:

```
mysql> SELECT 'a' = 'a ', 'a' LIKE 'a ';
+------------+---------------+
| 'a' = 'a ' | 'a' LIKE 'a ' |
+------------+---------------+
|          1 |             0 |
+------------+---------------+
1 row in set (0.00 sec)
```

  With `LIKE` you can use the following two wildcard characters in the pattern:

- `%` matches any number of characters, even zero characters.
- `_` matches exactly one character.

```
mysql> SELECT 'David!' LIKE 'David_';
        -> 1
mysql> SELECT 'David!' LIKE '%D%v%';
        -> 1
```

To test for literal instances of a wildcard character, precede it by the escape character. If you do not specify the `ESCAPE` character, `\` is assumed.

- `\%` matches one `%` character.
- `\_` matches one `_` character.

```
mysql> SELECT 'David!' LIKE 'David\_';
        -> 0
mysql> SELECT 'David_' LIKE 'David\_';
        -> 1
```

To specify a different escape character, use the `ESCAPE` clause:

```
mysql> SELECT 'David_' LIKE 'David|_' ESCAPE '|';
        -> 1
```

The escape sequence should be empty or one character long. The expression must evaluate as a constant at execution time. If the `NO_BACKSLASH_ESCAPES` SQL mode is enabled, the sequence cannot be empty.

The following two statements illustrate that string comparisons are not case-sensitive unless one of the operands is case-sensitive (uses a case-sensitive collation or is a binary string):

```
mysql> SELECT 'abc' LIKE 'ABC';
        -> 1
mysql> SELECT 'abc' LIKE _latin1 'ABC' COLLATE latin1_general_cs;
        -> 0
mysql> SELECT 'abc' LIKE _latin1 'ABC' COLLATE latin1_bin;
        -> 0
mysql> SELECT 'abc' LIKE BINARY 'ABC';
        -> 0
```

As an extension to standard SQL, MySQL permits `LIKE` on numeric expressions.

```
mysql> SELECT 10 LIKE '1%';
        -> 1
```

Note

Because MySQL uses C escape syntax in strings (for example, `\n` to represent a newline character), you must double any `\` that you use in `LIKE` strings. For example, to search for `\n`, specify it as `\\n`. To search for `\`, specify it as `\\\\`; this is because the backslashes are stripped once by the parser and again when the pattern match is made, leaving a single backslash to be matched against.

Exception: At the end of the pattern string, backslash can be specified as `\\`. At the end of the string, backslash stands for itself because there is nothing following to escape. Suppose that a table contains the following values:

```
mysql> SELECT filename FROM t1;
+--------------+
| filename     |
+--------------+
| C:           |
```

```
| C:\          |
| C:\Programs  |
| C:\Programs\ |
+--------------+
```

To test for values that end with backslash, you can match the values using either of the following patterns:

```
mysql> SELECT filename, filename LIKE '%\\' FROM t1;
+--------------+---------------------+
| filename     | filename LIKE '%\\' |
+--------------+---------------------+
| C:           |                   0 |
| C:\          |                   1 |
| C:\Programs  |                   0 |
| C:\Programs\ |                   1 |
+--------------+---------------------+

mysql> SELECT filename, filename LIKE '%\\\\' FROM t1;
+--------------+----------------------+
| filename     | filename LIKE '%\\\\' |
+--------------+----------------------+
| C:           |                    0 |
| C:\          |                    1 |
| C:\Programs  |                    0 |
| C:\Programs\ |                    1 |
+--------------+----------------------+
```

- *expr* `NOT LIKE` *pat* `[ESCAPE '`*escape_char*`']`

  This is the same as `NOT (`*expr* `LIKE` *pat* `[ESCAPE '`*escape_char*`'])`.

Note

Aggregate queries involving `NOT LIKE` comparisons with columns containing `NULL` may yield unexpected results. For example, consider the following table and data:

```
CREATE TABLE foo (bar VARCHAR(10));

INSERT INTO foo VALUES (NULL), (NULL);
```

The query `SELECT COUNT(*) FROM foo WHERE bar LIKE '%baz%';` returns `0`. You might assume that `SELECT COUNT(*) FROM foo WHERE bar NOT LIKE '%baz%';` would return `2`. However, this is not the case: The second query returns `0`. This is because `NULL NOT LIKE` *expr* always returns `NULL`, regardless of the value of *expr*. The same is true for aggregate queries involving `NULL` and comparisons using `NOT RLIKE` or `NOT REGEXP`. In such cases, you must test explicitly for `NOT NULL` using `OR` (and not `AND`), as shown here:

```
SELECT COUNT(*) FROM foo WHERE bar NOT LIKE '%baz%' OR bar IS NULL;
```

- `STRCMP(`*expr1*`,`*expr2*`)`

  `STRCMP()` returns `0` if the strings are the same, `-1` if the first argument is smaller than the second according to the current sort order, and `1` otherwise.

```
mysql> SELECT STRCMP('text', 'text2');
        -> -1
mysql> SELECT STRCMP('text2', 'text');
        -> 1
mysql> SELECT STRCMP('text', 'text');
```

```
        -> 0
```

`STRCMP()` performs the comparison using the collation of the arguments.

```
mysql> SET @s1 = _latin1 'x' COLLATE latin1_general_ci;
mysql> SET @s2 = _latin1 'X' COLLATE latin1_general_ci;
mysql> SET @s3 = _latin1 'x' COLLATE latin1_general_cs;
mysql> SET @s4 = _latin1 'X' COLLATE latin1_general_cs;
mysql> SELECT STRCMP(@s1, @s2), STRCMP(@s3, @s4);
+------------------+------------------+
| STRCMP(@s1, @s2) | STRCMP(@s3, @s4) |
+------------------+------------------+
|                0 |                1 |
+------------------+------------------+
```

If the collations are incompatible, one of the arguments must be converted to be compatible with the other. See Section 10.8.4, “Collation Coercibility in Expressions”.

```
mysql> SELECT STRCMP(@s1, @s3);
ERROR 1267 (HY000): Illegal mix of collations (latin1_general_ci,IMPLICIT)
and (latin1_general_cs,IMPLICIT) for operation 'strcmp'
mysql> SELECT STRCMP(@s1, @s3 COLLATE latin1_general_ci);
+-------------------------------------------+
| STRCMP(@s1, @s3 COLLATE latin1_general_ci) |
+-------------------------------------------+
|                                         0 |
+-------------------------------------------+
```

12.8.2 Regular Expressions

Table 12.14 Regular Expression Functions and Operators

Name	Description
`NOT REGEXP`	Negation of REGEXP
`REGEXP`	Whether string matches regular expression
`RLIKE`	Whether string matches regular expression

A regular expression is a powerful way of specifying a pattern for a complex search. This section discusses the operators available for regular expression matching and illustrates, with examples, some of the special characters and constructs that can be used for regular expression operations. See also Section 3.3.4.7, “Pattern Matching”.

MySQL uses Henry Spencer's implementation of regular expressions, which is aimed at conformance with POSIX 1003.2. MySQL uses the extended version to support regular expression pattern-matching operations in SQL statements. This section does not contain all the details that can be found in Henry Spencer's `regex(7)` manual page. That manual page is included in MySQL source distributions, in the `regex.7` file under the `regex` directory.

- Regular Expression Operators
- Regular Expression Syntax

Regular Expression Operators

- *`expr`* `NOT REGEXP` *`pat`*, *`expr`* `NOT RLIKE` *`pat`*

  This is the same as `NOT (`*`expr`* `REGEXP` *`pat`*`)`.

- *`expr`* `REGEXP` *`pat`*, *`expr`* `RLIKE` *`pat`*

  Returns 1 if the string *`expr`* matches the regular expression specified by the pattern *`pat`*, 0 otherwise. If either *`expr`* or *`pat`* is `NULL`, the return value is `NULL`.

  `RLIKE` is a synonym for `REGEXP`, provided for `mSQL` compatibility.

The pattern can be an extended regular expression, the syntax for which is discussed in Regular Expression Syntax. The pattern need not be a literal string. For example, it can be specified as a string expression or table column.

Note

Because MySQL uses the C escape syntax in strings (for example, `\n` to represent the newline character), you must double any `\` that you use in your `REGEXP` arguments.

Regular expression operations use the character set and collation of the string expression and pattern arguments when deciding the type of a character and performing the comparison. If the arguments have different character sets or collations, coercibility rules apply as described in Section 10.8.4, “Collation Coercibility in Expressions”. If either argument is a binary string, the arguments are handled in case-sensitive fashion as binary strings.

```
mysql> SELECT 'Michael!' REGEXP '.*';
+-----------------------+
| 'Michael!' REGEXP '.*' |
+-----------------------+
|                     1 |
+-----------------------+
mysql> SELECT 'new*\n*line' REGEXP 'new\\*.\\*line';
+--------------------------------------+
| 'new*\n*line' REGEXP 'new\\*.\\*line' |
+--------------------------------------+
|                                    0 |
+--------------------------------------+
mysql> SELECT 'a' REGEXP '^[a-d]';
+--------------------+
| 'a' REGEXP '^[a-d]' |
+--------------------+
|                  1 |
+--------------------+
mysql> SELECT 'a' REGEXP 'A', 'a' REGEXP BINARY 'A';
+---------------+-----------------------+
| 'a' REGEXP 'A' | 'a' REGEXP BINARY 'A' |
+---------------+-----------------------+
|              1 |                     0 |
+---------------+-----------------------+
```

Warning

The `REGEXP` and `RLIKE` operators work in byte-wise fashion, so they are not multibyte safe and may produce unexpected results with multibyte character sets. In addition, these operators compare characters by their byte values and accented characters may not compare as equal even if a given collation treats them as equal.

Regular Expression Syntax

A regular expression describes a set of strings. The simplest regular expression is one that has no special characters in it. For example, the regular expression `hello` matches `hello` and nothing else.

Nontrivial regular expressions use certain special constructs so that they can match more than one string. For example, the regular expression `hello|world` contains the `|` alternation operator and matches either the `hello` or `world`.

As a more complex example, the regular expression `B[an]*s` matches any of the strings `Bananas`, `Baaaaas`, `Bs`, and any other string starting with a `B`, ending with an `s`, and containing any number of `a` or `n` characters in between.

A regular expression for the `REGEXP` operator may use any of the following special characters and constructs:

- `^`

  Match the beginning of a string.

```
mysql> SELECT 'fo\nfo' REGEXP '^fo$';                -> 0
mysql> SELECT 'fofo' REGEXP '^fo';                   -> 1
```

- `$`

  Match the end of a string.

```
mysql> SELECT 'fo\no' REGEXP '^fo\no$';              -> 1
mysql> SELECT 'fo\no' REGEXP '^fo$';                 -> 0
```

- `.`

  Match any character (including carriage return and newline).

```
mysql> SELECT 'fofo' REGEXP '^f.*$';                 -> 1
mysql> SELECT 'fo\r\nfo' REGEXP '^f.*$';             -> 1
```

- `a*`

  Match any sequence of zero or more `a` characters.

```
mysql> SELECT 'Ban' REGEXP '^Ba*n';                  -> 1
mysql> SELECT 'Baaan' REGEXP '^Ba*n';                -> 1
mysql> SELECT 'Bn' REGEXP '^Ba*n';                   -> 1
```

- `a+`

  Match any sequence of one or more `a` characters.

```
mysql> SELECT 'Ban' REGEXP '^Ba+n';                  -> 1
mysql> SELECT 'Bn' REGEXP '^Ba+n';                   -> 0
```

- `a?`

  Match either zero or one `a` character.

```
mysql> SELECT 'Bn' REGEXP '^Ba?n';                   -> 1
mysql> SELECT 'Ban' REGEXP '^Ba?n';                  -> 1
mysql> SELECT 'Baan' REGEXP '^Ba?n';                 -> 0
```

- `de|abc`

  Alternation; match either of the sequences `de` or `abc`.

```
mysql> SELECT 'pi' REGEXP 'pi|apa';                  -> 1
mysql> SELECT 'axe' REGEXP 'pi|apa';                 -> 0
mysql> SELECT 'apa' REGEXP 'pi|apa';                 -> 1
mysql> SELECT 'apa' REGEXP '^(pi|apa)$';             -> 1
mysql> SELECT 'pi' REGEXP '^(pi|apa)$';              -> 1
mysql> SELECT 'pix' REGEXP '^(pi|apa)$';             -> 0
```

- `(abc)*`

  Match zero or more instances of the sequence `abc`.

```
mysql> SELECT 'pi' REGEXP '^(pi)*$';                 -> 1
mysql> SELECT 'pip' REGEXP '^(pi)*$';                -> 0
mysql> SELECT 'pipi' REGEXP '^(pi)*$';               -> 1
```

- `{1}`, `{2,3}`

  Repetition; `{n}` and `{m,n}` notation provide a more general way of writing regular expressions that match many occurrences of the previous atom (or “piece”) of the pattern. *`m`* and *`n`* are integers.

- `a*`

  Can be written as `a{0,}`.

- `a+`

  Can be written as `a{1,}`.

- `a?`

  Can be written as `a{0,1}`.

To be more precise, `a{n}` matches exactly *n* instances of `a`. `a{n,}` matches *n* or more instances of `a`. `a{m,n}` matches *m* through *n* instances of `a`, inclusive. If both *m* and *n* are given, *m* must be less than or equal to *n*.

m and *n* must be in the range from `0` to `RE_DUP_MAX` (default 255), inclusive.

```
mysql> SELECT 'abcde' REGEXP 'a[bcd]{2}e';             -> 0
mysql> SELECT 'abcde' REGEXP 'a[bcd]{3}e';             -> 1
mysql> SELECT 'abcde' REGEXP 'a[bcd]{1,10}e';          -> 1
```

- `[a-dX]`, `[^a-dX]`

  Matches any character that is (or is not, if `^` is used) either `a`, `b`, `c`, `d` or `X`. A `-` character between two other characters forms a range that matches all characters from the first character to the second. For example, `[0-9]` matches any decimal digit. To include a literal `]` character, it must immediately follow the opening bracket `[`. To include a literal `-` character, it must be written first or last. Any character that does not have a defined special meaning inside a `[]` pair matches only itself.

```
mysql> SELECT 'aXbc' REGEXP '[a-dXYZ]';                -> 1
mysql> SELECT 'aXbc' REGEXP '^[a-dXYZ]$';              -> 0
mysql> SELECT 'aXbc' REGEXP '^[a-dXYZ]+$';             -> 1
mysql> SELECT 'aXbc' REGEXP '^[^a-dXYZ]+$';            -> 0
mysql> SELECT 'gheis' REGEXP '^[^a-dXYZ]+$';           -> 1
mysql> SELECT 'gheisa' REGEXP '^[^a-dXYZ]+$';          -> 0
```

- `[.characters.]`

  Within a bracket expression (written using `[` and `]`), matches the sequence of characters of that collating element. `characters` is either a single character or a character name like `newline`. The following table lists the permissible character names.

  The following table shows the permissible character names and the characters that they match. For characters given as numeric values, the values are represented in octal.

Name	Character	Name	Character
NUL	0	SOH	001
STX	002	ETX	003
EOT	004	ENQ	005
ACK	006	BEL	007
alert	007	BS	010
backspace	'\b'	HT	011
tab	'\t'	LF	012
newline	'\n'	VT	013
vertical-tab	'\v'	FF	014
form-feed	'\f'	CR	015

Name	Character	Name	Character
carriage-return	'\r'	SO	016
SI	017	DLE	020
DC1	021	DC2	022
DC3	023	DC4	024
NAK	025	SYN	026
ETB	027	CAN	030
EM	031	SUB	032
ESC	033	IS4	034
FS	034	IS3	035
GS	035	IS2	036
RS	036	IS1	037
US	037	space	' '
exclamation-mark	'!'	quotation-mark	'"'
number-sign	'#'	dollar-sign	'$'
percent-sign	'%'	ampersand	'&'
apostrophe	'\''	left-parenthesis	'('
right-parenthesis	')'	asterisk	'*'
plus-sign	'+'	comma	','
hyphen	'-'	hyphen-minus	'-'
period	'.'	full-stop	'.'
slash	'/'	solidus	'/'
zero	'0'	one	'1'
two	'2'	three	'3'
four	'4'	five	'5'
six	'6'	seven	'7'
eight	'8'	nine	'9'
colon	':'	semicolon	';'
less-than-sign	'<'	equals-sign	'='
greater-than-sign	'>'	question-mark	'?'
commercial-at	'@'	left-square-bracket	'['
backslash	'\\'	reverse-solidus	'\\'
right-square-bracket	']'	circumflex	'^'
circumflex-accent	'^'	underscore	'_'
low-line	'_'	grave-accent	'`'
left-brace	'{'	left-curly-bracket	'{'
vertical-line	'|'	right-brace	'}'
right-curly-bracket	'}'	tilde	'~'

Name	Character	Name	Character
DEL	177		

```
mysql> SELECT '~' REGEXP '[[.~.]]';                    -> 1
mysql> SELECT '~' REGEXP '[[.tilde.]]';                -> 1
```

- `[=character_class=]`

  Within a bracket expression (written using `[` and `]`), `[=character_class=]` represents an equivalence class. It matches all characters with the same collation value, including itself. For example, if `o` and `(+)` are the members of an equivalence class, `[[=o=]]`, `[[=(+)=]]`, and `[o(+)]` are all synonymous. An equivalence class may not be used as an endpoint of a range.

- `[:character_class:]`

  Within a bracket expression (written using `[` and `]`), `[:character_class:]` represents a character class that matches all characters belonging to that class. The following table lists the standard class names. These names stand for the character classes defined in the `ctype(3)` manual page. A particular locale may provide other class names. A character class may not be used as an endpoint of a range.

Character Class Name	Meaning
`alnum`	Alphanumeric characters
`alpha`	Alphabetic characters
`blank`	Whitespace characters
`cntrl`	Control characters
`digit`	Digit characters
`graph`	Graphic characters
`lower`	Lowercase alphabetic characters
`print`	Graphic or space characters
`punct`	Punctuation characters
`space`	Space, tab, newline, and carriage return
`upper`	Uppercase alphabetic characters
`xdigit`	Hexadecimal digit characters

```
mysql> SELECT 'justalnums' REGEXP '[[:alnum:]]+';       -> 1
mysql> SELECT '!!' REGEXP '[[:alnum:]]+';               -> 0
```

- `[[:<:]]`, `[[:>:]]`

  These markers stand for word boundaries. They match the beginning and end of words, respectively. A word is a sequence of word characters that is not preceded by or followed by word characters. A word character is an alphanumeric character in the `alnum` class or an underscore (_).

```
mysql> SELECT 'a word a' REGEXP '[[:<:]]word[[:>:]]';   -> 1
mysql> SELECT 'a xword a' REGEXP '[[:<:]]word[[:>:]]';  -> 0
```

To use a literal instance of a special character in a regular expression, precede it by two backslash (\) characters. The MySQL parser interprets one of the backslashes, and the regular expression library interprets the other. For example, to match the string `1+2` that contains the special `+` character, only the last of the following regular expressions is the correct one:

```
mysql> SELECT '1+2' REGEXP '1+2';                       -> 0
mysql> SELECT '1+2' REGEXP '1\+2';                      -> 0
mysql> SELECT '1+2' REGEXP '1\\+2';                     -> 1
```

12.8.3 Character Set and Collation of Function Results

MySQL has many operators and functions that return a string. This section answers the question: What is the character set and collation of such a string?

For simple functions that take string input and return a string result as output, the output's character set and collation are the same as those of the principal input value. For example, `UPPER(X)` returns a string with the same character string and collation as *X*. The same applies for `INSTR()`, `LCASE()`, `LOWER()`, `LTRIM()`, `MID()`, `REPEAT()`, `REPLACE()`, `REVERSE()`, `RIGHT()`, `RPAD()`, `RTRIM()`, `SOUNDEX()`, `SUBSTRING()`, `TRIM()`, `UCASE()`, and `UPPER()`.

Note

The `REPLACE()` function, unlike all other functions, always ignores the collation of the string input and performs a case-sensitive comparison.

If a string input or function result is a binary string, the string has the `binary` character set and collation. This can be checked by using the `CHARSET()` and `COLLATION()` functions, both of which return `binary` for a binary string argument:

```
mysql> SELECT CHARSET(BINARY 'a'), COLLATION(BINARY 'a');
+--------------------+----------------------+
| CHARSET(BINARY 'a') | COLLATION(BINARY 'a') |
+--------------------+----------------------+
| binary              | binary               |
+--------------------+----------------------+
```

For operations that combine multiple string inputs and return a single string output, the “aggregation rules” of standard SQL apply for determining the collation of the result:

- If an explicit `COLLATE Y` occurs, use *Y*.
- If explicit `COLLATE Y` and `COLLATE Z` occur, raise an error.
- Otherwise, if all collations are *Y*, use *Y*.
- Otherwise, the result has no collation.

For example, with `CASE ... WHEN a THEN b WHEN b THEN c COLLATE X END`, the resulting collation is *X*. The same applies for `UNION`, `||`, `CONCAT()`, `ELT()`, `GREATEST()`, `IF()`, and `LEAST()`.

For operations that convert to character data, the character set and collation of the strings that result from the operations are defined by the `character_set_connection` and `collation_connection` system variables that determine the default connection character set and collation (see Section 10.4, “Connection Character Sets and Collations”). This applies only to `CAST()`, `CONV()`, `FORMAT()`, `HEX()`, and `SPACE()`.

If there is any question about the character set or collation of the result returned by a string function, use the `CHARSET()` or `COLLATION()` function to find out:

```
mysql> SELECT USER(), CHARSET(USER()), COLLATION(USER());
+----------------+-----------------+-------------------+
| USER()         | CHARSET(USER()) | COLLATION(USER()) |
+----------------+-----------------+-------------------+
| test@localhost | utf8            | utf8_general_ci   |
+----------------+-----------------+-------------------+
mysql> SELECT CHARSET(COMPRESS('abc')), COLLATION(COMPRESS('abc'));
+--------------------------+----------------------------+
| CHARSET(COMPRESS('abc')) | COLLATION(COMPRESS('abc')) |
+--------------------------+----------------------------+
| binary                   | binary                     |
+--------------------------+----------------------------+
```

12.9 What Calendar Is Used By MySQL?

MySQL uses what is known as a *proleptic Gregorian calendar*.

Every country that has switched from the Julian to the Gregorian calendar has had to discard at least ten days during the switch. To see how this works, consider the month of October 1582, when the first Julian-to-Gregorian switch occurred.

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
1	2	3	4	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31

There are no dates between October 4 and October 15. This discontinuity is called the *cutover*. Any dates before the cutover are Julian, and any dates following the cutover are Gregorian. Dates during a cutover are nonexistent.

A calendar applied to dates when it was not actually in use is called *proleptic*. Thus, if we assume there was never a cutover and Gregorian rules always rule, we have a proleptic Gregorian calendar. This is what is used by MySQL, as is required by standard SQL. For this reason, dates prior to the cutover stored as MySQL `DATE` or `DATETIME` values must be adjusted to compensate for the difference. It is important to realize that the cutover did not occur at the same time in all countries, and that the later it happened, the more days were lost. For example, in Great Britain, it took place in 1752, when Wednesday September 2 was followed by Thursday September 14. Russia remained on the Julian calendar until 1918, losing 13 days in the process, and what is popularly referred to as its “October Revolution” occurred in November according to the Gregorian calendar.

12.10 Full-Text Search Functions

```
MATCH (col1,col2,...) AGAINST (expr [search_modifier])

search_modifier:
  {
       IN NATURAL LANGUAGE MODE
     | IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION
     | IN BOOLEAN MODE
     | WITH QUERY EXPANSION
  }
```

MySQL has support for full-text indexing and searching:

- A full-text index in MySQL is an index of type `FULLTEXT`.
- Full-text indexes can be used only with `InnoDB` or `MyISAM` tables, and can be created only for `CHAR`, `VARCHAR`, or `TEXT` columns.
- A `FULLTEXT` index definition can be given in the `CREATE TABLE` statement when a table is created, or added later using `ALTER TABLE` or `CREATE INDEX`.
- For large data sets, it is much faster to load your data into a table that has no `FULLTEXT` index and then create the index after that, than to load data into a table that has an existing `FULLTEXT` index.

Full-text searching is performed using `MATCH() ... AGAINST` syntax. `MATCH()` takes a comma-separated list that names the columns to be searched. `AGAINST` takes a string to search for, and an optional modifier that indicates what type of search to perform. The search string must be a string value that is constant during query evaluation. This rules out, for example, a table column because that can differ for each row.

There are three types of full-text searches:

- A natural language search interprets the search string as a phrase in natural human language (a phrase in free text). There are no special operators, with the exception of double quote (") characters. The stopword list applies. For more information about stopword lists, see Section 12.10.4, “Full-Text Stopwords”.

Full-text searches are natural language searches if the `IN NATURAL LANGUAGE MODE` modifier is given or if no modifier is given. For more information, see Section 12.10.1, “Natural Language Full-Text Searches”.

- A boolean search interprets the search string using the rules of a special query language. The string contains the words to search for. It can also contain operators that specify requirements such that a word must be present or absent in matching rows, or that it should be weighted higher or lower than usual. Certain common words (stopwords) are omitted from the search index and do not match if present in the search string. The `IN BOOLEAN MODE` modifier specifies a boolean search. For more information, see Section 12.10.2, “Boolean Full-Text Searches”.
- A query expansion search is a modification of a natural language search. The search string is used to perform a natural language search. Then words from the most relevant rows returned by the search are added to the search string and the search is done again. The query returns the rows from the second search. The `IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION` or `WITH QUERY EXPANSION` modifier specifies a query expansion search. For more information, see Section 12.10.3, “Full-Text Searches with Query Expansion”.

For information about `FULLTEXT` query performance, see Section 8.3.4, “Column Indexes”.

For more information about `InnoDB FULLTEXT` indexes, see Section 14.6.2.3, “InnoDB FULLTEXT Indexes”.

Constraints on full-text searching are listed in Section 12.10.5, “Full-Text Restrictions”.

The `myisam_ftdump` utility dumps the contents of a `MyISAM` full-text index. This may be helpful for debugging full-text queries. See Section 4.6.2, “`myisam_ftdump` — Display Full-Text Index information”.

12.10.1 Natural Language Full-Text Searches

By default or with the `IN NATURAL LANGUAGE MODE` modifier, the `MATCH()` function performs a natural language search for a string against a *text collection*. A collection is a set of one or more columns included in a `FULLTEXT` index. The search string is given as the argument to `AGAINST()`. For each row in the table, `MATCH()` returns a relevance value; that is, a similarity measure between the search string and the text in that row in the columns named in the `MATCH()` list.

```
mysql> CREATE TABLE articles (
          id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
          title VARCHAR(200),
          body TEXT,
          FULLTEXT (title,body)
        ) ENGINE=InnoDB;
Query OK, 0 rows affected (0.08 sec)

mysql> INSERT INTO articles (title,body) VALUES
        ('MySQL Tutorial','DBMS stands for DataBase ...'),
        ('How To Use MySQL Well','After you went through a ...'),
        ('Optimizing MySQL','In this tutorial we show ...'),
        ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
        ('MySQL vs. YourSQL','In the following database comparison ...'),
        ('MySQL Security','When configured properly, MySQL ...');
Query OK, 6 rows affected (0.01 sec)
Records: 6  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM articles
        WHERE MATCH (title,body)
        AGAINST ('database' IN NATURAL LANGUAGE MODE);
+----+-------------------+------------------------------------------+
| id | title             | body                                     |
+----+-------------------+------------------------------------------+
|  1 | MySQL Tutorial    | DBMS stands for DataBase ...             |
|  5 | MySQL vs. YourSQL | In the following database comparison ... |
+----+-------------------+------------------------------------------+
2 rows in set (0.00 sec)
```

By default, the search is performed in case-insensitive fashion. To perform a case-sensitive full-text search, use a binary collation for the indexed columns. For example, a column that uses the `latin1` character set of can be assigned a collation of `latin1_bin` to make it case-sensitive for full-text searches.

When `MATCH()` is used in a `WHERE` clause, as in the example shown earlier, the rows returned are automatically sorted with the highest relevance first. Relevance values are nonnegative floating-point numbers. Zero relevance means no similarity. Relevance is computed based on the number of words in the row (document), the number of unique words in the row, the total number of words in the collection, and the number of rows that contain a particular word.

Note

The term “document” may be used interchangeably with the term “row”, and both terms refer to the indexed part of the row. The term “collection” refers to the indexed columns and encompasses all rows.

To simply count matches, you could use a query like this:

```
mysql> SELECT COUNT(*) FROM articles
    WHERE MATCH (title,body)
    AGAINST ('database' IN NATURAL LANGUAGE MODE);
+----------+
| COUNT(*) |
+----------+
|        2 |
+----------+
1 row in set (0.00 sec)
```

You might find it quicker to rewrite the query as follows:

```
mysql> SELECT
    COUNT(IF(MATCH (title,body) AGAINST ('database' IN NATURAL LANGUAGE MODE), 1, NULL))
    AS count
    FROM articles;
+-------+
| count |
+-------+
|     2 |
+-------+
1 row in set (0.03 sec)
```

The first query does some extra work (sorting the results by relevance) but also can use an index lookup based on the `WHERE` clause. The index lookup might make the first query faster if the search matches few rows. The second query performs a full table scan, which might be faster than the index lookup if the search term was present in most rows.

For natural-language full-text searches, the columns named in the `MATCH()` function must be the same columns included in some `FULLTEXT` index in your table. For the preceding query, the columns named in the `MATCH()` function (`title` and `body`) are the same as those named in the definition of the `article` table's `FULLTEXT` index. To search the `title` or `body` separately, you would create separate `FULLTEXT` indexes for each column.

You can also perform a boolean search or a search with query expansion. These search types are described in Section 12.10.2, “Boolean Full-Text Searches”, and Section 12.10.3, “Full-Text Searches with Query Expansion”.

A full-text search that uses an index can name columns only from a single table in the `MATCH()` clause because an index cannot span multiple tables. For `MyISAM` tables, a boolean search can be done in the absence of an index (albeit more slowly), in which case it is possible to name columns from multiple tables.

The preceding example is a basic illustration that shows how to use the `MATCH()` function where rows are returned in order of decreasing relevance. The next example shows how to retrieve the relevance

values explicitly. Returned rows are not ordered because the SELECT statement includes neither WHERE nor ORDER BY clauses:

```
mysql> SELECT id, MATCH (title,body)
    AGAINST ('Tutorial' IN NATURAL LANGUAGE MODE) AS score
    FROM articles;
+----+--------------------+
| id | score              |
+----+--------------------+
|  1 | 0.22764469683170319 |
|  2 |                  0 |
|  3 | 0.22764469683170319 |
|  4 |                  0 |
|  5 |                  0 |
|  6 |                  0 |
+----+--------------------+
6 rows in set (0.00 sec)
```

The following example is more complex. The query returns the relevance values and it also sorts the rows in order of decreasing relevance. To achieve this result, specify MATCH() twice: once in the SELECT list and once in the WHERE clause. This causes no additional overhead, because the MySQL optimizer notices that the two MATCH() calls are identical and invokes the full-text search code only once.

```
mysql> SELECT id, body, MATCH (title,body) AGAINST
    ('Security implications of running MySQL as root'
    IN NATURAL LANGUAGE MODE) AS score
    FROM articles WHERE MATCH (title,body) AGAINST
    ('Security implications of running MySQL as root'
    IN NATURAL LANGUAGE MODE);
+----+-------------------------------------+-----------------+
| id | body                                | score           |
+----+-------------------------------------+-----------------+
|  4 | 1. Never run mysqld as root. 2. ... | 1.5219271183014 |
|  6 | When configured properly, MySQL ... | 1.3114095926285 |
+----+-------------------------------------+-----------------+
2 rows in set (0.00 sec)
```

A phrase that is enclosed within double quote (") characters matches only rows that contain the phrase *literally, as it was typed*. The full-text engine splits the phrase into words and performs a search in the FULLTEXT index for the words. Nonword characters need not be matched exactly: Phrase searching requires only that matches contain exactly the same words as the phrase and in the same order. For example, "test phrase" matches "test, phrase". If the phrase contains no words that are in the index, the result is empty. For example, if all words are either stopwords or shorter than the minimum length of indexed words, the result is empty.

The MySQL FULLTEXT implementation regards any sequence of true word characters (letters, digits, and underscores) as a word. That sequence may also contain apostrophes ('), but not more than one in a row. This means that aaa'bbb is regarded as one word, but aaa''bbb is regarded as two words. Apostrophes at the beginning or the end of a word are stripped by the FULLTEXT parser; 'aaa'bbb' would be parsed as aaa'bbb.

The FULLTEXT parser determines where words start and end by looking for certain delimiter characters; for example, (space), , (comma), and . (period). If words are not separated by delimiters (as in, for example, Chinese), the FULLTEXT parser cannot determine where a word begins or ends. To be able to add words or other indexed terms in such languages to a FULLTEXT index, you must preprocess them so that they are separated by some arbitrary delimiter.

It is possible to write a plugin that replaces the built-in full-text parser (only supported with MyISAM). For details, see The MySQL Plugin API. For example parser plugin source code, see the plugin/fulltext directory of a MySQL source distribution.

Some words are ignored in full-text searches:

- Any word that is too short is ignored. The default minimum length of words that are found by full-text searches is three characters for InnoDB search indexes, or four characters for

MyISAM. You can control the cutoff by setting a configuration option before creating the index: innodb_ft_min_token_size configuration option for InnoDB search indexes, or ft_min_word_len for MyISAM.

- Words in the stopword list are ignored. A stopword is a word such as “the” or “some” that is so common that it is considered to have zero semantic value. There is a built-in stopword list, but it can be overridden by a user-defined list. The stopword lists and related configuration options are different for InnoDB search indexes and MyISAM ones. Stopword processing is controlled by the configuration options innodb_ft_enable_stopword, innodb_ft_server_stopword_table, and innodb_ft_user_stopword_table for InnoDB search indexes, and ft_stopword_file for MyISAM ones.

See Section 12.10.4, “Full-Text Stopwords” to view default stopword lists and how to change them. The default minimum word length can be changed as described in Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”.

Every correct word in the collection and in the query is weighted according to its significance in the collection or query. Thus, a word that is present in many documents has a lower weight, because it has lower semantic value in this particular collection. Conversely, if the word is rare, it receives a higher weight. The weights of the words are combined to compute the relevance of the row. This technique works best with large collections.

MyISAM Limitation

For very small tables, word distribution does not adequately reflect their semantic value, and this model may sometimes produce bizarre results for search indexes on MyISAM tables. For example, although the word “MySQL” is present in every row of the articles table shown earlier, a search for the word in a MyISAM search index produces no results:

```
mysql> SELECT * FROM articles
    WHERE MATCH (title,body)
    AGAINST ('MySQL' IN NATURAL LANGUAGE MODE);
Empty set (0.00 sec)
```

The search result is empty because the word “MySQL” is present in at least 50% of the rows, and so is effectively treated as a stopword. This filtering technique is more suitable for large data sets, where you might not want the result set to return every second row from a 1GB table, than for small data sets where it might cause poor results for popular terms.

The 50% threshold can surprise you when you first try full-text searching to see how it works, and makes InnoDB tables more suited to experimentation with full-text searches. If you create a MyISAM table and insert only one or two rows of text into it, every word in the text occurs in at least 50% of the rows. As a result, no search returns any results until the table contains more rows. Users who need to bypass the 50% limitation can build search indexes on InnoDB tables, or use the boolean search mode explained in Section 12.10.2, “Boolean Full-Text Searches”.

12.10.2 Boolean Full-Text Searches

MySQL can perform boolean full-text searches using the IN BOOLEAN MODE modifier. With this modifier, certain characters have special meaning at the beginning or end of words in the search string. In the following query, the + and - operators indicate that a word must be present or absent, respectively, for a match to occur. Thus, the query retrieves all the rows that contain the word “MySQL” but that do *not* contain the word “YourSQL”:

```
mysql> SELECT * FROM articles WHERE MATCH (title,body)
    AGAINST ('+MySQL -YourSQL' IN BOOLEAN MODE);
+----+-----------------------+-------------------------------------+
```

```
| id | title                 | body                                |
+----+-----------------------+-------------------------------------+
|  1 | MySQL Tutorial        | DBMS stands for DataBase ...        |
|  2 | How To Use MySQL Well | After you went through a ...        |
|  3 | Optimizing MySQL      | In this tutorial we show ...        |
|  4 | 1001 MySQL Tricks     | 1. Never run mysqld as root. 2. ... |
|  6 | MySQL Security        | When configured properly, MySQL ... |
+----+-----------------------+-------------------------------------+
```

Note

In implementing this feature, MySQL uses what is sometimes referred to as *implied Boolean logic*, in which

- `+` stands for `AND`
- `-` stands for `NOT`
- [*no operator*] implies `OR`

Boolean full-text searches have these characteristics:

- They do not automatically sort rows in order of decreasing relevance.
- `InnoDB` tables require a `FULLTEXT` index on all columns of the `MATCH()` expression to perform boolean queries. Boolean queries against a `MyISAM` search index can work even without a `FULLTEXT` index, although a search executed in this fashion would be quite slow.
- The minimum and maximum word length full-text parameters apply: `innodb_ft_min_token_size` and `innodb_ft_max_token_size` for `InnoDB` search indexes, and `ft_min_word_len` and `ft_max_word_len` for `MyISAM` ones.
- The stopword list applies, controlled by `innodb_ft_enable_stopword`, `innodb_ft_server_stopword_table`, and `innodb_ft_user_stopword_table` for `InnoDB` search indexes, and `ft_stopword_file` for `MyISAM` ones.
- `InnoDB` full-text search does not support the use of multiple operators on a single search word, as in this example: `'++apple'`. MyISAM full-text search successfully processes the same search ignoring all operators except for the operator immediately adjacent to the search word.
- `InnoDB` full-text search only supports leading plus or minus signs. For example, `InnoDB` supports `'+apple'` but does not support `'apple+'`. Specifying a trailing plus or minus sign causes `InnoDB` to report a syntax error.
- `InnoDB` full-text search does not support the use of a leading plus sign with wildcard (`'+*'`), a plus and minus sign combination (`'+-'`), or leading a plus and minus sign combination (`'+-apple'`). These invalid queries return a syntax error.
- `InnoDB` full-text search does not support the use of the `@` symbol in boolean full-text searches. The `@` symbol is reserved for use by the `@distance` proximity search operator.
- They do not use the 50% threshold that applies to `MyISAM` search indexes.

The boolean full-text search capability supports the following operators:

- `+`

  A leading or trailing plus sign indicates that this word *must* be present in each row that is returned. `InnoDB` only supports leading plus signs.

- `-`

  A leading or trailing minus sign indicates that this word must *not* be present in any of the rows that are returned. `InnoDB` only supports leading minus signs.

The `-` operator acts only to exclude rows that are otherwise matched by other search terms. Thus, a boolean-mode search that contains only terms preceded by `-` returns an empty result. It does not return “all rows except those containing any of the excluded terms.”

- (no operator)

  By default (when neither `+` nor `-` is specified), the word is optional, but the rows that contain it are rated higher. This mimics the behavior of `MATCH() ... AGAINST()` without the `IN BOOLEAN MODE` modifier.

- *`@distance`*

  This operator works on `InnoDB` tables only. It tests whether two or more words all start within a specified distance from each other, measured in words. Specify the search words within a double-quoted string immediately before the *`@distance`* operator, for example, `MATCH(col1) AGAINST('"word1 word2 word3" @8' IN BOOLEAN MODE)`

- `> <`

  These two operators are used to change a word's contribution to the relevance value that is assigned to a row. The `>` operator increases the contribution and the `<` operator decreases it. See the example following this list.

- `( )`

  Parentheses group words into subexpressions. Parenthesized groups can be nested.

- `~`

  A leading tilde acts as a negation operator, causing the word's contribution to the row's relevance to be negative. This is useful for marking “noise” words. A row containing such a word is rated lower than others, but is not excluded altogether, as it would be with the `-` operator.

- `*`

  The asterisk serves as the truncation (or wildcard) operator. Unlike the other operators, it is *appended* to the word to be affected. Words match if they begin with the word preceding the `*` operator.

  If a word is specified with the truncation operator, it is not stripped from a boolean query, even if it is too short or a stopword. Whether a word is too short is determined from the `innodb_ft_min_token_size` setting for `InnoDB` tables, or `ft_min_word_len` for `MyISAM` tables. The wildcarded word is considered as a prefix that must be present at the start of one or more words. If the minimum word length is 4, a search for `'+word +the*'` could return fewer rows than a search for `'+word +the'`, because the second query ignores the too-short search term `the`.

- `"`

  A phrase that is enclosed within double quote (`"`) characters matches only rows that contain the phrase *literally, as it was typed*. The full-text engine splits the phrase into words and performs a search in the `FULLTEXT` index for the words. Nonword characters need not be matched exactly: Phrase searching requires only that matches contain exactly the same words as the phrase and in the same order. For example, `"test phrase"` matches `"test, phrase"`.

  If the phrase contains no words that are in the index, the result is empty. The words might not be in the index because of a combination of factors: if they do not exist in the text, are stopwords, or are shorter than the minimum length of indexed words.

The following examples demonstrate some search strings that use boolean full-text operators:

- `'apple banana'`

Find rows that contain at least one of the two words.

- `'+apple +juice'`

  Find rows that contain both words.

- `'+apple macintosh'`

  Find rows that contain the word “apple”, but rank rows higher if they also contain “macintosh”.

- `'+apple -macintosh'`

  Find rows that contain the word “apple” but not “macintosh”.

- `'+apple ~macintosh'`

  Find rows that contain the word “apple”, but if the row also contains the word “macintosh”, rate it lower than if row does not. This is “softer” than a search for `'+apple -macintosh'`, for which the presence of “macintosh” causes the row not to be returned at all.

- `'+apple +(>turnover <strudel)'`

  Find rows that contain the words “apple” and “turnover”, or “apple” and “strudel” (in any order), but rank “apple turnover” higher than “apple strudel”.

- `'apple*'`

  Find rows that contain words such as “apple”, “apples”, “applesauce”, or “applet”.

- `'"some words"'`

  Find rows that contain the exact phrase “some words” (for example, rows that contain “some words of wisdom” but not “some noise words”).

Note

The `"` characters that enclose the phrase are operator characters that delimit the phrase. They are not the quotation marks that enclose the search string itself.

Relevancy Rankings for InnoDB Boolean Mode Search

`InnoDB` full-text search is modeled on the Sphinx full-text search engine, and the algorithms used are based on BM25 and TF-IDF ranking algorithms. For these reasons, relevancy rankings for `InnoDB` boolean full-text search may differ from `MyISAM` relevancy rankings.

`InnoDB` uses a variation of the “term frequency-inverse document frequency” (`TF-IDF`) weighting system to rank a document's relevance for a given full-text search query. The `TF-IDF` weighting is based on how frequently a word appears in a document, offset by how frequently the word appears in all documents in the collection. In other words, the more frequently a word appears in a document, and the less frequently the word appears in the document collection, the higher the document is ranked.

How Relevancy Ranking is Calculated

The term frequency (`TF`) value is the number of times that a word appears in a document. The inverse document frequency (`IDF`) value of a word is calculated using the following formula, where `total_records` is the number of records in the collection, and `matching_records` is the number of records that the search term appears in.

```
${IDF} = log10( ${total_records} / ${matching_records} )
```

When a document contains a word multiple times, the IDF value is multiplied by the TF value:

```
${TF} * ${IDF}
```

Using the `TF` and `IDF` values, the relevancy ranking for a document is calculated using this formula:

```
${rank} = ${TF} * ${IDF} * ${IDF}
```

The formula is demonstrated in the following examples.

Relevancy Ranking for a Single Word Search

This example demonstrates the relevancy ranking calculation for a single-word search.

```
mysql> CREATE TABLE articles (
id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
title VARCHAR(200),
body TEXT,
FULLTEXT (title,body)
) ENGINE=InnoDB;
Query OK, 0 rows affected (1.04 sec)

mysql> INSERT INTO articles (title,body) VALUES
('MySQL Tutorial','This database tutorial ...'),
("How To Use MySQL",'After you went through a ...'),
('Optimizing Your Database','In this database tutorial ...'),
('MySQL vs. YourSQL','When comparing databases ...'),
('MySQL Security','When configured properly, MySQL ...'),
('Database, Database, Database','database database database'),
('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
('MySQL Full-Text Indexes', 'MySQL fulltext indexes use a ..');
Query OK, 8 rows affected (0.06 sec)
Records: 8  Duplicates: 0  Warnings: 0

mysql> SELECT id, title, body, MATCH (title,body)  AGAINST ('database' IN BOOLEAN MODE)
AS score FROM articles ORDER BY score DESC;
+----+-----------------------------+-------------------------------------+----------------------+
| id | title                       | body                                | score                |
+----+-----------------------------+-------------------------------------+----------------------+
|  6 | Database, Database, Database | database database database          |  1.0886961221694946 |
|  3 | Optimizing Your Database     | In this database tutorial ...       | 0.36289870738983154 |
|  1 | MySQL Tutorial               | This database tutorial ...          | 0.18144935369491577 |
|  2 | How To Use MySQL             | After you went through a ...        |                   0 |
|  4 | MySQL vs. YourSQL            | When comparing databases ...        |                   0 |
|  5 | MySQL Security               | When configured properly, MySQL ... |                   0 |
|  7 | 1001 MySQL Tricks            | 1. Never run mysqld as root. 2. ... |                   0 |
|  8 | MySQL Full-Text Indexes      | MySQL fulltext indexes use a ..     |                   0 |
+----+-----------------------------+-------------------------------------+----------------------+
8 rows in set (0.00 sec)
```

There are 8 records in total, with 3 that match the “database” search term. The first record (`id 6`) contains the search term 6 times and has a relevancy ranking of `1.0886961221694946`. This ranking value is calculated using a `TF` value of 6 (the “database” search term appears 6 times in record `id 6`) and an `IDF` value of 0.42596873216370745, which is calculated as follows (where 8 is the total number of records and 3 is the number of records that the search term appears in):

```
${IDF} = log10( 8 / 3 ) = 0.42596873216370745
```

The `TF` and `IDF` values are then entered into the ranking formula:

```
${rank} = ${TF} * ${IDF} * ${IDF}
```

Performing the calculation in the MySQL command-line client returns a ranking value of 1.088696164686938.

```
mysql> SELECT 6*log10(8/3)*log10(8/3);
+-------------------------+
| 6*log10(8/3)*log10(8/3) |
+-------------------------+
|       1.088696164686938 |
+-------------------------+
```

```
1 row in set (0.00 sec)
```

Note

You may notice a slight difference in the ranking values returned by the `SELECT ... MATCH ... AGAINST` statement and the MySQL command-line client (`1.0886961221694946` versus `1.088696164686938`). The difference is due to how the casts between integers and floats/doubles are performed internally by `InnoDB` (along with related precision and rounding decisions), and how they are performed elsewhere, such as in the MySQL command-line client or other types of calculators.

Relevancy Ranking for a Multiple Word Search

This example demonstrates the relevancy ranking calculation for a multiple-word full-text search based on the `articles` table and data used in the previous example.

If you search on more than one word, the relevancy ranking value is a sum of the relevancy ranking value for each word, as shown in this formula:

```
${rank} = ${TF} * ${IDF} * ${IDF} + ${TF} * ${IDF} * ${IDF}
```

Performing a search on two terms ('mysql tutorial') returns the following results:

```
mysql> SELECT id, title, body, MATCH (title,body)  AGAINST ('mysql tutorial' IN BOOLEAN MODE)
    AS score FROM articles ORDER BY score DESC;
+----+-----------------------------+-------------------------------------+----------------------+
| id | title                       | body                                | score                |
+----+-----------------------------+-------------------------------------+----------------------+
|  1 | MySQL Tutorial              | This database tutorial ...          |   0.7405621409416199 |
|  3 | Optimizing Your Database    | In this database tutorial ...       |   0.3624762296676636 |
|  5 | MySQL Security              | When configured properly, MySQL ... | 0.031219376251101494 |
|  8 | MySQL Full-Text Indexes     | MySQL fulltext indexes use a ..     | 0.031219376251101494 |
|  2 | How To Use MySQL            | After you went through a ...        | 0.015609688125550747 |
|  4 | MySQL vs. YourSQL           | When comparing databases ...        | 0.015609688125550747 |
|  7 | 1001 MySQL Tricks           | 1. Never run mysqld as root. 2. ... | 0.015609688125550747 |
|  6 | Database, Database, Database | database database database          |                    0 |
+----+-----------------------------+-------------------------------------+----------------------+
8 rows in set (0.00 sec)
```

In the first record (`id 8`), 'mysql' appears once and 'tutorial' appears twice. There are six matching records for 'mysql' and two matching records for 'tutorial'. The MySQL command-line client returns the expected ranking value when inserting these values into the ranking formula for a multiple word search:

```
mysql> SELECT (1*log10(8/6)*log10(8/6)) + (2*log10(8/2)*log10(8/2));
+-----------------------------------------------------+
| (1*log10(8/6)*log10(8/6)) + (2*log10(8/2)*log10(8/2)) |
+-----------------------------------------------------+
|                                  0.7405621541938003 |
+-----------------------------------------------------+
1 row in set (0.00 sec)
```

Note

The slight difference in the ranking values returned by the `SELECT ... MATCH ... AGAINST` statement and the MySQL command-line client is explained in the preceding example.

12.10.3 Full-Text Searches with Query Expansion

Full-text search supports query expansion (and in particular, its variant “blind query expansion”). This is generally useful when a search phrase is too short, which often means that the user is relying on implied knowledge that the full-text search engine lacks. For example, a user searching for “database” may really mean that “MySQL”, “Oracle”, “DB2”, and “RDBMS” all are phrases that should match “databases” and should be returned, too. This is implied knowledge.

Blind query expansion (also known as automatic relevance feedback) is enabled by adding `WITH QUERY EXPANSION` or `IN NATURAL LANGUAGE MODE WITH QUERY EXPANSION` following the search phrase. It works by performing the search twice, where the search phrase for the second search is the original search phrase concatenated with the few most highly relevant documents from the first search. Thus, if one of these documents contains the word “databases” and the word “MySQL”, the second search finds the documents that contain the word “MySQL” even if they do not contain the word “database”. The following example shows this difference:

```
mysql> SELECT * FROM articles
    WHERE MATCH (title,body)
    AGAINST ('database' IN NATURAL LANGUAGE MODE);
+----+-------------------+----------------------------------------+
| id | title             | body                                   |
+----+-------------------+----------------------------------------+
|  1 | MySQL Tutorial    | DBMS stands for DataBase ...           |
|  5 | MySQL vs. YourSQL | In the following database comparison ... |
+----+-------------------+----------------------------------------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM articles
    WHERE MATCH (title,body)
    AGAINST ('database' WITH QUERY EXPANSION);
+----+-----------------------+----------------------------------------+
| id | title                 | body                                   |
+----+-----------------------+----------------------------------------+
|  5 | MySQL vs. YourSQL     | In the following database comparison ... |
|  1 | MySQL Tutorial        | DBMS stands for DataBase ...           |
|  3 | Optimizing MySQL      | In this tutorial we show ...           |
|  6 | MySQL Security        | When configured properly, MySQL ...    |
|  2 | How To Use MySQL Well | After you went through a ...           |
|  4 | 1001 MySQL Tricks     | 1. Never run mysqld as root. 2. ...    |
+----+-----------------------+----------------------------------------+
6 rows in set (0.00 sec)
```

Another example could be searching for books by Georges Simenon about Maigret, when a user is not sure how to spell “Maigret”. A search for “Megre and the reluctant witnesses” finds only “Maigret and the Reluctant Witnesses” without query expansion. A search with query expansion finds all books with the word “Maigret” on the second pass.

Note

Because blind query expansion tends to increase noise significantly by returning nonrelevant documents, use it only when a search phrase is short.

12.10.4 Full-Text Stopwords

The stopword list is loaded and searched for full-text queries using the server character set and collation (the values of the `character_set_server` and `collation_server` system variables). False hits or misses might occur for stopword lookups if the stopword file or columns used for full-text indexing or searches have a character set or collation different from `character_set_server` or `collation_server`.

Case sensitivity of stopword lookups depends on the server collation. For example, lookups are case-insensitive if the collation is `latin1_swedish_ci`, whereas lookups are case-sensitive if the collation is `latin1_general_cs` or `latin1_bin`.

- Stopwords for InnoDB Search Indexes
- Stopwords for MyISAM Search Indexes

Stopwords for InnoDB Search Indexes

`InnoDB` has a relatively short list of default stopwords, because documents from technical, literary, and other sources often use short words as keywords or in significant phrases. For example, you might

search for “to be or not to be” and expect to get a sensible result, rather than having all those words ignored.

To see the default `InnoDB` stopword list, query the `INFORMATION_SCHEMA.INNODB_FT_DEFAULT_STOPWORD` table.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_DEFAULT_STOPWORD;
+-------+
| value |
+-------+
| a     |
| about |
| an    |
| are   |
| as    |
| at    |
| be    |
| by    |
| com   |
| de    |
| en    |
| for   |
| from  |
| how   |
| i     |
| in    |
| is    |
| it    |
| la    |
| of    |
| on    |
| or    |
| that  |
| the   |
| this  |
| to    |
| was   |
| what  |
| when  |
| where |
| who   |
| will  |
| with  |
| und   |
| the   |
| www   |
+-------+
36 rows in set (0.00 sec)
```

To define your own stopword list for all `InnoDB` tables, define a table with the same structure as the `INNODB_FT_DEFAULT_STOPWORD` table, populate it with stopwords, and set the value of the `innodb_ft_server_stopword_table` option to a value in the form *`db_name/table_name`* before creating the full-text index. The stopword table must have a single `VARCHAR` column named `value`. The following example demonstrates creating and configuring a new global stopword table for `InnoDB`.

```
-- Create a new stopword table

mysql> CREATE TABLE my_stopwords(value VARCHAR(30)) ENGINE = INNODB;
Query OK, 0 rows affected (0.01 sec)

-- Insert stopwords (for simplicity, a single stopword is used in this example)

mysql> INSERT INTO my_stopwords(value) VALUES ('Ishmael');
Query OK, 1 row affected (0.00 sec)

-- Create the table

mysql> CREATE TABLE opening_lines (
id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
opening_line TEXT(500),
```

```
author VARCHAR(200),
title VARCHAR(200)
) ENGINE=InnoDB;
Query OK, 0 rows affected (0.01 sec)

-- Insert data into the table

mysql> INSERT INTO opening_lines(opening_line,author,title) VALUES
('Call me Ishmael.','Herman Melville','Moby-Dick'),
('A screaming comes across the sky.','Thomas Pynchon','Gravity\'s Rainbow'),
('I am an invisible man.','Ralph Ellison','Invisible Man'),
('Where now? Who now? When now?','Samuel Beckett','The Unnamable'),
('It was love at first sight.','Joseph Heller','Catch-22'),
('All this happened, more or less.','Kurt Vonnegut','Slaughterhouse-Five'),
('Mrs. Dalloway said she would buy the flowers herself.','Virginia Woolf','Mrs. Dalloway'),
('It was a pleasure to burn.','Ray Bradbury','Fahrenheit 451');
Query OK, 8 rows affected (0.00 sec)
Records: 8  Duplicates: 0  Warnings: 0

-- Set the innodb_ft_server_stopword_table option to the new stopword table

mysql> SET GLOBAL innodb_ft_server_stopword_table = 'test/my_stopwords';
Query OK, 0 rows affected (0.00 sec)

-- Create the full-text index (which rebuilds the table if no FTS_DOC_ID column is defined)

mysql> CREATE FULLTEXT INDEX idx ON opening_lines(opening_line);
Query OK, 0 rows affected, 1 warning (1.17 sec)
Records: 0  Duplicates: 0  Warnings: 1
```

Verify that the specified stopword ('Ishmael') does not appear by querying the words in `INFORMATION_SCHEMA.INNODB_FT_INDEX_TABLE`.

Note

By default, words less than 3 characters in length or greater than 84 characters in length do not appear in an `InnoDB` full-text search index. Maximum and minimum word length values are configurable using the `innodb_ft_max_token_size` and `innodb_ft_min_token_size` variables.

```
mysql> SET GLOBAL innodb_ft_aux_table='test/opening_lines';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT word FROM INFORMATION_SCHEMA.INNODB_FT_INDEX_TABLE LIMIT 15;
+-----------+
| word      |
+-----------+
| across    |
| all       |
| burn      |
| buy       |
| call      |
| comes     |
| dalloway  |
| first     |
| flowers   |
| happened  |
| herself   |
| invisible |
| less      |
| love      |
| man       |
+-----------+
15 rows in set (0.00 sec)
```

To create stopword lists on a table-by-table basis, create other stopword tables and use the `innodb_ft_user_stopword_table` option to specify the stopword table that you want to use before you create the full-text index.

Stopwords for MyISAM Search Indexes

The stopword file is loaded and searched using `latin1` if `character_set_server` is `ucs2`, `utf16`, `utf16le`, or `utf32`.

To override the default stopword list for MyISAM tables, set the `ft_stopword_file` system variable. (See Section 5.1.7, “Server System Variables”.) The variable value should be the path name of the file containing the stopword list, or the empty string to disable stopword filtering. The server looks for the file in the data directory unless an absolute path name is given to specify a different directory. After changing the value of this variable or the contents of the stopword file, restart the server and rebuild your `FULLTEXT` indexes.

The stopword list is free-form, separating stopwords with any nonalphanumeric character such as newline, space, or comma. Exceptions are the underscore character (_) and a single apostrophe (') which are treated as part of a word. The character set of the stopword list is the server's default character set; see Section 10.3.2, “Server Character Set and Collation”.

The following list shows the default stopwords for `MyISAM` search indexes. In a MySQL source distribution, you can find this list in the `storage/myisam/ft_static.c` file.

```
a's           able          about         above         according
accordingly   across        actually      after         afterwards
again         against       ain't         all           allow
allows        almost        alone         along         already
also          although      always        am            among
amongst       an            and           another       any
anybody       anyhow        anyone        anything      anyway
anyways       anywhere      apart         appear        appreciate
appropriate   are           aren't        around        as
aside         ask           asking        associated    at
available     away          awfully       be            became
because       become        becomes       becoming      been
before        beforehand    behind        being         believe
below         beside        besides       best          better
between       beyond        both          brief         but
by            c'mon         c's           came          can
can't         cannot        cant          cause         causes
certain       certainly     changes       clearly       co
com           come          comes         concerning    consequently
consider      considering   contain       containing    contains
corresponding could         couldn't      course        currently
definitely    described     despite       did           didn't
different     do            does          doesn't       doing
don't         done          down          downwards     during
each          edu           eg            eight         either
else          elsewhere     enough        entirely      especially
et            etc           even          ever          every
everybody     everyone      everything    everywhere    ex
exactly       example       except        far           few
fifth         first         five          followed      following
follows       for           former        formerly      forth
four          from          further       furthermore   get
gets          getting       given         gives         go
goes          going         gone          got           gotten
greetings     had           hadn't        happens       hardly
has           hasn't        have          haven't       having
he            he's          hello         help          hence
her           here          here's        hereafter     hereby
herein        hereupon      hers          herself       hi
him           himself       his           hither        hopefully
how           howbeit       however       i'd           i'll
i'm           i've          ie            if            ignored
immediate     in            inasmuch      inc           indeed
indicate      indicated     indicates     inner         insofar
instead       into          inward        is            isn't
it            it'd          it'll         it's          its
itself        just          keep          keeps         kept
know          known         knows         last          lately
```

```
later         latter        latterly      least         less
lest          let           let's         like          liked
likely        little        look          looking       looks
ltd           mainly        many          may           maybe
me            mean          meanwhile     merely        might
more          moreover      most          mostly        much
must          my            myself        name          namely
nd            near          nearly        necessary     need
needs         neither       never         nevertheless  new
next          nine          no            nobody        non
none          noone         nor           normally      not
nothing       novel         now           nowhere       obviously
of            off           often         oh            ok
okay          old           on            once          one
ones          only          onto          or            other
others        otherwise     ought         our           ours
ourselves     out           outside       over          overall
own           particular    particularly  per           perhaps
placed        please        plus          possible      presumably
probably      provides      que           quite         qv
rather        rd            re            really        reasonably
regarding     regardless    regards       relatively    respectively
right         said          same          saw           say
saying        says          second        secondly      see
seeing        seem          seemed        seeming       seems
seen          self          selves        sensible      sent
serious       seriously     seven         several       shall
she           should        shouldn't     since         six
so            some          somebody      somehow       someone
something     sometime      sometimes     somewhat      somewhere
soon          sorry         specified     specify       specifying
still         sub           such          sup           sure
t's           take          taken         tell          tends
th            than          thank         thanks        thanx
that          that's        thats         the           their
theirs        them          themselves    then          thence
there         there's       thereafter    thereby       therefore
therein       theres        thereupon     these         they
they'd        they'll       they're       they've       think
third         this          thorough      thoroughly    those
though        three         through       throughout    thru
thus          to            together      too           took
toward        towards       tried         tries         truly
try           trying        twice         two           un
under         unfortunately unless        unlikely      until
unto          up            upon          us            use
used          useful        uses          using         usually
value         various       very          via           viz
vs            want          wants         was           wasn't
way           we            we'd          we'll         we're
we've         welcome       well          went          were
weren't       what          what's        whatever      when
whence        whenever      where         where's       whereafter
whereas       whereby       wherein       whereupon     wherever
whether       which         while         whither       who
who's         whoever       whole         whom          whose
why           will          willing       wish          with
within        without       won't         wonder        would
wouldn't      yes           yet           you           you'd
you'll        you're        you've        your          yours
yourself      yourselves    zero
```

12.10.5 Full-Text Restrictions

- Full-text searches are supported for `InnoDB` and `MyISAM` tables only.
- Full-text searches are not supported for partitioned tables. See Section 19.6, “Restrictions and Limitations on Partitioning”.
- Full-text searches can be used with most multibyte character sets. The exception is that for Unicode, the `utf8` character set can be used, but not the `ucs2` character set. Although `FULLTEXT` indexes on

`ucs2` columns cannot be used, you can perform `IN BOOLEAN MODE` searches on a `ucs2` column that has no such index.

The remarks for `utf8` also apply to `utf8mb4`, and the remarks for `ucs2` also apply to `utf16`, `utf16le`, and `utf32`.

- Ideographic languages such as Chinese and Japanese do not have word delimiters. Therefore, the `FULLTEXT` parser *cannot determine where words begin and end in these and other such languages*.
- Although the use of multiple character sets within a single table is supported, all columns in a `FULLTEXT` index must use the same character set and collation.
- The `MATCH()` column list must match exactly the column list in some `FULLTEXT` index definition for the table, unless this `MATCH()` is `IN BOOLEAN MODE` on a `MyISAM` table. For `MyISAM` tables, boolean-mode searches can be done on nonindexed columns, although they are likely to be slow.
- The argument to `AGAINST()` must be a string value that is constant during query evaluation. This rules out, for example, a table column because that can differ for each row.
- Index hints are more limited for `FULLTEXT` searches than for non-`FULLTEXT` searches. See Section 8.9.3, “Index Hints”.
- For `InnoDB`, all DML operations (`INSERT`, `UPDATE`, `DELETE`) involving columns with full-text indexes are processed at transaction commit time. For example, for an `INSERT` operation, an inserted string is tokenized and decomposed into individual words. The individual words are then added to full-text index tables when the transaction is committed. As a result, full-text searches only return committed data.
- The '%' character is not a supported wildcard character for full-text searches.

12.10.6 Fine-Tuning MySQL Full-Text Search

MySQL's full-text search capability has few user-tunable parameters. You can exert more control over full-text searching behavior if you have a MySQL source distribution because some changes require source code modifications. See Section 2.9, “Installing MySQL from Source”.

Full-text search is carefully tuned for effectiveness. Modifying the default behavior in most cases can actually decrease effectiveness. *Do not alter the MySQL sources unless you know what you are doing*.

Most full-text variables described in this section must be set at server startup time. A server restart is required to change them; they cannot be modified while the server is running.

Some variable changes require that you rebuild the `FULLTEXT` indexes in your tables. Instructions for doing so are given later in this section.

- Configuring Minimum and Maximum Word Length
- Configuring the Natural Language Search Threshold
- Modifying Boolean Full-Text Search Operators
- Character Set Modifications
- Rebuilding InnoDB Full-Text Indexes
- Optimizing InnoDB Full-Text Indexes
- Rebuilding MyISAM Full-Text Indexes

Configuring Minimum and Maximum Word Length

The minimum and maximum lengths of words to be indexed are defined by the `innodb_ft_min_token_size` and `innodb_ft_max_token_size` for `InnoDB` search indexes, and

`ft_min_word_len` and `ft_max_word_len` for `MyISAM` ones. After changing any of these options, rebuild your `FULLTEXT` indexes for the change to take effect. For example, to make two-character words searchable, you could put the following lines in an option file:

```
[mysqld]
innodb_ft_min_token_size=2
ft_min_word_len=2
```

Then restart the server and rebuild your `FULLTEXT` indexes. For `MyISAM` tables, note the remarks regarding `myisamchk` in the instructions that follow for rebuilding `MyISAM` full-text indexes.

Configuring the Natural Language Search Threshold

For `MyISAM` search indexes, the 50% threshold for natural language searches is determined by the particular weighting scheme chosen. To disable it, look for the following line in `storage/myisam/ftdefs.h`:

```
#define GWS_IN_USE GWS_PROB
```

Change that line to this:

```
#define GWS_IN_USE GWS_FREQ
```

Then recompile MySQL. There is no need to rebuild the indexes in this case.

Note

By making this change, you *severely* decrease MySQL's ability to provide adequate relevance values for the `MATCH()` function. If you really need to search for such common words, it would be better to search using `IN BOOLEAN MODE` instead, which does not observe the 50% threshold.

Modifying Boolean Full-Text Search Operators

To change the operators used for boolean full-text searches on `MyISAM` tables, set the `ft_boolean_syntax` system variable. (`InnoDB` does not have an equivalent setting.) This variable can be changed while the server is running, but you must have privileges sufficient to set global system variables (see Section 5.1.8.1, “System Variable Privileges”). No rebuilding of indexes is necessary in this case.

Character Set Modifications

You can change the set of characters that are considered word characters in several ways, as described in the following list. After making the modification, rebuild the indexes for each table that contains any `FULLTEXT` indexes. Suppose that you want to treat the hyphen character ('-') as a word character. Use one of these methods:

- Modify the MySQL source: In `storage/innobase/handler/ha_innodb.cc` (for `InnoDB`), or in `storage/myisam/ftdefs.h` (for `MyISAM`), see the `true_word_char()` and `misc_word_char()` macros. Add `'-'` to one of those macros and recompile MySQL.

- Modify a character set file: This requires no recompilation. The `true_word_char()` macro uses a “character type” table to distinguish letters and numbers from other characters. . You can edit the contents of the `<ctype><map>` array in one of the character set XML files to specify that `'-'` is a “letter.” Then use the given character set for your `FULLTEXT` indexes. For information about the `<ctype><map>` array format, see Section 10.13.1, “Character Definition Arrays”.

- Add a new collation for the character set used by the indexed columns, and alter the columns to use that collation. For general information about adding collations, see Section 10.14, “Adding a Collation to a Character Set”. For an example specific to full-text indexing, see Section 12.10.7, “Adding a User-Defined Collation for Full-Text Indexing”.

Rebuilding InnoDB Full-Text Indexes

For the changes to take effect, FULLTEXT indexes must be rebuilt after modifying any of the following full-text index variables: innodb_ft_min_token_size; innodb_ft_max_token_size; innodb_ft_server_stopword_table; innodb_ft_user_stopword_table; innodb_ft_enable_stopword. Modifying innodb_ft_min_token_size or innodb_ft_max_token_size requires restarting the server.

To rebuild FULLTEXT indexes for an InnoDB table, use ALTER TABLE with the DROP INDEX and ADD INDEX options to drop and re-create each index.

Optimizing InnoDB Full-Text Indexes

Running OPTIMIZE TABLE on a table with a full-text index rebuilds the full-text index, removing deleted Document IDs and consolidating multiple entries for the same word, where possible.

To optimize a full-text index, enable innodb_optimize_fulltext_only and run OPTIMIZE TABLE.

```
mysql> set GLOBAL innodb_optimize_fulltext_only=ON;
Query OK, 0 rows affected (0.01 sec)

mysql> OPTIMIZE TABLE opening_lines;
+--------------------+----------+----------+----------+
| Table              | Op       | Msg_type | Msg_text |
+--------------------+----------+----------+----------+
| test.opening_lines | optimize | status   | OK       |
+--------------------+----------+----------+----------+
1 row in set (0.01 sec)
```

To avoid lengthy rebuild times for full-text indexes on large tables, you can use the innodb_ft_num_word_optimize option to perform the optimization in stages. The innodb_ft_num_word_optimize option defines the number of words that are optimized each time OPTIMIZE TABLE is run. The default setting is 2000, which means that 2000 words are optimized each time OPTIMIZE TABLE is run. Subsequent OPTIMIZE TABLE operations continue from where the preceding OPTIMIZE TABLE operation ended.

Rebuilding MyISAM Full-Text Indexes

If you modify full-text variables that affect indexing (ft_min_word_len, ft_max_word_len, or ft_stopword_file), or if you change the stopword file itself, you must rebuild your FULLTEXT indexes after making the changes and restarting the server.

To rebuild the FULLTEXT indexes for a MyISAM table, it is sufficient to do a QUICK repair operation:

```
mysql> REPAIR TABLE tbl_name QUICK;
```

Alternatively, use ALTER TABLE as just described. In some cases, this may be faster than a repair operation.

Each table that contains any FULLTEXT index must be repaired as just shown. Otherwise, queries for the table may yield incorrect results, and modifications to the table cause the server to see the table as corrupt and in need of repair.

If you use myisamchk to perform an operation that modifies MyISAM table indexes (such as repair or analyze), the FULLTEXT indexes are rebuilt using the *default* full-text parameter values for minimum word length, maximum word length, and stopword file unless you specify otherwise. This can result in queries failing.

The problem occurs because these parameters are known only by the server. They are not stored in MyISAM index files. To avoid the problem if you have modified the minimum or maximum word length or stopword file values used by the server, specify the same ft_min_word_len, ft_max_word_len, and ft_stopword_file values for myisamchk that you use for mysqld. For example, if you have set the minimum word length to 3, you can repair a table with myisamchk like this:

```
myisamchk --recover --ft_min_word_len=3 tbl_name.MYI
```

To ensure that `myisamchk` and the server use the same values for full-text parameters, place each one in both the `[mysqld]` and `[myisamchk]` sections of an option file:

```
[mysqld]
ft_min_word_len=3

[myisamchk]
ft_min_word_len=3
```

An alternative to using `myisamchk` for `MyISAM` table index modification is to use the `REPAIR TABLE`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, or `ALTER TABLE` statements. These statements are performed by the server, which knows the proper full-text parameter values to use.

12.10.7 Adding a User-Defined Collation for Full-Text Indexing

This section describes how to add a user-defined collation for full-text searches using the built-in full-text parser. The sample collation is like `latin1_swedish_ci` but treats the `'-'` character as a letter rather than as a punctuation character so that it can be indexed as a word character. General information about adding collations is given in Section 10.14, “Adding a Collation to a Character Set”; it is assumed that you have read it and are familiar with the files involved.

To add a collation for full-text indexing, use the following procedure. The instructions here add a collation for a simple character set, which as discussed in Section 10.14, “Adding a Collation to a Character Set”, can be created using a configuration file that describes the character set properties. For a complex character set such as Unicode, create collations using C source files that describe the character set properties.

1. Add a collation to the `Index.xml` file. The permitted range of IDs for user-defined collations is given in Section 10.14.2, “Choosing a Collation ID”. The ID must be unused, so choose a value different from 1025 if that ID is already taken on your system.

   ```
   <charset name="latin1">
   ...
   <collation name="latin1_fulltext_ci" id="1025"/>
   </charset>
   ```

2. Declare the sort order for the collation in the `latin1.xml` file. In this case, the order can be copied from `latin1_swedish_ci`:

   ```
   <collation name="latin1_fulltext_ci">
   <map>
   00 01 02 03 04 05 06 07 08 09 0A 0B 0C 0D 0E 0F
   10 11 12 13 14 15 16 17 18 19 1A 1B 1C 1D 1E 1F
   20 21 22 23 24 25 26 27 28 29 2A 2B 2C 2D 2E 2F
   30 31 32 33 34 35 36 37 38 39 3A 3B 3C 3D 3E 3F
   40 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
   50 51 52 53 54 55 56 57 58 59 5A 5B 5C 5D 5E 5F
   60 41 42 43 44 45 46 47 48 49 4A 4B 4C 4D 4E 4F
   50 51 52 53 54 55 56 57 58 59 5A 7B 7C 7D 7E 7F
   80 81 82 83 84 85 86 87 88 89 8A 8B 8C 8D 8E 8F
   90 91 92 93 94 95 96 97 98 99 9A 9B 9C 9D 9E 9F
   A0 A1 A2 A3 A4 A5 A6 A7 A8 A9 AA AB AC AD AE AF
   B0 B1 B2 B3 B4 B5 B6 B7 B8 B9 BA BB BC BD BE BF
   41 41 41 41 5C 5B 5C 43 45 45 45 45 49 49 49 49
   44 4E 4F 4F 4F 4F 5D D7 D8 55 55 55 59 59 DE DF
   41 41 41 41 5C 5B 5C 43 45 45 45 45 49 49 49 49
   44 4E 4F 4F 4F 4F 5D F7 D8 55 55 55 59 59 DE FF
   </map>
   </collation>
   ```

3. Modify the `ctype` array in `latin1.xml`. Change the value corresponding to 0x2D (which is the code for the `'-'` character) from 10 (punctuation) to 01 (uppercase letter). In the following array, this is the element in the fourth row down, third value from the end.

```
<ctype>
<map>
00
20 20 20 20 20 20 20 20 20 28 28 28 28 28 20 20
20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
48 10 10 10 10 10 10 10 10 10 10 10 01 10 10 10
84 84 84 84 84 84 84 84 84 10 10 10 10 10 10 10
10 81 81 81 81 81 81 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 01 01 01 10 10 10 10 10 10
10 82 82 82 82 82 82 02 02 02 02 02 02 02 02 02
02 02 02 02 02 02 02 02 02 02 02 10 10 10 10 20
10 00 10 02 10 10 10 10 10 10 01 10 01 00 01 00
00 10 10 10 10 10 10 10 10 10 02 10 02 00 02 01
48 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
10 10 10 10 10 10 10 10 10 10 10 10 10 10 10 10
01 01 01 01 01 01 01 01 01 01 01 01 01 01 01 01
01 01 01 01 01 01 01 10 01 01 01 01 01 01 01 02
02 02 02 02 02 02 02 02 02 02 02 02 02 02 02 02
02 02 02 02 02 02 02 10 02 02 02 02 02 02 02 02
</map>
</ctype>
```

4. Restart the server.

5. To employ the new collation, include it in the definition of columns that are to use it:

```
mysql> DROP TABLE IF EXISTS t1;
Query OK, 0 rows affected (0.13 sec)

mysql> CREATE TABLE t1 (
    a TEXT CHARACTER SET latin1 COLLATE latin1_fulltext_ci,
    FULLTEXT INDEX(a)
    ) ENGINE=InnoDB;
Query OK, 0 rows affected (0.47 sec)
```

6. Test the collation to verify that hyphen is considered as a word character:

```
mysql> INSERT INTO t1 VALUEs ('----'),('....'),('abcd');
Query OK, 3 rows affected (0.22 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM t1 WHERE MATCH a AGAINST ('----' IN BOOLEAN MODE);
+------+
| a    |
+------+
| ---- |
+------+
1 row in set (0.00 sec)
```

12.11 Cast Functions and Operators

Table 12.15 Cast Functions and Operators

Name	Description
BINARY	Cast a string to a binary string
CAST()	Cast a value as a certain type
CONVERT()	Cast a value as a certain type

Cast functions and operators enable conversion of values from one data type to another.

CONVERT() with a USING clause converts data between different character sets:

```
CONVERT(expr USING transcoding_name)
```

In MySQL, transcoding names are the same as the corresponding character set names.

Examples:

```
SELECT CONVERT('test' USING utf8);
SELECT CONVERT(_latin1'Müller' USING utf8);
INSERT INTO utf8_table (utf8_column)
    SELECT CONVERT(latin1_column USING utf8) FROM latin1_table;
```

To convert strings between different character sets, you can also use `CONVERT(expr, type)` syntax (without `USING`), or `CAST(expr AS type)`, which is equivalent:

```
CONVERT(string, CHAR[(N)] CHARACTER SET charset_name)
CAST(string AS CHAR[(N)] CHARACTER SET charset_name)
```

Examples:

```
SELECT CONVERT('test', CHAR CHARACTER SET utf8);
SELECT CAST('test' AS CHAR CHARACTER SET utf8);
```

If you specify `CHARACTER SET charset_name` as just shown, the character set and collation of the result are `charset_name` and the default collation of `charset_name`. If you omit `CHARACTER SET charset_name`, the character set and collation of the result are defined by the `character_set_connection` and `collation_connection` system variables that determine the default connection character set and collation (see Section 10.4, “Connection Character Sets and Collations”).

A `COLLATE` clause is not permitted within a `CONVERT()` or `CAST()` call, but you can apply it to the function result. For example, these are legal:

```
SELECT CONVERT('test' USING utf8) COLLATE utf8_bin;
SELECT CONVERT('test', CHAR CHARACTER SET utf8) COLLATE utf8_bin;
SELECT CAST('test' AS CHAR CHARACTER SET utf8) COLLATE utf8_bin;
```

But these are illegal:

```
SELECT CONVERT('test' USING utf8 COLLATE utf8_bin);
SELECT CONVERT('test', CHAR CHARACTER SET utf8 COLLATE utf8_bin);
SELECT CAST('test' AS CHAR CHARACTER SET utf8 COLLATE utf8_bin);
```

Normally, you cannot compare a `BLOB` value or other binary string in case-insensitive fashion because binary strings use the `binary` character set, which has no collation with the concept of lettercase. To perform a case-insensitive comparison, first use the `CONVERT()` or `CAST()` function to convert the value to a nonbinary string. Comparisons of the resulting string use its collation. For example, if the conversion result character set has a case-insensitive collation, a `LIKE` operation is not case-sensitive. That is true for the following operation because the default `latin1` collation (`latin1_swedish_ci`) is not case-sensitive:

```
SELECT 'A' LIKE CONVERT(blob_col USING latin1)
  FROM tbl_name;
```

To specify a particular collation for the converted string, use a `COLLATE` clause following the `CONVERT()` call:

```
SELECT 'A' LIKE CONVERT(blob_col USING latin1) COLLATE latin1_german1_ci
  FROM tbl_name;
```

To use a different character set, substitute its name for `latin1` in the preceding statements (and similarly to use a different collation).

`CONVERT()` and `CAST()` can be used more generally for comparing strings represented in different character sets. For example, a comparison of these strings results in an error because they have different character sets:

```
mysql> SET @s1 = _latin1 'abc', @s2 = _latin2 'abc';
mysql> SELECT @s1 = @s2;
ERROR 1267 (HY000): Illegal mix of collations (latin1_swedish_ci,IMPLICIT)
and (latin2_general_ci,IMPLICIT) for operation '='
```

Converting one of the strings to a character set compatible with the other enables the comparison to occur without error:

```
mysql> SELECT @s1 = CONVERT(@s2 USING latin1);
+---------------------------------+
| @s1 = CONVERT(@s2 USING latin1) |
+---------------------------------+
|                               1 |
+---------------------------------+
```

For string literals, another way to specify the character set is to use a character set introducer. `_latin1` and `_latin2` in the preceding example are instances of introducers. Unlike conversion functions such as `CAST()`, or `CONVERT()`, which convert a string from one character set to another, an introducer designates a string literal as having a particular character set, with no conversion involved. For more information, see Section 10.3.8, “Character Set Introducers”.

Character set conversion is also useful preceding lettercase conversion of binary strings. `LOWER()` and `UPPER()` are ineffective when applied directly to binary strings because the concept of lettercase does not apply. To perform lettercase conversion of a binary string, first convert it to a nonbinary string using a character set appropriate for the data stored in the string:

```
mysql> SET @str = BINARY 'New York';
mysql> SELECT LOWER(@str), LOWER(CONVERT(@str USING latin1));
+-------------+-----------------------------------+
| LOWER(@str) | LOWER(CONVERT(@str USING latin1)) |
+-------------+-----------------------------------+
| New York    | new york                          |
+-------------+-----------------------------------+
```

Be aware that if you convert an indexed column using `BINARY`, `CAST()`, or `CONVERT()`, MySQL may not be able to use the index efficiently.

The cast functions are useful for creating a column with a specific type in a `CREATE TABLE ... SELECT` statement:

```
mysql> CREATE TABLE new_table SELECT CAST('2000-01-01' AS DATE) AS c1;
mysql> SHOW CREATE TABLE new_table\G
*************************** 1. row ***************************
       Table: new_table
Create Table: CREATE TABLE `new_table` (
  `c1` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

The cast functions are useful for sorting `ENUM` columns in lexical order. Normally, sorting of `ENUM` columns occurs using the internal numeric values. Casting the values to `CHAR` results in a lexical sort:

```
SELECT enum_col FROM tbl_name ORDER BY CAST(enum_col AS CHAR);
```

`CAST()` also changes the result if you use it as part of a more complex expression such as `CONCAT('Date: ',CAST(NOW() AS DATE))`.

For temporal values, there is little need to use `CAST()` to extract data in different formats. Instead, use a function such as `EXTRACT()`, `DATE_FORMAT()`, or `TIME_FORMAT()`. See Section 12.7, “Date and Time Functions”.

To cast a string to a number, you normally need do nothing other than use the string value in numeric context:

```
mysql> SELECT 1+'1';
       -> 2
```

That is also true for hexadecimal and bit literals, which are binary strings by default:

```
mysql> SELECT X'41', X'41'+0;
        -> 'A', 65
mysql> SELECT b'1100001', b'1100001'+0;
```

```
        -> 'a', 97
```

A string used in an arithmetic operation is converted to a floating-point number during expression evaluation.

A number used in string context is converted to a string:

```
mysql> SELECT CONCAT('hello you ',2);
        -> 'hello you 2'
```

For information about implicit conversion of numbers to strings, see Section 12.3, “Type Conversion in Expression Evaluation”.

MySQL supports arithmetic with both signed and unsigned 64-bit values. For numeric operators (such as + or -) where one of the operands is an unsigned integer, the result is unsigned by default (see Section 12.6.1, “Arithmetic Operators”). To override this, use the SIGNED or UNSIGNED cast operator to cast a value to a signed or unsigned 64-bit integer, respectively.

```
mysql> SELECT 1 - 2;
        -> -1
mysql> SELECT CAST(1 - 2 AS UNSIGNED);
        -> 18446744073709551615
mysql> SELECT CAST(CAST(1 - 2 AS UNSIGNED) AS SIGNED);
        -> -1
```

If either operand is a floating-point value, the result is a floating-point value and is not affected by the preceding rule. (In this context, DECIMAL column values are regarded as floating-point values.)

```
mysql> SELECT CAST(1 AS UNSIGNED) - 2.0;
        -> -1.0
```

The SQL mode affects the result of conversion operations (see Section 5.1.10, “Server SQL Modes”). Examples:

- For conversion of a “zero” date string to a date, CONVERT() and CAST() return NULL and produce a warning when the NO_ZERO_DATE SQL mode is enabled.
- For integer subtraction, if the NO_UNSIGNED_SUBTRACTION SQL mode is enabled, the subtraction result is signed even if any operand is unsigned.

The following list describes the available cast functions and operators:

- BINARY *expr*

  The BINARY operator converts the expression to a binary string (a string that has the binary character set and binary collation). A common use for BINARY is to force a character string comparison to be done byte by byte using numeric byte values rather than character by character. The BINARY operator also causes trailing spaces in comparisons to be significant. For information about the differences between the binary collation of the binary character set and the _bin collations of nonbinary character sets, see Section 10.8.5, “The binary Collation Compared to _bin Collations”.

  ```
  mysql> SELECT 'a' = 'A';
          -> 1
  mysql> SELECT BINARY 'a' = 'A';
          -> 0
  mysql> SELECT 'a' = 'a ';
          -> 1
  mysql> SELECT BINARY 'a' = 'a ';
          -> 0
  ```

  In a comparison, BINARY affects the entire operation; it can be given before either operand with the same result.

  To convert a string expression to a binary string, these constructs are equivalent:

```
BINARY expr
CAST(expr AS BINARY)
CONVERT(expr USING BINARY)
```

If a value is a string literal, it can be designated as a binary string without performing any conversion by using the `_binary` character set introducer:

```
mysql> SELECT 'a' = 'A';
        -> 1
mysql> SELECT _binary 'a' = 'A';
        -> 0
```

For information about introducers, see Section 10.3.8, “Character Set Introducers”.

The `BINARY` operator in expressions differs in effect from the `BINARY` attribute in character column definitions. A character column defined with the `BINARY` attribute is assigned the table default character set and the binary (`_bin`) collation of that character set. Every nonbinary character set has a `_bin` collation. For example, if the table default character set is `utf8`, these two column definitions are equivalent:

```
CHAR(10) BINARY
CHAR(10) CHARACTER SET utf8 COLLATE utf8_bin
```

The use of `CHARACTER SET binary` in the definition of a `CHAR`, `VARCHAR`, or `TEXT` column causes the column to be treated as the corresponding binary string data type. For example, the following pairs of definitions are equivalent:

```
CHAR(10) CHARACTER SET binary
BINARY(10)

VARCHAR(10) CHARACTER SET binary
VARBINARY(10)

TEXT CHARACTER SET binary
BLOB
```

- `CAST(expr AS type)`

  The `CAST()` function takes an expression of any type and produces a result value of the specified type, similar to `CONVERT()`. For more information, see the description of `CONVERT()`.

- `CONVERT(expr USING transcoding_name)`, `CONVERT(expr,type)`

  The `CONVERT()` function takes an expression of any type and produces a result value of the specified type.

  `CONVERT(... USING ...)` is standard SQL syntax. The non-`USING` form of `CONVERT()` is ODBC syntax.

  `CONVERT(expr USING transcoding_name)` converts data between different character sets. In MySQL, transcoding names are the same as the corresponding character set names. For example, this statement converts the string `'abc'` in the default character set to the corresponding string in the `utf8` character set:

  ```
  SELECT CONVERT('abc' USING utf8);
  ```

  `CONVERT(expr, type)` syntax (without `USING`) takes an expression and a `type` value specifying the result type. This operation may also be expressed as `CAST(expr AS type)`, which is equivalent. These `type` values are permitted:

  - `BINARY[(N)]`

    Produces a string with the `BINARY` data type. For a description of how this affects comparisons, see Section 11.3.3, “The BINARY and VARBINARY Types”. If the optional length `N` is given,

`BINARY(N)` causes the cast to use no more than `N` bytes of the argument. Values shorter than `N` bytes are padded with `0x00` bytes to a length of `N`.

- `CHAR[(N)] [charset_info]`

  Produces a string with the `CHAR` data type. If the optional length `N` is given, `CHAR(N)` causes the cast to use no more than `N` characters of the argument. No padding occurs for values shorter than `N` characters.

  With no `charset_info` clause, `CHAR` produces a string with the default character set. To specify the character set explicitly, these `charset_info` values are permitted:

  - `CHARACTER SET charset_name`: Produces a string with the given character set.
  - `ASCII`: Shorthand for `CHARACTER SET latin1`.
  - `UNICODE`: Shorthand for `CHARACTER SET ucs2`.

  In all cases, the string has the character set default collation.

- `DATE`

  Produces a `DATE` value.

- `DATETIME`

  Produces a `DATETIME` value.

- `DECIMAL[(M[,D])]`

  Produces a `DECIMAL` value. If the optional `M` and `D` values are given, they specify the maximum number of digits (the precision) and the number of digits following the decimal point (the scale).

- `NCHAR[(N)]`

  Like `CHAR`, but produces a string with the national character set. See Section 10.3.7, “The National Character Set”.

  Unlike `CHAR`, `NCHAR` does not permit trailing character set information to be specified.

- `SIGNED [INTEGER]`

  Produces a signed integer value.

- `TIME`

  Produces a `TIME` value.

- `UNSIGNED [INTEGER]`

  Produces an unsigned integer value.

12.12 XML Functions

Table 12.16 XML Functions

Name	Description
`ExtractValue()`	Extract a value from an XML string using XPath notation
`UpdateXML()`	Return replaced XML fragment

This section discusses XML and related functionality in MySQL.

Note

It is possible to obtain XML-formatted output from MySQL in the `mysql` and `mysqldump` clients by invoking them with the `--xml` option. See Section 4.5.1, "`mysql` — The MySQL Command-Line Client", and Section 4.5.4, "`mysqldump` — A Database Backup Program".

Two functions providing basic XPath 1.0 (XML Path Language, version 1.0) capabilities are available. Some basic information about XPath syntax and usage is provided later in this section; however, an in-depth discussion of these topics is beyond the scope of this manual, and you should refer to the XML Path Language (XPath) 1.0 standard for definitive information. A useful resource for those new to XPath or who desire a refresher in the basics is the Zvon.org XPath Tutorial, which is available in several languages.

Note

These functions remain under development. We continue to improve these and other aspects of XML and XPath functionality in MySQL 5.6 and onwards. You may discuss these, ask questions about them, and obtain help from other users with them in the MySQL XML User Forum.

XPath expressions used with these functions support user variables and local stored program variables. User variables are weakly checked; variables local to stored programs are strongly checked (see also Bug #26518):

- **User variables (weak checking).** Variables using the syntax `$@variable_name` (that is, user variables) are not checked. No warnings or errors are issued by the server if a variable has the wrong type or has previously not been assigned a value. This also means the user is fully responsible for any typographical errors, since no warnings are given if (for example) `$@myvairable` is used where `$@myvariable` was intended.

  Example:

```
mysql> SET @xml = '<a><b>X</b><b>Y</b></a>';
Query OK, 0 rows affected (0.00 sec)

mysql> SET @i =1, @j = 2;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @i, ExtractValue(@xml, '//b[$@i]');
+------+-------------------------------+
| @i   | ExtractValue(@xml, '//b[$@i]') |
+------+-------------------------------+
|    1 | X                             |
+------+-------------------------------+
1 row in set (0.00 sec)

mysql> SELECT @j, ExtractValue(@xml, '//b[$@j]');
+------+-------------------------------+
| @j   | ExtractValue(@xml, '//b[$@j]') |
+------+-------------------------------+
|    2 | Y                             |
+------+-------------------------------+
1 row in set (0.00 sec)

mysql> SELECT @k, ExtractValue(@xml, '//b[$@k]');
+------+-------------------------------+
| @k   | ExtractValue(@xml, '//b[$@k]') |
+------+-------------------------------+
| NULL |                               |
+------+-------------------------------+
1 row in set (0.00 sec)
```

- **Variables in stored programs (strong checking).** Variables using the syntax `$variable_name` can be declared and used with these functions when they are called inside stored programs. Such

variables are local to the stored program in which they are defined, and are strongly checked for type and value.

Example:

```
mysql> DELIMITER |

mysql> CREATE PROCEDURE myproc ()
    -> BEGIN
    ->   DECLARE i INT DEFAULT 1;
    ->   DECLARE xml VARCHAR(25) DEFAULT '<a>X</a><a>Y</a><a>Z</a>';
    ->
    ->   WHILE i < 4 DO
    ->     SELECT xml, i, ExtractValue(xml, '//a[$i]');
    ->     SET i = i+1;
    ->   END WHILE;
    -> END |
Query OK, 0 rows affected (0.01 sec)

mysql> DELIMITER ;

mysql> CALL myproc();
+-------------------------+---+-----------------------------+
| xml                     | i | ExtractValue(xml, '//a[$i]') |
+-------------------------+---+-----------------------------+
| <a>X</a><a>Y</a><a>Z</a> | 1 | X                           |
+-------------------------+---+-----------------------------+
1 row in set (0.00 sec)

+-------------------------+---+-----------------------------+
| xml                     | i | ExtractValue(xml, '//a[$i]') |
+-------------------------+---+-----------------------------+
| <a>X</a><a>Y</a><a>Z</a> | 2 | Y                           |
+-------------------------+---+-----------------------------+
1 row in set (0.01 sec)

+-------------------------+---+-----------------------------+
| xml                     | i | ExtractValue(xml, '//a[$i]') |
+-------------------------+---+-----------------------------+
| <a>X</a><a>Y</a><a>Z</a> | 3 | Z                           |
+-------------------------+---+-----------------------------+
1 row in set (0.01 sec)
```

Parameters. Variables used in XPath expressions inside stored routines that are passed in as parameters are also subject to strong checking.

Expressions containing user variables or variables local to stored programs must otherwise (except for notation) conform to the rules for XPath expressions containing variables as given in the XPath 1.0 specification.

Note

A user variable used to store an XPath expression is treated as an empty string. Because of this, it is not possible to store an XPath expression as a user variable. (Bug #32911)

- `ExtractValue(xml_frag, xpath_expr)`

  `ExtractValue()` takes two string arguments, a fragment of XML markup *`xml_frag`* and an XPath expression *`xpath_expr`* (also known as a *locator*); it returns the text (`CDATA`) of the first text node which is a child of the element or elements matched by the XPath expression.

  Using this function is the equivalent of performing a match using the *`xpath_expr`* after appending `/text()`. In other words, `ExtractValue('<a><b>Sakila</b></a>', '/a/b')` and `ExtractValue('<a><b>Sakila</b></a>', '/a/b/text()')` produce the same result.

If multiple matches are found, the content of the first child text node of each matching element is returned (in the order matched) as a single, space-delimited string.

If no matching text node is found for the expression (including the implicit `/text()`)—for whatever reason, as long as *`xpath_expr`* is valid, and *`xml_frag`* consists of elements which are properly nested and closed—an empty string is returned. No distinction is made between a match on an empty element and no match at all. This is by design.

If you need to determine whether no matching element was found in *`xml_frag`* or such an element was found but contained no child text nodes, you should test the result of an expression that uses the XPath `count()` function. For example, both of these statements return an empty string, as shown here:

```
mysql> SELECT ExtractValue('<a><b/></a>', '/a/b');
+-----------------------------------+
| ExtractValue('<a><b/></a>', '/a/b') |
+-----------------------------------+
|                                   |
+-----------------------------------+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a><c/></a>', '/a/b');
+-----------------------------------+
| ExtractValue('<a><c/></a>', '/a/b') |
+-----------------------------------+
|                                   |
+-----------------------------------+
1 row in set (0.00 sec)
```

Note

In MySQL 5.6.28 and MySQL 5.6.29, when `ExtractValue()` failed to find a match for the supplied expression, it returned `NULL`. This issue was resolved in MySQL 5.6.30. (Bug #22552615)

However, you can determine whether there was actually a matching element using the following:

```
mysql> SELECT ExtractValue('<a><b/></a>', 'count(/a/b)');
+-----------------------------------+
| ExtractValue('<a><b/></a>', 'count(/a/b)') |
+-----------------------------------+
| 1                                 |
+-----------------------------------+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a><c/></a>', 'count(/a/b)');
+-----------------------------------+
| ExtractValue('<a><c/></a>', 'count(/a/b)') |
+-----------------------------------+
| 0                                 |
+-----------------------------------+
1 row in set (0.01 sec)
```

Important

`ExtractValue()` returns only `CDATA`, and does not return any tags that might be contained within a matching tag, nor any of their content (see the result returned as `val1` in the following example).

```
mysql> SELECT
    ->   ExtractValue('<a>ccc<b>ddd</b></a>', '/a') AS val1,
    ->   ExtractValue('<a>ccc<b>ddd</b></a>', '/a/b') AS val2,
    ->   ExtractValue('<a>ccc<b>ddd</b></a>', '//b') AS val3,
    ->   ExtractValue('<a>ccc<b>ddd</b></a>', '/b') AS val4,
    ->   ExtractValue('<a>ccc<b>ddd</b><b>eee</b></a>', '//b') AS val5;
```

```
+------+------+------+------+---------+
| val1 | val2 | val3 | val4 | val5    |
+------+------+------+------+---------+
| ccc  | ddd  | ddd  |      | ddd eee |
+------+------+------+------+---------+
```

This function uses the current SQL collation for making comparisons with `contains()`, performing the same collation aggregation as other string functions (such as `CONCAT()`), in taking into account the collation coercibility of their arguments; see Section 10.8.4, “Collation Coercibility in Expressions”, for an explanation of the rules governing this behavior.

(Previously, binary—that is, case-sensitive—comparison was always used.)

`NULL` is returned if *`xml_frag`* contains elements which are not properly nested or closed, and a warning is generated, as shown in this example:

```
mysql> SELECT ExtractValue('<a>c</a><b', '//a');
+-----------------------------------+
| ExtractValue('<a>c</a><b', '//a') |
+-----------------------------------+
| NULL                              |
+-----------------------------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1525
Message: Incorrect XML value: 'parse error at line 1 pos 11:
         END-OF-INPUT unexpected ('>' wanted)'
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a>c</a><b/>', '//a');
+-------------------------------------+
| ExtractValue('<a>c</a><b/>', '//a') |
+-------------------------------------+
| c                                   |
+-------------------------------------+
1 row in set (0.00 sec)
```

- `UpdateXML(`*`xml_target`*`, `*`xpath_expr`*`, `*`new_xml`*`)`

  This function replaces a single portion of a given fragment of XML markup *`xml_target`* with a new XML fragment *`new_xml`*, and then returns the changed XML. The portion of *`xml_target`* that is replaced matches an XPath expression *`xpath_expr`* supplied by the user.

  If no expression matching *`xpath_expr`* is found, or if multiple matches are found, the function returns the original *`xml_target`* XML fragment. All three arguments should be strings.

```
mysql> SELECT
    ->   UpdateXML('<a><b>ccc</b><d></d></a>', '/a', '<e>fff</e>') AS val1,
    ->   UpdateXML('<a><b>ccc</b><d></d></a>', '/b', '<e>fff</e>') AS val2,
    ->   UpdateXML('<a><b>ccc</b><d></d></a>', '//b', '<e>fff</e>') AS val3,
    ->   UpdateXML('<a><b>ccc</b><d></d></a>', '/a/d', '<e>fff</e>') AS val4,
    ->   UpdateXML('<a><d></d><b>ccc</b><d></d></a>', '/a/d', '<e>fff</e>') AS val5
    -> \G

*************************** 1. row ***************************
val1: <e>fff</e>
val2: <a><b>ccc</b><d></d></a>
val3: <a><e>fff</e><d></d></a>
val4: <a><b>ccc</b><e>fff</e></a>
val5: <a><d></d><b>ccc</b><d></d></a>
```

Note

A discussion in depth of XPath syntax and usage are beyond the scope of this manual. Please see the XML Path Language (XPath) 1.0 specification

for definitive information. A useful resource for those new to XPath or who are wishing a refresher in the basics is the Zvon.org XPath Tutorial, which is available in several languages.

Descriptions and examples of some basic XPath expressions follow:

- */tag*

  Matches *<tag/>* if and only if *<tag/>* is the root element.

  Example: */a* has a match in *<a>* because it matches the outermost (root) tag. It does not match the inner *a* element in *<a/>* because in this instance it is the child of another element.

- */tag1/tag2*

  Matches *<tag2/>* if and only if it is a child of *<tag1/>*, and *<tag1/>* is the root element.

  Example: */a/b* matches the *b* element in the XML fragment *<a>* because it is a child of the root element *a*. It does not have a match in *<a/>* because in this case, *b* is the root element (and hence the child of no other element). Nor does the XPath expression have a match in *<a><c></c>*; here, *b* is a descendant of *a*, but not actually a child of *a*.

  This construct is extendable to three or more elements. For example, the XPath expression */a/b/c* matches the *c* element in the fragment *<a><c/>*.

- *//tag*

  Matches any instance of *<tag>*.

  Example: *//a* matches the *a* element in any of the following: *<a><c/>*; *<c><a>*; *<c><a/></c>*.

  // can be combined with */*. For example, *//a/b* matches the *b* element in either of the fragments *<a>* or *<c><a></c>*.

Note

//tag is the equivalent of */descendant-or-self::*/tag*. A common error is to confuse this with */descendant-or-self::tag*, although the latter expression can actually lead to very different results, as can be seen here:

```
mysql> SET @xml = '<a><b><c>w</c><b>x</b><d>y</d>z</b></a>';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @xml;
+---------------------------------------+
| @xml                                  |
+---------------------------------------+
| <a><b><c>w</c><b>x</b><d>y</d>z</b></a> |
+---------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue(@xml, '//b[1]');
+-------------------------------+
| ExtractValue(@xml, '//b[1]') |
+-------------------------------+
| x z                           |
+-------------------------------+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue(@xml, '//b[2]');
+-------------------------------+
| ExtractValue(@xml, '//b[2]') |
+-------------------------------+
```

```
|                              |
+------------------------------+
1 row in set (0.01 sec)

mysql> SELECT ExtractValue(@xml, '/descendant-or-self::*/b[1]');
+-------------------------------------------------+
| ExtractValue(@xml, '/descendant-or-self::*/b[1]') |
+-------------------------------------------------+
| x z                                             |
+-------------------------------------------------+
1 row in set (0.06 sec)

mysql> SELECT ExtractValue(@xml, '/descendant-or-self::*/b[2]');
+-------------------------------------------------+
| ExtractValue(@xml, '/descendant-or-self::*/b[2]') |
+-------------------------------------------------+
|                                                 |
+-------------------------------------------------+
1 row in set (0.00 sec)


mysql> SELECT ExtractValue(@xml, '/descendant-or-self::b[1]');
+---------------------------------------------+
| ExtractValue(@xml, '/descendant-or-self::b[1]') |
+---------------------------------------------+
| z                                           |
+---------------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT ExtractValue(@xml, '/descendant-or-self::b[2]');
+---------------------------------------------+
| ExtractValue(@xml, '/descendant-or-self::b[2]') |
+---------------------------------------------+
| x                                           |
+---------------------------------------------+
1 row in set (0.00 sec)
```

- The `*` operator acts as a “wildcard” that matches any element. For example, the expression `/*/b` matches the *b* element in either of the XML fragments `<a><b/></a>` or `<c><b/></c>`. However, the expression does not produce a match in the fragment `<b><a/></b>` because *b* must be a child of some other element. The wildcard may be used in any position: The expression `/*/b/*` matches any child of a *b* element that is itself not the root element.

- You can match any of several locators using the `|` (`UNION`) operator. For example, the expression `//b|//c` matches all *b* and *c* elements in the XML target.

- It is also possible to match an element based on the value of one or more of its attributes. This done using the syntax *`tag`*`[@`*`attribute`*`="`*`value`*`"]`. For example, the expression `//b[@id="idB"]` matches the second *b* element in the fragment `<a><b id="idA"/><c/><b id="idB"/></a>`. To match against *any* element having *`attribute`*`="`*`value`*`"`, use the XPath expression `//*[`*`attribute`*`="`*`value`*`"]`.

  To filter multiple attribute values, simply use multiple attribute-comparison clauses in succession. For example, the expression `//b[@c="x"][@d="y"]` matches the element `<b c="x" d="y"/>` occurring anywhere in a given XML fragment.

  To find elements for which the same attribute matches any of several values, you can use multiple locators joined by the `|` operator. For example, to match all *b* elements whose *c* attributes have either of the values 23 or 17, use the expression `//b[@c="23"]|//b[@c="17"]`. You can also use the logical `or` operator for this purpose: `//b[@c="23" or @c="17"]`.

Note

The difference between `or` and `|` is that `or` joins conditions, while `|` joins result sets.

XPath Limitations. The XPath syntax supported by these functions is currently subject to the following limitations:

- Nodeset-to-nodeset comparison (such as `'/a/b[@c=@d]'`) is not supported.
- All of the standard XPath comparison operators are supported. (Bug #22823)
- Relative locator expressions are resolved in the context of the root node. For example, consider the following query and result:

```
mysql> SELECT ExtractValue(
    ->   '<a><b c="1">X</b><b c="2">Y</b></a>',
    ->    'a/b'
    -> ) AS result;
+--------+
| result |
+--------+
| X Y    |
+--------+
1 row in set (0.03 sec)
```

  In this case, the locator `a/b` resolves to `/a/b`.

  Relative locators are also supported within predicates. In the following example, `d[../@c="1"]` is resolved as `/a/b[@c="1"]/d`:

```
mysql> SELECT ExtractValue(
    ->      '<a>
    ->        <b c="1"><d>X</d></b>
    ->        <b c="2"><d>X</d></b>
    ->      </a>',
    ->      'a/b/d[../@c="1"]')
    -> AS result;
+--------+
| result |
+--------+
| X      |
+--------+
1 row in set (0.00 sec)
```

- Locators prefixed with expressions that evaluate as scalar values—including variable references, literals, numbers, and scalar function calls—are not permitted, and their use results in an error.
- The `::` operator is not supported in combination with node types such as the following:
  - *`axis`*`::comment()`
  - *`axis`*`::text()`
  - *`axis`*`::processing-instructions()`
  - *`axis`*`::node()`

  However, name tests (such as *`axis`*`::`*`name`* and *`axis`*`::*`) are supported, as shown in these examples:

```
mysql> SELECT ExtractValue('<a><b>x</b><c>y</c></a>','/a/child::b');
+-----------------------------------------------------+
| ExtractValue('<a><b>x</b><c>y</c></a>','/a/child::b') |
+-----------------------------------------------------+
| x                                                   |
+-----------------------------------------------------+
1 row in set (0.02 sec)

mysql> SELECT ExtractValue('<a><b>x</b><c>y</c></a>','/a/child::*');
+-----------------------------------------------------+
| ExtractValue('<a><b>x</b><c>y</c></a>','/a/child::*') |
+-----------------------------------------------------+
```

```
| x y                                                   |
+-------------------------------------------------------+
1 row in set (0.01 sec)
```

- “Up-and-down” navigation is not supported in cases where the path would lead “above” the root element. That is, you cannot use expressions which match on descendants of ancestors of a given element, where one or more of the ancestors of the current element is also an ancestor of the root element (see Bug #16321).
- The following XPath functions are not supported, or have known issues as indicated:
  - `id()`
  - `lang()`
  - `local-name()`
  - `name()`
  - `namespace-uri()`
  - `normalize-space()`
  - `starts-with()`
  - `string()`
  - `substring-after()`
  - `substring-before()`
  - `translate()`
- The following axes are not supported:
  - `following-sibling`
  - `following`
  - `preceding-sibling`
  - `preceding`

XPath expressions passed as arguments to `ExtractValue()` and `UpdateXML()` may contain the colon character (`:`) in element selectors, which enables their use with markup employing XML namespaces notation. For example:

```
mysql> SET @xml = '<a>111<b:c>222<d>333</d><e:f>444</e:f></b:c></a>';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT ExtractValue(@xml, '//e:f');
+-----------------------------+
| ExtractValue(@xml, '//e:f') |
+-----------------------------+
| 444                         |
+-----------------------------+
1 row in set (0.00 sec)

mysql> SELECT UpdateXML(@xml, '//b:c', '<g:h>555</g:h>');
+-------------------------------------------+
| UpdateXML(@xml, '//b:c', '<g:h>555</g:h>') |
+-------------------------------------------+
| <a>111<g:h>555</g:h></a>                  |
+-------------------------------------------+
1 row in set (0.00 sec)
```

This is similar in some respects to what is permitted by Apache Xalan and some other parsers, and is much simpler than requiring namespace declarations or the use of the `namespace-uri()` and `local-name()` functions.

Error handling. For both `ExtractValue()` and `UpdateXML()`, the XPath locator used must be valid and the XML to be searched must consist of elements which are properly nested and closed. If the locator is invalid, an error is generated:

```
mysql> SELECT ExtractValue('<a>c</a><b/>', '/&a');
ERROR 1105 (HY000): XPATH syntax error: '&a'
```

If *`xml_frag`* does not consist of elements which are properly nested and closed, `NULL` is returned and a warning is generated, as shown in this example:

```
mysql> SELECT ExtractValue('<a>c</a><b', '//a');
+----------------------------------+
| ExtractValue('<a>c</a><b', '//a') |
+----------------------------------+
| NULL                             |
+----------------------------------+
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1525
Message: Incorrect XML value: 'parse error at line 1 pos 11:
         END-OF-INPUT unexpected ('>' wanted)'
1 row in set (0.00 sec)

mysql> SELECT ExtractValue('<a>c</a><b/>', '//a');
+-----------------------------------+
| ExtractValue('<a>c</a><b/>', '//a') |
+-----------------------------------+
| c                                 |
+-----------------------------------+
1 row in set (0.00 sec)
```

Important

The replacement XML used as the third argument to `UpdateXML()` is *not* checked to determine whether it consists solely of elements which are properly nested and closed.

XPath Injection. *code injection* occurs when malicious code is introduced into the system to gain unauthorized access to privileges and data. It is based on exploiting assumptions made by developers about the type and content of data input from users. XPath is no exception in this regard.

A common scenario in which this can happen is the case of application which handles authorization by matching the combination of a login name and password with those found in an XML file, using an XPath expression like this one:

```
//user[login/text()='neapolitan' and password/text()='1c3cr34m']/attribute::id
```

This is the XPath equivalent of an SQL statement like this one:

```
SELECT id FROM users WHERE login='neapolitan' AND password='1c3cr34m';
```

A PHP application employing XPath might handle the login process like this:

```
<?php

  $file     =   "users.xml";

  $login    =   $POST["login"];
  $password =   $POST["password"];

  $xpath = "//user[login/text()=$login and password/text()=$password]/attribute::id";
```

```
  if( file_exists($file) )
  {
    $xml = simplexml_load_file($file);

    if($result = $xml->xpath($xpath))
      echo "You are now logged in as user $result[0].";
    else
      echo "Invalid login name or password.";
  }
  else
    exit("Failed to open $file.");

?>
```

No checks are performed on the input. This means that a malevolent user can “short-circuit” the test by entering `' or 1=1` for both the login name and password, resulting in `$xpath` being evaluated as shown here:

```
//user[login/text()='' or 1=1 and password/text()='' or 1=1]/attribute::id
```

Since the expression inside the square brackets always evaluates as `true`, it is effectively the same as this one, which matches the `id` attribute of every `user` element in the XML document:

```
//user/attribute::id
```

One way in which this particular attack can be circumvented is simply by quoting the variable names to be interpolated in the definition of `$xpath`, forcing the values passed from a Web form to be converted to strings:

```
$xpath = "//user[login/text()='$login' and password/text()='$password']/attribute::id";
```

This is the same strategy that is often recommended for preventing SQL injection attacks. In general, the practices you should follow for preventing XPath injection attacks are the same as for preventing SQL injection:

- Never accepted untested data from users in your application.
- Check all user-submitted data for type; reject or convert data that is of the wrong type
- Test numeric data for out of range values; truncate, round, or reject values that are out of range. Test strings for illegal characters and either strip them out or reject input containing them.
- Do not output explicit error messages that might provide an unauthorized user with clues that could be used to compromise the system; log these to a file or database table instead.

Just as SQL injection attacks can be used to obtain information about database schemas, so can XPath injection be used to traverse XML files to uncover their structure, as discussed in Amit Klein's paper Blind XPath Injection (PDF file, 46KB).

It is also important to check the output being sent back to the client. Consider what can happen when we use the MySQL `ExtractValue()` function:

```
mysql> SELECT ExtractValue(
    ->     LOAD_FILE('users.xml'),
    ->     '//user[login/text()="" or 1=1 and password/text()="" or 1=1]/attribute::id'
    -> ) AS id;
+-------------------------------+
| id                            |
+-------------------------------+
| 00327 13579 02403 42354 28570 |
+-------------------------------+
1 row in set (0.01 sec)
```

Because `ExtractValue()` returns multiple matches as a single space-delimited string, this injection attack provides every valid ID contained within `users.xml` to the user as a single row of output. As an extra safeguard, you should also test output before returning it to the user. Here is a simple example:

```
mysql> SELECT @id = ExtractValue(
    ->     LOAD_FILE('users.xml'),
    ->     '//user[login/text()="" or 1=1 and password/text()="" or 1=1]/attribute::id'
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT IF(
    ->     INSTR(@id, ' ') = 0,
    ->     @id,
    ->     'Unable to retrieve user ID')
    -> AS singleID;
+---------------------------+
| singleID                  |
+---------------------------+
| Unable to retrieve user ID |
+---------------------------+
1 row in set (0.00 sec)
```

In general, the guidelines for returning data to users securely are the same as for accepting user input. These can be summed up as:

- Always test outgoing data for type and permissible values.
- Never permit unauthorized users to view error messages that might provide information about the application that could be used to exploit it.

12.13 Bit Functions and Operators

Table 12.17 Bit Functions and Operators

Name	Description
`&`	Bitwise AND
`>>`	Right shift
`<<`	Left shift
`^`	Bitwise XOR
`BIT_COUNT()`	Return the number of bits that are set
`|`	Bitwise OR
`~`	Bitwise inversion

Bit functions and operators comprise `BIT_COUNT()`, `BIT_AND()`, `BIT_OR()`, `BIT_XOR()`, `&`, `|`, `^`, `~`, `<<`, and `>>`. (`BIT_AND()`, `BIT_OR()`, and `BIT_XOR()` are aggregate functions described at Section 12.19.1, “Aggregate Function Descriptions”.) Bit functions and operators require `BIGINT` (64-bit integer) arguments and return `BIGINT` values, so they have a maximum range of 64 bits. Arguments of other types (such as the `BINARY` and `VARBINARY` binary string types) are converted to `BIGINT` and truncation might occur.

The following list describes available bit functions and operators:

- `|`

  Bitwise OR.

  The result is an unsigned 64-bit integer.

  ```
  mysql> SELECT 29 | 15;
          -> 31
  ```

- `&`

  Bitwise AND.

The result is an unsigned 64-bit integer.

```
mysql> SELECT 29 & 15;
        -> 13
```

- `^`

  Bitwise XOR.

  The result is an unsigned 64-bit integer.

```
mysql> SELECT 1 ^ 1;
        -> 0
mysql> SELECT 1 ^ 0;
        -> 1
mysql> SELECT 11 ^ 3;
        -> 8
```

- `<<`

  Shifts a longlong (`BIGINT`) number to the left.

  The result is an unsigned 64-bit integer. The value is truncated to 64 bits. In particular, if the shift count is greater or equal to the width of an unsigned 64-bit number, the result is zero.

```
mysql> SELECT 1 << 2;
        -> 4
```

- `>>`

  Shifts a longlong (`BIGINT`) number to the right.

  The result is an unsigned 64-bit integer. The value is truncated to 64 bits. In particular, if the shift count is greater or equal to the width of an unsigned 64-bit number, the result is zero.

```
mysql> SELECT 4 >> 2;
        -> 1
```

- `~`

  Invert all bits.

  The result is an unsigned 64-bit integer.

```
mysql> SELECT 5 & ~1;
        -> 4
```

- `BIT_COUNT(N)`

  Returns the number of bits that are set in the argument *`N`* as an unsigned 64-bit integer, or `NULL` if the argument is `NULL`.

```
mysql> SELECT BIT_COUNT(29), BIT_COUNT(b'101010');
        -> 4, 3
```

12.14 Encryption and Compression Functions

Table 12.18 Encryption Functions

Name	Description
`AES_DECRYPT()`	Decrypt using AES
`AES_ENCRYPT()`	Encrypt using AES
`COMPRESS()`	Return result as a binary string

Name	Description
DECODE()	Decode a string encrypted using ENCODE()
DES_DECRYPT()	Decrypt a string
DES_ENCRYPT()	Encrypt a string
ENCODE()	Encode a string
ENCRYPT()	Encrypt a string
MD5()	Calculate MD5 checksum
OLD_PASSWORD() (deprecated)	Return the value of the pre-4.1 implementation of PASSWORD
PASSWORD()	Calculate and return a password string
RANDOM_BYTES() (introduced 5.6.17)	Return a random byte vector
SHA1(), SHA()	Calculate an SHA-1 160-bit checksum
SHA2()	Calculate an SHA-2 checksum
UNCOMPRESS()	Uncompress a string compressed
UNCOMPRESSED_LENGTH()	Return the length of a string before compression
VALIDATE_PASSWORD_STRENGTH()	Determine strength of password

Many encryption and compression functions return strings for which the result might contain arbitrary byte values. If you want to store these results, use a column with a VARBINARY or BLOB binary string data type. This avoids potential problems with trailing space removal or character set conversion that would change data values, such as may occur if you use a nonbinary string data type (CHAR, VARCHAR, TEXT).

Some encryption functions return strings of ASCII characters: MD5(), OLD_PASSWORD(), PASSWORD(), SHA(), SHA1(), SHA2(). Their return value is a string that has a character set and collation determined by the character_set_connection and collation_connection system variables. This is a nonbinary string unless the character set is binary.

If an application stores values from a function such as MD5() or SHA1() that returns a string of hex digits, more efficient storage and comparisons can be obtained by converting the hex representation to binary using UNHEX() and storing the result in a BINARY(*N*) column. Each pair of hexadecimal digits requires one byte in binary form, so the value of *N* depends on the length of the hex string. *N* is 16 for an MD5() value and 20 for a SHA1() value. For SHA2(), *N* ranges from 28 to 32 depending on the argument specifying the desired bit length of the result.

The size penalty for storing the hex string in a CHAR column is at least two times, up to eight times if the value is stored in a column that uses the utf8 character set (where each character uses 4 bytes). Storing the string also results in slower comparisons because of the larger values and the need to take character set collation rules into account.

Suppose that an application stores MD5() string values in a CHAR(32) column:

```
CREATE TABLE md5_tbl (md5_val CHAR(32), ...);
INSERT INTO md5_tbl (md5_val, ...) VALUES(MD5('abcdef'), ...);
```

To convert hex strings to more compact form, modify the application to use UNHEX() and BINARY(16) instead as follows:

```
CREATE TABLE md5_tbl (md5_val BINARY(16), ...);
INSERT INTO md5_tbl (md5_val, ...) VALUES(UNHEX(MD5('abcdef')), ...);
```

Applications should be prepared to handle the very rare case that a hashing function produces the same value for two different input values. One way to make collisions detectable is to make the hash column a primary key.

Note

Exploits for the MD5 and SHA-1 algorithms have become known. You may wish to consider using another one-way encryption function described in this section instead, such as `SHA2()`.

Caution

Passwords or other sensitive values supplied as arguments to encryption functions are sent as cleartext to the MySQL server unless an SSL connection is used. Also, such values appear in any MySQL logs to which they are written. To avoid these types of exposure, applications can encrypt sensitive values on the client side before sending them to the server. The same considerations apply to encryption keys. To avoid exposing these, applications can use stored procedures to encrypt and decrypt values on the server side.

- `AES_DECRYPT(`*`crypt_str`*`,`*`key_str`*`[,`*`init_vector`*`])`

  This function decrypts data using the official AES (Advanced Encryption Standard) algorithm. For more information, see the description of `AES_ENCRYPT()`.

  The optional initialization vector argument, *`init_vector`*, is available as of MySQL 5.6.17. As of that version, statements that use `AES_DECRYPT()` are unsafe for statement-based replication and cannot be stored in the query cache.

- `AES_ENCRYPT(`*`str`*`,`*`key_str`*`[,`*`init_vector`*`])`

  `AES_ENCRYPT()` and `AES_DECRYPT()` implement encryption and decryption of data using the official AES (Advanced Encryption Standard) algorithm, previously known as “Rijndael.” The AES standard permits various key lengths. By default these functions implement AES with a 128-bit key length. As of MySQL 5.6.17, key lengths of 196 or 256 bits can be used, as described later. The key length is a trade off between performance and security.

  `AES_ENCRYPT()` encrypts the string *`str`* using the key string *`key_str`* and returns a binary string containing the encrypted output. `AES_DECRYPT()` decrypts the encrypted string *`crypt_str`* using the key string *`key_str`* and returns the original plaintext string. If either function argument is `NULL`, the function returns `NULL`.

  The *`str`* and *`crypt_str`* arguments can be any length, and padding is automatically added to *`str`* so it is a multiple of a block as required by block-based algorithms such as AES. This padding is automatically removed by the `AES_DECRYPT()` function. The length of *`crypt_str`* can be calculated using this formula:

```
16 * (trunc(string_length / 16) + 1)
```

  For a key length of 128 bits, the most secure way to pass a key to the *`key_str`* argument is to create a truly random 128-bit value and pass it as a binary value. For example:

```
INSERT INTO t
VALUES (1,AES_ENCRYPT('text',UNHEX('F3229A0B371ED2D9441B830D21A390C3')));
```

  A passphrase can be used to generate an AES key by hashing the passphrase. For example:

```
INSERT INTO t
VALUES (1,AES_ENCRYPT('text', UNHEX(SHA2('My secret passphrase',512))));
```

  Do not pass a password or passphrase directly to *`crypt_str`*, hash it first. Previous versions of this documentation suggested the former approach, but it is no longer recommended as the examples shown here are more secure.

If AES_DECRYPT() detects invalid data or incorrect padding, it returns NULL. However, it is possible for AES_DECRYPT() to return a non-NULL value (possibly garbage) if the input data or the key is invalid.

As of MySQL 5.6.17, AES_ENCRYPT() and AES_DECRYPT() permit control of the block encryption mode and take an optional *init_vector* initialization vector argument:

- The block_encryption_mode system variable controls the mode for block-based encryption algorithms. Its default value is aes-128-ecb, which signifies encryption using a key length of 128 bits and ECB mode. For a description of the permitted values of this variable, see Section 5.1.7, "Server System Variables".
- The optional *init_vector* argument provides an initialization vector for block encryption modes that require it.

For modes that require the optional *init_vector* argument, it must be 16 bytes or longer (bytes in excess of 16 are ignored). An error occurs if *init_vector* is missing.

For modes that do not require *init_vector*, it is ignored and a warning is generated if it is specified.

A random string of bytes to use for the initialization vector can be produced by calling RANDOM_BYTES(16). For encryption modes that require an initialization vector, the same vector must be used for encryption and decryption.

```
mysql> SET block_encryption_mode = 'aes-256-cbc';
mysql> SET @key_str = SHA2('My secret passphrase',512);
mysql> SET @init_vector = RANDOM_BYTES(16);
mysql> SET @crypt_str = AES_ENCRYPT('text',@key_str,@init_vector);
mysql> SELECT AES_DECRYPT(@crypt_str,@key_str,@init_vector);
+-------------------------------------------------+
| AES_DECRYPT(@crypt_str,@key_str,@init_vector) |
+-------------------------------------------------+
| text                                            |
+-------------------------------------------------+
```

The following table lists each permitted block encryption mode, the SSL libraries that support it, and whether the initialization vector argument is required.

Block Encryption Mode	SSL Libraries that Support Mode	Initialization Vector Required
ECB	OpenSSL, yaSSL	No
CBC	OpenSSL, yaSSL	Yes
CFB1	OpenSSL	Yes
CFB8	OpenSSL	Yes
CFB128	OpenSSL	Yes
OFB	OpenSSL	Yes

As of MySQL 5.6.17, statements that use AES_ENCRYPT() or AES_DECRYPT() are unsafe for statement-based replication and cannot be stored in the query cache.

- COMPRESS(*string_to_compress*)

  Compresses a string and returns the result as a binary string. This function requires MySQL to have been compiled with a compression library such as zlib. Otherwise, the return value is always NULL. The compressed string can be uncompressed with UNCOMPRESS().

```
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',1000)));
        -> 21
```

```
mysql> SELECT LENGTH(COMPRESS(''));
        -> 0
mysql> SELECT LENGTH(COMPRESS('a'));
        -> 13
mysql> SELECT LENGTH(COMPRESS(REPEAT('a',16)));
        -> 15
```

The compressed string contents are stored the following way:

- Empty strings are stored as empty strings.
- Nonempty strings are stored as a 4-byte length of the uncompressed string (low byte first), followed by the compressed string. If the string ends with space, an extra `.` character is added to avoid problems with endspace trimming should the result be stored in a `CHAR` or `VARCHAR` column. (However, use of nonbinary string data types such as `CHAR` or `VARCHAR` to store compressed strings is not recommended anyway because character set conversion may occur. Use a `VARBINARY` or `BLOB` binary string column instead.)

- `DECODE(crypt_str,pass_str)`

  Decrypts the encrypted string *`crypt_str`* using *`pass_str`* as the password. *`crypt_str`* should be a string returned from `ENCODE()`.

- `DES_DECRYPT(crypt_str[,key_str])`

  Decrypts a string encrypted with `DES_ENCRYPT()`. If an error occurs, this function returns `NULL`.

  This function works only if MySQL has been configured with SSL support. See Section 6.3, “Using Encrypted Connections”.

  If no *`key_str`* argument is given, `DES_DECRYPT()` examines the first byte of the encrypted string to determine the DES key number that was used to encrypt the original string, and then reads the key from the DES key file to decrypt the message. For this to work, the user must have the `SUPER` privilege. The key file can be specified with the `--des-key-file` server option.

  If you pass this function a *`key_str`* argument, that string is used as the key for decrypting the message.

  If the *`crypt_str`* argument does not appear to be an encrypted string, MySQL returns the given *`crypt_str`*.

- `DES_ENCRYPT(str[,{key_num|key_str}])`

  Encrypts the string with the given key using the Triple-DES algorithm.

  This function works only if MySQL has been configured with SSL support. See Section 6.3, “Using Encrypted Connections”.

  The encryption key to use is chosen based on the second argument to `DES_ENCRYPT()`, if one was given. With no argument, the first key from the DES key file is used. With a *`key_num`* argument, the

given key number (0 to 9) from the DES key file is used. With a *`key_str`* argument, the given key string is used to encrypt *`str`*.

The key file can be specified with the `--des-key-file` server option.

The return string is a binary string where the first character is `CHAR(128 | key_num)`. If an error occurs, `DES_ENCRYPT()` returns `NULL`.

The 128 is added to make it easier to recognize an encrypted key. If you use a string key, *`key_num`* is 127.

The string length for the result is given by this formula:

```
new_len = orig_len + (8 - (orig_len % 8)) + 1
```

Each line in the DES key file has the following format:

```
key_num des_key_str
```

Each *`key_num`* value must be a number in the range from `0` to `9`. Lines in the file may be in any order. *`des_key_str`* is the string that is used to encrypt the message. There should be at least one space between the number and the key. The first key is the default key that is used if you do not specify any key argument to `DES_ENCRYPT()`.

You can tell MySQL to read new key values from the key file with the `FLUSH DES_KEY_FILE` statement. This requires the `RELOAD` privilege.

One benefit of having a set of default keys is that it gives applications a way to check for the existence of encrypted column values, without giving the end user the right to decrypt those values.

```
mysql> SELECT customer_address FROM customer_table
     > WHERE crypted_credit_card = DES_ENCRYPT('credit_card_number');
```

- `ENCODE(str,pass_str)`

  Encrypt *`str`* using *`pass_str`* as the password. The result is a binary string of the same length as *`str`*. To decrypt the result, use `DECODE()`.

  The `ENCODE()` function should no longer be used. If you still need to use `ENCODE()`, a salt value must be used with it to reduce risk. For example:

  ```
  ENCODE('cleartext', CONCAT('my_random_salt','my_secret_password'))
  ```

  A new random salt value must be used whenever a password is updated.

- `ENCRYPT(str[,salt])`

  Encrypts *`str`* using the Unix `crypt()` system call and returns a binary string. The *`salt`* argument must be a string with at least two characters or the result is `NULL`. If no *`salt`* argument is given, a random value is used.

  ```
  mysql> SELECT ENCRYPT('hello');
         -> 'VxuFAJXVARROc'
  ```

  `ENCRYPT()` ignores all but the first eight characters of *`str`*, at least on some systems. This behavior is determined by the implementation of the underlying `crypt()` system call.

  The use of `ENCRYPT()` with the `ucs2`, `utf16`, `utf16le`, or `utf32` multibyte character sets is not recommended because the system call expects a string terminated by a zero byte.

  If `crypt()` is not available on your system (as is the case with Windows), `ENCRYPT()` always returns `NULL`.

- `MD5(str)`

  Calculates an MD5 128-bit checksum for the string. The value is returned as a string of 32 hexadecimal digits, or `NULL` if the argument was `NULL`. The return value can, for example, be used as a hash key. See the notes at the beginning of this section about storing hash values efficiently.

  The return value is a string in the connection character set.

```
mysql> SELECT MD5('testing');
        -> 'ae2b1fca515949e5d54fb22b8ed95575'
```

  This is the "RSA Data Security, Inc. MD5 Message-Digest Algorithm."

  See the note regarding the MD5 algorithm at the beginning this section.

- `OLD_PASSWORD(str)`

  `OLD_PASSWORD()` was added when the implementation of `PASSWORD()` was changed in MySQL 4.1 to improve security. `OLD_PASSWORD()` returns the value of the pre-4.1 implementation of `PASSWORD()` as a string, and is intended to permit you to reset passwords for any pre-4.1 clients that need to connect to your MySQL server without locking them out. See Section 6.1.2.4, "Password Hashing in MySQL".

  The return value is a string in the connection character set.

Note

Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated; expect support for them to be removed in a future MySQL release. Consequently, `OLD_PASSWORD()` is also deprecated.

- `PASSWORD(str)`

  Returns a hashed password string calculated from the cleartext password `str`. The return value is a string in the connection character set, or `NULL` if the argument is `NULL`. This function is the SQL interface to the algorithm used by the server to encrypt MySQL passwords for storage in the `mysql.user` grant table.

  The `old_passwords` system variable controls the password hashing method used by the `PASSWORD()` function. It also influences password hashing performed by `CREATE USER` and `GRANT` statements that specify a password using an `IDENTIFIED BY` clause.

  The following table shows, for each password hashing method, the permitted value of `old_passwords` and which authentication plugins use the hashing method.

Password Hashing Method	old_passwords Value	Associated Authentication Plugin
MySQL 4.1 native hashing	0	`mysql_native_password`
Pre-4.1 ("old") hashing	1	`mysql_old_password`
SHA-256 hashing	2	`sha256_password`

Note

Passwords that use the pre-4.1 hashing method are less secure than passwords that use the native password hashing method and should be avoided. Pre-4.1 passwords are deprecated; expect support for them to be removed in a future MySQL release. Consequently, `old_passwords=1`, which causes `PASSWORD()` to generate pre-4.1 password hashes, is also

deprecated. For account upgrade instructions, see Section 6.4.1.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”.

If `old_passwords=1`, `PASSWORD(str)` returns the same value as `OLD_PASSWORD(str)`. The latter function is not affected by the value of `old_passwords`.

```
mysql> SET old_passwords = 0;
mysql> SELECT PASSWORD('mypass'), OLD_PASSWORD('mypass');
+-------------------------------------------+------------------------+
| PASSWORD('mypass')                        | OLD_PASSWORD('mypass') |
+-------------------------------------------+------------------------+
| *6C8989366EAF75BB670AD8EA7A7FC1176A95CEF4 | 6f8c114b58f2ce9e       |
+-------------------------------------------+------------------------+

mysql> SET old_passwords = 1;
mysql> SELECT PASSWORD('mypass'), OLD_PASSWORD('mypass');
+-------------------+------------------------+
| PASSWORD('mypass') | OLD_PASSWORD('mypass') |
+-------------------+------------------------+
| 6f8c114b58f2ce9e   | 6f8c114b58f2ce9e       |
+-------------------+------------------------+
```

SHA-256 password hashing (`old_passwords=2`) uses a random salt value, which makes the result from `PASSWORD()` nondeterministic. Consequently, statements that use this function are not safe for statement-based replication and cannot be stored in the query cache.

Encryption performed by `PASSWORD()` is one-way (not reversible). It is not the same type of encryption as used for Unix passwords; for that, use `ENCRYPT()`.

Note

`PASSWORD()` is used by the authentication system in MySQL Server; you should *not* use it in your own applications. For that purpose, consider a more secure function such as `AES_ENCRYPT()` or `SHA2()` instead. Also see RFC 2195, section 2 (Challenge-Response Authentication Mechanism (CRAM)), for more information about handling passwords and authentication securely in your applications.

Caution

Under some circumstances, statements that invoke `PASSWORD()` may be recorded in server logs or on the client side in a history file such as `~/.mysql_history`, which means that cleartext passwords may be read by anyone having read access to that information. For information about the conditions under which this occurs for the server logs and how to control it, see Section 6.1.2.3, “Passwords and Logging”. For similar information about client-side logging, see Section 4.5.1.3, “mysql Client Logging”.

- `RANDOM_BYTES(len)`

  This function returns a binary string of *`len`* random bytes generated using the random number generator of the SSL library. Permitted values of *`len`* range from 1 to 1024. For values outside that range, an error occurs.

  `RANDOM_BYTES()` can be used to provide the initialization vector for the `AES_DECRYPT()` and `AES_ENCRYPT()` functions. For use in that context, *`len`* must be at least 16. Larger values are permitted, but bytes in excess of 16 are ignored.

  `RANDOM_BYTES()` generates a random value, which makes its result nondeterministic. Consequently, statements that use this function are unsafe for statement-based replication and cannot be stored in the query cache.

  This function is available as of MySQL 5.6.17.

- `SHA1(str)`, `SHA(str)`

  Calculates an SHA-1 160-bit checksum for the string, as described in RFC 3174 (Secure Hash Algorithm). The value is returned as a string of 40 hexadecimal digits, or `NULL` if the argument was `NULL`. One of the possible uses for this function is as a hash key. See the notes at the beginning of this section about storing hash values efficiently. `SHA()` is synonymous with `SHA1()`.

  The return value is a string in the connection character set.

```
mysql> SELECT SHA1('abc');
        -> 'a9993e364706816aba3e25717850c26c9cd0d89d'
```

  `SHA1()` can be considered a cryptographically more secure equivalent of `MD5()`. However, see the note regarding the MD5 and SHA-1 algorithms at the beginning this section.

- `SHA2(str, hash_length)`

  Calculates the SHA-2 family of hash functions (SHA-224, SHA-256, SHA-384, and SHA-512). The first argument is the plaintext string to be hashed. The second argument indicates the desired bit length of the result, which must have a value of 224, 256, 384, 512, or 0 (which is equivalent to 256). If either argument is `NULL` or the hash length is not one of the permitted values, the return value is `NULL`. Otherwise, the function result is a hash value containing the desired number of bits. See the notes at the beginning of this section about storing hash values efficiently.

  The return value is a string in the connection character set.

```
mysql> SELECT SHA2('abc', 224);
        -> '23097d223405d8228642a477bda255b32aadbce4bda0b3f7e36c9da7'
```

  This function works only if MySQL has been configured with SSL support. See Section 6.3, “Using Encrypted Connections”.

  `SHA2()` can be considered cryptographically more secure than `MD5()` or `SHA1()`.

- `UNCOMPRESS(string_to_uncompress)`

  Uncompresses a string compressed by the `COMPRESS()` function. If the argument is not a compressed value, the result is `NULL`. This function requires MySQL to have been compiled with a compression library such as `zlib`. Otherwise, the return value is always `NULL`.

```
mysql> SELECT UNCOMPRESS(COMPRESS('any string'));
        -> 'any string'
mysql> SELECT UNCOMPRESS('any string');
        -> NULL
```

- `UNCOMPRESSED_LENGTH(compressed_string)`

  Returns the length that the compressed string had before being compressed.

```
mysql> SELECT UNCOMPRESSED_LENGTH(COMPRESS(REPEAT('a',30)));
        -> 30
```

- `VALIDATE_PASSWORD_STRENGTH(str)`

  Given an argument representing a plaintext password, this function returns an integer to indicate how strong the password is. The return value ranges from 0 (weak) to 100 (strong).

  Password assessment by `VALIDATE_PASSWORD_STRENGTH()` is done by the `validate_password` plugin. If that plugin is not installed, the function always returns 0. For information about installing `validate_password`, see Section 6.4.3, “The Password Validation Plugin”. To examine or configure the parameters that affect password testing, check or set the

system variables implemented by `validate_password`. See Section 6.4.3.2, “Password Validation Plugin Options and Variables”.

The password is subjected to increasingly strict tests and the return value reflects which tests were satisfied, as shown in the following table.

Password Test	Return Value
Length < 4	0
Length ≥ 4 and < `validate_password_length`	25
Satisfies policy 1 (`LOW`)	50
Satisfies policy 2 (`MEDIUM`)	75
Satisfies policy 3 (`STRONG`)	100

12.15 Locking Functions

This section describes functions used to manipulate user-level locks.

Table 12.19 Locking Functions

Name	Description
`GET_LOCK()`	Get a named lock
`IS_FREE_LOCK()`	Whether the named lock is free
`IS_USED_LOCK()`	Whether the named lock is in use; return connection identifier if true
`RELEASE_LOCK()`	Release the named lock

- `GET_LOCK(`*`str`*`,`*`timeout`*`)`

  Tries to obtain a lock with a name given by the string *`str`*, using a timeout of *`timeout`* seconds. A negative *`timeout`* value means infinite timeout. The lock is exclusive. While held by one session, other sessions cannot obtain a lock of the same name.

  Returns `1` if the lock was obtained successfully, `0` if the attempt timed out (for example, because another client has previously locked the name), or `NULL` if an error occurred (such as running out of memory or the thread was killed with `mysqladmin kill`).

  A lock obtained with `GET_LOCK()` is released explicitly by executing `RELEASE_LOCK()` or implicitly when your session terminates (either normally or abnormally). Locks obtained with `GET_LOCK()` are not released when transactions commit or roll back.

  Important

  The behavior of `GET_LOCK()` changes in MySQL 5.7. In consideration of future upgrades, limit the *`str`* value to 64 characters or less and do not rely on subsequent calls to `GET_LOCK()` releasing previous locks.

  `GET_LOCK()` can be used to implement application locks or to simulate record locks. Names are locked on a server-wide basis. If a name has been locked within one session, `GET_LOCK()` blocks any request by another session for a lock with the same name. This enables clients that agree on a given lock name to use the name to perform cooperative advisory locking. But be aware that it also enables a client that is not among the set of cooperating clients to lock a name, either inadvertently or deliberately, and thus prevent any of the cooperating clients from locking that name. One way to reduce the likelihood of this is to use lock names that are database-specific or application-specific. For example, use lock names of the form *`db_name.str`* or *`app_name.str`*.

```
mysql> SELECT GET_LOCK('lock1',10);
```

```
        -> 1
mysql> SELECT IS_FREE_LOCK('lock2');
        -> 1
mysql> SELECT GET_LOCK('lock2',10);
        -> 1
mysql> SELECT RELEASE_LOCK('lock2');
        -> 1
mysql> SELECT RELEASE_LOCK('lock1');
        -> NULL
```

The second `RELEASE_LOCK()` call returns `NULL` because the lock `'lock1'` was automatically released by the second `GET_LOCK()` call.

If multiple clients are waiting for a lock, the order in which they acquire it is undefined. Applications should not assume that clients acquire the lock in the same order that they issued the lock requests.

`GET_LOCK()` is unsafe for statement-based replication. A warning is logged if you use this function when `binlog_format` is set to `STATEMENT`.

- `IS_FREE_LOCK(str)`

  Checks whether the lock named *`str`* is free to use (that is, not locked). Returns `1` if the lock is free (no one is using the lock), `0` if the lock is in use, and `NULL` if an error occurs (such as an incorrect argument).

  This function is unsafe for statement-based replication. A warning is logged if you use this function when `binlog_format` is set to `STATEMENT`.

- `IS_USED_LOCK(str)`

  Checks whether the lock named *`str`* is in use (that is, locked). If so, it returns the connection identifier of the client session that holds the lock. Otherwise, it returns `NULL`.

  This function is unsafe for statement-based replication. A warning is logged if you use this function when `binlog_format` is set to `STATEMENT`.

- `RELEASE_LOCK(str)`

  Releases the lock named by the string *`str`* that was obtained with `GET_LOCK()`. Returns `1` if the lock was released, `0` if the lock was not established by this thread (in which case the lock is not released), and `NULL` if the named lock did not exist. The lock does not exist if it was never obtained by a call to `GET_LOCK()` or if it has previously been released.

  The `DO` statement is convenient to use with `RELEASE_LOCK()`. See Section 13.2.3, “DO Statement”.

  This function is unsafe for statement-based replication. A warning is logged if you use this function when `binlog_format` is set to `STATEMENT`.

12.16 Information Functions

Table 12.20 Information Functions

Name	Description
`BENCHMARK()`	Repeatedly execute an expression
`CHARSET()`	Return the character set of the argument
`COERCIBILITY()`	Return the collation coercibility value of the string argument
`COLLATION()`	Return the collation of the string argument
`CONNECTION_ID()`	Return the connection ID (thread ID) for the connection
`CURRENT_USER()`, `CURRENT_USER`	The authenticated user name and host name

Name	Description
DATABASE()	Return the default (current) database name
FOUND_ROWS()	For a SELECT with a LIMIT clause, the number of rows that would be returned were there no LIMIT clause
LAST_INSERT_ID()	Value of the AUTOINCREMENT column for the last INSERT
ROW_COUNT()	The number of rows updated
SCHEMA()	Synonym for DATABASE()
SESSION_USER()	Synonym for USER()
SYSTEM_USER()	Synonym for USER()
USER()	The user name and host name provided by the client
VERSION()	Return a string that indicates the MySQL server version

- BENCHMARK(*count*,*expr*)

  The BENCHMARK() function executes the expression *expr* repeatedly *count* times. It may be used to time how quickly MySQL processes the expression. The result value is 0, or NULL for inappropriate arguments such as a NULL or negative repeat count.

  The intended use is from within the mysql client, which reports query execution times:

```
mysql> SELECT BENCHMARK(1000000,AES_ENCRYPT('hello','goodbye'));
+-------------------------------------------------+
| BENCHMARK(1000000,AES_ENCRYPT('hello','goodbye')) |
+-------------------------------------------------+
|                                               0 |
+-------------------------------------------------+
1 row in set (4.74 sec)
```

  The time reported is elapsed time on the client end, not CPU time on the server end. It is advisable to execute BENCHMARK() several times, and to interpret the result with regard to how heavily loaded the server machine is.

  BENCHMARK() is intended for measuring the runtime performance of scalar expressions, which has some significant implications for the way that you use it and interpret the results:

  - Only scalar expressions can be used. Although the expression can be a subquery, it must return a single column and at most a single row. For example, BENCHMARK(10, (SELECT * FROM t)) fails if the table t has more than one column or more than one row.
  - Executing a SELECT *expr* statement *N* times differs from executing SELECT BENCHMARK(*N*, *expr*) in terms of the amount of overhead involved. The two have very different execution profiles and you should not expect them to take the same amount of time. The former involves the parser, optimizer, table locking, and runtime evaluation *N* times each. The latter involves only runtime evaluation *N* times, and all the other components just once. Memory structures already allocated are reused, and runtime optimizations such as local caching of results already evaluated for aggregate functions can alter the results. Use of BENCHMARK() thus measures performance of the runtime component by giving more weight to that component and removing the “noise” introduced by the network, parser, optimizer, and so forth.

- CHARSET(*str*)

  Returns the character set of the string argument.

```
mysql> SELECT CHARSET('abc');
        -> 'latin1'
mysql> SELECT CHARSET(CONVERT('abc' USING utf8));
        -> 'utf8'
mysql> SELECT CHARSET(USER());
        -> 'utf8'
```

- `COERCIBILITY(str)`

  Returns the collation coercibility value of the string argument.

```
mysql> SELECT COERCIBILITY('abc' COLLATE latin1_swedish_ci);
        -> 0
mysql> SELECT COERCIBILITY(USER());
        -> 3
mysql> SELECT COERCIBILITY('abc');
        -> 4
mysql> SELECT COERCIBILITY(1000);
        -> 5
```

  The return values have the meanings shown in the following table. Lower values have higher precedence.

Coercibility	Meaning	Example
0	Explicit collation	Value with `COLLATE` clause
1	No collation	Concatenation of strings with different collations
2	Implicit collation	Column value, stored routine parameter or local variable
3	System constant	`USER()` return value
4	Coercible	Literal string
5	Numeric	Numeric or temporal value
5	Ignorable	`NULL` or an expression derived from `NULL`

  For more information, see Section 10.8.4, “Collation Coercibility in Expressions”.

- `COLLATION(str)`

  Returns the collation of the string argument.

```
mysql> SELECT COLLATION('abc');
        -> 'latin1_swedish_ci'
mysql> SELECT COLLATION(_utf8'abc');
        -> 'utf8_general_ci'
```

- `CONNECTION_ID()`

  Returns the connection ID (thread ID) for the connection. Every connection has an ID that is unique among the set of currently connected clients.

  The value returned by `CONNECTION_ID()` is the same type of value as displayed in the `ID` column of the `INFORMATION_SCHEMA.PROCESSLIST` table, the `Id` column of `SHOW PROCESSLIST` output, and the `PROCESSLIST_ID` column of the Performance Schema `threads` table.

```
mysql> SELECT CONNECTION_ID();
        -> 23786
```

  Warning

  Changing the session value of the `pseudo_thread_id` system variable changes the value returned by the `CONNECTION_ID()` function.

- `CURRENT_USER`, `CURRENT_USER()`

  Returns the user name and host name combination for the MySQL account that the server used to authenticate the current client. This account determines your access privileges. The return value is a string in the `utf8` character set.

The value of `CURRENT_USER()` can differ from the value of `USER()`.

```
mysql> SELECT USER();
        -> 'davida@localhost'
mysql> SELECT * FROM mysql.user;
ERROR 1044: Access denied for user ''@'localhost' to
database 'mysql'
mysql> SELECT CURRENT_USER();
        -> '@localhost'
```

The example illustrates that although the client specified a user name of `davida` (as indicated by the value of the `USER()` function), the server authenticated the client using an anonymous user account (as seen by the empty user name part of the `CURRENT_USER()` value). One way this might occur is that there is no account listed in the grant tables for `davida`.

Within a stored program or view, `CURRENT_USER()` returns the account for the user who defined the object (as given by its `DEFINER` value) unless defined with the `SQL SECURITY INVOKER` characteristic. In the latter case, `CURRENT_USER()` returns the object's invoker.

Triggers and events have no option to define the `SQL SECURITY` characteristic, so for these objects, `CURRENT_USER()` returns the account for the user who defined the object. To return the invoker, use `USER()` or `SESSION_USER()`.

The following statements support use of the `CURRENT_USER()` function to take the place of the name of (and, possibly, a host for) an affected user or a definer; in such cases, `CURRENT_USER()` is expanded where and as needed:

- `DROP USER`
- `RENAME USER`
- `GRANT`
- `REVOKE`
- `CREATE FUNCTION`
- `CREATE PROCEDURE`
- `CREATE TRIGGER`
- `CREATE EVENT`
- `CREATE VIEW`
- `ALTER EVENT`
- `ALTER VIEW`
- `SET PASSWORD`

For information about the implications that this expansion of `CURRENT_USER()` has for replication, see Section 17.4.1.8, “Replication of CURRENT_USER()”.

- `DATABASE()`

  Returns the default (current) database name as a string in the `utf8` character set. If there is no default database, `DATABASE()` returns `NULL`. Within a stored routine, the default database is the database that the routine is associated with, which is not necessarily the same as the database that is the default in the calling context.

  ```
  mysql> SELECT DATABASE();
  ```

```
        -> 'test'
```

If there is no default database, `DATABASE()` returns `NULL`.

- `FOUND_ROWS()`

  A `SELECT` statement may include a `LIMIT` clause to restrict the number of rows the server returns to the client. In some cases, it is desirable to know how many rows the statement would have returned without the `LIMIT`, but without running the statement again. To obtain this row count, include an `SQL_CALC_FOUND_ROWS` option in the `SELECT` statement, and then invoke `FOUND_ROWS()` afterward:

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM tbl_name
    -> WHERE id > 100 LIMIT 10;
mysql> SELECT FOUND_ROWS();
```

  The second `SELECT` returns a number indicating how many rows the first `SELECT` would have returned had it been written without the `LIMIT` clause.

  In the absence of the `SQL_CALC_FOUND_ROWS` option in the most recent successful `SELECT` statement, `FOUND_ROWS()` returns the number of rows in the result set returned by that statement. If the statement includes a `LIMIT` clause, `FOUND_ROWS()` returns the number of rows up to the limit. For example, `FOUND_ROWS()` returns 10 or 60, respectively, if the statement includes `LIMIT 10` or `LIMIT 50, 10`.

  The row count available through `FOUND_ROWS()` is transient and not intended to be available past the statement following the `SELECT SQL_CALC_FOUND_ROWS` statement. If you need to refer to the value later, save it:

```
mysql> SELECT SQL_CALC_FOUND_ROWS * FROM ... ;
mysql> SET @rows = FOUND_ROWS();
```

  If you are using `SELECT SQL_CALC_FOUND_ROWS`, MySQL must calculate how many rows are in the full result set. However, this is faster than running the query again without `LIMIT`, because the result set need not be sent to the client.

  `SQL_CALC_FOUND_ROWS` and `FOUND_ROWS()` can be useful in situations when you want to restrict the number of rows that a query returns, but also determine the number of rows in the full result set without running the query again. An example is a Web script that presents a paged display

containing links to the pages that show other sections of a search result. Using FOUND_ROWS() enables you to determine how many other pages are needed for the rest of the result.

The use of SQL_CALC_FOUND_ROWS and FOUND_ROWS() is more complex for UNION statements than for simple SELECT statements, because LIMIT may occur at multiple places in a UNION. It may be applied to individual SELECT statements in the UNION, or global to the UNION result as a whole.

The intent of SQL_CALC_FOUND_ROWS for UNION is that it should return the row count that would be returned without a global LIMIT. The conditions for use of SQL_CALC_FOUND_ROWS with UNION are:

- The SQL_CALC_FOUND_ROWS keyword must appear in the first SELECT of the UNION.
- The value of FOUND_ROWS() is exact only if UNION ALL is used. If UNION without ALL is used, duplicate removal occurs and the value of FOUND_ROWS() is only approximate.
- If no LIMIT is present in the UNION, SQL_CALC_FOUND_ROWS is ignored and returns the number of rows in the temporary table that is created to process the UNION.

Beyond the cases described here, the behavior of FOUND_ROWS() is undefined (for example, its value following a SELECT statement that fails with an error).

Important

FOUND_ROWS() is not replicated reliably using statement-based replication. This function is automatically replicated using row-based replication.

- LAST_INSERT_ID(), LAST_INSERT_ID(*expr*)

  With no argument, LAST_INSERT_ID() returns a 64-bit value representing the first automatically generated value successfully inserted for an AUTO_INCREMENT column as a result of the most recently executed INSERT statement. The value of LAST_INSERT_ID() remains unchanged if no rows are successfully inserted.

  With an argument, LAST_INSERT_ID() returns an unsigned integer.

  For example, after inserting a row that generates an AUTO_INCREMENT value, you can get the value like this:

```
mysql> SELECT LAST_INSERT_ID();
        -> 195
```

  The currently executing statement does not affect the value of LAST_INSERT_ID(). Suppose that you generate an AUTO_INCREMENT value with one statement, and then refer to LAST_INSERT_ID() in a multiple-row INSERT statement that inserts rows into a table with its own AUTO_INCREMENT column. The value of LAST_INSERT_ID() remains stable in the second statement; its value for the second and later rows is not affected by the earlier row insertions. (However, if you mix references to LAST_INSERT_ID() and LAST_INSERT_ID(*expr*), the effect is undefined.)

  If the previous statement returned an error, the value of LAST_INSERT_ID() is undefined. For transactional tables, if the statement is rolled back due to an error, the value of LAST_INSERT_ID() is left undefined. For manual ROLLBACK, the value of LAST_INSERT_ID() is not restored to that before the transaction; it remains as it was at the point of the ROLLBACK.

  Prior to MySQL 5.6.15, this function was not replicated correctly if replication filtering rules were in use. (Bug #17234370, Bug #69861)

  Within the body of a stored routine (procedure or function) or a trigger, the value of LAST_INSERT_ID() changes the same way as for statements executed outside the body of these

kinds of objects. The effect of a stored routine or trigger upon the value of `LAST_INSERT_ID()` that is seen by following statements depends on the kind of routine:

- If a stored procedure executes statements that change the value of `LAST_INSERT_ID()`, the changed value is seen by statements that follow the procedure call.
- For stored functions and triggers that change the value, the value is restored when the function or trigger ends, so statements following it do not see a changed value.

The ID that was generated is maintained in the server on a *per-connection basis*. This means that the value returned by the function to a given client is the first `AUTO_INCREMENT` value generated for most recent statement affecting an `AUTO_INCREMENT` column *by that client*. This value cannot be affected by other clients, even if they generate `AUTO_INCREMENT` values of their own. This behavior ensures that each client can retrieve its own ID without concern for the activity of other clients, and without the need for locks or transactions.

The value of `LAST_INSERT_ID()` is not changed if you set the `AUTO_INCREMENT` column of a row to a non-“magic” value (that is, a value that is not `NULL` and not `0`).

Important

If you insert multiple rows using a single `INSERT` statement, `LAST_INSERT_ID()` returns the value generated for the *first* inserted row *only*. The reason for this is to make it possible to reproduce easily the same `INSERT` statement against some other server.

For example:

```
mysql> USE test;

mysql> CREATE TABLE t (
       id INT AUTO_INCREMENT NOT NULL PRIMARY KEY,
       name VARCHAR(10) NOT NULL
       );

mysql> INSERT INTO t VALUES (NULL, 'Bob');

mysql> SELECT * FROM t;
+----+------+
| id | name |
+----+------+
|  1 | Bob  |
+----+------+

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                1 |
+------------------+

mysql> INSERT INTO t VALUES
       (NULL, 'Mary'), (NULL, 'Jane'), (NULL, 'Lisa');

mysql> SELECT * FROM t;
+----+------+
| id | name |
+----+------+
|  1 | Bob  |
|  2 | Mary |
|  3 | Jane |
|  4 | Lisa |
+----+------+

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
```

```
+------------------+
|                2 |
+------------------+
```

Although the second `INSERT` statement inserted three new rows into `t`, the ID generated for the first of these rows was `2`, and it is this value that is returned by `LAST_INSERT_ID()` for the following `SELECT` statement.

If you use `INSERT IGNORE` and the row is ignored, the `LAST_INSERT_ID()` remains unchanged from the current value (or 0 is returned if the connection has not yet performed a successful `INSERT`) and, for non-transactional tables, the `AUTO_INCREMENT` counter is not incremented. For `InnoDB` tables, the `AUTO_INCREMENT` counter is incremented if `innodb_autoinc_lock_mode` is set to `1` or `2`, as demonstrated in the following example:

```
mysql> USE test;

mysql> SELECT @@innodb_autoinc_lock_mode;
+----------------------------+
| @@innodb_autoinc_lock_mode |
+----------------------------+
|                          1 |
+----------------------------+

mysql> CREATE TABLE `t` (
       `id` INT(11) NOT NULL AUTO_INCREMENT,
       `val` INT(11) DEFAULT NULL,
       PRIMARY KEY (`id`),
       UNIQUE KEY `i1` (`val`)
       ) ENGINE=InnoDB DEFAULT CHARSET=latin1;

# Insert two rows

mysql> INSERT INTO t (val) VALUES (1),(2);

# With auto_increment_offset=1, the inserted rows
# result in an AUTO_INCREMENT value of 3

mysql> SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE `t` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `val` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `i1` (`val`)
) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=latin1

# LAST_INSERT_ID() returns the first automatically generated
# value that is successfully inserted for the AUTO_INCREMENT column

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                1 |
+------------------+

# The attempted insertion of duplicate rows fail but errors are ignored

mysql> INSERT IGNORE INTO t (val) VALUES (1),(2);
Query OK, 0 rows affected (0.00 sec)
Records: 2  Duplicates: 2  Warnings: 0

# With innodb_autoinc_lock_mode=1, the AUTO_INCREMENT counter
# is incremented for the ignored rows

mysql> SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE `t` (
```

```
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `val` int(11) DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `i1` (`val`)
) ENGINE=InnoDB AUTO_INCREMENT=5 DEFAULT CHARSET=latin1

# The LAST_INSERT_ID is unchanged because the previous insert was unsuccessful

mysql> SELECT LAST_INSERT_ID();
+------------------+
| LAST_INSERT_ID() |
+------------------+
|                1 |
+------------------+
```

For more information, see Section 14.6.1.6, “AUTO_INCREMENT Handling in InnoDB”.

If *`expr`* is given as an argument to `LAST_INSERT_ID()`, the value of the argument is returned by the function and is remembered as the next value to be returned by `LAST_INSERT_ID()`. This can be used to simulate sequences:

1. Create a table to hold the sequence counter and initialize it:

   ```
   mysql> CREATE TABLE sequence (id INT NOT NULL);
   mysql> INSERT INTO sequence VALUES (0);
   ```

2. Use the table to generate sequence numbers like this:

   ```
   mysql> UPDATE sequence SET id=LAST_INSERT_ID(id+1);
   mysql> SELECT LAST_INSERT_ID();
   ```

   The `UPDATE` statement increments the sequence counter and causes the next call to `LAST_INSERT_ID()` to return the updated value. The `SELECT` statement retrieves that value. The `mysql_insert_id()` C API function can also be used to get the value. See mysql_insert_id().

You can generate sequences without calling `LAST_INSERT_ID()`, but the utility of using the function this way is that the ID value is maintained in the server as the last automatically generated value. It is multi-user safe because multiple clients can issue the `UPDATE` statement and get their own sequence value with the `SELECT` statement (or `mysql_insert_id()`), without affecting or being affected by other clients that generate their own sequence values.

The `mysql_insert_id()` function is only updated after the `INSERT` and `UPDATE` statements, so you cannot use the C API function to retrieve the value for `LAST_INSERT_ID(`*`expr`*`)` after executing other SQL statements like `SELECT` or `SET`.

- `ROW_COUNT()`

  `ROW_COUNT()` returns a value as follows:

  - DDL statements: 0. This applies to statements such as `CREATE TABLE` or `DROP TABLE`.
  - DML statements other than `SELECT`: The number of affected rows. This applies to statements such as `UPDATE`, `INSERT`, or `DELETE` (as before), but now also to statements such as `ALTER TABLE` and `LOAD DATA`.
  - `SELECT`: -1 if the statement returns a result set, or the number of rows “affected” if it does not. For example, for `SELECT * FROM t1`, `ROW_COUNT()` returns -1. For `SELECT * FROM t1 INTO OUTFILE '`*`file_name`*`'`, `ROW_COUNT()` returns the number of rows written to the file.
  - `SIGNAL` statements: 0.

  For `UPDATE` statements, the affected-rows value by default is the number of rows actually changed. If you specify the `CLIENT_FOUND_ROWS` flag to `mysql_real_connect()` when connecting to

mysqld, the affected-rows value is the number of rows “found”; that is, matched by the WHERE clause.

For REPLACE statements, the affected-rows value is 2 if the new row replaced an old row, because in this case, one row was inserted after the duplicate was deleted.

For INSERT ... ON DUPLICATE KEY UPDATE statements, the affected-rows value per row is 1 if the row is inserted as a new row, 2 if an existing row is updated, and 0 if an existing row is set to its current values. If you specify the CLIENT_FOUND_ROWS flag, the affected-rows value is 1 (not 0) if an existing row is set to its current values.

The ROW_COUNT() value is similar to the value from the mysql_affected_rows() C API function and the row count that the mysql client displays following statement execution.

```
mysql> INSERT INTO t VALUES(1),(2),(3);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT ROW_COUNT();
+-------------+
| ROW_COUNT() |
+-------------+
|           3 |
+-------------+
1 row in set (0.00 sec)

mysql> DELETE FROM t WHERE i IN(1,2);
Query OK, 2 rows affected (0.00 sec)

mysql> SELECT ROW_COUNT();
+-------------+
| ROW_COUNT() |
+-------------+
|           2 |
+-------------+
1 row in set (0.00 sec)
```

Important

ROW_COUNT() is not replicated reliably using statement-based replication. This function is automatically replicated using row-based replication.

- SCHEMA()

  This function is a synonym for DATABASE().

- SESSION_USER()

  SESSION_USER() is a synonym for USER().

- SYSTEM_USER()

  SYSTEM_USER() is a synonym for USER().

- USER()

  Returns the current MySQL user name and host name as a string in the utf8 character set.

```
mysql> SELECT USER();
        -> 'davida@localhost'
```

The value indicates the user name you specified when connecting to the server, and the client host from which you connected. The value can be different from that of CURRENT_USER().

- `VERSION()`

  Returns a string that indicates the MySQL server version. The string uses the `utf8` character set. The value might have a suffix in addition to the version number. See the description of the `version` system variable in Section 5.1.7, “Server System Variables”.

  This function is unsafe for statement-based replication. A warning is logged if you use this function when `binlog_format` is set to `STATEMENT`.

```
mysql> SELECT VERSION();
        -> '5.6.51-standard'
```

12.17 Spatial Analysis Functions

MySQL provides functions to perform various operations on spatial data. These functions can be grouped into several major categories according to the type of operation they perform:

- Functions that create geometries in various formats (WKT, WKB, internal)
- Functions that convert geometries between formats
- Functions that access qualitative or quantitative properties of a geometry
- Functions that describe relations between two geometries
- Functions that create new geometries from existing ones

For general background about MySQL support for using spatial data, see Section 11.4, “Spatial Data Types”.

12.17.1 Spatial Function Reference

The following table lists each spatial function and provides a short description of each one.

Table 12.21 Spatial Functions

Name	Description
`Area()`	Return Polygon or MultiPolygon area
`AsBinary()`, `AsWKB()`	Convert from internal geometry format to WKB
`AsText()`, `AsWKT()`	Convert from internal geometry format to WKT
`Buffer()`	Return geometry of points within given distance from geometry
`Centroid()`	Return centroid as a point
`Contains()`	Whether MBR of one geometry contains MBR of another
`Crosses()`	Whether one geometry crosses another
`Dimension()`	Dimension of geometry
`Disjoint()`	Whether MBRs of two geometries are disjoint
`EndPoint()`	End Point of LineString
`Envelope()`	Return MBR of geometry
`Equals()`	Whether MBRs of two geometries are equal
`ExteriorRing()`	Return exterior ring of Polygon
`GeomCollFromText()`, `GeometryCollectionFromText()`	Return geometry collection from WKT
`GeomCollFromWKB()`, `GeometryCollectionFromWKB()`	Return geometry collection from WKB
`GeometryCollection()`	Construct geometry collection from geometries

Name	Description
GeometryN()	Return N-th geometry from geometry collection
GeometryType()	Return name of geometry type
GeomFromText(), GeometryFromText()	Return geometry from WKT
GeomFromWKB(), GeometryFromWKB()	Return geometry from WKB
GLength()	Return length of LineString
InteriorRingN()	Return N-th interior ring of Polygon
Intersects()	Whether MBRs of two geometries intersect
IsClosed()	Whether a geometry is closed and simple
IsEmpty()	Whether a geometry is empty
IsSimple()	Whether a geometry is simple
LineFromText(), LineStringFromText()	Construct LineString from WKT
LineFromWKB(), LineStringFromWKB()	Construct LineString from WKB
LineString()	Construct LineString from Point values
MBRContains()	Whether MBR of one geometry contains MBR of another
MBRDisjoint()	Whether MBRs of two geometries are disjoint
MBREqual()	Whether MBRs of two geometries are equal
MBRIntersects()	Whether MBRs of two geometries intersect
MBROverlaps()	Whether MBRs of two geometries overlap
MBRTouches()	Whether MBRs of two geometries touch
MBRWithin()	Whether MBR of one geometry is within MBR of another
MLineFromText(), MultiLineStringFromText()	Construct MultiLineString from WKT
MLineFromWKB(), MultiLineStringFromWKB()	Construct MultiLineString from WKB
MPointFromText(), MultiPointFromText()	Construct MultiPoint from WKT
MPointFromWKB(), MultiPointFromWKB()	Construct MultiPoint from WKB
MPolyFromText(), MultiPolygonFromText()	Construct MultiPolygon from WKT
MPolyFromWKB(), MultiPolygonFromWKB()	Construct MultiPolygon from WKB
MultiLineString()	Contruct MultiLineString from LineString values
MultiPoint()	Construct MultiPoint from Point values
MultiPolygon()	Construct MultiPolygon from Polygon values
NumGeometries()	Return number of geometries in geometry collection
NumInteriorRings()	Return number of interior rings in Polygon
NumPoints()	Return number of points in LineString
Overlaps()	Whether MBRs of two geometries overlap
Point()	Construct Point from coordinates

Name	Description
PointFromText()	Construct Point from WKT
PointFromWKB()	Construct Point from WKB
PointN()	Return N-th point from LineString
PolyFromText(), PolygonFromText()	Construct Polygon from WKT
PolyFromWKB(), PolygonFromWKB()	Construct Polygon from WKB
Polygon()	Construct Polygon from LineString arguments
SRID()	Return spatial reference system ID for geometry
ST_Area()	Return Polygon or MultiPolygon area
ST_AsBinary(), ST_AsWKB()	Convert from internal geometry format to WKB
ST_AsText(), ST_AsWKT()	Convert from internal geometry format to WKT
ST_Buffer()	Return geometry of points within given distance from geometry
ST_Centroid()	Return centroid as a point
ST_Contains()	Whether one geometry contains another
ST_Crosses()	Whether one geometry crosses another
ST_Difference()	Return point set difference of two geometries
ST_Dimension()	Dimension of geometry
ST_Disjoint()	Whether one geometry is disjoint from another
ST_Distance()	The distance of one geometry from another
ST_EndPoint()	End Point of LineString
ST_Envelope()	Return MBR of geometry
ST_Equals()	Whether one geometry is equal to another
ST_ExteriorRing()	Return exterior ring of Polygon
ST_GeomCollFromText(), ST_GeometryCollectionFromText(), ST_GeomCollFromTxt()	Return geometry collection from WKT
ST_GeomCollFromWKB(), ST_GeometryCollectionFromWKB()	Return geometry collection from WKB
ST_GeometryN()	Return N-th geometry from geometry collection
ST_GeometryType()	Return name of geometry type
ST_GeomFromText(), ST_GeometryFromText()	Return geometry from WKT
ST_GeomFromWKB(), ST_GeometryFromWKB()	Return geometry from WKB
ST_InteriorRingN()	Return N-th interior ring of Polygon
ST_Intersection()	Return point set intersection of two geometries
ST_Intersects()	Whether one geometry intersects another
ST_IsClosed()	Whether a geometry is closed and simple
ST_IsEmpty()	Whether a geometry is empty
ST_IsSimple()	Whether a geometry is simple

Name	Description
ST_LineFromText(), ST_LineStringFromText()	Construct LineString from WKT
ST_LineFromWKB(), ST_LineStringFromWKB()	Construct LineString from WKB
ST_NumGeometries()	Return number of geometries in geometry collection
ST_NumInteriorRing(), ST_NumInteriorRings()	Return number of interior rings in Polygon
ST_NumPoints()	Return number of points in LineString
ST_Overlaps()	Whether one geometry overlaps another
ST_PointFromText()	Construct Point from WKT
ST_PointFromWKB()	Construct Point from WKB
ST_PointN()	Return N-th point from LineString
ST_PolyFromText(), ST_PolygonFromText()	Construct Polygon from WKT
ST_PolyFromWKB(), ST_PolygonFromWKB()	Construct Polygon from WKB
ST_SRID()	Return spatial reference system ID for geometry
ST_StartPoint()	Start Point of LineString
ST_SymDifference()	Return point set symmetric difference of two geometries
ST_Touches()	Whether one geometry touches another
ST_Union()	Return point set union of two geometries
ST_Within()	Whether one geometry is within another
ST_X()	Return X coordinate of Point
ST_Y()	Return Y coordinate of Point
StartPoint()	Start Point of LineString
Touches()	Whether one geometry touches another
Within()	Whether MBR of one geometry is within MBR of another
X()	Return X coordinate of Point
Y()	Return Y coordinate of Point

12.17.2 Argument Handling by Spatial Functions

Spatial values, or geometries, have the properties described in Section 11.4.2.2, “Geometry Class”. The following discussion lists general spatial function argument-handling characteristics. Specific functions or groups of functions may have additional or different argument-handling characteristics, as discussed in the sections where those function descriptions occur. Where that is true, those descriptions take precedence over the general discussion here.

Spatial functions are defined only for valid geometry values. If an invalid geometry is passed to a spatial function, the result is undefined.

The spatial reference identifier (SRID) of a geometry identifies the coordinate space in which the geometry is defined. In MySQL, the SRID value is an integer associated with the geometry value. The maximum usable SRID value is $2^{32}-1$. If a larger value is given, only the lower 32 bits are used.

Geometry values produced by any spatial function inherit the SRID of the geometry arguments.

12.17.3 Functions That Create Geometry Values from WKT Values

These functions take as arguments a Well-Known Text (WKT) representation and, optionally, a spatial reference system identifier (SRID). They return the corresponding geometry.

`GeomFromText()` and `ST_GeomFromText()` accept a WKT value of any geometry type as their first argument. Other functions provide type-specific construction functions for construction of geometry values of each geometry type.

For a description of WKT format, see Well-Known Text (WKT) Format.

- `GeomCollFromText(wkt [, srid])`, `GeometryCollectionFromText(wkt [, srid])`

  `ST_GeomCollFromText()`, `ST_GeometryCollectionFromText()`, `GeomCollFromText()`, and `GeometryCollectionFromText()` are synonyms. For more information, see the description of `ST_GeomCollFromText()`.

- `GeomFromText(wkt [, srid])`, `GeometryFromText(wkt [, srid])`

  `ST_GeomFromText()`, `ST_GeometryFromText()`, `GeomFromText()`, and `GeometryFromText()` are synonyms. For more information, see the description of `ST_GeomFromText()`.

- `LineFromText(wkt [, srid])`, `LineStringFromText(wkt [, srid])`

  `ST_LineFromText()`, `ST_LineStringFromText()`, `LineFromText()`, and `LineStringFromText()` are synonyms. For more information, see the description of `ST_LineFromText()`.

- `MLineFromText(wkt [, srid])`, `MultiLineStringFromText(wkt [, srid])`

  Constructs a `MultiLineString` value using its WKT representation and SRID.

  The result is `NULL` if the geometry argument is `NULL` or not a syntactically well-formed geometry.

- `MPointFromText(wkt [, srid])`, `MultiPointFromText(wkt [, srid])`

  Constructs a `MultiPoint` value using its WKT representation and SRID.

  The result is `NULL` if the geometry argument is `NULL` or not a syntactically well-formed geometry.

- `MPolyFromText(wkt [, srid])`, `MultiPolygonFromText(wkt [, srid])`

  Constructs a `MultiPolygon` value using its WKT representation and SRID.

  The result is `NULL` if the geometry argument is `NULL` or not a syntactically well-formed geometry.

- `PointFromText(wkt [, srid])`

  `ST_PointFromText()` and `PointFromText()` are synonyms. For more information, see the description of `ST_PointFromText()`.

- `PolyFromText(wkt [, srid])`, `PolygonFromText(wkt [, srid])`

  `ST_PolyFromText()`, `ST_PolygonFromText()`, `PolyFromText()`, and `PolygonFromText()` are synonyms. For more information, see the description of `ST_PolyFromText()`.

- `ST_GeomCollFromText(wkt [, srid])`, `ST_GeometryCollectionFromText(wkt [, srid])`

  Constructs a `GeometryCollection` value using its WKT representation and SRID.

  The result is `NULL` if the geometry argument is `NULL` or not a syntactically well-formed geometry.

```
mysql> SET @g = "MULTILINESTRING((10 10, 11 11), (9 9, 10 10))";
mysql> SELECT ST_AsText(ST_GeomCollFromText(@g));
+-----------------------------------------+
| ST_AsText(ST_GeomCollFromText(@g))      |
+-----------------------------------------+
| MULTILINESTRING((10 10,11 11),(9 9,10 10)) |
+-----------------------------------------+
```

`ST_GeomCollFromText()`, `ST_GeometryCollectionFromText()`, `GeomCollFromText()`, and `GeometryCollectionFromText()` are synonyms.

- `ST_GeomFromText(wkt [, srid])`, `ST_GeometryFromText(wkt [, srid])`

  Constructs a geometry value of any type using its WKT representation and SRID.

  The result is `NULL` if the geometry argument is `NULL` or not a syntactically well-formed geometry.

  `ST_GeomFromText()`, `ST_GeometryFromText()`, `GeomFromText()`, and `GeometryFromText()` are synonyms.

- `ST_LineFromText(wkt [, srid])`, `ST_LineStringFromText(wkt [, srid])`

  Constructs a `LineString` value using its WKT representation and SRID.

  The result is `NULL` if the geometry argument is `NULL` or not a syntactically well-formed geometry.

  `ST_LineFromText()`, `ST_LineStringFromText()`, `LineFromText()`, and `LineStringFromText()` are synonyms.

- `ST_PointFromText(wkt [, srid])`

  Constructs a `Point` value using its WKT representation and SRID.

  The result is `NULL` if the geometry argument is `NULL` or not a syntactically well-formed geometry.

  `ST_PointFromText()` and `PointFromText()` are synonyms.

- `ST_PolyFromText(wkt [, srid])`, `ST_PolygonFromText(wkt [, srid])`

  Constructs a `Polygon` value using its WKT representation and SRID.

  The result is `NULL` if the geometry argument is `NULL` or not a syntactically well-formed geometry.

  `ST_PolyFromText()`, `ST_PolygonFromText()`, `PolyFromText()`, and `PolygonFromText()` are synonyms.

12.17.4 Functions That Create Geometry Values from WKB Values

These functions take as arguments a `BLOB` containing a Well-Known Binary (WKB) representation and, optionally, a spatial reference system identifier (SRID). They return the corresponding geometry.

`GeomFromWKB()` and `ST_GeomFromWKB()` accept a WKB value of any geometry type as their first argument. Other functions provide type-specific construction functions for construction of geometry values of each geometry type.

These functions also accept geometry objects as returned by the functions in Section 12.17.5, “MySQL-Specific Functions That Create Geometry Values”. Thus, those functions may be used to provide the first argument to the functions in this section.

For a description of WKB format, see Well-Known Binary (WKB) Format.

- `GeomCollFromWKB(wkb [, srid])`, `GeometryCollectionFromWKB(wkb [, srid])`

`ST_GeomCollFromWKB()`, `ST_GeometryCollectionFromWKB()`, `GeomCollFromWKB()`, and `GeometryCollectionFromWKB()` are synonyms. For more information, see the description of `ST_GeomCollFromWKB()`.

- `GeomFromWKB(wkb [, srid])`, `GeometryFromWKB(wkb [, srid])`

  `ST_GeomFromWKB()`, `ST_GeometryFromWKB()`, `GeomFromWKB()`, and `GeometryFromWKB()` are synonyms. For more information, see the description of `ST_GeomFromWKB()`.

- `LineFromWKB(wkb [, srid])`, `LineStringFromWKB(wkb [, srid])`

  `ST_LineFromWKB()`, `ST_LineStringFromWKB()`, `LineFromWKB()`, and `LineStringFromWKB()` are synonyms. For more information, see the description of `ST_LineFromWKB()`.

- `MLineFromWKB(wkb [, srid])`, `MultiLineStringFromWKB(wkb [, srid])`

  Constructs a `MultiLineString` value using its WKB representation and SRID.

  The result is `NULL` if the WKB or SRID argument is `NULL`.

- `MPointFromWKB(wkb [, srid])`, `MultiPointFromWKB(wkb [, srid])`

  Constructs a `MultiPoint` value using its WKB representation and SRID.

  The result is `NULL` if the WKB or SRID argument is `NULL`.

- `MPolyFromWKB(wkb [, srid])`, `MultiPolygonFromWKB(wkb [, srid])`

  Constructs a `MultiPolygon` value using its WKB representation and SRID.

  The result is `NULL` if the WKB or SRID argument is `NULL`.

- `PointFromWKB(wkb [, srid])`

  `ST_PointFromWKB()` and `PointFromWKB()` are synonyms. For more information, see the description of `ST_PointFromWKB()`.

- `PolyFromWKB(wkb [, srid])`, `PolygonFromWKB(wkb [, srid])`

  `ST_PolyFromWKB()`, `ST_PolygonFromWKB()`, `PolyFromWKB()`, and `PolygonFromWKB()` are synonyms. For more information, see the description of `ST_PolyFromWKB()`.

- `ST_GeomCollFromWKB(wkb [, srid])`, `ST_GeometryCollectionFromWKB(wkb [, srid])`

  Constructs a `GeometryCollection` value using its WKB representation and SRID.

  The result is `NULL` if the WKB or SRID argument is `NULL`.

  `ST_GeomCollFromWKB()`, `ST_GeometryCollectionFromWKB()`, `GeomCollFromWKB()`, and `GeometryCollectionFromWKB()` are synonyms.

- `ST_GeomFromWKB(wkb [, srid])`, `ST_GeometryFromWKB(wkb [, srid])`

  Constructs a geometry value of any type using its WKB representation and SRID.

  The result is `NULL` if the WKB or SRID argument is `NULL`.

  `ST_GeomFromWKB()`, `ST_GeometryFromWKB()`, `GeomFromWKB()`, and `GeometryFromWKB()` are synonyms.

- `ST_LineFromWKB(wkb [, srid])`, `ST_LineStringFromWKB(wkb [, srid])`

Constructs a `LineString` value using its WKB representation and SRID.

The result is `NULL` if the WKB or SRID argument is `NULL`.

`ST_LineFromWKB()`, `ST_LineStringFromWKB()`, `LineFromWKB()`, and `LineStringFromWKB()` are synonyms.

- `ST_PointFromWKB(wkb [, srid])`

  Constructs a `Point` value using its WKB representation and SRID.

  The result is `NULL` if the WKB or SRID argument is `NULL`.

  `ST_PointFromWKB()` and `PointFromWKB()` are synonyms.

- `ST_PolyFromWKB(wkb [, srid])`, `ST_PolygonFromWKB(wkb [, srid])`

  Constructs a `Polygon` value using its WKB representation and SRID.

  The result is `NULL` if the WKB or SRID argument is `NULL`.

  `ST_PolyFromWKB()`, `ST_PolygonFromWKB()`, `PolyFromWKB()`, and `PolygonFromWKB()` are synonyms.

12.17.5 MySQL-Specific Functions That Create Geometry Values

MySQL provides a set of useful nonstandard functions for creating geometry values. The functions described in this section are MySQL extensions to the OpenGIS specification.

These functions produce geometry objects from either WKB values or geometry objects as arguments. If any argument is not a proper WKB or geometry representation of the proper object type, the return value is `NULL`.

For example, you can insert the geometry return value from `Point()` directly into a `POINT` column:

```
INSERT INTO t1 (pt_col) VALUES(Point(1,2));
```

- `GeometryCollection(g [, g] ...)`

  Constructs a `GeometryCollection` value from the geometry arguments.

  If an argument contains a nonsupported geometry, the return value is `NULL`.

- `LineString(pt [, pt] ...)`

  Constructs a `LineString` value from a number of `Point` or WKB `Point` arguments. If the number of arguments is less than two, the return value is `NULL`.

- `MultiLineString(ls [, ls] ...)`

  Constructs a `MultiLineString` value using `LineString` or WKB `LineString` arguments.

- `MultiPoint(pt [, pt2] ...)`

  Constructs a `MultiPoint` value using `Point` or WKB `Point` arguments.

- `MultiPolygon(poly [, poly] ...)`

  Constructs a `MultiPolygon` value from a set of `Polygon` or WKB `Polygon` arguments.

- `Point(x, y)`

  Constructs a `Point` using its coordinates.

- `Polygon(ls [, ls] ...)`

  Constructs a `Polygon` value from a number of `LineString` or WKB `LineString` arguments. If any argument does not represent a `LinearRing` (that is, not a closed and simple `LineString`), the return value is `NULL`.

12.17.6 Geometry Format Conversion Functions

MySQL supports the functions listed in this section for converting geometry values from internal geometry format to WKT or WKB format.

There are also functions to convert a string from WKT or WKB format to internal geometry format. See Section 12.17.3, “Functions That Create Geometry Values from WKT Values”, and Section 12.17.4, “Functions That Create Geometry Values from WKB Values”.

- `AsBinary(g)`, `AsWKB(g)`

  `ST_AsBinary()`, `ST_AsWKB()`, `AsBinary()`, and `AsWKB()` are synonyms. For more information, see the description of `ST_AsBinary()`.

- `AsText(g)`, `AsWKT(g)`

  `ST_AsText()`, `ST_AsWKT()`, `AsText()`, and `AsWKT()` are synonyms. For more information, see the description of `ST_AsText()`.

- `ST_AsBinary(g)`, `ST_AsWKB(g)`

  Converts a value in internal geometry format to its WKB representation and returns the binary result.

  The result is `NULL` if the geometry argument is `NULL` or not a syntactically well-formed geometry.

  ```
  SELECT ST_AsBinary(g) FROM geom;
  ```

  `ST_AsBinary()`, `ST_AsWKB()`, `AsBinary()`, and `AsWKB()` are synonyms.

- `ST_AsText(g)`, `ST_AsWKT(g)`

  Converts a value in internal geometry format to its WKT representation and returns the string result.

  The result is `NULL` if the geometry argument is `NULL` or not a syntactically well-formed geometry.

  ```
  mysql> SET @g = 'LineString(1 1,2 2,3 3)';
  mysql> SELECT ST_AsText(ST_GeomFromText(@g));
  +------------------------------+
  | ST_AsText(ST_GeomFromText(@g)) |
  +------------------------------+
  | LINESTRING(1 1,2 2,3 3)        |
  +------------------------------+
  ```

  `ST_AsText()`, `ST_AsWKT()`, `AsText()`, and `AsWKT()` are synonyms.

12.17.7 Geometry Property Functions

Each function that belongs to this group takes a geometry value as its argument and returns some quantitative or qualitative property of the geometry. Some functions restrict their argument type. Such functions return `NULL` if the argument is of an incorrect geometry type. For example, the `ST_Area()` polygon function returns `NULL` if the object type is neither `Polygon` nor `MultiPolygon`.

12.17.7.1 General Geometry Property Functions

The functions listed in this section do not restrict their argument and accept a geometry value of any type.

- `Dimension(g)`

`ST_Dimension()` and `Dimension()` are synonyms. For more information, see the description of `ST_Dimension()`.

- `Envelope(g)`

  `ST_Envelope()` and `Envelope()` are synonyms. For more information, see the description of `ST_Envelope()`.

- `GeometryType(g)`

  `ST_GeometryType()` and `GeometryType()` are synonyms. For more information, see the description of `ST_GeometryType()`.

- `IsEmpty(g)`

  `ST_IsEmpty()` and `IsEmpty()` are synonyms. For more information, see the description of `ST_IsEmpty()`.

- `IsSimple(g)`

  `ST_IsSimple()` and `IsSimple()` are synonyms. For more information, see the description of `ST_IsSimple()`.

- `SRID(g)`

  `ST_SRID()` and `SRID()` are synonyms. For more information, see the description of `ST_SRID()`.

- `ST_Dimension(g)`

  Returns the inherent dimension of the geometry value *g*, or `NULL` if the argument is `NULL`. The dimension can be −1, 0, 1, or 2. The meaning of these values is given in Section 11.4.2.2, “Geometry Class”.

```
mysql> SELECT ST_Dimension(ST_GeomFromText('LineString(1 1,2 2)'));
+------------------------------------------------------+
| ST_Dimension(ST_GeomFromText('LineString(1 1,2 2)')) |
+------------------------------------------------------+
|                                                    1 |
+------------------------------------------------------+
```

  `ST_Dimension()` and `Dimension()` are synonyms.

- `ST_Envelope(g)`

  Returns the minimum bounding rectangle (MBR) for the geometry value *g*, or `NULL` if the argument is `NULL`. The result is returned as a `Polygon` value that is defined by the corner points of the bounding box:

```
POLYGON((MINX MINY, MAXX MINY, MAXX MAXY, MINX MAXY, MINX MINY))
```

```
mysql> SELECT ST_AsText(ST_Envelope(ST_GeomFromText('LineString(1 1,2 2)')));
+---------------------------------------------------------------+
| ST_AsText(ST_Envelope(ST_GeomFromText('LineString(1 1,2 2)'))) |
+---------------------------------------------------------------+
| POLYGON((1 1,2 1,2 2,1 2,1 1))                                |
+---------------------------------------------------------------+
```

  `ST_Envelope()` and `Envelope()` are synonyms.

- `ST_GeometryType(g)`

  Returns a binary string indicating the name of the geometry type of which the geometry instance *g* is a member, or `NULL` if the argument is `NULL`. The name corresponds to one of the instantiable `Geometry` subclasses.

```
mysql> SELECT ST_GeometryType(ST_GeomFromText('POINT(1 1)'));
+---------------------------------------------+
| ST_GeometryType(ST_GeomFromText('POINT(1 1)')) |
+---------------------------------------------+
| POINT                                       |
+---------------------------------------------+
```

`ST_GeometryType()` and `GeometryType()` are synonyms.

- `ST_IsEmpty(g)`

  This function is a placeholder that returns 0 for any valid geometry value, 1 for any invalid geometry value, or `NULL` if the argument is `NULL`.

  MySQL does not support GIS `EMPTY` values such as `POINT EMPTY`.

  `ST_IsEmpty()` and `IsEmpty()` are synonyms.

- `ST_IsSimple(g)`

  Returns 1 if the geometry value *g* has no anomalous geometric points, such as self-intersection or self-tangency. `ST_IsSimple()` returns 0 if the argument is not simple, and `NULL` if the argument is `NULL`.

  The descriptions of the instantiable geometric classes given under Section 11.4.2, “The OpenGIS Geometry Model” includes the specific conditions that cause class instances to be classified as not simple.

  `ST_IsSimple()` and `IsSimple()` are synonyms.

- `ST_SRID(g)`

  Returns an integer indicating the spatial reference system ID associated with the geometry value *g*, or `NULL` if the argument is `NULL`.

```
mysql> SELECT ST_SRID(ST_GeomFromText('LineString(1 1,2 2)',101));
+-----------------------------------------------------+
| ST_SRID(ST_GeomFromText('LineString(1 1,2 2)',101)) |
+-----------------------------------------------------+
|                                                 101 |
+-----------------------------------------------------+
```

`ST_SRID()` and `SRID()` are synonyms.

12.17.7.2 Point Property Functions

A `Point` consists of X and Y coordinates, which may be obtained using the following functions:

- `ST_X(p)`

  Returns the X-coordinate value for the `Point` object *p* as a double-precision number.

```
mysql> SELECT ST_X(Point(56.7, 53.34));
+--------------------------+
| ST_X(Point(56.7, 53.34)) |
+--------------------------+
|                     56.7 |
+--------------------------+
```

`ST_X()` and `X()` are synonyms.

- `ST_Y(p)`

  Returns the Y-coordinate value for the `Point` object *p* as a double-precision number.

```
mysql> SELECT ST_Y(Point(56.7, 53.34));
+--------------------------+
| ST_Y(Point(56.7, 53.34)) |
+--------------------------+
|                    53.34 |
+--------------------------+
```

`ST_Y()` and `Y()` are synonyms.

- `X(p)`

  `ST_X()` and `X()` are synonyms. For more information, see the description of `ST_X()`.

- `Y(p)`

  `ST_Y()` and `Y()` are synonyms. For more information, see the description of `ST_Y()`.

12.17.7.3 LineString and MultiLineString Property Functions

A `LineString` consists of `Point` values. You can extract particular points of a `LineString`, count the number of points that it contains, or obtain its length.

Some functions in this section also work for `MultiLineString` values.

- `EndPoint(ls)`

  `ST_EndPoint()` and `EndPoint()` are synonyms. For more information, see the description of `ST_EndPoint()`.

- `GLength(ls)`

  Returns a double-precision number indicating the length of the `LineString` or `MultiLineString` value *`ls`* in its associated spatial reference. The length of a `MultiLineString` value is equal to the sum of the lengths of its elements. If the argument is `NULL` or an empty geometry, the return value is `NULL`.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT GLength(GeomFromText(@ls));
+---------------------------+
| GLength(GeomFromText(@ls)) |
+---------------------------+
|         2.8284271247461903 |
+---------------------------+

mysql> SET @mls = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';
mysql> SELECT GLength(GeomFromText(@mls));
+----------------------------+
| GLength(GeomFromText(@mls)) |
+----------------------------+
|           4.242640687119286 |
+----------------------------+
```

`GLength()` is a nonstandard name. It corresponds to the OpenGIS `Length()` function. (There is an existing SQL function `Length()` that calculates the length of string values.)

- `IsClosed(ls)`

  `ST_IsClosed()` and `IsClosed()` are synonyms. For more information, see the description of `ST_IsClosed()`.

- `NumPoints(ls)`

  `ST_NumPoints()` and `NumPoints()` are synonyms. For more information, see the description of `ST_NumPoints()`.

- `PointN(ls, N)`

`ST_PointN()` and `PointN()` are synonyms. For more information, see the description of `ST_PointN()`.

- `ST_EndPoint(ls)`

  Returns the `Point` that is the endpoint of the `LineString` value *`ls`*. If the argument is `NULL` or an empty geometry, the return value is `NULL`.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT ST_AsText(ST_EndPoint(ST_GeomFromText(@ls)));
+-------------------------------------------+
| ST_AsText(ST_EndPoint(ST_GeomFromText(@ls))) |
+-------------------------------------------+
| POINT(3 3)                                |
+-------------------------------------------+
```

  `ST_EndPoint()` and `EndPoint()` are synonyms.

- `ST_IsClosed(ls)`

  For a `LineString` value *`ls`*, `ST_IsClosed()` returns 1 if *`ls`* is closed (that is, its `ST_StartPoint()` and `ST_EndPoint()` values are the same). If the argument is `NULL` or an empty geometry, the return value is `NULL`.

  For a `MultiLineString` value *`ls`*, `ST_IsClosed()` returns 1 if *`ls`* is closed (that is, the `ST_StartPoint()` and `ST_EndPoint()` values are the same for each `LineString` in *`ls`*).

  `ST_IsClosed()` returns 0 if *`ls`* is not closed.

```
mysql> SET @ls1 = 'LineString(1 1,2 2,3 3,2 2)';
mysql> SET @ls2 = 'LineString(1 1,2 2,3 3,1 1)';

mysql> SELECT ST_IsClosed(ST_GeomFromText(@ls1));
+-----------------------------------+
| ST_IsClosed(ST_GeomFromText(@ls1)) |
+-----------------------------------+
|                                 0 |
+-----------------------------------+

mysql> SELECT ST_IsClosed(ST_GeomFromText(@ls2));
+-----------------------------------+
| ST_IsClosed(ST_GeomFromText(@ls2)) |
+-----------------------------------+
|                                 1 |
+-----------------------------------+

mysql> SET @ls3 = 'MultiLineString((1 1,2 2,3 3),(4 4,5 5))';

mysql> SELECT ST_IsClosed(ST_GeomFromText(@ls3));
+-----------------------------------+
| ST_IsClosed(ST_GeomFromText(@ls3)) |
+-----------------------------------+
|                                 0 |
+-----------------------------------+
```

  `ST_IsClosed()` and `IsClosed()` are synonyms.

- `ST_NumPoints(ls)`

  Returns the number of `Point` objects in the `LineString` value *`ls`*. If the argument is `NULL` or an empty geometry, the return value is `NULL`.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT ST_NumPoints(ST_GeomFromText(@ls));
+-----------------------------------+
| ST_NumPoints(ST_GeomFromText(@ls)) |
+-----------------------------------+
```

```
|                                   3 |
+-------------------------------------+
```

`ST_NumPoints()` and `NumPoints()` are synonyms.

- `ST_PointN(ls, N)`

  Returns the *N*-th `Point` in the `Linestring` value *ls*. Points are numbered beginning with 1. If any argument is `NULL` or the geometry argument is an empty geometry, the return value is `NULL`.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT ST_AsText(ST_PointN(ST_GeomFromText(@ls),2));
+---------------------------------------------+
| ST_AsText(ST_PointN(ST_GeomFromText(@ls),2)) |
+---------------------------------------------+
| POINT(2 2)                                  |
+---------------------------------------------+
```

  `ST_PointN()` and `PointN()` are synonyms.

- `ST_StartPoint(ls)`

  Returns the `Point` that is the start point of the `LineString` value *ls*. If the argument is `NULL` or an empty geometry, the return value is `NULL`.

```
mysql> SET @ls = 'LineString(1 1,2 2,3 3)';
mysql> SELECT ST_AsText(ST_StartPoint(ST_GeomFromText(@ls)));
+-----------------------------------------------+
| ST_AsText(ST_StartPoint(ST_GeomFromText(@ls))) |
+-----------------------------------------------+
| POINT(1 1)                                    |
+-----------------------------------------------+
```

  `ST_StartPoint()` and `StartPoint()` are synonyms.

- `StartPoint(ls)`

  `ST_StartPoint()` and `StartPoint()` are synonyms. For more information, see the description of `ST_StartPoint()`.

12.17.7.4 Polygon and MultiPolygon Property Functions

Functions in this section return properties of `Polygon` or `MultiPolygon` values.

- `Area({poly|mpoly})`

  `ST_Area()` and `Area()` are synonyms. For more information, see the description of `ST_Area()`.

- `Centroid({poly|mpoly})`

  `ST_Centroid()` and `Centroid()` are synonyms. For more information, see the description of `ST_Centroid()`.

- `ExteriorRing(poly)`

  `ST_ExteriorRing()` and `ExteriorRing()` are synonyms. For more information, see the description of `ST_ExteriorRing()`.

- `InteriorRingN(poly, N)`

  `ST_InteriorRingN()` and `InteriorRingN()` are synonyms. For more information, see the description of `ST_InteriorRingN()`.

- `NumInteriorRings(poly)`

`ST_NumInteriorRings()` and `NumInteriorRings()` are synonyms. For more information, see the description of `ST_NumInteriorRings()`.

- `ST_Area({poly|mpoly})`

  Returns a double-precision number indicating the area of the `Polygon` or `MultiPolygon` argument, as measured in its spatial reference system. For arguments of dimension 0 or 1, the result is 0. If the argument is an empty geometry the return value is 0. If the argument is `NULL` the return value is `NULL`.

```
mysql> SET @poly =
       'Polygon((0 0,0 3,3 0,0 0),(1 1,1 2,2 1,1 1))';
mysql> SELECT ST_Area(ST_GeomFromText(@poly));
+---------------------------------+
| ST_Area(ST_GeomFromText(@poly)) |
+---------------------------------+
|                               4 |
+---------------------------------+

mysql> SET @mpoly =
       'MultiPolygon(((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1)))';
mysql> SELECT ST_Area(ST_GeomFromText(@mpoly));
+----------------------------------+
| ST_Area(ST_GeomFromText(@mpoly)) |
+----------------------------------+
|                                8 |
+----------------------------------+
```

  `ST_Area()` and `Area()` are synonyms.

- `ST_Centroid({poly|mpoly})`

  Returns the mathematical centroid for the `Polygon` or `MultiPolygon` argument as a `Point`. The result is not guaranteed to be on the `MultiPolygon`. If the argument is `NULL` or an empty geometry, the return value is `NULL`.

```
mysql> SET @poly =
       ST_GeomFromText('POLYGON((0 0,10 0,10 10,0 10,0 0),(5 5,7 5,7 7,5 7,5 5))');
mysql> SELECT GeometryType(@poly),ST_AsText(ST_Centroid(@poly));
+--------------------+-----------------------------------------+
| GeometryType(@poly) | ST_AsText(ST_Centroid(@poly))           |
+--------------------+-----------------------------------------+
| POLYGON            | POINT(4.958333333333333 4.958333333333333) |
+--------------------+-----------------------------------------+
```

  `ST_Centroid()` and `Centroid()` are synonyms.

- `ST_ExteriorRing(poly)`

  Returns the exterior ring of the `Polygon` value *`poly`* as a `LineString`. If the argument is `NULL` or an empty geometry, the return value is `NULL`.

```
mysql> SET @poly =
       'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT ST_AsText(ST_ExteriorRing(ST_GeomFromText(@poly)));
+-------------------------------------------------+
| ST_AsText(ST_ExteriorRing(ST_GeomFromText(@poly))) |
+-------------------------------------------------+
| LINESTRING(0 0,0 3,3 3,3 0,0 0)                 |
+-------------------------------------------------+
```

  `ST_ExteriorRing()` and `ExteriorRing()` are synonyms.

- `ST_InteriorRingN(poly, N)`

  Returns the *N*-th interior ring for the `Polygon` value *poly* as a `LineString`. Rings are numbered beginning with 1. If the argument is `NULL` or an empty geometry, the return value is `NULL`.

```
mysql> SET @poly =
       'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT ST_AsText(ST_InteriorRingN(ST_GeomFromText(@poly),1));
+------------------------------------------------------+
| ST_AsText(ST_InteriorRingN(ST_GeomFromText(@poly),1)) |
+------------------------------------------------------+
| LINESTRING(1 1,1 2,2 2,2 1,1 1)                      |
+------------------------------------------------------+
```

  `ST_InteriorRingN()` and `InteriorRingN()` are synonyms.

- `ST_NumInteriorRings(poly)`

  Returns the number of interior rings in the `Polygon` value *poly*. If the argument is `NULL` or an empty geometry, the return value is `NULL`.

```
mysql> SET @poly =
       'Polygon((0 0,0 3,3 3,3 0,0 0),(1 1,1 2,2 2,2 1,1 1))';
mysql> SELECT ST_NumInteriorRings(ST_GeomFromText(@poly));
+---------------------------------------------+
| ST_NumInteriorRings(ST_GeomFromText(@poly)) |
+---------------------------------------------+
|                                           1 |
+---------------------------------------------+
```

  `ST_NumInteriorRings()` and `NumInteriorRings()` are synonyms.

12.17.7.5 GeometryCollection Property Functions

These functions return properties of `GeometryCollection` values.

- `GeometryN(gc, N)`

  `ST_GeometryN()` and `GeometryN()` are synonyms. For more information, see the description of `ST_GeometryN()`.

- `NumGeometries(gc)`

  `ST_NumGeometries()` and `NumGeometries()` are synonyms. For more information, see the description of `ST_NumGeometries()`.

- `ST_GeometryN(gc, N)`

  Returns the *N*-th geometry in the `GeometryCollection` value *gc*. Geometries are numbered beginning with 1. If any argument is `NULL` or the geometry argument is an empty geometry, the return value is `NULL`.

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';
mysql> SELECT ST_AsText(ST_GeometryN(ST_GeomFromText(@gc),1));
+-----------------------------------------------+
| ST_AsText(ST_GeometryN(ST_GeomFromText(@gc),1)) |
+-----------------------------------------------+
| POINT(1 1)                                    |
+-----------------------------------------------+
```

  `ST_GeometryN()` and `GeometryN()` are synonyms.

- `ST_NumGeometries(gc)`

  Returns the number of geometries in the `GeometryCollection` value *gc*. If the argument is `NULL` or an empty geometry, the return value is `NULL`.

```
mysql> SET @gc = 'GeometryCollection(Point(1 1),LineString(2 2, 3 3))';
mysql> SELECT ST_NumGeometries(ST_GeomFromText(@gc));
+---------------------------------------+
| ST_NumGeometries(ST_GeomFromText(@gc)) |
+---------------------------------------+
|                                     2 |
+---------------------------------------+
```

`ST_NumGeometries()` and `NumGeometries()` are synonyms.

12.17.8 Spatial Operator Functions

OpenGIS proposes a number of functions that can produce geometries. They are designed to implement spatial operators.

- `Buffer(g, d)`

  `ST_Buffer()` and `Buffer()` are synonyms. For more information, see the description of `ST_Buffer()`.

- `ST_Buffer(g, d)`

  Returns a geometry that represents all points whose distance from the geometry value *g* is less than or equal to a distance of *d*, or `NULL` if any argument is `NULL`.

  `ST_Buffer()` supports negative distances for polygons, multipolygons, and geometry collections containing polygons or multipolygons. For point, multipoint, linestring, multilinestring, and geometry collections not containing any polygons or multipolygons, `ST_Buffer()` with a negative distance returns `NULL`.

  `ST_Buffer()` and `Buffer()` are synonyms.

- `ST_Difference(g1, g2)`

  Returns a geometry that represents the point set difference of the geometry values *g1* and *g2*. If any argument is `NULL`, the return value is `NULL`.

```
mysql> SET @g1 = Point(1,1), @g2 = Point(2,2);
mysql> SELECT ST_AsText(ST_Difference(@g1, @g2));
+-----------------------------------+
| ST_AsText(ST_Difference(@g1, @g2)) |
+-----------------------------------+
| POINT(1 1)                        |
+-----------------------------------+
```

- `ST_Intersection(g1, g2)`

  Returns a geometry that represents the point set intersection of the geometry values *g1* and *g2*. If any argument is `NULL`, the return value is `NULL`.

```
mysql> SET @g1 = ST_GeomFromText('LineString(1 1, 3 3)');
mysql> SET @g2 = ST_GeomFromText('LineString(1 3, 3 1)');
mysql> SELECT ST_AsText(ST_Intersection(@g1, @g2));
+-------------------------------------+
| ST_AsText(ST_Intersection(@g1, @g2)) |
+-------------------------------------+
| POINT(2 2)                          |
+-------------------------------------+
```

- `ST_SymDifference(g1, g2)`

  Returns a geometry that represents the point set symmetric difference of the geometry values *g1* and *g2*, which is defined as:

```
g1 symdifference g2 := (g1 union g2) difference (g1 intersection g2)
```

Or, in function call notation:

```
ST_SymDifference(g1, g2) = ST_Difference(ST_Union(g1, g2), ST_Intersection(g1, g2))
```

If any argument is NULL, the return value is NULL.

```
mysql> SET @g1 = Point(1,1), @g2 = Point(2,2);
mysql> SELECT ST_AsText(ST_SymDifference(@g1, @g2));
+--------------------------------------+
| ST_AsText(ST_SymDifference(@g1, @g2)) |
+--------------------------------------+
| MULTIPOINT(1 1,2 2)                  |
+--------------------------------------+
```

- `ST_Union(g1, g2)`

  Returns a geometry that represents the point set union of the geometry values *g1* and *g2*. If any argument is NULL, the return value is NULL.

```
mysql> SET @g1 = ST_GeomFromText('LineString(1 1, 3 3)');
mysql> SET @g2 = ST_GeomFromText('LineString(1 3, 3 1)');
mysql> SELECT ST_AsText(ST_Union(@g1, @g2));
+-------------------------------------+
| ST_AsText(ST_Union(@g1, @g2))       |
+-------------------------------------+
| MULTILINESTRING((1 1,3 3),(3 1,1 3)) |
+-------------------------------------+
```

In addition, Section 12.17.7, “Geometry Property Functions”, discusses several functions that construct new geometries from existing ones. See that section for descriptions of these functions:

- `ST_Envelope(g)`
- `StartPoint(ls)`
- `ST_EndPoint(ls)`
- `ST_PointN(ls, N)`
- `ST_ExteriorRing(poly)`
- `ST_InteriorRingN(poly, N)`
- `ST_GeometryN(gc, N)`

12.17.9 Functions That Test Spatial Relations Between Geometry Objects

The functions described in this section take two geometries as arguments and return a qualitative or quantitative relation between them.

MySQL implements two sets of functions using function names defined by the OpenGIS specification. One set tests the relationship between two geometry values using precise object shapes, the other set uses object minimum bounding rectangles (MBRs).

There is also a MySQL-specific set of MBR-based functions available to test the relationship between two geometry values.

12.17.9.1 Spatial Relation Functions That Use Object Shapes

The OpenGIS specification defines the following functions to test the relationship between two geometry values *g1* and *g2*, using precise object shapes. The return values 1 and 0 indicate true and false, respectively, except for `ST_Distance()`, which returns distance values.

- `ST_Contains(g1, g2)`

Returns 1 or 0 to indicate whether *g1* completely contains *g2*. This tests the opposite relationship as `ST_Within()`.

- `Crosses(g1, g2)`

  `ST_Crosses()` and `Crosses()` are synonyms. For more information, see the description of `ST_Crosses()`.

- `ST_Crosses(g1, g2)`

  The term *spatially crosses* denotes a spatial relation between two given geometries that has the following properties:

  - The two geometries intersect.
  - Their intersection results in a geometry that has a dimension that is one less than the maximum dimension of the two given geometries.
  - Their intersection is not equal to either of the two given geometries.

  This function returns 1 or 0 to indicate whether *g1* spatially crosses *g2*. If *g1* is a `Polygon` or a `MultiPolygon`, or if *g2* is a `Point` or a `MultiPoint`, the return value is `NULL`.

  `ST_Crosses()` and `Crosses()` are synonyms.

- `ST_Disjoint(g1, g2)`

  Returns 1 or 0 to indicate whether *g1* is spatially disjoint from (does not intersect) *g2*.

- `ST_Distance(g1, g2)`

  Returns the distance between *g1* and *g2*.

```
mysql> SET @g1 = Point(1,1);
mysql> SET @g2 = Point(2,2);
mysql> SELECT ST_Distance(@g1, @g2);
+-----------------------+
| ST_Distance(@g1, @g2) |
+-----------------------+
|    1.4142135623730951 |
+-----------------------+
```

- `ST_Equals(g1, g2)`

  Returns 1 or 0 to indicate whether *g1* is spatially equal to *g2*.

```
mysql> SET @g1 = Point(1,1), @g2 = Point(2,2);
mysql> SELECT ST_Equals(@g1, @g1), ST_Equals(@g1, @g2);
+---------------------+---------------------+
| ST_Equals(@g1, @g1) | ST_Equals(@g1, @g2) |
+---------------------+---------------------+
|                   1 |                   0 |
+---------------------+---------------------+
```

- `ST_Intersects(g1, g2)`

  Returns 1 or 0 to indicate whether *g1* spatially intersects *g2*.

- `ST_Overlaps(g1, g2)`

  Two geometries *spatially overlap* if they intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

  This function returns 1 or 0 to indicate whether *g1* spatially overlaps *g2*.

- `ST_Touches(g1, g2)`

  Two geometries *spatially touch* if their interiors do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

  This function returns 1 or 0 to indicate whether *g1* spatially touches *g2*.

  `ST_Touches()` and `Touches()` are synonyms.

- `ST_Within(g1, g2)`

  Returns 1 or 0 to indicate whether *g1* is spatially within *g2*. This tests the opposite relationship as `ST_Contains()`.

- `Touches(g1, g2)`

  `ST_Touches()` and `Touches()` are synonyms. For more information, see the description of `ST_Touches()`.

12.17.9.2 Spatial Relation Functions That Use Minimum Bounding Rectangles

MySQL provides several MySQL-specific functions that test the relationship between minimum bounding rectangles (MBRs) of two geometries *g1* and *g2*. The return values 1 and 0 indicate true and false, respectively.

A corresponding set of MBR functions defined according to the OpenGIS specification is described later in this section.

- `MBRContains(g1, g2)`

  Returns 1 or 0 to indicate whether the minimum bounding rectangle of *g1* contains the minimum bounding rectangle of *g2*. This tests the opposite relationship as `MBRWithin()`.

```
mysql> SET @g1 = ST_GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = ST_GeomFromText('Point(1 1)');
mysql> SELECT MBRContains(@g1,@g2), MBRWithin(@g2,@g1);
+----------------------+--------------------+
| MBRContains(@g1,@g2) | MBRWithin(@g2,@g1) |
+----------------------+--------------------+
|                    1 |                  1 |
+----------------------+--------------------+
```

  `MBRContains()` and `Contains()` are synonyms.

- `MBRDisjoint(g1, g2)`

  Returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries *g1* and *g2* are disjoint (do not intersect).

  `MBRDisjoint()` and `Disjoint()` are synonyms.

- `MBREqual(g1, g2)`

  Returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries *g1* and *g2* are the same.

- `MBRIntersects(g1, g2)`

  Returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries *g1* and *g2* intersect.

  `MBRIntersects()` and `Intersects()` are synonyms.

- `MBROverlaps(g1, g2)`

Two geometries *spatially overlap* if they intersect and their intersection results in a geometry of the same dimension but not equal to either of the given geometries.

This function returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries *g1* and *g2* overlap.

`MBROverlaps()` and `Overlaps()` are synonyms.

- `MBRTouches(g1, g2)`

  Two geometries *spatially touch* if their interiors do not intersect, but the boundary of one of the geometries intersects either the boundary or the interior of the other.

  This function returns 1 or 0 to indicate whether the minimum bounding rectangles of the two geometries *g1* and *g2* touch.

- `MBRWithin(g1, g2)`

  Returns 1 or 0 to indicate whether the minimum bounding rectangle of *g1* is within the minimum bounding rectangle of *g2*. This tests the opposite relationship as `MBRContains()`.

```
mysql> SET @g1 = ST_GeomFromText('Polygon((0 0,0 3,3 3,3 0,0 0))');
mysql> SET @g2 = ST_GeomFromText('Polygon((0 0,0 5,5 5,5 0,0 0))');
mysql> SELECT MBRWithin(@g1,@g2), MBRWithin(@g2,@g1);
+-------------------+-------------------+
| MBRWithin(@g1,@g2) | MBRWithin(@g2,@g1) |
+-------------------+-------------------+
|                 1 |                 0 |
+-------------------+-------------------+
```

  `MBRWithin()` and `Within()` are synonyms.

The OpenGIS specification defines the following functions that test the relationship between two geometry values *g1* and *g2*. The MySQL implementation uses minimum bounding rectangles, so these functions return the same result as the corresponding MBR-based functions described earlier in this section. The return values 1 and 0 indicate true and false, respectively.

- `Contains(g1, g2)`

  `MBRContains()` and `Contains()` are synonyms. For more information, see the description of `MBRContains()`.

- `Disjoint(g1, g2)`

  `MBRDisjoint()` and `Disjoint()` are synonyms. For more information, see the description of `MBRDisjoint()`.

- `Equals(g1, g2)`

  Returns 1 or 0 to indicate whether *g1* is spatially equal to *g2*.

- `Intersects(g1, g2)`

  `MBRIntersects()` and `Intersects()` are synonyms. For more information, see the description of `MBRIntersects()`.

- `Overlaps(g1, g2)`

  `MBROverlaps()` and `Overlaps()` are synonyms. For more information, see the description of `MBROverlaps()`.

- `Within(g1, g2)`

`MBRWithin()` and `Within()` are synonyms. For more information, see the description of `MBRWithin()`.

12.18 Functions Used with Global Transaction IDs

The functions described in this section are used with GTID-based replication. It is important to keep in mind that all of these functions take string representations of GTID sets as arguments—as such, the GTID sets must always be quoted when used with them.

The union of two GTID sets is simply their representations as strings, joined together with an interposed comma. In other words, you can define a very simple function for obtaining the union of two GTID sets, similar to that created here:

```
CREATE FUNCTION GTID_UNION(g1 TEXT, g2 TEXT)
    RETURNS TEXT DETERMINISTIC
    RETURN CONCAT(g1,',',g2);
```

For more information about GTIDs and how these GTID functions are used in practice, see Section 17.1.3, “Replication with Global Transaction Identifiers”.

Table 12.22 GTID Functions

Name	Description
`GTID_SUBSET()`	Return true if all GTIDs in subset are also in set; otherwise false.
`GTID_SUBTRACT()`	Return all GTIDs in set that are not in subset.
`SQL_THREAD_WAIT_AFTER_GTIDS()` (deprecated)	Obsolete. Use `WAIT_FOR_EXECUTED_GTID_SET()`.
`WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS()`	Use `WAIT_FOR_EXECUTED_GTID_SET()`.

- `GTID_SUBSET(subset,set)`

  Given two sets of global transaction IDs *`subset`* and *`set`*, returns true (1) if all GTIDs in *`subset`* are also in *`set`*. Returns false (0) otherwise.

  The GTID sets used with this function are represented as strings, as shown in the following examples:

```
mysql> SELECT GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:23',
    ->     '3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57')\G
*************************** 1. row ***************************
GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:23',
    '3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57'): 1
1 row in set (0.00 sec)

mysql> SELECT GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:23-25',
    ->     '3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57')\G
*************************** 1. row ***************************
GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:23-25',
    '3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57'): 1
1 row in set (0.00 sec)

mysql> SELECT GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:20-25',
    ->     '3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57')\G
*************************** 1. row ***************************
GTID_SUBSET('3E11FA47-71CA-11E1-9E33-C80AA9429562:20-25',
    '3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57'): 0
1 row in set (0.00 sec)
```

- `GTID_SUBTRACT(set,subset)`

  Given two sets of global transaction IDs *`subset`* and *`set`*, returns only those GTIDs from *`set`* that are not in *`subset`*.

All GTID sets used with this function are represented as strings and must be quoted, as shown in these examples:

```
mysql> SELECT GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
    ->     '3E11FA47-71CA-11E1-9E33-C80AA9429562:21')\G
*************************** 1. row ***************************
GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
    '3E11FA47-71CA-11E1-9E33-C80AA9429562:21'): 3e11fa47-71ca-11e1-9e33-c80aa9429562:22-57
1 row in set (0.00 sec)

mysql> SELECT GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
    ->     '3E11FA47-71CA-11E1-9E33-C80AA9429562:20-25')\G
*************************** 1. row ***************************
GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
    '3E11FA47-71CA-11E1-9E33-C80AA9429562:20-25'): 3e11fa47-71ca-11e1-9e33-c80aa9429562:26-57
1 row in set (0.00 sec)

mysql> SELECT GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
    ->     '3E11FA47-71CA-11E1-9E33-C80AA9429562:23-24')\G
*************************** 1. row ***************************
GTID_SUBTRACT('3E11FA47-71CA-11E1-9E33-C80AA9429562:21-57',
    '3E11FA47-71CA-11E1-9E33-C80AA9429562:23-24'): 3e11fa47-71ca-11e1-9e33-c80aa9429562:21-22:25-57
1 row in set (0.01 sec)
```

- `SQL_THREAD_WAIT_AFTER_GTIDS(gtid_set[, timeout])`

  `SQL_THREAD_WAIT_AFTER_GTIDS()` was added in MySQL 5.6.5, and replaced by `WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS()` in MySQL 5.6.9. (Bug #14775984)

  For more information, see Section 17.1.3, “Replication with Global Transaction Identifiers”.

- `WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS(gtid_set[, timeout])`

  Wait until the replica SQL thread has executed all of the transactions whose global transaction identifiers are contained in *`gtid_set`* (see Section 17.1.3.1, “GTID Concepts”, for a definition of “GTID sets”), or until *`timeout`* seconds have elapsed, whichever occurs first. *`timeout`* is optional; the default timeout is 0 seconds, in which case the source simply waits until all of the transactions in the GTID set have been executed.

  Prior to MySQL 5.6.9, `WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS()` was named `SQL_THREAD_WAIT_AFTER_GTIDS()`. (Bug #14775984)

  For more information, see Section 17.1.3, “Replication with Global Transaction Identifiers”.

  GTID sets used with this function are represented as strings and so must be quoted as shown in the following example:

  ```
  mysql> SELECT WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS('3E11FA47-71CA-11E1-9E33-C80AA9429562:1-5');
         -> 5
  ```

  The return value is the number of transactional events that were executed. Prior to MySQL 5.6.8, this function behaved unpredictably if no timeout was set and it was invoked while GTID-based replication was not active; in MySQL 5.6.8 and later, the function returns `NULL` whenever `gtid_mode` is `OFF`. (Bug #14640065)

12.19 Aggregate Functions

Aggregate functions operate on sets of values. They are often used with a `GROUP BY` clause to group values into subsets.

12.19.1 Aggregate Function Descriptions

This section describes aggregate functions that operate on sets of values. They are often used with a `GROUP BY` clause to group values into subsets.

Table 12.23 Aggregate Functions

Name	Description
AVG()	Return the average value of the argument
BIT_AND()	Return bitwise AND
BIT_OR()	Return bitwise OR
BIT_XOR()	Return bitwise XOR
COUNT()	Return a count of the number of rows returned
COUNT(DISTINCT)	Return the count of a number of different values
GROUP_CONCAT()	Return a concatenated string
MAX()	Return the maximum value
MIN()	Return the minimum value
STD()	Return the population standard deviation
STDDEV()	Return the population standard deviation
STDDEV_POP()	Return the population standard deviation
STDDEV_SAMP()	Return the sample standard deviation
SUM()	Return the sum
VAR_POP()	Return the population standard variance
VAR_SAMP()	Return the sample variance
VARIANCE()	Return the population standard variance

Unless otherwise stated, aggregate functions ignore NULL values.

If you use an aggregate function in a statement containing no GROUP BY clause, it is equivalent to grouping on all rows. For more information, see Section 12.19.3, “MySQL Handling of GROUP BY”.

For numeric arguments, the variance and standard deviation functions return a DOUBLE value. The SUM() and AVG() functions return a DECIMAL value for exact-value arguments (integer or DECIMAL), and a DOUBLE value for approximate-value arguments (FLOAT or DOUBLE).

The SUM() and AVG() aggregate functions do not work with temporal values. (They convert the values to numbers, losing everything after the first nonnumeric character.) To work around this problem, convert to numeric units, perform the aggregate operation, and convert back to a temporal value. Examples:

```
SELECT SEC_TO_TIME(SUM(TIME_TO_SEC(time_col))) FROM tbl_name;
SELECT FROM_DAYS(SUM(TO_DAYS(date_col))) FROM tbl_name;
```

Functions such as SUM() or AVG() that expect a numeric argument cast the argument to a number if necessary. For SET or ENUM values, the cast operation causes the underlying numeric value to be used.

The BIT_AND(), BIT_OR(), and BIT_XOR() aggregate functions perform bit operations. They require BIGINT (64-bit integer) arguments and return BIGINT values. Arguments of other types are converted to BIGINT and truncation might occur.

- AVG([DISTINCT] *expr*)

  Returns the average value of *expr*. The DISTINCT option can be used to return the average of the distinct values of *expr*.

  If there are no matching rows, AVG() returns NULL.

```
mysql> SELECT student_name, AVG(test_score)
```

```
       FROM student
       GROUP BY student_name;
```

- `BIT_AND(expr)`

  Returns the bitwise `AND` of all bits in *`expr`*. The calculation is performed with 64-bit (`BIGINT`) precision.

  If there are no matching rows, `BIT_AND()` returns a neutral value (all bits set to 1).

- `BIT_OR(expr)`

  Returns the bitwise `OR` of all bits in *`expr`*. The calculation is performed with 64-bit (`BIGINT`) precision.

  If there are no matching rows, `BIT_OR()` returns a neutral value (all bits set to 0).

- `BIT_XOR(expr)`

  Returns the bitwise `XOR` of all bits in *`expr`*. The calculation is performed with 64-bit (`BIGINT`) precision.

  If there are no matching rows, `BIT_XOR()` returns a neutral value (all bits set to 0).

- `COUNT(expr)`

  Returns a count of the number of non-`NULL` values of *`expr`* in the rows retrieved by a `SELECT` statement. The result is a `BIGINT` value.

  If there are no matching rows, `COUNT()` returns `0`.

```
mysql> SELECT student.student_name,COUNT(*)
       FROM student,course
       WHERE student.student_id=course.student_id
       GROUP BY student_name;
```

  `COUNT(*)` is somewhat different in that it returns a count of the number of rows retrieved, whether or not they contain `NULL` values.

  For transactional storage engines such as `InnoDB`, storing an exact row count is problematic. Multiple transactions may be occurring at the same time, each of which may affect the count.

  `InnoDB` does not keep an internal count of rows in a table because concurrent transactions might “see” different numbers of rows at the same time. Consequently, `SELECT COUNT(*)` statements only count rows visible to the current transaction.

  To process a `SELECT COUNT(*)` statement, `InnoDB` scans an index of the table, which takes some time if the index is not entirely in the buffer pool. For a faster count, create a counter table and let your application update it according to the inserts and deletes it does. However, this method may not scale well in situations where thousands of concurrent transactions are initiating updates to the same counter table. If an approximate row count is sufficient, use `SHOW TABLE STATUS`.

  `InnoDB` handles `SELECT COUNT(*)` and `SELECT COUNT(1)` operations in the same way. There is no performance difference.

  For `MyISAM` tables, `COUNT(*)` is optimized to return very quickly if the `SELECT` retrieves from one table, no other columns are retrieved, and there is no `WHERE` clause. For example:

```
mysql> SELECT COUNT(*) FROM student;
```

  This optimization only applies to `MyISAM` tables, because an exact row count is stored for this storage engine and can be accessed very quickly. `COUNT(1)` is only subject to the same optimization if the first column is defined as `NOT NULL`.

- `COUNT(DISTINCT expr,[expr...])`

  Returns a count of the number of rows with different non-`NULL` *`expr`* values.

  If there are no matching rows, `COUNT(DISTINCT)` returns `0`.

```
mysql> SELECT COUNT(DISTINCT results) FROM student;
```

  In MySQL, you can obtain the number of distinct expression combinations that do not contain `NULL` by giving a list of expressions. In standard SQL, you would have to do a concatenation of all expressions inside `COUNT(DISTINCT ...)`.

- `GROUP_CONCAT(expr)`

  This function returns a string result with the concatenated non-`NULL` values from a group. It returns `NULL` if there are no non-`NULL` values. The full syntax is as follows:

```
GROUP_CONCAT([DISTINCT] expr [,expr ...]
             [ORDER BY {unsigned_integer | col_name | expr}
                 [ASC | DESC] [,col_name ...]]
             [SEPARATOR str_val])
```

```
mysql> SELECT student_name,
         GROUP_CONCAT(test_score)
       FROM student
       GROUP BY student_name;
```

  Or:

```
mysql> SELECT student_name,
         GROUP_CONCAT(DISTINCT test_score
                      ORDER BY test_score DESC SEPARATOR ' ')
       FROM student
       GROUP BY student_name;
```

  In MySQL, you can get the concatenated values of expression combinations. To eliminate duplicate values, use the `DISTINCT` clause. To sort values in the result, use the `ORDER BY` clause. To sort in reverse order, add the `DESC` (descending) keyword to the name of the column you are sorting by in the `ORDER BY` clause. The default is ascending order; this may be specified explicitly using the `ASC` keyword. The default separator between values in a group is comma (`,`). To specify a separator explicitly, use `SEPARATOR` followed by the string literal value that should be inserted between group values. To eliminate the separator altogether, specify `SEPARATOR ''`.

  The result is truncated to the maximum length that is given by the `group_concat_max_len` system variable, which has a default value of 1024. The value can be set higher, although the effective maximum length of the return value is constrained by the value of `max_allowed_packet`. The syntax to change the value of `group_concat_max_len` at runtime is as follows, where *`val`* is an unsigned integer:

```
SET [GLOBAL | SESSION] group_concat_max_len = val;
```

  The return value is a nonbinary or binary string, depending on whether the arguments are nonbinary or binary strings. The result type is `TEXT` or `BLOB` unless `group_concat_max_len` is less than or equal to 512, in which case the result type is `VARCHAR` or `VARBINARY`.

  See also `CONCAT()` and `CONCAT_WS()`: Section 12.8, “String Functions and Operators”.

- `MAX([DISTINCT] expr)`

  Returns the maximum value of *`expr`*. `MAX()` may take a string argument; in such cases, it returns the maximum string value. See Section 8.3.1, “How MySQL Uses Indexes”. The `DISTINCT` keyword

can be used to find the maximum of the distinct values of *expr*, however, this produces the same result as omitting DISTINCT.

If there are no matching rows, MAX() returns NULL.

```
mysql> SELECT student_name, MIN(test_score), MAX(test_score)
       FROM student
       GROUP BY student_name;
```

For MAX(), MySQL currently compares ENUM and SET columns by their string value rather than by the string's relative position in the set. This differs from how ORDER BY compares them.

- MIN([DISTINCT] *expr*)

  Returns the minimum value of *expr*. MIN() may take a string argument; in such cases, it returns the minimum string value. See Section 8.3.1, “How MySQL Uses Indexes”. The DISTINCT keyword can be used to find the minimum of the distinct values of *expr*, however, this produces the same result as omitting DISTINCT.

  If there are no matching rows, MIN() returns NULL.

  ```
  mysql> SELECT student_name, MIN(test_score), MAX(test_score)
         FROM student
         GROUP BY student_name;
  ```

  For MIN(), MySQL currently compares ENUM and SET columns by their string value rather than by the string's relative position in the set. This differs from how ORDER BY compares them.

- STD(*expr*)

  Returns the population standard deviation of *expr*. STD() is a synonym for the standard SQL function STDDEV_POP(), provided as a MySQL extension.

  If there are no matching rows, STD() returns NULL.

- STDDEV(*expr*)

  Returns the population standard deviation of *expr*. STDDEV() is a synonym for the standard SQL function STDDEV_POP(), provided for compatibility with Oracle.

  If there are no matching rows, STDDEV() returns NULL.

- STDDEV_POP(*expr*)

  Returns the population standard deviation of *expr* (the square root of VAR_POP()). You can also use STD() or STDDEV(), which are equivalent but not standard SQL.

  If there are no matching rows, STDDEV_POP() returns NULL.

- STDDEV_SAMP(*expr*)

  Returns the sample standard deviation of *expr* (the square root of VAR_SAMP().

  If there are no matching rows, STDDEV_SAMP() returns NULL.

- SUM([DISTINCT] *expr*)

  Returns the sum of *expr*. If the return set has no rows, SUM() returns NULL. The DISTINCT keyword can be used to sum only the distinct values of *expr*.

  If there are no matching rows, SUM() returns NULL.

- `VAR_POP(expr)`

  Returns the population standard variance of *`expr`*. It considers rows as the whole population, not as a sample, so it has the number of rows as the denominator. You can also use `VARIANCE()`, which is equivalent but is not standard SQL.

  If there are no matching rows, `VAR_POP()` returns `NULL`.

- `VAR_SAMP(expr)`

  Returns the sample variance of *`expr`*. That is, the denominator is the number of rows minus one.

  If there are no matching rows, `VAR_SAMP()` returns `NULL`.

- `VARIANCE(expr)`

  Returns the population standard variance of *`expr`*. `VARIANCE()` is a synonym for the standard SQL function `VAR_POP()`, provided as a MySQL extension.

  If there are no matching rows, `VARIANCE()` returns `NULL`.

12.19.2 GROUP BY Modifiers

The `GROUP BY` clause permits a `WITH ROLLUP` modifier that causes summary output to include extra rows that represent higher-level (that is, super-aggregate) summary operations. `ROLLUP` thus enables you to answer questions at multiple levels of analysis with a single query. For example, `ROLLUP` can be used to provide support for OLAP (Online Analytical Processing) operations.

Suppose that a `sales` table has `year`, `country`, `product`, and `profit` columns for recording sales profitability:

```
CREATE TABLE sales
(
    year    INT,
    country VARCHAR(20),
    product VARCHAR(32),
    profit  INT
);
```

To summarize table contents per year, use a simple `GROUP BY` like this:

```
mysql> SELECT year, SUM(profit) AS profit
       FROM sales
       GROUP BY year;
+------+--------+
| year | profit |
+------+--------+
| 2000 |   4525 |
| 2001 |   3010 |
+------+--------+
```

The output shows the total (aggregate) profit for each year. To also determine the total profit summed over all years, you must add up the individual values yourself or run an additional query. Or you can use `ROLLUP`, which provides both levels of analysis with a single query. Adding a `WITH ROLLUP` modifier to the `GROUP BY` clause causes the query to produce another (super-aggregate) row that shows the grand total over all year values:

```
mysql> SELECT year, SUM(profit) AS profit
       FROM sales
       GROUP BY year WITH ROLLUP;
+------+--------+
| year | profit |
+------+--------+
| 2000 |   4525 |
| 2001 |   3010 |
| NULL |   7535 |
+------+--------+
```

The `NULL` value in the `year` column identifies the grand total super-aggregate line.

`ROLLUP` has a more complex effect when there are multiple `GROUP BY` columns. In this case, each time there is a change in value in any but the last grouping column, the query produces an extra super-aggregate summary row.

For example, without `ROLLUP`, a summary of the `sales` table based on `year`, `country`, and `product` might look like this, where the output indicates summary values only at the year/country/product level of analysis:

```
mysql> SELECT year, country, product, SUM(profit) AS profit
       FROM sales
       GROUP BY year, country, product;
+------+---------+------------+--------+
| year | country | product    | profit |
+------+---------+------------+--------+
| 2000 | Finland | Computer   |   1500 |
| 2000 | Finland | Phone      |    100 |
| 2000 | India   | Calculator |    150 |
| 2000 | India   | Computer   |   1200 |
| 2000 | USA     | Calculator |     75 |
| 2000 | USA     | Computer   |   1500 |
| 2001 | Finland | Phone      |     10 |
| 2001 | USA     | Calculator |     50 |
| 2001 | USA     | Computer   |   2700 |
| 2001 | USA     | TV         |    250 |
+------+---------+------------+--------+
```

With `ROLLUP` added, the query produces several extra rows:

```
mysql> SELECT year, country, product, SUM(profit) AS profit
       FROM sales
       GROUP BY year, country, product WITH ROLLUP;
+------+---------+------------+--------+
| year | country | product    | profit |
+------+---------+------------+--------+
| 2000 | Finland | Computer   |   1500 |
| 2000 | Finland | Phone      |    100 |
| 2000 | Finland | NULL       |   1600 |
| 2000 | India   | Calculator |    150 |
| 2000 | India   | Computer   |   1200 |
| 2000 | India   | NULL       |   1350 |
| 2000 | USA     | Calculator |     75 |
| 2000 | USA     | Computer   |   1500 |
| 2000 | USA     | NULL       |   1575 |
| 2000 | NULL    | NULL       |   4525 |
| 2001 | Finland | Phone      |     10 |
| 2001 | Finland | NULL       |     10 |
| 2001 | USA     | Calculator |     50 |
| 2001 | USA     | Computer   |   2700 |
| 2001 | USA     | TV         |    250 |
| 2001 | USA     | NULL       |   3000 |
| 2001 | NULL    | NULL       |   3010 |
| NULL | NULL    | NULL       |   7535 |
+------+---------+------------+--------+
```

Now the output includes summary information at four levels of analysis, not just one:

- Following each set of product rows for a given year and country, an extra super-aggregate summary row appears showing the total for all products. These rows have the `product` column set to `NULL`.
- Following each set of rows for a given year, an extra super-aggregate summary row appears showing the total for all countries and products. These rows have the `country` and `products` columns set to `NULL`.
- Finally, following all other rows, an extra super-aggregate summary row appears showing the grand total for all years, countries, and products. This row has the `year`, `country`, and `products` columns set to `NULL`.

The NULL indicators in each super-aggregate row are produced when the row is sent to the client. The server looks at the columns named in the GROUP BY clause following the leftmost one that has changed value. For any column in the result set with a name that matches any of those names, its value is set to NULL. (If you specify grouping columns by column position, the server identifies which columns to set to NULL by position.)

Because the NULL values in the super-aggregate rows are placed into the result set at such a late stage in query processing, you can test them as NULL values only in the select list or HAVING clause. You cannot test them as NULL values in join conditions or the WHERE clause to determine which rows to select. For example, you cannot add WHERE product IS NULL to the query to eliminate from the output all but the super-aggregate rows.

The NULL values do appear as NULL on the client side and can be tested as such using any MySQL client programming interface. However, at this point, you cannot distinguish whether a NULL represents a regular grouped value or a super-aggregate value.

Other Considerations When using ROLLUP

The following discussion lists some behaviors specific to the MySQL implementation of ROLLUP.

When you use ROLLUP, you cannot also use an ORDER BY clause to sort the results. In other words, ROLLUP and ORDER BY are mutually exclusive in MySQL. However, you still have some control over sort order. GROUP BY in MySQL implicitly sorts results by default (in the absence of ASC or DESC designators). However, implicit GROUP BY sorting in MySQL is deprecated. To achieve a specific sort order of grouped results:

- Use explicit ASC and DESC keywords with columns named in the GROUP BY list to specify sort order for individual columns. In this case, the super-aggregate summary rows added by ROLLUP still appear after the rows from which they are calculated, regardless of the sort order.
- To work around the restriction that prevents using ROLLUP with ORDER BY, generate the grouped result set as a derived table and apply ORDER BY to it. For example:

```
mysql> SELECT * FROM
         (SELECT year, SUM(profit) AS profit
         FROM sales GROUP BY year WITH ROLLUP) AS dt
       ORDER BY year DESC;
+------+--------+
| year | profit |
+------+--------+
| 2001 |   3010 |
| 2000 |   4525 |
| NULL |   7535 |
+------+--------+
```

  In this case, the super-aggregate summary rows sort with the rows from which they are calculated, and their placement depends on sort order (at the beginning for ascending sort, at the end for descending sort).

LIMIT can be used to restrict the number of rows returned to the client. LIMIT is applied after ROLLUP, so the limit applies against the extra rows added by ROLLUP. For example:

```
mysql> SELECT year, country, product, SUM(profit) AS profit
       FROM sales
       GROUP BY year, country, product WITH ROLLUP
       LIMIT 5;
+------+---------+------------+--------+
| year | country | product    | profit |
+------+---------+------------+--------+
| 2000 | Finland | Computer   |   1500 |
| 2000 | Finland | Phone      |    100 |
| 2000 | Finland | NULL       |   1600 |
| 2000 | India   | Calculator |    150 |
| 2000 | India   | Computer   |   1200 |
+------+---------+------------+--------+
```

Using `LIMIT` with `ROLLUP` may produce results that are more difficult to interpret, because there is less context for understanding the super-aggregate rows.

A MySQL extension permits a column that does not appear in the `GROUP BY` list to be named in the select list. (For information about nonaggregated columns and `GROUP BY`, see Section 12.19.3, "MySQL Handling of GROUP BY".) In this case, the server is free to choose any value from this nonaggregated column in summary rows, and this includes the extra rows added by `WITH ROLLUP`. For example, in the following query, `country` is a nonaggregated column that does not appear in the `GROUP BY` list and values chosen for this column are nondeterministic:

```
mysql> SELECT year, country, SUM(profit) AS profit
       FROM sales
       GROUP BY year WITH ROLLUP;
+------+---------+--------+
| year | country | profit |
+------+---------+--------+
| 2000 | India   |   4525 |
| 2001 | USA     |   3010 |
| NULL | USA     |   7535 |
+------+---------+--------+
```

This behavior is permitted when the `ONLY_FULL_GROUP_BY` SQL mode is not enabled. If that mode is enabled, the server rejects the query as illegal because `country` is not listed in the `GROUP BY` clause.

12.19.3 MySQL Handling of GROUP BY

In standard SQL, a query that includes a `GROUP BY` clause cannot refer to nonaggregated columns in the select list that are not named in the `GROUP BY` clause. For example, this query is illegal in standard SQL because the nonaggregated `name` column in the select list does not appear in the `GROUP BY`:

```
SELECT o.custid, c.name, MAX(o.payment)
  FROM orders AS o, customers AS c
  WHERE o.custid = c.custid
  GROUP BY o.custid;
```

For the query to be legal, the `name` column must be omitted from the select list or named in the `GROUP BY` clause.

MySQL extends the standard SQL use of `GROUP BY` so that the select list can refer to nonaggregated columns not named in the `GROUP BY` clause. This means that the preceding query is legal in MySQL. You can use this feature to get better performance by avoiding unnecessary column sorting and grouping. However, this is useful primarily when all values in each nonaggregated column not named in the `GROUP BY` are the same for each group. The server is free to choose any value from each group, so unless they are the same, the values chosen are nondeterministic. Furthermore, the selection of values from each group cannot be influenced by adding an `ORDER BY` clause. Result set sorting occurs after values have been chosen, and `ORDER BY` does not affect which values within each group the server chooses.

A similar MySQL extension applies to the `HAVING` clause. In standard SQL, a query cannot refer to nonaggregated columns in the `HAVING` clause that are not named in the `GROUP BY` clause. To simplify calculations, a MySQL extension permits references to such columns. This extension assumes that the nongrouped columns have the same group-wise values. Otherwise, the result is nondeterministic.

To disable the MySQL `GROUP BY` extension and enable standard SQL behavior, enable the `ONLY_FULL_GROUP_BY` SQL mode. In this case, columns not named in the `GROUP BY` clause cannot be used in the select list or `HAVING` clause unless enclosed in an aggregate function.

The select list extension also applies to `ORDER BY`. That is, you can refer to nonaggregated columns in the `ORDER BY` clause that do not appear in the `GROUP BY` clause. (However, as mentioned previously, `ORDER BY` does not affect which values are chosen from nonaggregated columns; it only sorts them after they have been chosen.) This extension does not apply if the `ONLY_FULL_GROUP_BY` SQL mode is enabled.

If a query has aggregate functions and no `GROUP BY` clause, it cannot have nonaggregated columns in the select list, `HAVING` condition, or `ORDER BY` list with `ONLY_FULL_GROUP_BY` enabled:

```
mysql> SELECT name, MAX(age) FROM t;
ERROR 1140 (42000): Mixing of GROUP columns (MIN(),MAX(),COUNT(),...)
with no GROUP columns is illegal if there is no GROUP BY clause
```

Without `GROUP BY`, there is a single group and it is nondeterministic which `name` value to choose for the group.

Another MySQL extension to standard SQL permits references in the `HAVING` clause to aliased expressions in the select list. Enabling `ONLY_FULL_GROUP_BY` prevents this. For example, the following query returns `name` values that occur only once in table `orders`; the query is accepted regardless of whether `ONLY_FULL_GROUP_BY` is enabled:

```
SELECT name, COUNT(name) FROM orders
  GROUP BY name
  HAVING COUNT(name) = 1;
```

The following query is accepted only if `ONLY_FULL_GROUP_BY` is disabled.

```
SELECT name, COUNT(name) AS c FROM orders
  GROUP BY name
  HAVING c = 1;
```

If you are trying to follow standard SQL, you can use only column expressions in `GROUP BY` clauses. As a workaround, use an alias for the expression:

```
SELECT id, FLOOR(value/100) AS val
  FROM tbl_name
  GROUP BY id, val;
```

MySQL permits noncolumn expressions in `GROUP BY` clauses, so the alias is unnecessary:

```
SELECT id, FLOOR(value/100)
  FROM tbl_name
  GROUP BY id, FLOOR(value/100);
```

12.20 Miscellaneous Functions

Table 12.24 Miscellaneous Functions

Name	Description
`DEFAULT()`	Return the default value for a table column
`INET_ATON()`	Return the numeric value of an IP address
`INET_NTOA()`	Return the IP address from a numeric value
`INET6_ATON()`	Return the numeric value of an IPv6 address
`INET6_NTOA()`	Return the IPv6 address from a numeric value
`IS_IPV4()`	Whether argument is an IPv4 address
`IS_IPV4_COMPAT()`	Whether argument is an IPv4-compatible address
`IS_IPV4_MAPPED()`	Whether argument is an IPv4-mapped address
`IS_IPV6()`	Whether argument is an IPv6 address
`MASTER_POS_WAIT()`	Block until the replica has read and applied all updates up to the specified position
`NAME_CONST()`	Cause the column to have the given name
`SLEEP()`	Sleep for a number of seconds
`UUID()`	Return a Universal Unique Identifier (UUID)
`UUID_SHORT()`	Return an integer-valued universal identifier

Name	Description
VALUES()	Define the values to be used during an INSERT

- DEFAULT(*col_name*)

  Returns the default value for a table column. An error results if the column has no default value.

```
mysql> UPDATE t SET i = DEFAULT(i)+1 WHERE id < 100;
```

- FORMAT(*X*,*D*)

  Formats the number *X* to a format like '#,###,###.##', rounded to *D* decimal places, and returns the result as a string. For details, see Section 12.8, “String Functions and Operators”.

- INET_ATON(*expr*)

  Given the dotted-quad representation of an IPv4 network address as a string, returns an integer that represents the numeric value of the address in network byte order (big endian). INET_ATON() returns NULL if it does not understand its argument.

```
mysql> SELECT INET_ATON('10.0.5.9');
        -> 167773449
```

  For this example, the return value is calculated as $10\times256^3 + 0\times256^2 + 5\times256 + 9$.

  INET_ATON() may or may not return a non-NULL result for short-form IP addresses (such as '127.1' as a representation of '127.0.0.1'). Because of this, INET_ATON()a should not be used for such addresses.

Note

To store values generated by INET_ATON(), use an INT UNSIGNED column rather than INT, which is signed. If you use a signed column, values corresponding to IP addresses for which the first octet is greater than 127 cannot be stored correctly. See Section 11.1.7, “Out-of-Range and Overflow Handling”.

- INET_NTOA(*expr*)

  Given a numeric IPv4 network address in network byte order, returns the dotted-quad string representation of the address as a string in the connection character set. INET_NTOA() returns NULL if it does not understand its argument.

```
mysql> SELECT INET_NTOA(167773449);
        -> '10.0.5.9'
```

- INET6_ATON(*expr*)

  Given an IPv6 or IPv4 network address as a string, returns a binary string that represents the numeric value of the address in network byte order (big endian). Because numeric-format IPv6 addresses require more bytes than the largest integer type, the representation returned by this function has the VARBINARY data type: VARBINARY(16) for IPv6 addresses and VARBINARY(4) for IPv4 addresses. If the argument is not a valid address, INET6_ATON() returns NULL.

  The following examples use HEX() to display the INET6_ATON() result in printable form:

```
mysql> SELECT HEX(INET6_ATON('fdfe::5a55:caff:fefa:9089'));
        -> 'FDFE0000000000005A55CAFFFEFA9089'
mysql> SELECT HEX(INET6_ATON('10.0.5.9'));
        -> '0A000509'
```

  INET6_ATON() observes several constraints on valid arguments. These are given in the following list along with examples.

- A trailing zone ID is not permitted, as in `fe80::3%1` or `fe80::3%eth0`.
- A trailing network mask is not permitted, as in `2001:45f:3:ba::/64` or `198.51.100.0/24`.
- For values representing IPv4 addresses, only classless addresses are supported. Classful addresses such as `198.51.1` are rejected. A trailing port number is not permitted, as in `198.51.100.2:8080`. Hexadecimal numbers in address components are not permitted, as in `198.0xa0.1.2`. Octal numbers are not supported: `198.51.010.1` is treated as `198.51.10.1`, not `198.51.8.1`. These IPv4 constraints also apply to IPv6 addresses that have IPv4 address parts, such as IPv4-compatible or IPv4-mapped addresses.

To convert an IPv4 address *`expr`* represented in numeric form as an `INT` value to an IPv6 address represented in numeric form as a `VARBINARY` value, use this expression:

```
INET6_ATON(INET_NTOA(expr))
```

For example:

```
mysql> SELECT HEX(INET6_ATON(INET_NTOA(167773449)));
        -> '0A000509'
```

- `INET6_NTOA(`*`expr`*`)`

  Given an IPv6 or IPv4 network address represented in numeric form as a binary string, returns the string representation of the address as a string in the connection character set. If the argument is not a valid address, `INET6_NTOA()` returns `NULL`.

  `INET6_NTOA()` has these properties:

  - It does not use operating system functions to perform conversions, thus the output string is platform independent.
  - The return string has a maximum length of 39 (4 x 8 + 7). Given this statement:

    ```
    CREATE TABLE t AS SELECT INET6_NTOA(expr) AS c1;
    ```

    The resulting table would have this definition:

    ```
    CREATE TABLE t (c1 VARCHAR(39) CHARACTER SET utf8 DEFAULT NULL);
    ```

  - The return string uses lowercase letters for IPv6 addresses.

```
mysql> SELECT INET6_NTOA(INET6_ATON('fdfe::5a55:caff:fefa:9089'));
        -> 'fdfe::5a55:caff:fefa:9089'
mysql> SELECT INET6_NTOA(INET6_ATON('10.0.5.9'));
        -> '10.0.5.9'

mysql> SELECT INET6_NTOA(UNHEX('FDFE0000000000005A55CAFFFEFA9089'));
        -> 'fdfe::5a55:caff:fefa:9089'
mysql> SELECT INET6_NTOA(UNHEX('0A000509'));
        -> '10.0.5.9'
```

- `IS_IPV4(`*`expr`*`)`

  Returns 1 if the argument is a valid IPv4 address specified as a string, 0 otherwise.

```
mysql> SELECT IS_IPV4('10.0.5.9'), IS_IPV4('10.0.5.256');
```

```
        -> 1, 0
```

For a given argument, if `IS_IPV4()` returns 1, `INET_ATON()` (and `INET6_ATON()`) returns a value that is not `NULL`. The converse statement is not true: In some cases, `INET_ATON()` returns a value other than `NULL` when `IS_IPV4()` returns 0.

As implied by the preceding remarks, `IS_IPV4()` is more strict than `INET_ATON()` about what constitutes a valid IPv4 address, so it may be useful for applications that need to perform strong checks against invalid values. Alternatively, use `INET6_ATON()` to convert IPv4 addresses to internal form and check for a `NULL` result (which indicates an invalid address). `INET6_ATON()` is equally strong as `IS_IPV4()` about checking IPv4 addresses.

- `IS_IPV4_COMPAT(`*`expr`*`)`

  This function takes an IPv6 address represented in numeric form as a binary string, as returned by `INET6_ATON()`. It returns 1 if the argument is a valid IPv4-compatible IPv6 address, 0 otherwise. IPv4-compatible addresses have the form `::`*`ipv4_address`*.

```
mysql> SELECT IS_IPV4_COMPAT(INET6_ATON('::10.0.5.9'));
        -> 1
mysql> SELECT IS_IPV4_COMPAT(INET6_ATON('::ffff:10.0.5.9'));
        -> 0
```

  The IPv4 part of an IPv4-compatible address can also be represented using hexadecimal notation. For example, `198.51.100.1` has this raw hexadecimal value:

```
mysql> SELECT HEX(INET6_ATON('198.51.100.1'));
        -> 'C6336401'
```

  Expressed in IPv4-compatible form, `::198.51.100.1` is equivalent to `::c0a8:0001` or (without leading zeros) `::c0a8:1`

```
mysql> SELECT
    ->   IS_IPV4_COMPAT(INET6_ATON('::198.51.100.1')),
    ->   IS_IPV4_COMPAT(INET6_ATON('::c0a8:0001')),
    ->   IS_IPV4_COMPAT(INET6_ATON('::c0a8:1'));
        -> 1, 1, 1
```

- `IS_IPV4_MAPPED(`*`expr`*`)`

  This function takes an IPv6 address represented in numeric form as a binary string, as returned by `INET6_ATON()`. It returns 1 if the argument is a valid IPv4-mapped IPv6 address, 0 otherwise. IPv4-mapped addresses have the form `::ffff:`*`ipv4_address`*.

```
mysql> SELECT IS_IPV4_MAPPED(INET6_ATON('::10.0.5.9'));
        -> 0
mysql> SELECT IS_IPV4_MAPPED(INET6_ATON('::ffff:10.0.5.9'));
        -> 1
```

  As with `IS_IPV4_COMPAT()` the IPv4 part of an IPv4-mapped address can also be represented using hexadecimal notation:

```
mysql> SELECT
    ->   IS_IPV4_MAPPED(INET6_ATON('::ffff:198.51.100.1')),
    ->   IS_IPV4_MAPPED(INET6_ATON('::ffff:c0a8:0001')),
    ->   IS_IPV4_MAPPED(INET6_ATON('::ffff:c0a8:1'));
        -> 1, 1, 1
```

- `IS_IPV6(`*`expr`*`)`

  Returns 1 if the argument is a valid IPv6 address specified as a string, 0 otherwise. This function does not consider IPv4 addresses to be valid IPv6 addresses.

```
mysql> SELECT IS_IPV6('10.0.5.9'), IS_IPV6('::1');
```

```
        -> 0, 1
```

For a given argument, if `IS_IPV6()` returns 1, `INET6_ATON()` returns a value other than `NULL`.

- `MASTER_POS_WAIT(log_name,log_pos[,timeout])`

  This function is useful for control of source/replica synchronization. It blocks until the replica has read and applied all updates up to the specified position in the source log. The return value is the number of log events the replica had to wait for to advance to the specified position. The function returns `NULL` if the replica SQL thread is not started, the replica's source information is not initialized, the arguments are incorrect, or an error occurs. It returns `-1` if the timeout has been exceeded. If the replica SQL thread stops while `MASTER_POS_WAIT()` is waiting, the function returns `NULL`. If the replica is past the specified position, the function returns immediately.

  On a multithreaded replica, the function waits until expiry of the limit set by the `slave_checkpoint_group` or `slave_checkpoint_period` system variable, when the checkpoint operation is called to update the status of the replica. Depending on the setting for the system variables, the function might therefore return some time after the specified position was reached.

  If a *`timeout`* value is specified, `MASTER_POS_WAIT()` stops waiting when *`timeout`* seconds have elapsed. *`timeout`* must be greater than 0; a zero or negative *`timeout`* means no timeout.

  This function is unsafe for statement-based replication. A warning is logged if you use this function when `binlog_format` is set to `STATEMENT`.

- `NAME_CONST(name,value)`

  Returns the given value. When used to produce a result set column, `NAME_CONST()` causes the column to have the given name. The arguments should be constants.

```
mysql> SELECT NAME_CONST('myname', 14);
+--------+
| myname |
+--------+
|     14 |
+--------+
```

  This function is for internal use only. The server uses it when writing statements from stored programs that contain references to local program variables, as described in Section 20.7, “Stored Program Binary Logging”. You might see this function in the output from `mysqlbinlog`.

  For your applications, you can obtain exactly the same result as in the example just shown by using simple aliasing, like this:

```
mysql> SELECT 14 AS myname;
+--------+
| myname |
+--------+
|     14 |
+--------+
1 row in set (0.00 sec)
```

  See Section 13.2.9, “SELECT Statement”, for more information about column aliases.

- `SLEEP(duration)`

  Sleeps (pauses) for the number of seconds given by the *`duration`* argument, then returns 0. If `SLEEP()` is interrupted, it returns 1. The duration may have a fractional part.

  When sleep returns normally (without interruption), it returns 0:

```
mysql> SELECT SLEEP(1000);
+-------------+
```

```
| SLEEP(1000) |
+-------------+
|           0 |
+-------------+
```

When `SLEEP()` is the only thing invoked by a query that is interrupted, it returns 1 and the query itself returns no error. This statement is interrupted using `KILL QUERY` from another session:

```
mysql> SELECT SLEEP(1000);
+-------------+
| SLEEP(1000) |
+-------------+
|           1 |
+-------------+
```

When `SLEEP()` is only part of a query that is interrupted, the query returns an error. This statement is interrupted using `KILL QUERY` from another session:

```
mysql> SELECT 1 FROM t1 WHERE SLEEP(1000);
ERROR 1317 (70100): Query execution was interrupted
```

This function is unsafe for statement-based replication. A warning is logged if you use this function when `binlog_format` is set to `STATEMENT`.

- `UUID()`

  Returns a Universal Unique Identifier (UUID) generated according to RFC 4122, “A Universally Unique IDentifier (UUID) URN Namespace” (http://www.ietf.org/rfc/rfc4122.txt).

  A UUID is designed as a number that is globally unique in space and time. Two calls to `UUID()` are expected to generate two different values, even if these calls are performed on two separate devices not connected to each other.

  Warning

  Although `UUID()` values are intended to be unique, they are not necessarily unguessable or unpredictable. If unpredictability is required, UUID values should be generated some other way.

  `UUID()` returns a value that conforms to UUID version 1 as described in RFC 4122. The value is a 128-bit number represented as a `utf8` string of five hexadecimal numbers in `aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee` format:

  - The first three numbers are generated from the low, middle, and high parts of a timestamp. The high part also includes the UUID version number.
  - The fourth number preserves temporal uniqueness in case the timestamp value loses monotonicity (for example, due to daylight saving time).
  - The fifth number is an IEEE 802 node number that provides spatial uniqueness. A random number is substituted if the latter is not available (for example, because the host device has no Ethernet card, or it is unknown how to find the hardware address of an interface on the host operating system). In this case, spatial uniqueness cannot be guaranteed. Nevertheless, a collision should have *very* low probability.

    The MAC address of an interface is taken into account only on FreeBSD, Linux, and Windows. On other operating systems, MySQL uses a randomly generated 48-bit number.

  ```
  mysql> SELECT UUID();
         -> '6ccd780c-baba-1026-9564-5b8c656024db'
  ```

  This function is unsafe for statement-based replication. A warning is logged if you use this function when `binlog_format` is set to `STATEMENT`.

- `UUID_SHORT()`

  Returns a “short” universal identifier as a 64-bit unsigned integer. Values returned by `UUID_SHORT()` differ from the string-format 128-bit identifiers returned by the `UUID()` function and have different uniqueness properties. The value of `UUID_SHORT()` is guaranteed to be unique if the following conditions hold:

  - The `server_id` value of the current server is between 0 and 255 and is unique among your set of source and replica servers
  - You do not set back the system time for your server host between `mysqld` restarts
  - You invoke `UUID_SHORT()` on average fewer than 16 million times per second between `mysqld` restarts

  The `UUID_SHORT()` return value is constructed this way:

```
  (server_id & 255) << 56
+ (server_startup_time_in_seconds << 24)
+ incremented_variable++;
```

```
mysql> SELECT UUID_SHORT();
        -> 92395783831158784
```

  Note

  `UUID_SHORT()` does not work with statement-based replication.

- `VALUES(col_name)`

  In an `INSERT ... ON DUPLICATE KEY UPDATE` statement, you can use the `VALUES(col_name)` function in the `UPDATE` clause to refer to column values from the `INSERT` portion of the statement. In other words, `VALUES(col_name)` in the `UPDATE` clause refers to the value of `col_name` that would be inserted, had no duplicate-key conflict occurred. This function is especially useful in multiple-row inserts. The `VALUES()` function is meaningful only in the `ON DUPLICATE KEY UPDATE` clause of `INSERT` statements and returns `NULL` otherwise. See Section 13.2.5.2, “INSERT ... ON DUPLICATE KEY UPDATE Statement”.

```
mysql> INSERT INTO table (a,b,c) VALUES (1,2,3),(4,5,6)
    -> ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

12.21 Precision Math

MySQL provides support for precision math: numeric value handling that results in extremely accurate results and a high degree control over invalid values. Precision math is based on these two features:

- SQL modes that control how strict the server is about accepting or rejecting invalid data.
- The MySQL library for fixed-point arithmetic.

These features have several implications for numeric operations and provide a high degree of compliance with standard SQL:

- **Precise calculations**: For exact-value numbers, calculations do not introduce floating-point errors. Instead, exact precision is used. For example, MySQL treats a number such as `.0001` as an exact value rather than as an approximation, and summing it 10,000 times produces a result of exactly `1`, not a value that is merely “close” to 1.
- **Well-defined rounding behavior**: For exact-value numbers, the result of `ROUND()` depends on its argument, not on environmental factors such as how the underlying C library works.
- **Platform independence**: Operations on exact numeric values are the same across different platforms such as Windows and Unix.

- **Control over handling of invalid values**: Overflow and division by zero are detectable and can be treated as errors. For example, you can treat a value that is too large for a column as an error rather than having the value truncated to lie within the range of the column's data type. Similarly, you can treat division by zero as an error rather than as an operation that produces a result of `NULL`. The choice of which approach to take is determined by the setting of the server SQL mode.

The following discussion covers several aspects of how precision math works, including possible incompatibilities with older applications. At the end, some examples are given that demonstrate how MySQL handles numeric operations precisely. For information about controlling the SQL mode, see Section 5.1.10, “Server SQL Modes”.

12.21.1 Types of Numeric Values

The scope of precision math for exact-value operations includes the exact-value data types (integer and `DECIMAL` types) and exact-value numeric literals. Approximate-value data types and numeric literals are handled as floating-point numbers.

Exact-value numeric literals have an integer part or fractional part, or both. They may be signed. Examples: `1`, `.2`, `3.4`, `-5`, `-6.78`, `+9.10`.

Approximate-value numeric literals are represented in scientific notation with a mantissa and exponent. Either or both parts may be signed. Examples: `1.2E3`, `1.2E-3`, `-1.2E3`, `-1.2E-3`.

Two numbers that look similar may be treated differently. For example, `2.34` is an exact-value (fixed-point) number, whereas `2.34E0` is an approximate-value (floating-point) number.

The `DECIMAL` data type is a fixed-point type and calculations are exact. In MySQL, the `DECIMAL` type has several synonyms: `NUMERIC`, `DEC`, `FIXED`. The integer types also are exact-value types.

The `FLOAT` and `DOUBLE` data types are floating-point types and calculations are approximate. In MySQL, types that are synonymous with `FLOAT` or `DOUBLE` are `DOUBLE PRECISION` and `REAL`.

12.21.2 DECIMAL Data Type Characteristics

This section discusses the characteristics of the `DECIMAL` data type (and its synonyms), with particular regard to the following topics:

- Maximum number of digits
- Storage format
- Storage requirements
- The nonstandard MySQL extension to the upper range of `DECIMAL` columns

The declaration syntax for a `DECIMAL` column is `DECIMAL(M,D)`. The ranges of values for the arguments are as follows:

- *`M`* is the maximum number of digits (the precision). It has a range of 1 to 65.
- *`D`* is the number of digits to the right of the decimal point (the scale). It has a range of 0 to 30 and must be no larger than *`M`*.

If *`D`* is omitted, the default is 0. If *`M`* is omitted, the default is 10.

The maximum value of 65 for *`M`* means that calculations on `DECIMAL` values are accurate up to 65 digits. This limit of 65 digits of precision also applies to exact-value numeric literals, so the maximum range of such literals differs from before. (There is also a limit on how long the text of `DECIMAL` literals can be; see Section 12.21.3, “Expression Handling”.)

Values for `DECIMAL` columns are stored using a binary format that packs nine decimal digits into 4 bytes. The storage requirements for the integer and fractional parts of each value are determined

separately. Each multiple of nine digits requires 4 bytes, and any remaining digits left over require some fraction of 4 bytes. The storage required for remaining digits is given by the following table.

Leftover Digits	Number of Bytes
0	0
1–2	1
3–4	2
5–6	3
7–9	4

For example, a `DECIMAL(18,9)` column has nine digits on either side of the decimal point, so the integer part and the fractional part each require 4 bytes. A `DECIMAL(20,6)` column has fourteen integer digits and six fractional digits. The integer digits require four bytes for nine of the digits and 3 bytes for the remaining five digits. The six fractional digits require 3 bytes.

`DECIMAL` columns do not store a leading `+` character or `-` character or leading `0` digits. If you insert `+0003.1` into a `DECIMAL(5,1)` column, it is stored as `3.1`. For negative numbers, a literal `-` character is not stored.

`DECIMAL` columns do not permit values larger than the range implied by the column definition. For example, a `DECIMAL(3,0)` column supports a range of `-999` to `999`. A `DECIMAL(M,D)` column permits up to *`M`* - *`D`* digits to the left of the decimal point.

The SQL standard requires that the precision of `NUMERIC(M,D)` be *exactly* *`M`* digits. For `DECIMAL(M,D)`, the standard requires a precision of at least *`M`* digits but permits more. In MySQL, `DECIMAL(M,D)` and `NUMERIC(M,D)` are the same, and both have a precision of exactly *`M`* digits.

For a full explanation of the internal format of `DECIMAL` values, see the file `strings/decimal.c` in a MySQL source distribution. The format is explained (with an example) in the `decimal2bin()` function.

12.21.3 Expression Handling

With precision math, exact-value numbers are used as given whenever possible. For example, numbers in comparisons are used exactly as given without a change in value. In strict SQL mode, for `INSERT` into a column with an exact data type (`DECIMAL` or integer), a number is inserted with its exact value if it is within the column range. When retrieved, the value should be the same as what was inserted. (If strict SQL mode is not enabled, truncation for `INSERT` is permissible.)

Handling of a numeric expression depends on what kind of values the expression contains:

- If any approximate values are present, the expression is approximate and is evaluated using floating-point arithmetic.
- If no approximate values are present, the expression contains only exact values. If any exact value contains a fractional part (a value following the decimal point), the expression is evaluated using `DECIMAL` exact arithmetic and has a precision of 65 digits. The term “exact” is subject to the limits of what can be represented in binary. For example, `1.0/3.0` can be approximated in decimal notation as `.333...`, but not written as an exact number, so `(1.0/3.0)*3.0` does not evaluate to exactly `1.0`.
- Otherwise, the expression contains only integer values. The expression is exact and is evaluated using integer arithmetic and has a precision the same as `BIGINT` (64 bits).

If a numeric expression contains any strings, they are converted to double-precision floating-point values and the expression is approximate.

Inserts into numeric columns are affected by the SQL mode, which is controlled by the `sql_mode` system variable. (See Section 5.1.10, “Server SQL Modes”.) The following discussion mentions

strict mode (selected by the STRICT_ALL_TABLES or STRICT_TRANS_TABLES mode values) and ERROR_FOR_DIVISION_BY_ZERO. To turn on all restrictions, you can simply use TRADITIONAL mode, which includes both strict mode values and ERROR_FOR_DIVISION_BY_ZERO:

```
SET sql_mode='TRADITIONAL';
```

If a number is inserted into an exact type column (DECIMAL or integer), it is inserted with its exact value if it is within the column range and precision.

If the value has too many digits in the fractional part, rounding occurs and a note is generated. Rounding is done as described in Section 12.21.4, “Rounding Behavior”. Truncation due to rounding of the fractional part is not an error, even in strict mode.

If the value has too many digits in the integer part, it is too large (out of range) and is handled as follows:

- If strict mode is not enabled, the value is truncated to the nearest legal value and a warning is generated.
- If strict mode is enabled, an overflow error occurs.

For DECIMAL literals, in addition to the precision limit of 65 digits, there is a limit on how long the text of the literal can be. If the value exceeds approximately 80 characters, unexpected results can occur. For example:

```
mysql> SELECT
       CAST(00000000000000000000000000000000000000000000000000000000000000000000000000000020.01 AS DE
+------------------+
| val              |
+------------------+
| 9999999999999.99 |
+------------------+
1 row in set, 2 warnings (0.00 sec)

mysql> SHOW WARNINGS;
+---------+------+-----------------------------------------------+
| Level   | Code | Message                                       |
+---------+------+-----------------------------------------------+
| Warning | 1292 | Truncated incorrect DECIMAL value: ''         |
| Warning | 1264 | Out of range value for column 'val' at row 1 |
+---------+------+-----------------------------------------------+
2 rows in set (0.00 sec)
```

Underflow is not detected, so underflow handling is undefined.

For inserts of strings into numeric columns, conversion from string to number is handled as follows if the string has nonnumeric contents:

- A string that does not begin with a number cannot be used as a number and produces an error in strict mode, or a warning otherwise. This includes the empty string.
- A string that begins with a number can be converted, but the trailing nonnumeric portion is truncated. If the truncated portion contains anything other than spaces, this produces an error in strict mode, or a warning otherwise.

By default, division by zero produces a result of NULL and no warning. By setting the SQL mode appropriately, division by zero can be restricted.

With the ERROR_FOR_DIVISION_BY_ZERO SQL mode enabled, MySQL handles division by zero differently:

- If strict mode is not enabled, a warning occurs.
- If strict mode is enabled, inserts and updates involving division by zero are prohibited, and an error occurs.

In other words, inserts and updates involving expressions that perform division by zero can be treated as errors, but this requires `ERROR_FOR_DIVISION_BY_ZERO` in addition to strict mode.

Suppose that we have this statement:

```
INSERT INTO t SET i = 1/0;
```

This is what happens for combinations of strict and `ERROR_FOR_DIVISION_BY_ZERO` modes.

`sql_mode` Value	Result
`''` (Default)	No warning, no error; `i` is set to `NULL`.
strict	No warning, no error; `i` is set to `NULL`.
`ERROR_FOR_DIVISION_BY_ZERO`	Warning, no error; `i` is set to `NULL`.
strict,`ERROR_FOR_DIVISION_BY_ZERO`	Error condition; no row is inserted.

12.21.4 Rounding Behavior

This section discusses precision math rounding for the `ROUND()` function and for inserts into columns with exact-value types (`DECIMAL` and integer).

The `ROUND()` function rounds differently depending on whether its argument is exact or approximate:

- For exact-value numbers, `ROUND()` uses the “round half up” rule: A value with a fractional part of .5 or greater is rounded up to the next integer if positive or down to the next integer if negative. (In other words, it is rounded away from zero.) A value with a fractional part less than .5 is rounded down to the next integer if positive or up to the next integer if negative. (In other words, it is rounded toward zero.)
- For approximate-value numbers, the result depends on the C library. On many systems, this means that `ROUND()` uses the “round to nearest even” rule: A value with a fractional part exactly half way between two integers is rounded to the nearest even integer.

The following example shows how rounding differs for exact and approximate values:

```
mysql> SELECT ROUND(2.5), ROUND(25E-1);
+------------+--------------+
| ROUND(2.5) | ROUND(25E-1) |
+------------+--------------+
| 3          |            2 |
+------------+--------------+
```

For inserts into a `DECIMAL` or integer column, the target is an exact data type, so rounding uses “round half away from zero,” regardless of whether the value to be inserted is exact or approximate:

```
mysql> CREATE TABLE t (d DECIMAL(10,0));
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t VALUES(2.5),(2.5E0);
Query OK, 2 rows affected, 2 warnings (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 2

mysql> SHOW WARNINGS;
+-------+------+----------------------------------------+
| Level | Code | Message                                |
+-------+------+----------------------------------------+
| Note  | 1265 | Data truncated for column 'd' at row 1 |
| Note  | 1265 | Data truncated for column 'd' at row 2 |
+-------+------+----------------------------------------+
2 rows in set (0.00 sec)

mysql> SELECT d FROM t;
+------+
| d    |
```

```
+------+
|    3 |
|    3 |
+------+
2 rows in set (0.00 sec)
```

The `SHOW WARNINGS` statement displays the notes that are generated by truncation due to rounding of the fractional part. Such truncation is not an error, even in strict SQL mode (see Section 12.21.3, “Expression Handling”).

12.21.5 Precision Math Examples

This section provides some examples that show precision math query results in MySQL. These examples demonstrate the principles described in Section 12.21.3, “Expression Handling”, and Section 12.21.4, “Rounding Behavior”.

Example 1. Numbers are used with their exact value as given when possible:

```
mysql> SELECT (.1 + .2) = .3;
+----------------+
| (.1 + .2) = .3 |
+----------------+
|              1 |
+----------------+
```

For floating-point values, results are inexact:

```
mysql> SELECT (.1E0 + .2E0) = .3E0;
+----------------------+
| (.1E0 + .2E0) = .3E0 |
+----------------------+
|                    0 |
+----------------------+
```

Another way to see the difference in exact and approximate value handling is to add a small number to a sum many times. Consider the following stored procedure, which adds `.0001` to a variable 1,000 times.

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 0;
  DECLARE d DECIMAL(10,4) DEFAULT 0;
  DECLARE f FLOAT DEFAULT 0;
  WHILE i < 10000 DO
    SET d = d + .0001;
    SET f = f + .0001E0;
    SET i = i + 1;
  END WHILE;
  SELECT d, f;
END;
```

The sum for both `d` and `f` logically should be 1, but that is true only for the decimal calculation. The floating-point calculation introduces small errors:

```
+--------+------------------+
| d      | f                |
+--------+------------------+
| 1.0000 | 0.99999999999991 |
+--------+------------------+
```

Example 2. Multiplication is performed with the scale required by standard SQL. That is, for two numbers *`X1`* and *`X2`* that have scale *`S1`* and *`S2`*, the scale of the result is *`S1 + S2`*:

```
mysql> SELECT .01 * .01;
+-----------+
| .01 * .01 |
+-----------+
```

```
| 0.0001    |
+-----------+
```

Example 3. Rounding behavior for exact-value numbers is well-defined:

Rounding behavior (for example, with the `ROUND()` function) is independent of the implementation of the underlying C library, which means that results are consistent from platform to platform.

- Rounding for exact-value columns (`DECIMAL` and integer) and exact-valued numbers uses the "round half away from zero" rule. A value with a fractional part of .5 or greater is rounded away from zero to the nearest integer, as shown here:

```
mysql> SELECT ROUND(2.5), ROUND(-2.5);
+------------+-------------+
| ROUND(2.5) | ROUND(-2.5) |
+------------+-------------+
| 3          | -3          |
+------------+-------------+
```

- Rounding for floating-point values uses the C library, which on many systems uses the "round to nearest even" rule. A value with a fractional part exactly half way between two integers is rounded to the nearest even integer:

```
mysql> SELECT ROUND(2.5E0), ROUND(-2.5E0);
+--------------+---------------+
| ROUND(2.5E0) | ROUND(-2.5E0) |
+--------------+---------------+
|            2 |            -2 |
+--------------+---------------+
```

Example 4. In strict mode, inserting a value that is out of range for a column causes an error, rather than truncation to a legal value.

When MySQL is not running in strict mode, truncation to a legal value occurs:

```
mysql> SET sql_mode='';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO t SET i = 128;
Query OK, 1 row affected, 1 warning (0.00 sec)

mysql> SELECT i FROM t;
+------+
| i    |
+------+
|  127 |
+------+
1 row in set (0.00 sec)
```

However, an error occurs if strict mode is in effect:

```
mysql> SET sql_mode='STRICT_ALL_TABLES';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 128;
ERROR 1264 (22003): Out of range value adjusted for column 'i' at row 1

mysql> SELECT i FROM t;
Empty set (0.00 sec)
```

Example 5: In strict mode and with `ERROR_FOR_DIVISION_BY_ZERO` set, division by zero causes an error, not a result of `NULL`.

In nonstrict mode, division by zero has a result of `NULL`:

```
mysql> SET sql_mode='';
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 1 / 0;
Query OK, 1 row affected (0.00 sec)

mysql> SELECT i FROM t;
+------+
| i    |
+------+
| NULL |
+------+
1 row in set (0.03 sec)
```

However, division by zero is an error if the proper SQL modes are in effect:

```
mysql> SET sql_mode='STRICT_ALL_TABLES,ERROR_FOR_DIVISION_BY_ZERO';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t (i TINYINT);
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO t SET i = 1 / 0;
ERROR 1365 (22012): Division by 0

mysql> SELECT i FROM t;
Empty set (0.01 sec)
```

Example 6. Exact-value literals are evaluated as exact values.

Approximate-value literals are evaluated using floating point, but exact-value literals are handled as `DECIMAL`:

```
mysql> CREATE TABLE t SELECT 2.5 AS a, 25E-1 AS b;
Query OK, 1 row affected (0.01 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> DESCRIBE t;
+-------+----------------------+------+-----+---------+-------+
| Field | Type                 | Null | Key | Default | Extra |
+-------+----------------------+------+-----+---------+-------+
| a     | decimal(2,1) unsigned | NO   |     | 0.0     |       |
| b     | double               | NO   |     | 0       |       |
+-------+----------------------+------+-----+---------+-------+
2 rows in set (0.01 sec)
```

Example 7. If the argument to an aggregate function is an exact numeric type, the result is also an exact numeric type, with a scale at least that of the argument.

Consider these statements:

```
mysql> CREATE TABLE t (i INT, d DECIMAL, f FLOAT);
mysql> INSERT INTO t VALUES(1,1,1);
mysql> CREATE TABLE y SELECT AVG(i), AVG(d), AVG(f) FROM t;
```

The result is a double only for the floating-point argument. For exact type arguments, the result is also an exact type:

```
mysql> DESCRIBE y;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| AVG(i) | decimal(14,4) | YES  |     | NULL    |       |
| AVG(d) | decimal(14,4) | YES  |     | NULL    |       |
| AVG(f) | double        | YES  |     | NULL    |       |
```

```
+--------+---------------+------+-----+---------+-------+
```

The result is a double only for the floating-point argument. For exact type arguments, the result is also an exact type.

Chapter 13 SQL Statements

Table of Contents

13.1 Data Definition Statements 1496
13.1.1 ALTER DATABASE Statement 1496
13.1.2 ALTER EVENT Statement 1497
13.1.3 ALTER FUNCTION Statement 1499
13.1.4 ALTER LOGFILE GROUP Statement 1499
13.1.5 ALTER PROCEDURE Statement 1500
13.1.6 ALTER SERVER Statement 1501
13.1.7 ALTER TABLE Statement 1501
13.1.8 ALTER TABLESPACE Statement 1520
13.1.9 ALTER VIEW Statement 1521
13.1.10 CREATE DATABASE Statement 1521
13.1.11 CREATE EVENT Statement 1522
13.1.12 CREATE FUNCTION Statement 1526
13.1.13 CREATE INDEX Statement 1526
13.1.14 CREATE LOGFILE GROUP Statement 1531
13.1.15 CREATE PROCEDURE and CREATE FUNCTION Statements 1533
13.1.16 CREATE SERVER Statement 1538
13.1.17 CREATE TABLE Statement 1539
13.1.18 CREATE TABLESPACE Statement 1572
13.1.19 CREATE TRIGGER Statement 1574
13.1.20 CREATE VIEW Statement 1576
13.1.21 DROP DATABASE Statement 1580
13.1.22 DROP EVENT Statement 1581
13.1.23 DROP FUNCTION Statement 1581
13.1.24 DROP INDEX Statement 1581
13.1.25 DROP LOGFILE GROUP Statement 1582
13.1.26 DROP PROCEDURE and DROP FUNCTION Statements 1582
13.1.27 DROP SERVER Statement 1583
13.1.28 DROP TABLE Statement 1583
13.1.29 DROP TABLESPACE Statement 1584
13.1.30 DROP TRIGGER Statement 1584
13.1.31 DROP VIEW Statement 1584
13.1.32 RENAME TABLE Statement 1585
13.1.33 TRUNCATE TABLE Statement 1586
13.2 Data Manipulation Statements 1587
13.2.1 CALL Statement 1587
13.2.2 DELETE Statement 1588
13.2.3 DO Statement 1592
13.2.4 HANDLER Statement 1593
13.2.5 INSERT Statement 1594
13.2.6 LOAD DATA Statement 1603
13.2.7 LOAD XML Statement 1613
13.2.8 REPLACE Statement 1620
13.2.9 SELECT Statement 1622
13.2.10 Subqueries 1637
13.2.11 UPDATE Statement 1649
13.3 Transactional and Locking Statements 1653
13.3.1 START TRANSACTION, COMMIT, and ROLLBACK Statements 1653
13.3.2 Statements That Cannot Be Rolled Back 1656
13.3.3 Statements That Cause an Implicit Commit 1656
13.3.4 SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Statements 1657
13.3.5 LOCK TABLES and UNLOCK TABLES Statements 1657

13.3.6 SET TRANSACTION Statement 1663
13.3.7 XA Transactions 1666
13.4 Replication Statements 1670
13.4.1 SQL Statements for Controlling Replication Source Servers 1670
13.4.2 SQL Statements for Controlling Replica Servers 1672
13.5 Prepared Statements 1682
13.5.1 PREPARE Statement 1685
13.5.2 EXECUTE Statement 1686
13.5.3 DEALLOCATE PREPARE Statement 1686
13.6 Compound Statements 1687
13.6.1 BEGIN ... END Compound Statement 1687
13.6.2 Statement Labels 1687
13.6.3 DECLARE Statement 1688
13.6.4 Variables in Stored Programs 1688
13.6.5 Flow Control Statements 1690
13.6.6 Cursors 1694
13.6.7 Condition Handling 1696
13.7 Database Administration Statements 1720
13.7.1 Account Management Statements 1720
13.7.2 Table Maintenance Statements 1739
13.7.3 Plugin and User-Defined Function Statements 1749
13.7.4 SET Statements 1752
13.7.5 SHOW Statements 1756
13.7.6 Other Administrative Statements 1804
13.8 Utility Statements 1814
13.8.1 DESCRIBE Statement 1814
13.8.2 EXPLAIN Statement 1814
13.8.3 HELP Statement 1816
13.8.4 USE Statement 1818

This chapter describes the syntax for the SQL statements supported by MySQL.

13.1 Data Definition Statements

13.1.1 ALTER DATABASE Statement

```
ALTER {DATABASE | SCHEMA} [db_name]
    alter_option ...
ALTER {DATABASE | SCHEMA} db_name
    UPGRADE DATA DIRECTORY NAME

alter_option: {
    [DEFAULT] CHARACTER SET [=] charset_name
  | [DEFAULT] COLLATE [=] collation_name
}
```

`ALTER DATABASE` enables you to change the overall characteristics of a database. These characteristics are stored in the `db.opt` file in the database directory. This statement requires the `ALTER` privilege on the database. `ALTER SCHEMA` is a synonym for `ALTER DATABASE`.

The database name can be omitted from the first syntax, in which case the statement applies to the default database. An error occurs if there is no default database.

- Character Set and Collation Options
- Upgrading from Versions Older than MySQL 5.1

Character Set and Collation Options

The `CHARACTER SET` clause changes the default database character set. The `COLLATE` clause changes the default database collation. For information about character set and collation names, see Chapter 10, *Character Sets, Collations, Unicode*.

To see the available character sets and collations, use the `SHOW CHARACTER SET` and `SHOW COLLATION` statements, respectively. See Section 13.7.5.4, “SHOW CHARACTER SET Statement”, and Section 13.7.5.5, “SHOW COLLATION Statement”.

A stored routine that uses the database defaults when the routine is created includes those defaults as part of its definition. (In a stored routine, variables with character data types use the database defaults if the character set or collation are not specified explicitly. See Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”.) If you change the default character set or collation for a database, any stored routines that are to use the new defaults must be dropped and recreated.

Upgrading from Versions Older than MySQL 5.1

The syntax that includes the `UPGRADE DATA DIRECTORY NAME` clause updates the name of the directory associated with the database to use the encoding implemented in MySQL 5.1 for mapping database names to database directory names (see Section 9.2.4, “Mapping of Identifiers to File Names”). This clause is for use under these conditions:

- It is intended when upgrading MySQL to 5.1 or later from older versions.
- It is intended to update a database directory name to the current encoding format if the name contains special characters that need encoding.
- The statement is used by `mysqlcheck` (as invoked by `mysql_upgrade`).

For example, if a database in MySQL 5.0 has the name `a-b-c`, the name contains instances of the `-` (dash) character. In MySQL 5.0, the database directory is also named `a-b-c`, which is not necessarily safe for all file systems. In MySQL 5.1 and later, the same database name is encoded as `a@002db@002dc` to produce a file system-neutral directory name.

When a MySQL installation is upgraded to MySQL 5.1 or later from an older version,the server displays a name such as `a-b-c` (which is in the old format) as `#mysql50#a-b-c`, and you must refer to the name using the `#mysql50#` prefix. Use `UPGRADE DATA DIRECTORY NAME` in this case to explicitly tell the server to re-encode the database directory name to the current encoding format:

```
ALTER DATABASE `#mysql50#a-b-c` UPGRADE DATA DIRECTORY NAME;
```

After executing this statement, you can refer to the database as `a-b-c` without the special `#mysql50#` prefix.

13.1.2 ALTER EVENT Statement

```
ALTER
    [DEFINER = user]
    EVENT event_name
    [ON SCHEDULE schedule]
    [ON COMPLETION [NOT] PRESERVE]
    [RENAME TO new_event_name]
    [ENABLE | DISABLE | DISABLE ON SLAVE]
    [COMMENT 'string']
    [DO event_body]
```

The `ALTER EVENT` statement changes one or more of the characteristics of an existing event without the need to drop and recreate it. The syntax for each of the `DEFINER`, `ON SCHEDULE`, `ON COMPLETION`, `COMMENT`, `ENABLE` / `DISABLE`, and `DO` clauses is exactly the same as when used with `CREATE EVENT`. (See Section 13.1.11, “CREATE EVENT Statement”.)

Any user can alter an event defined on a database for which that user has the `EVENT` privilege. When a user executes a successful `ALTER EVENT` statement, that user becomes the definer for the affected event.

`ALTER EVENT` works only with an existing event:

```
mysql> ALTER EVENT no_such_event
     >     ON SCHEDULE
     >       EVERY '2:3' DAY_HOUR;
ERROR 1517 (HY000): Unknown event 'no_such_event'
```

In each of the following examples, assume that the event named `myevent` is defined as shown here:

```
CREATE EVENT myevent
    ON SCHEDULE
      EVERY 6 HOUR
    COMMENT 'A sample comment.'
    DO
      UPDATE myschema.mytable SET mycol = mycol + 1;
```

The following statement changes the schedule for `myevent` from once every six hours starting immediately to once every twelve hours, starting four hours from the time the statement is run:

```
ALTER EVENT myevent
    ON SCHEDULE
      EVERY 12 HOUR
    STARTS CURRENT_TIMESTAMP + INTERVAL 4 HOUR;
```

It is possible to change multiple characteristics of an event in a single statement. This example changes the SQL statement executed by `myevent` to one that deletes all records from `mytable`; it also changes the schedule for the event such that it executes once, one day after this `ALTER EVENT` statement is run.

```
ALTER EVENT myevent
    ON SCHEDULE
      AT CURRENT_TIMESTAMP + INTERVAL 1 DAY
    DO
      TRUNCATE TABLE myschema.mytable;
```

Specify the options in an `ALTER EVENT` statement only for those characteristics that you want to change; omitted options keep their existing values. This includes any default values for `CREATE EVENT` such as `ENABLE`.

To disable `myevent`, use this `ALTER EVENT` statement:

```
ALTER EVENT myevent
    DISABLE;
```

The `ON SCHEDULE` clause may use expressions involving built-in MySQL functions and user variables to obtain any of the *`timestamp`* or *`interval`* values which it contains. You cannot use stored routines or user-defined functions in such expressions, and you cannot use any table references; however, you can use `SELECT FROM DUAL`. This is true for both `ALTER EVENT` and `CREATE EVENT` statements. References to stored routines, user-defined functions, and tables in such cases are specifically not permitted, and fail with an error (see Bug #22830).

Although an `ALTER EVENT` statement that contains another `ALTER EVENT` statement in its `DO` clause appears to succeed, when the server attempts to execute the resulting scheduled event, the execution fails with an error.

To rename an event, use the `ALTER EVENT` statement's `RENAME TO` clause. This statement renames the event `myevent` to `yourevent`:

```
ALTER EVENT myevent
    RENAME TO yourevent;
```

You can also move an event to a different database using `ALTER EVENT ... RENAME TO ...` and *`db_name.event_name`* notation, as shown here:

```
ALTER EVENT olddb.myevent
    RENAME TO newdb.myevent;
```

To execute the previous statement, the user executing it must have the EVENT privilege on both the olddb and newdb databases.

Note

There is no RENAME EVENT statement.

The value DISABLE ON SLAVE is used on a replica instead of ENABLE or DISABLE to indicate an event that was created on the source and replicated to the replica, but that is not executed on the replica. Normally, DISABLE ON SLAVE is set automatically as required; however, there are some circumstances under which you may want or need to change it manually. See Section 17.4.1.16, "Replication of Invoked Features", for more information.

13.1.3 ALTER FUNCTION Statement

```
ALTER FUNCTION func_name [characteristic ...]

characteristic: {
    COMMENT 'string'
  | LANGUAGE SQL
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
}
```

This statement can be used to change the characteristics of a stored function. More than one change may be specified in an ALTER FUNCTION statement. However, you cannot change the parameters or body of a stored function using this statement; to make such changes, you must drop and re-create the function using DROP FUNCTION and CREATE FUNCTION.

You must have the ALTER ROUTINE privilege for the function. (That privilege is granted automatically to the function creator.) If binary logging is enabled, the ALTER FUNCTION statement might also require the SUPER privilege, as described in Section 20.7, "Stored Program Binary Logging".

13.1.4 ALTER LOGFILE GROUP Statement

```
ALTER LOGFILE GROUP logfile_group
    ADD UNDOFILE 'file_name'
    [INITIAL_SIZE [=] size]
    [WAIT]
    ENGINE [=] engine_name
```

This statement adds an UNDO file named '*file_name*' to an existing log file group *logfile_group*. An ALTER LOGFILE GROUP statement has one and only one ADD UNDOFILE clause. No DROP UNDOFILE clause is currently supported.

Note

All NDB Cluster Disk Data objects share the same namespace. This means that *each Disk Data object* must be uniquely named (and not merely each Disk Data object of a given type). For example, you cannot have a tablespace and an undo log file with the same name, or an undo log file and a data file with the same name.

The optional INITIAL_SIZE parameter sets the UNDO file's initial size in bytes; if not specified, the initial size defaults to 134217728 (128 MB). Prior to MySQL NDB Cluster 7.3.2, this value was required to be specified using digits; in MySQL NDB Cluster 7.3.2 and later, you may optionally follow *size* with a one-letter abbreviation for an order of magnitude, similar to those used in my.cnf. Generally, this is one of the letters M (megabytes) or G (gigabytes). (Bug #13116514, Bug #16104705, Bug #62858)

On 32-bit systems, the maximum supported value for INITIAL_SIZE is 4294967296 (4 GB). (Bug #29186)

The minimum allowed value for `INITIAL_SIZE` is 1048576 (1 MB). (Bug #29574)

Note

`WAIT` is parsed but otherwise ignored. This keyword currently has no effect, and is intended for future expansion.

The `ENGINE` parameter (required) determines the storage engine which is used by this log file group, with *`engine_name`* being the name of the storage engine. Currently, the only accepted values for *`engine_name`* are "`NDBCLUSTER`" and "`NDB`". The two values are equivalent.

Here is an example, which assumes that the log file group `lg_3` has already been created using `CREATE LOGFILE GROUP` (see Section 13.1.14, "CREATE LOGFILE GROUP Statement"):

```
ALTER LOGFILE GROUP lg_3
    ADD UNDOFILE 'undo_10.dat'
    INITIAL_SIZE=32M
    ENGINE=NDBCLUSTER;
```

When `ALTER LOGFILE GROUP` is used with `ENGINE = NDBCLUSTER` (alternatively, `ENGINE = NDB`), an `UNDO` log file is created on each NDB Cluster data node. You can verify that the `UNDO` files were created and obtain information about them by querying the `INFORMATION_SCHEMA.FILES` table. For example:

```
mysql> SELECT FILE_NAME, LOGFILE_GROUP_NUMBER, EXTRA
    -> FROM INFORMATION_SCHEMA.FILES
    -> WHERE LOGFILE_GROUP_NAME = 'lg_3';
+-------------+--------------------+----------------+
| FILE_NAME   | LOGFILE_GROUP_NUMBER | EXTRA          |
+-------------+--------------------+----------------+
| newdata.dat |                  0 | CLUSTER_NODE=3 |
| newdata.dat |                  0 | CLUSTER_NODE=4 |
| undo_10.dat |                 11 | CLUSTER_NODE=3 |
| undo_10.dat |                 11 | CLUSTER_NODE=4 |
+-------------+--------------------+----------------+
4 rows in set (0.01 sec)
```

(See Section 21.31.1, "The INFORMATION_SCHEMA FILES Table".)

Memory used for `UNDO_BUFFER_SIZE` comes from the global pool whose size is determined by the value of the `SharedGlobalMemory` data node configuration parameter. This includes any default value implied for this option by the setting of the `InitialLogFileGroup` data node configuration parameter.

`ALTER LOGFILE GROUP` is useful only with Disk Data storage for NDB Cluster. For more information, see Section 18.5.10, "NDB Cluster Disk Data Tables".

13.1.5 ALTER PROCEDURE Statement

```
ALTER PROCEDURE proc_name [characteristic ...]

characteristic: {
    COMMENT 'string'
  | LANGUAGE SQL
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
}
```

This statement can be used to change the characteristics of a stored procedure. More than one change may be specified in an `ALTER PROCEDURE` statement. However, you cannot change the parameters or body of a stored procedure using this statement; to make such changes, you must drop and re-create the procedure using `DROP PROCEDURE` and `CREATE PROCEDURE`.

You must have the `ALTER ROUTINE` privilege for the procedure. By default, that privilege is granted automatically to the procedure creator. This behavior can be changed by disabling the

`automatic_sp_privileges` system variable. See Section 20.2.2, “Stored Routines and MySQL Privileges”.

13.1.6 ALTER SERVER Statement

```
ALTER SERVER  server_name
    OPTIONS (option [, option] ...)
```

Alters the server information for *`server_name`*, adjusting any of the options permitted in the `CREATE SERVER` statement. The corresponding fields in the `mysql.servers` table are updated accordingly. This statement requires the `SUPER` privilege.

For example, to update the `USER` option:

```
ALTER SERVER s OPTIONS (USER 'sally');
```

`ALTER SERVER` causes an implicit commit. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

`ALTER SERVER` is not written to the binary log, regardless of the logging format that is in use.

In MySQL 5.6.11 only, `gtid_next` must be set to `AUTOMATIC` before issuing this statement. (Bug #16062608, Bug #16715809, Bug #69045)

13.1.7 ALTER TABLE Statement

```
ALTER [ONLINE | OFFLINE] [IGNORE] TABLE tbl_name
    [alter_option [, alter_option] ...]
    [partition_options]

alter_option: {
    table_options
  | ADD [COLUMN] col_name column_definition
        [FIRST | AFTER col_name]
  | ADD [COLUMN] (col_name column_definition,...)
  | ADD {INDEX | KEY} [index_name]
        [index_type] (key_part,...) [index_option] ...
  | ADD {FULLTEXT | SPATIAL} [INDEX | KEY] [index_name]
        (key_part,...) [index_option] ...
  | ADD [CONSTRAINT [symbol]] PRIMARY KEY
        [index_type] (key_part,...)
        [index_option] ...
  | ADD [CONSTRAINT [symbol]] UNIQUE [INDEX | KEY]
        [index_name] [index_type] (key_part,...)
        [index_option] ...
  | ADD [CONSTRAINT [symbol]] FOREIGN KEY
        [index_name] (col_name,...)
        reference_definition
  | ADD CHECK (expr)
  | ALGORITHM [=] {DEFAULT | INPLACE | COPY}
  | ALTER [COLUMN] col_name {
        SET DEFAULT {literal | (expr)}
      | DROP DEFAULT
    }
  | CHANGE [COLUMN] old_col_name new_col_name column_definition
        [FIRST | AFTER col_name]
  | [DEFAULT] CHARACTER SET [=] charset_name [COLLATE [=] collation_name]
  | CONVERT TO CHARACTER SET charset_name [COLLATE collation_name]
  | {DISABLE | ENABLE} KEYS
  | {DISCARD | IMPORT} TABLESPACE
  | DROP [COLUMN] col_name
  | DROP {INDEX | KEY} index_name
  | DROP PRIMARY KEY
  | DROP FOREIGN KEY fk_symbol
  | FORCE
  | LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
  | MODIFY [COLUMN] col_name column_definition
        [FIRST | AFTER col_name]
  | ORDER BY col_name [, col_name] ...
```

```
  | RENAME [TO | AS] new_tbl_name
}

partition_options:
    partition_option [partition_option] ...

partition_option: {
    ADD PARTITION (partition_definition)
  | DROP PARTITION partition_names
  | TRUNCATE PARTITION {partition_names | ALL}
  | COALESCE PARTITION number
  | REORGANIZE PARTITION partition_names INTO (partition_definitions)
  | EXCHANGE PARTITION partition_name WITH TABLE tbl_name
  | ANALYZE PARTITION {partition_names | ALL}
  | CHECK PARTITION {partition_names | ALL}
  | OPTIMIZE PARTITION {partition_names | ALL}
  | REBUILD PARTITION {partition_names | ALL}
  | REPAIR PARTITION {partition_names | ALL}
  | REMOVE PARTITIONING
}

key_part:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH}

index_option: {
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
}

table_options:
    table_option [[,] table_option] ...

table_option: {
    AUTO_INCREMENT [=] value
  | AVG_ROW_LENGTH [=] value
  | [DEFAULT] CHARACTER SET [=] charset_name
  | CHECKSUM [=] {0 | 1}
  | [DEFAULT] COLLATE [=] collation_name
  | COMMENT [=] 'string'
  | CONNECTION [=] 'connect_string'
  | {DATA | INDEX} DIRECTORY [=] 'absolute path to directory'
  | DELAY_KEY_WRITE [=] {0 | 1}
  | ENGINE [=] engine_name
  | INSERT_METHOD [=] { NO | FIRST | LAST }
  | KEY_BLOCK_SIZE [=] value
  | MAX_ROWS [=] value
  | MIN_ROWS [=] value
  | PACK_KEYS [=] {0 | 1 | DEFAULT}
  | PASSWORD [=] 'string'
  | ROW_FORMAT [=] {DEFAULT | DYNAMIC | FIXED | COMPRESSED | REDUNDANT | COMPACT}
  | STATS_AUTO_RECALC [=] {DEFAULT | 0 | 1}
  | STATS_PERSISTENT [=] {DEFAULT | 0 | 1}
  | STATS_SAMPLE_PAGES [=] value
  | TABLESPACE tablespace_name [STORAGE {DISK | MEMORY}]
  | UNION [=] (tbl_name[,tbl_name]...)
}
```

`ALTER TABLE` changes the structure of a table. For example, you can add or delete columns, create or destroy indexes, change the type of existing columns, or rename columns or the table itself. You can also change characteristics such as the storage engine used for the table or the table comment.

- To use `ALTER TABLE`, you need `ALTER`, `CREATE`, and `INSERT` privileges for the table. Renaming a table requires `ALTER` and `DROP` on the old table, `ALTER`, `CREATE`, and `INSERT` on the new table.
- Following the table name, specify the alterations to be made. If none are given, `ALTER TABLE` does nothing.

- The syntax for many of the permissible alterations is similar to clauses of the `CREATE TABLE` statement. *`column_definition`* clauses use the same syntax for `ADD` and `CHANGE` as for `CREATE TABLE`. For more information, see Section 13.1.17, “CREATE TABLE Statement”.
- The word `COLUMN` is optional and can be omitted.
- Multiple `ADD`, `ALTER`, `DROP`, and `CHANGE` clauses are permitted in a single `ALTER TABLE` statement, separated by commas. This is a MySQL extension to standard SQL, which permits only one of each clause per `ALTER TABLE` statement. For example, to drop multiple columns in a single statement, do this:

```
ALTER TABLE t2 DROP COLUMN c, DROP COLUMN d;
```

- If a storage engine does not support an attempted `ALTER TABLE` operation, a warning may result. Such warnings can be displayed with `SHOW WARNINGS`. See Section 13.7.5.41, “SHOW WARNINGS Statement”. For information on troubleshooting `ALTER TABLE`, see Section B.3.6.1, “Problems with ALTER TABLE”.
- For usage examples, see Section 13.1.7.2, “ALTER TABLE Examples”.
- With the `mysql_info()` C API function, you can find out how many rows were copied by `ALTER TABLE`, and (when `IGNORE` is used) how many rows were deleted due to duplication of unique key values. See mysql_info().

There are several additional aspects to the `ALTER TABLE` statement, described under the following topics in this section:

- Table Options
- Performance and Space Requirements
- Concurrency Control
- Adding and Dropping Columns
- Renaming, Redefining, and Reordering Columns
- Primary Keys and Indexes
- Foreign Keys and Other Constraints
- Changing the Character Set
- Discarding and Importing InnoDB Tablespaces
- Row Order for MyISAM Tables
- Partitioning Options

Table Options

`table_options` signifies table options of the kind that can be used in the `CREATE TABLE` statement, such as `ENGINE`, `AUTO_INCREMENT`, `AVG_ROW_LENGTH`, `MAX_ROWS`, or `ROW_FORMAT`.

For descriptions of all table options, see Section 13.1.17, “CREATE TABLE Statement”. However, `ALTER TABLE` ignores `DATA DIRECTORY` and `INDEX DIRECTORY` when given as table options. `ALTER TABLE` permits them only as partitioning options, and, as of MySQL 5.6.35, requires that you have the `FILE` privilege.

Use of table options with `ALTER TABLE` provides a convenient way of altering single table characteristics. For example:

- If `t1` is currently not an `InnoDB` table, this statement changes its storage engine to `InnoDB`:

```
ALTER TABLE t1 ENGINE = InnoDB;
```

  - See Section 14.6.1.5, “Converting Tables from MyISAM to InnoDB” for considerations when switching tables to the `InnoDB` storage engine.
  - When you specify an `ENGINE` clause, `ALTER TABLE` rebuilds the table. This is true even if the table already has the specified storage engine.
  - Running `ALTER TABLE tbl_name ENGINE=INNODB` on an existing `InnoDB` table performs a “null” `ALTER TABLE` operation, which can be used to defragment an `InnoDB` table, as described in Section 14.12.4, “Defragmenting a Table”. Running `ALTER TABLE tbl_name FORCE` on an `InnoDB` table performs the same function.
  - As of MySQL 5.6.17, `ALTER TABLE tbl_name ENGINE=INNODB` and `ALTER TABLE tbl_name FORCE` use online DDL. For more information, see Section 14.13, “InnoDB and Online DDL”.
  - The outcome of attempting to change the storage engine of a table is affected by whether the desired storage engine is available and the setting of the `NO_ENGINE_SUBSTITUTION` SQL mode, as described in Section 5.1.10, “Server SQL Modes”.
  - To prevent inadvertent loss of data, `ALTER TABLE` cannot be used to change the storage engine of a table to `MERGE` or `BLACKHOLE`.

- To change the `InnoDB` table to use compressed row-storage format:

```
ALTER TABLE t1 ROW_FORMAT = COMPRESSED;
```

- To reset the current auto-increment value:

```
ALTER TABLE t1 AUTO_INCREMENT = 13;
```

  You cannot reset the counter to a value less than or equal to the value that is currently in use. For both `InnoDB` and `MyISAM`, if the value is less than or equal to the maximum value currently in the `AUTO_INCREMENT` column, the value is reset to the current maximum `AUTO_INCREMENT` column value plus one.

- To change the default table character set:

```
ALTER TABLE t1 CHARACTER SET = utf8;
```

  See also Changing the Character Set.

- To add (or change) a table comment:

```
ALTER TABLE t1 COMMENT = 'New table comment';
```

To verify that the table options were changed as intended, use `SHOW CREATE TABLE`, or query the `INFORMATION_SCHEMA.TABLES` table.

Performance and Space Requirements

`ALTER TABLE` operations are processed using one of the following algorithms:

- `COPY`: Operations are performed on a copy of the original table, and table data is copied from the original table to the new table row by row. Concurrent DML is not permitted.
- `INPLACE`: Operations avoid copying table data but may rebuild the table in place. An exclusive metadata lock on the table may be taken briefly during preparation and execution phases of the operation. Typically, concurrent DML is supported.

The `ALGORITHM` clause is optional. If the `ALGORITHM` clause is omitted (or if you specify `ALGORITHM=DEFAULT`), MySQL uses `ALGORITHM=INPLACE` for storage engines and `ALTER TABLE` clauses that support it. Otherwise, `ALGORITHM=COPY` is used.

Note

`ALGORITHM=DEFAULT`, `ALGORITHM=INPLACE`, and `ALGORITHM=COPY` are not supported for `ALTER TABLE` operations on partitioned tables. MySQL 5.6.11 and later supports `ALGORITHM=1` and `ALGORITHM=2` for use with upgrades and downgrades; see Section 13.1.7.1, "ALTER TABLE Partition Operations", for more information.

Specifying an `ALGORITHM` clause requires the operation to use the specified algorithm for clauses and storage engines that support it, or fail with an error otherwise. Specifying `ALGORITHM=DEFAULT` is the same as omitting the `ALGORITHM` clause.

`ALTER TABLE` operations that use the `COPY` algorithm wait for other operations that are modifying the table to complete. After alterations are applied to the table copy, data is copied over, the original table is deleted, and the table copy is renamed to the name of the original table. While the `ALTER TABLE` operation executes, the original table is readable by other sessions (with the exception noted shortly). Updates and writes to the table started after the `ALTER TABLE` operation begins are stalled until the new table is ready, then are automatically redirected to the new table. The temporary copy of the table is created in the database directory of the original table unless it is a `RENAME TO` operation that moves the table to a database that resides in a different directory.

The exception referred to earlier is that `ALTER TABLE` blocks reads (not just writes) at the point where it is ready to install a new version of the table `.frm` file, discard the old file, and clear outdated table structures from the table and table definition caches. At this point, it must acquire an exclusive lock. To do so, it waits for current readers to finish, and blocks new reads and writes.

An `ALTER TABLE` operation that uses the `COPY` algorithm prevents concurrent DML operations. Concurrent queries are still allowed. That is, a table-copying operation always includes at least the concurrency restrictions of `LOCK=SHARED` (allow queries but not DML). You can further restrict concurrency for operations that support the `LOCK` clause by specifying `LOCK=EXCLUSIVE`, which prevents DML and queries. For more information, see Concurrency Control.

Note

In MySQL 5.6, `LOCK` is not supported for `ALTER TABLE` operations on partitioned tables.

To force use of the `COPY` algorithm for an `ALTER TABLE` operation that would otherwise not use it, enable the `old_alter_table` system variable or specify `ALGORITHM=COPY`. If there is a conflict between the `old_alter_table` setting and an `ALGORITHM` clause with a value other than `DEFAULT`, the `ALGORITHM` clause takes precedence.

For `InnoDB` tables, an `ALTER TABLE` operation that uses the `COPY` algorithm on a table that resides in a shared tablespace such as the system tablespace can increase the amount of space used by the tablespace. Such operations require as much additional space as the data in the table plus indexes. For a table residing in a shared tablespace, the additional space used during the operation is not released back to the operating system as it is for a table that resides in a file-per-table tablespace.

For information about space requirements for online DDL operations, see Section 14.13.3, "Online DDL Space Requirements".

`ALTER TABLE` operations that use the `INPLACE` algorithm include:

- `ALTER TABLE` operations supported by the `InnoDB` online DDL feature. See Section 14.13.1, "Online DDL Operations".
- Renaming a table. MySQL renames files that correspond to the table *`tbl_name`* without making a copy. (You can also use the `RENAME TABLE` statement to rename tables. See Section 13.1.32, "RENAME TABLE Statement".) Privileges granted specifically for the renamed table are not migrated to the new name. They must be changed manually.

- Operations that only modify table metadata. These operations are immediate because the server only alters the table `.frm` file, not touch table contents. Metadata-only operations include:
  - Renaming a column.
  - Changing the default value of a column (except for `NDB` tables; see Limitations of NDB online operations).
  - Modifying the definition of an `ENUM` or `SET` column by adding new enumeration or set members to the *end* of the list of valid member values, as long as the storage size of the data type does not change. For example, adding a member to a `SET` column that has 8 members changes the required storage per value from 1 byte to 2 bytes; this requires a table copy. Adding members in the middle of the list causes renumbering of existing members, which requires a table copy.
- Adding or dropping a secondary index, for `InnoDB` and `NDB` tables. See Section 14.13, "InnoDB and Online DDL".
- For `NDB` tables, operations that add and drop indexes on variable-width columns. These operations occur online, without table copying and without blocking concurrent DML actions for most of their duration. See Section 18.5.11, "Online Operations with ALTER TABLE in NDB Cluster".

As of MySQL 5.6.16, `ALTER TABLE` upgrades MySQL 5.5 temporal columns to 5.6 format for `ADD COLUMN`, `CHANGE COLUMN`, `MODIFY COLUMN`, `ADD INDEX`, and `FORCE` operations. This conversion cannot be done using the `INPLACE` algorithm because the table must be rebuilt, so specifying `ALGORITHM=INPLACE` in these cases results in an error. Specify `ALGORITHM=COPY` if necessary.

NDB Cluster supports online `ALTER TABLE` operations using the `ALGORITHM=INPLACE` syntax in MySQL NDB Cluster 7.3 and later. NDB Cluster also supports an older syntax specific to NDB that uses the `ONLINE` and `OFFLINE` keywords. These keywords are deprecated beginning with MySQL NDB Cluster 7.3; they continue to be supported in MySQL NDB Cluster 7.4, but are subject to removal in a future version of NDB Cluster. See Section 18.5.11, "Online Operations with ALTER TABLE in NDB Cluster", for the exact syntax and other particulars.

If an `ALTER TABLE` operation on a multicolumn index used to partition a table by `KEY` changes the order of the columns, it can only be performed using `ALGORITHM=COPY`.

MySQL NDB Cluster 7.3 (and later) supports online operations using the same `ALGORITHM=INPLACE` syntax used with the standard MySQL Server. NDB Cluster 7.3 and 7.4 also continue to support online `ALTER TABLE` operations using the `ONLINE` and `OFFLINE` keywords for backwards compatibility, but these keywords are deprecated in NDB 7.3 and 7.4, and no longer supported as of NDB 7.5. `NDB` does not support changing a tablespace online. See Section 18.5.11, "Online Operations with ALTER TABLE in NDB Cluster", for more information.

`ALTER TABLE` with `ADD PARTITION`, `DROP PARTITION`, `COALESCE PARTITION`, `REBUILD PARTITION`, or `REORGANIZE PARTITION` does not create temporary tables (except when used with `NDB` tables); however, these operations can and do create temporary partition files.

`ADD` or `DROP` operations for `RANGE` or `LIST` partitions are immediate operations or nearly so. `ADD` or `COALESCE` operations for `HASH` or `KEY` partitions copy data between all partitions, unless `LINEAR HASH` or `LINEAR KEY` was used; this is effectively the same as creating a new table, although the `ADD` or `COALESCE` operation is performed partition by partition. `REORGANIZE` operations copy only changed partitions and do not touch unchanged ones.

For `MyISAM` tables, you can speed up index re-creation (the slowest part of the alteration process) by setting the `myisam_sort_buffer_size` system variable to a high value.

Concurrency Control

For `ALTER TABLE` operations that support it, you can use the `LOCK` clause to control the level of concurrent reads and writes on a table while it is being altered. Specifying a non-default value for this clause enables you to require a certain amount of concurrent access or exclusivity during the alter

operation, and halts the operation if the requested degree of locking is not available. The parameters for the `LOCK` clause are:

- `LOCK = DEFAULT`

  Maximum level of concurrency for the given `ALGORITHM` clause (if any) and `ALTER TABLE` operation: Permit concurrent reads and writes if supported. If not, permit concurrent reads if supported. If not, enforce exclusive access.

- `LOCK = NONE`

  If supported, permit concurrent reads and writes. Otherwise, an error occurs.

- `LOCK = SHARED`

  If supported, permit concurrent reads but block writes. Writes are blocked even if concurrent writes are supported by the storage engine for the given `ALGORITHM` clause (if any) and `ALTER TABLE` operation. If concurrent reads are not supported, an error occurs.

- `LOCK = EXCLUSIVE`

  Enforce exclusive access. This is done even if concurrent reads/writes are supported by the storage engine for the given `ALGORITHM` clause (if any) and `ALTER TABLE` operation.

Note

Pending `INSERT DELAYED` statements are lost if a table is write locked and `ALTER TABLE` is used to modify the table structure.

Adding and Dropping Columns

Use `ADD` to add new columns to a table, and `DROP` to remove existing columns. `DROP col_name` is a MySQL extension to standard SQL.

To add a column at a specific position within a table row, use `FIRST` or `AFTER col_name`. The default is to add the column last.

If a table contains only one column, the column cannot be dropped. If what you intend is to remove the table, use the `DROP TABLE` statement instead.

If columns are dropped from a table, the columns are also removed from any index of which they are a part. If all columns that make up an index are dropped, the index is dropped as well.

Renaming, Redefining, and Reordering Columns

The `CHANGE`, `MODIFY`, and `ALTER` clauses enable the names and definitions of existing columns to be altered. They have these comparative characteristics:

- `CHANGE`:
  - Can rename a column and change its definition, or both.
  - Has more capability than `MODIFY`, but at the expense of convenience for some operations. `CHANGE` requires naming the column twice if not renaming it.
  - With `FIRST` or `AFTER`, can reorder columns.
- `MODIFY`:
  - Can change a column definition but not its name.
  - More convenient than `CHANGE` to change a column definition without renaming it.
  - With `FIRST` or `AFTER`, can reorder columns.

- `ALTER`: Used only to change a column default value.

`CHANGE` is a MySQL extension to standard SQL. `MODIFY` is a MySQL extension for Oracle compatibility.

To alter a column to change both its name and definition, use `CHANGE`, specifying the old and new names and the new definition. For example, to rename an `INT NOT NULL` column from `a` to `b` and change its definition to use the `BIGINT` data type while retaining the `NOT NULL` attribute, do this:

```
ALTER TABLE t1 CHANGE a b BIGINT NOT NULL;
```

To change a column definition but not its name, use `CHANGE` or `MODIFY`. With `CHANGE`, the syntax requires two column names, so you must specify the same name twice to leave the name unchanged. For example, to change the definition of column `b`, do this:

```
ALTER TABLE t1 CHANGE b b INT NOT NULL;
```

`MODIFY` is more convenient to change the definition without changing the name because it requires the column name only once:

```
ALTER TABLE t1 MODIFY b INT NOT NULL;
```

To change a column name but not its definition, use `CHANGE`. The syntax requires a column definition, so to leave the definition unchanged, you must respecify the definition the column currently has. For example, to rename an `INT NOT NULL` column from `b` to `a`, do this:

```
ALTER TABLE t1 CHANGE b a INT NOT NULL;
```

For column definition changes using `CHANGE` or `MODIFY`, the definition must include the data type and all attributes that should apply to the new column, other than index attributes such as `PRIMARY KEY` or `UNIQUE`. Attributes present in the original definition but not specified for the new definition are not carried forward. Suppose that a column `col1` is defined as `INT UNSIGNED DEFAULT 1 COMMENT 'my column'` and you modify the column as follows, intending to change only `INT` to `BIGINT`:

```
ALTER TABLE t1 MODIFY col1 BIGINT;
```

That statement changes the data type from `INT` to `BIGINT`, but it also drops the `UNSIGNED`, `DEFAULT`, and `COMMENT` attributes. To retain them, the statement must include them explicitly:

```
ALTER TABLE t1 MODIFY col1 BIGINT UNSIGNED DEFAULT 1 COMMENT 'my column';
```

For data type changes using `CHANGE` or `MODIFY`, MySQL tries to convert existing column values to the new type as well as possible.

Warning

This conversion may result in alteration of data. For example, if you shorten a string column, values may be truncated. To prevent the operation from succeeding if conversions to the new data type would result in loss of data, enable strict SQL mode before using `ALTER TABLE` (see Section 5.1.10, “Server SQL Modes”).

If you use `CHANGE` or `MODIFY` to shorten a column for which an index exists on the column, and the resulting column length is less than the index length, MySQL shortens the index automatically.

For columns renamed by `CHANGE`, MySQL automatically renames these references to the renamed column:

- Indexes that refer to the old column, including indexes and disabled `MyISAM` indexes.
- Foreign keys that refer to the old column.

For columns renamed by `CHANGE`, MySQL does not automatically rename these references to the renamed column:

- Partition expressions that refer to the renamed column. You must use CHANGE to redefine such expressions in the same ALTER TABLE statement as the one that renames the column.
- Views and stored programs that refer to the renamed column. You must manually alter the definition of these objects to refer to the new column name.

To reorder columns within a table, use FIRST and AFTER in CHANGE or MODIFY operations.

ALTER ... SET DEFAULT or ALTER ... DROP DEFAULT specify a new default value for a column or remove the old default value, respectively. If the old default is removed and the column can be NULL, the new default is NULL. If the column cannot be NULL, MySQL assigns a default value as described in Section 11.5, “Data Type Default Values”.

Primary Keys and Indexes

DROP PRIMARY KEY drops the primary key. If there is no primary key, an error occurs. For information about the performance characteristics of primary keys, especially for InnoDB tables, see Section 8.3.2, “Primary Key Optimization”.

If you add a UNIQUE INDEX or PRIMARY KEY to a table, MySQL stores it before any nonunique index to permit detection of duplicate keys as early as possible.

IGNORE is a MySQL extension to standard SQL. It controls how ALTER TABLE works if there are duplicates on unique keys in the new table or if warnings occur when strict mode is enabled. If IGNORE is not specified, the copy is aborted and rolled back if duplicate-key errors occur. If IGNORE is specified, only one row is used of rows with duplicates on a unique key. The other conflicting rows are deleted. Incorrect values are truncated to the closest matching acceptable value.

As of MySQL 5.6.17, the IGNORE clause is deprecated and its use generates a warning. IGNORE is removed in MySQL 5.7.

DROP INDEX removes an index. This is a MySQL extension to standard SQL. See Section 13.1.24, “DROP INDEX Statement”. To determine index names, use SHOW INDEX FROM *tbl_name*.

Some storage engines permit you to specify an index type when creating an index. The syntax for the *index_type* specifier is USING *type_name*. For details about USING, see Section 13.1.13, “CREATE INDEX Statement”. The preferred position is after the column list. Expect support for use of the option before the column list to be removed in a future MySQL release.

index_option values specify additional options for an index. For details about permissible *index_option* values, see Section 13.1.13, “CREATE INDEX Statement”.

If you use ALTER TABLE on a MyISAM table, all nonunique indexes are created in a separate batch (as for REPAIR TABLE). This should make ALTER TABLE much faster when you have many indexes.

For MyISAM tables, key updating can be controlled explicitly. Use ALTER TABLE ... DISABLE KEYS to tell MySQL to stop updating nonunique indexes. Then use ALTER TABLE ... ENABLE KEYS to re-create missing indexes. MyISAM does this with a special algorithm that is much faster than inserting keys one by one, so disabling keys before performing bulk insert operations should give a considerable speedup. Using ALTER TABLE ... DISABLE KEYS requires the INDEX privilege in addition to the privileges mentioned earlier.

While the nonunique indexes are disabled, they are ignored for statements such as SELECT and EXPLAIN that otherwise would use them.

After an ALTER TABLE statement, it may be necessary to run ANALYZE TABLE to update index cardinality information. See Section 13.7.5.23, “SHOW INDEX Statement”.

Foreign Keys and Other Constraints

The FOREIGN KEY and REFERENCES clauses are supported by the InnoDB and NDB storage engines, which implement ADD [CONSTRAINT [*symbol*]] FOREIGN KEY [*index_name*] (...)

`REFERENCES ... (...)`. See Section 13.1.17.5, “FOREIGN KEY Constraints”. For other storage engines, the clauses are parsed but ignored.

The `CHECK` constraint clause is parsed but ignored by all storage engines. See Section 13.1.17, “CREATE TABLE Statement”. The reason for accepting but ignoring syntax clauses is for compatibility, to make it easier to port code from other SQL servers, and to run applications that create tables with references. See Section 1.7.2, “MySQL Differences from Standard SQL”.

For `ALTER TABLE`, unlike `CREATE TABLE`, `ADD FOREIGN KEY` ignores *`index_name`* if given and uses an automatically generated foreign key name. As a workaround, include the `CONSTRAINT` clause to specify the foreign key name:

```
ADD CONSTRAINT name FOREIGN KEY (....) ...
```

Important

MySQL silently ignores inline `REFERENCES` specifications, where the references are defined as part of the column specification. MySQL accepts only `REFERENCES` clauses defined as part of a separate `FOREIGN KEY` specification.

Note

Partitioned `InnoDB` tables do not support foreign keys. This restriction does not apply to `NDB` tables, including those explicitly partitioned by `[LINEAR] KEY`. For more information, see Section 19.6.2, “Partitioning Limitations Relating to Storage Engines”.

The `InnoDB` and `NDB` storage engines support the use of `ALTER TABLE` to drop foreign keys:

```
ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;
```

Adding and dropping a foreign key in the same `ALTER TABLE` statement is supported for `ALTER TABLE ... ALGORITHM=INPLACE` but not for `ALTER TABLE ... ALGORITHM=COPY`.

The server prohibits changes to foreign key columns that have the potential to cause loss of referential integrity. A workaround is to use `ALTER TABLE ... DROP FOREIGN KEY` before changing the column definition and `ALTER TABLE ... ADD FOREIGN KEY` afterward. Examples of prohibited changes include:

- Changes to the data type of foreign key columns that may be unsafe. For example, changing `VARCHAR(20)` to `VARCHAR(30)` is permitted, but changing it to `VARCHAR(1024)` is not because that alters the number of length bytes required to store individual values.
- Changing a `NULL` column to `NOT NULL` in non-strict mode is prohibited to prevent converting `NULL` values to default non-`NULL` values, for which there are no corresponding values in the referenced table. The operation is permitted in strict mode, but an error is returned if any such conversion is required.

`ALTER TABLE tbl_name RENAME new_tbl_name` changes internally generated foreign key constraint names and user-defined foreign key constraint names that begin with the string “*`tbl_name`*_ibfk_” to reflect the new table name. `InnoDB` interprets foreign key constraint names that begin with the string “*`tbl_name`*_ibfk_” as internally generated names.

Changing the Character Set

To change the table default character set and all character columns (`CHAR`, `VARCHAR`, `TEXT`) to a new character set, use a statement like this:

```
ALTER TABLE tbl_name
CONVERT TO CHARACTER SET charset_name [COLLATE collation_name];
```

The statement also changes the collation of all character columns. If you specify no `COLLATE` clause to indicate which collation to use, the statement uses default collation for the character set. If this collation is inappropriate for the intended table use (for example, if it would change from a case-sensitive collation to a case-insensitive collation), specify a collation explicitly.

For a column that has a data type of `VARCHAR` or one of the `TEXT` types, `CONVERT TO CHARACTER SET` changes the data type as necessary to ensure that the new column is long enough to store as many characters as the original column. For example, a `TEXT` column has two length bytes, which store the byte-length of values in the column, up to a maximum of 65,535. For a `latin1 TEXT` column, each character requires a single byte, so the column can store up to 65,535 characters. If the column is converted to `utf8`, each character might require up to three bytes, for a maximum possible length of 3 × 65,535 = 196,605 bytes. That length does not fit in a `TEXT` column's length bytes, so MySQL converts the data type to `MEDIUMTEXT`, which is the smallest string type for which the length bytes can record a value of 196,605. Similarly, a `VARCHAR` column might be converted to `MEDIUMTEXT`.

To avoid data type changes of the type just described, do not use `CONVERT TO CHARACTER SET`. Instead, use `MODIFY` to change individual columns. For example:

```
ALTER TABLE t MODIFY latin1_text_col TEXT CHARACTER SET utf8;
ALTER TABLE t MODIFY latin1_varchar_col VARCHAR(M) CHARACTER SET utf8;
```

If you specify `CONVERT TO CHARACTER SET binary`, the `CHAR`, `VARCHAR`, and `TEXT` columns are converted to their corresponding binary string types (`BINARY`, `VARBINARY`, `BLOB`). This means that the columns no longer have a character set and a subsequent `CONVERT TO` operation does not apply to them.

If *`charset_name`* is `DEFAULT` in a `CONVERT TO CHARACTER SET` operation, the character set named by the `character_set_database` system variable is used.

Warning

The `CONVERT TO` operation converts column values between the original and named character sets. This is *not* what you want if you have a column in one character set (like `latin1`) but the stored values actually use some other, incompatible character set (like `utf8`). In this case, you have to do the following for each such column:

```
ALTER TABLE t1 CHANGE c1 c1 BLOB;
ALTER TABLE t1 CHANGE c1 c1 TEXT CHARACTER SET utf8;
```

The reason this works is that there is no conversion when you convert to or from `BLOB` columns.

To change only the *default* character set for a table, use this statement:

```
ALTER TABLE tbl_name DEFAULT CHARACTER SET charset_name;
```

The word `DEFAULT` is optional. The default character set is the character set that is used if you do not specify the character set for columns that you add to a table later (for example, with `ALTER TABLE ... ADD column`).

When the `foreign_key_checks` system variable is enabled, which is the default setting, character set conversion is not permitted on tables that include a character string column used in a foreign key constraint. The workaround is to disable `foreign_key_checks` before performing the character set conversion. You must perform the conversion on both tables involved in the foreign key constraint before re-enabling `foreign_key_checks`. If you re-enable `foreign_key_checks` after converting only one of the tables, an `ON DELETE CASCADE` or `ON UPDATE CASCADE` operation could corrupt data in the referencing table due to implicit conversion that occurs during these operations (Bug #45290, Bug #74816).

Discarding and Importing InnoDB Tablespaces

An InnoDB table created in its own file-per-table tablespace can be imported from a backup or from another MySQL server instance using the DISCARD TABLEPACE and IMPORT TABLESPACE clauses. See Section 14.6.1.3, “Importing InnoDB Tables”.

Row Order for MyISAM Tables

ORDER BY enables you to create the new table with the rows in a specific order. This option is useful primarily when you know that you query the rows in a certain order most of the time. By using this option after major changes to the table, you might be able to get higher performance. In some cases, it might make sorting easier for MySQL if the table is in order by the column that you want to order it by later.

Note

The table does not remain in the specified order after inserts and deletes.

ORDER BY syntax permits one or more column names to be specified for sorting, each of which optionally can be followed by ASC or DESC to indicate ascending or descending sort order, respectively. The default is ascending order. Only column names are permitted as sort criteria; arbitrary expressions are not permitted. This clause should be given last after any other clauses.

ORDER BY does not make sense for InnoDB tables because InnoDB always orders table rows according to the clustered index.

When used on a partitioned table, ALTER TABLE ... ORDER BY orders rows within each partition only.

Partitioning Options

partition_options signifies options that can be used with partitioned tables for repartitioning, to add, drop, discard, merge, and split partitions, and to perform partitioning maintenance.

It is possible for an ALTER TABLE statement to contain a PARTITION BY or REMOVE PARTITIONING clause in an addition to other alter specifications, but the PARTITION BY or REMOVE PARTITIONING clause must be specified last after any other specifications. The ADD PARTITION, DROP PARTITION, COALESCE PARTITION, REORGANIZE PARTITION, EXCHANGE PARTITION, ANALYZE PARTITION, CHECK PARTITION, and REPAIR PARTITION options cannot be combined with other alter specifications in a single ALTER TABLE, since the options just listed act on individual partitions.

For more information about partition options, see Section 13.1.17, “CREATE TABLE Statement”, and Section 13.1.7.1, “ALTER TABLE Partition Operations”. For information about and examples of ALTER TABLE ... EXCHANGE PARTITION statements, see Section 19.3.3, “Exchanging Partitions and Subpartitions with Tables”.

13.1.7.1 ALTER TABLE Partition Operations

Partitioning-related clauses for ALTER TABLE can be used with partitioned tables for repartitioning, to add, drop, discard, merge, and split partitions, and to perform partitioning maintenance.

- Simply using a *partition_options* clause with ALTER TABLE on a partitioned table repartitions the table according to the partitioning scheme defined by the *partition_options*. This clause always begins with PARTITION BY, and follows the same syntax and other rules as apply to the *partition_options* clause for CREATE TABLE (for more detailed information, see Section 13.1.17, “CREATE TABLE Statement”), and can also be used to partition an existing table that is not already partitioned. For example, consider a (nonpartitioned) table defined as shown here:

```
CREATE TABLE t1 (
    id INT,
    year_col INT
);
```

This table can be partitioned by HASH, using the id column as the partitioning key, into 8 partitions by means of this statement:

```
ALTER TABLE t1
    PARTITION BY HASH(id)
    PARTITIONS 8;
```

MySQL 5.6.11 and later supports an ALGORITHM option with [SUB]PARTITION BY [LINEAR] KEY. ALGORITHM=1 causes the server to use the same key-hashing functions as MySQL 5.1 when computing the placement of rows in partitions; ALGORITHM=2 means that the server employs the key-hashing functions implemented and used by default for new KEY partitioned tables in MySQL 5.5 and later. (Partitioned tables created with the key-hashing functions employed in MySQL 5.5 and later cannot be used by a MySQL 5.1 server.) Not specifying the option has the same effect as using ALGORITHM=2. This option is intended for use chiefly when upgrading or downgrading [LINEAR] KEY partitioned tables between MySQL 5.1 and later MySQL versions, or for creating tables partitioned by KEY or LINEAR KEY on a MySQL 5.5 or later server which can be used on a MySQL 5.1 server.

To upgrade a KEY partitioned table that was created in MySQL 5.1, first execute SHOW CREATE TABLE and note the exact columns and number of partitions shown. Now execute an ALTER TABLE statement using exactly the same column list and number of partitions as in the CREATE TABLE statement, while adding ALGORITHM=2 immediately following the PARTITION BY keywords. (You should also include the LINEAR keyword if it was used for the original table definition.) An example from a session in the mysql client is shown here:

```
mysql> SHOW CREATE TABLE p\G
*************************** 1. row ***************************
       Table: p
Create Table: CREATE TABLE `p` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `cd` datetime NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
/*!50100 PARTITION BY LINEAR KEY (id)
PARTITIONS 32 */
1 row in set (0.00 sec)

mysql> ALTER TABLE p PARTITION BY LINEAR KEY ALGORITHM=2 (id) PARTITIONS 32;
Query OK, 0 rows affected (5.34 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW CREATE TABLE p\G
*************************** 1. row ***************************
       Table: p
Create Table: CREATE TABLE `p` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `cd` datetime NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
/*!50100 PARTITION BY LINEAR KEY (id)
PARTITIONS 32 */
1 row in set (0.00 sec)
```

Downgrading a table created using the default key-hashing used in MySQL 5.5 and later to enable its use by a MySQL 5.1 server is similar, except in this case you should use ALGORITHM=1 to force the table's partitions to be rebuilt using the MySQL 5.1 key-hashing functions. It is recommended that you not do this except when necessary for compatibility with a MySQL 5.1 server, as the improved KEY hashing functions used by default in MySQL 5.5 and later provide fixes for a number of issues found in the older implementation.

Note

A table upgraded by means of ALTER TABLE ... PARTITION BY ALGORITHM=2 [LINEAR] KEY ... can no longer be used by a MySQL 5.1

> server. (Such a table would need to be downgraded with `ALTER TABLE ... PARTITION BY ALGORITHM=1 [LINEAR] KEY ...` before it could be used again by a MySQL 5.1 server.)

The table that results from using an `ALTER TABLE ... PARTITION BY` statement must follow the same rules as one created using `CREATE TABLE ... PARTITION BY`. This includes the rules governing the relationship between any unique keys (including any primary key) that the table might have, and the column or columns used in the partitioning expression, as discussed in Section 19.6.1, "Partitioning Keys, Primary Keys, and Unique Keys". The `CREATE TABLE ... PARTITION BY` rules for specifying the number of partitions also apply to `ALTER TABLE ... PARTITION BY`.

The *`partition_definition`* clause for `ALTER TABLE ADD PARTITION` supports the same options as the clause of the same name for the `CREATE TABLE` statement. (See Section 13.1.17, "CREATE TABLE Statement", for the syntax and description.) Suppose that you have the partitioned table created as shown here:

```
CREATE TABLE t1 (
    id INT,
    year_col INT
)
PARTITION BY RANGE (year_col) (
    PARTITION p0 VALUES LESS THAN (1991),
    PARTITION p1 VALUES LESS THAN (1995),
    PARTITION p2 VALUES LESS THAN (1999)
);
```

You can add a new partition `p3` to this table for storing values less than `2002` as follows:

```
ALTER TABLE t1 ADD PARTITION (PARTITION p3 VALUES LESS THAN (2002));
```

`DROP PARTITION` can be used to drop one or more `RANGE` or `LIST` partitions. This statement cannot be used with `HASH` or `KEY` partitions; instead, use `COALESCE PARTITION` (see below). Any data that was stored in the dropped partitions named in the *`partition_names`* list is discarded. For example, given the table `t1` defined previously, you can drop the partitions named `p0` and `p1` as shown here:

```
ALTER TABLE t1 DROP PARTITION p0, p1;
```

> **Note**
>
> `DROP PARTITION` does not work with tables that use the `NDB` storage engine. See Section 19.3.1, "Management of RANGE and LIST Partitions", and Section 18.1.7, "Known Limitations of NDB Cluster".

`ADD PARTITION` and `DROP PARTITION` do not currently support `IF [NOT] EXISTS`.

Renames of partitioned tables are supported. You can rename individual partitions indirectly using `ALTER TABLE ... REORGANIZE PARTITION`; however, this operation copies the partition's data.

To delete rows from selected partitions, use the `TRUNCATE PARTITION` option. This option takes a list of one or more comma-separated partition names. For example, consider the table `t1` as defined here:

```
CREATE TABLE t1 (
    id INT,
    year_col INT
)
PARTITION BY RANGE (year_col) (
    PARTITION p0 VALUES LESS THAN (1991),
    PARTITION p1 VALUES LESS THAN (1995),
    PARTITION p2 VALUES LESS THAN (1999),
    PARTITION p3 VALUES LESS THAN (2003),
    PARTITION p4 VALUES LESS THAN (2007)
```

```
);
```

To delete all rows from partition `p0`, use the following statement:

```
ALTER TABLE t1 TRUNCATE PARTITION p0;
```

The statement just shown has the same effect as the following `DELETE` statement:

```
DELETE FROM t1 WHERE year_col < 1991;
```

When truncating multiple partitions, the partitions do not have to be contiguous: This can greatly simplify delete operations on partitioned tables that would otherwise require very complex `WHERE` conditions if done with `DELETE` statements. For example, this statement deletes all rows from partitions `p1` and `p3`:

```
ALTER TABLE t1 TRUNCATE PARTITION p1, p3;
```

An equivalent `DELETE` statement is shown here:

```
DELETE FROM t1 WHERE
    (year_col >= 1991 AND year_col < 1995)
    OR
    (year_col >= 2003 AND year_col < 2007);
```

If you use the `ALL` keyword in place of the list of partition names, the statement acts on all table partitions.

`TRUNCATE PARTITION` merely deletes rows; it does not alter the definition of the table itself, or of any of its partitions.

Note

`TRUNCATE PARTITION` does not work with subpartitions.

To verify that the rows were dropped, check the `INFORMATION_SCHEMA.PARTITIONS` table, using a query such as this one:

```
SELECT PARTITION_NAME, TABLE_ROWS
    FROM INFORMATION_SCHEMA.PARTITIONS
    WHERE TABLE_NAME = 't1';
```

`TRUNCATE PARTITION` is supported only for partitioned tables that use the `MyISAM`, `InnoDB`, or `MEMORY` storage engine. It also works on `BLACKHOLE` tables (but has no effect). It is not supported for `ARCHIVE` tables.

`COALESCE PARTITION` can be used with a table that is partitioned by `HASH` or `KEY` to reduce the number of partitions by *`number`*. Suppose that you have created table `t2` as follows:

```
CREATE TABLE t2 (
    name VARCHAR (30),
    started DATE
)
PARTITION BY HASH( YEAR(started) )
```

```
PARTITIONS 6;
```

To reduce the number of partitions used by `t2` from 6 to 4, use the following statement:

```
ALTER TABLE t2 COALESCE PARTITION 2;
```

The data contained in the last *`number`* partitions is merged into the remaining partitions. In this case, partitions 4 and 5 are merged into the first 4 partitions (the partitions numbered 0, 1, 2, and 3).

To change some but not all the partitions used by a partitioned table, you can use `REORGANIZE PARTITION`. This statement can be used in several ways:

- To merge a set of partitions into a single partition. This is done by naming several partitions in the *`partition_names`* list and supplying a single definition for *`partition_definition`*.
- To split an existing partition into several partitions. Accomplish this by naming a single partition for *`partition_names`* and providing multiple *`partition_definitions`*.
- To change the ranges for a subset of partitions defined using `VALUES LESS THAN` or the value lists for a subset of partitions defined using `VALUES IN`.
- This statement may also be used without the *`partition_names`* `INTO (`*`partition_definitions`*`)` option on tables that are automatically partitioned using `HASH` partitioning to force redistribution of data. (Currently, only `NDB` tables are automatically partitioned in this way.) This is useful in NDB Cluster where, after you have added new NDB Cluster data nodes online to an existing NDB Cluster, you wish to redistribute existing NDB Cluster table data to the new data nodes. In such cases, you should invoke the statement with the `ONLINE` option; in other words, as shown here:

  ```
  ALTER ONLINE TABLE table REORGANIZE PARTITION;
  ```

  You cannot perform other DDL concurrently with online table reorganization—that is, no other DDL statements can be issued while an `ALTER ONLINE TABLE ... REORGANIZE PARTITION` statement is executing. For more information about adding NDB Cluster data nodes online, see Section 18.5.7, “Adding NDB Cluster Data Nodes Online”.

  `ALTER ONLINE TABLE ... REORGANIZE PARTITION` does not work with tables which were created using the `MAX_ROWS` option, because it uses the constant `MAX_ROWS` value specified in the original `CREATE TABLE` statement to determine the number of partitions required, so no new partitions are created. Instead, you can use `ALTER ONLINE TABLE ... MAX_ROWS=`*`rows`* to increase the maximum number of rows for such a table; in this case, `ALTER ONLINE TABLE ... REORGANIZE PARTITION` is not needed (and causes an error if executed). The value of *`rows`* must be greater than the value specified for `MAX_ROWS` in the original `CREATE TABLE` statement for this to work.

  Attempting to use `REORGANIZE PARTITION` without the *`partition_names`* `INTO (`*`partition_definitions`*`)` option on explicitly partitioned tables results in the error `REORGANIZE PARTITION without parameters can only be used on auto-partitioned tables using HASH partitioning`.

  Note

  For partitions that have not been explicitly named, MySQL automatically provides the default names `p0`, `p1`, `p2`, and so on. The same is true with regard to subpartitions.

  For more detailed information about and examples of `ALTER TABLE ... REORGANIZE PARTITION` statements, see Section 19.3.1, “Management of RANGE and LIST Partitions”.

- To exchange a table partition or subpartition with a table, use the `ALTER TABLE ... EXCHANGE PARTITION` statement—that is, to move any existing rows in the partition or subpartition to the

nonpartitioned table, and any existing rows in the nonpartitioned table to the table partition or subpartition.

For usage information and examples, see Section 19.3.3, “Exchanging Partitions and Subpartitions with Tables”.

- Several options provide partition maintenance and repair functionality analogous to that implemented for nonpartitioned tables by statements such as `CHECK TABLE` and `REPAIR TABLE` (which are also supported for partitioned tables; for more information, see Section 13.7.2, “Table Maintenance Statements”). These include `ANALYZE PARTITION`, `CHECK PARTITION`, `OPTIMIZE PARTITION`, `REBUILD PARTITION`, and `REPAIR PARTITION`. Each of these options takes a *`partition_names`* clause consisting of one or more names of partitions, separated by commas. The partitions must already exist in the table to be altered. You can also use the `ALL` keyword in place of *`partition_names`*, in which case the statement acts on all table partitions. For more information and examples, see Section 19.3.4, “Maintenance of Partitions”.

  Some MySQL storage engines, such as `InnoDB`, do not support per-partition optimization. For a partitioned table using such a storage engine, `ALTER TABLE ... OPTIMIZE PARTITION` causes the entire table to rebuilt and analyzed, and an appropriate warning to be issued. (Bug #11751825, Bug #42822)

  To work around this problem, use the statements `ALTER TABLE ... REBUILD PARTITION` and `ALTER TABLE ... ANALYZE PARTITION` instead.

  The `ANALYZE PARTITION`, `CHECK PARTITION`, `OPTIMIZE PARTITION`, and `REPAIR PARTITION` options are not permitted for tables which are not partitioned.

- `REMOVE PARTITIONING` enables you to remove a table's partitioning without otherwise affecting the table or its data. This option can be combined with other `ALTER TABLE` options such as those used to add, drop, or rename columns or indexes.

- Using the `ENGINE` option with `ALTER TABLE` changes the storage engine used by the table without affecting the partitioning.

When `ALTER TABLE ... EXCHANGE PARTITION` or `ALTER TABLE ... TRUNCATE PARTITION` is run against a partitioned table that uses `MyISAM` (or another storage engine that makes use of table-level locking), only those partitions that are actually read from are locked. (This does not apply to partitioned tables using a storage enginethat employs row-level locking, such as `InnoDB`.) See Section 19.6.4, “Partitioning and Locking”.

It is possible for an `ALTER TABLE` statement to contain a `PARTITION BY` or `REMOVE PARTITIONING` clause in an addition to other alter specifications, but the `PARTITION BY` or `REMOVE PARTITIONING` clause must be specified last after any other specifications.

The `ADD PARTITION`, `DROP PARTITION`, `COALESCE PARTITION`, `REORGANIZE PARTITION`, `ANALYZE PARTITION`, `CHECK PARTITION`, and `REPAIR PARTITION` options cannot be combined with other alter specifications in a single `ALTER TABLE`, since the options just listed act on individual partitions. For more information, see Section 13.1.7.1, “ALTER TABLE Partition Operations”.

Only a single instance of any one of the following options can be used in a given `ALTER TABLE` statement: `PARTITION BY`, `ADD PARTITION`, `DROP PARTITION`, `TRUNCATE PARTITION`, `EXCHANGE PARTITION`, `REORGANIZE PARTITION`, or `COALESCE PARTITION`, `ANALYZE PARTITION`, `CHECK PARTITION`, `OPTIMIZE PARTITION`, `REBUILD PARTITION`, `REMOVE PARTITIONING`.

For example, the following two statements are invalid:

```
ALTER TABLE t1 ANALYZE PARTITION p1, ANALYZE PARTITION p2;

ALTER TABLE t1 ANALYZE PARTITION p1, CHECK PARTITION p2;
```

In the first case, you can analyze partitions `p1` and `p2` of table `t1` concurrently using a single statement with a single `ANALYZE PARTITION` option that lists both of the partitions to be analyzed, like this:

```
ALTER TABLE t1 ANALYZE PARTITION p1, p2;
```

In the second case, it is not possible to perform `ANALYZE` and `CHECK` operations on different partitions of the same table concurrently. Instead, you must issue two separate statements, like this:

```
ALTER TABLE t1 ANALYZE PARTITION p1;
ALTER TABLE t1 CHECK PARTITION p2;
```

`ANALYZE`, `CHECK`, `OPTIMIZE`, `REBUILD`, `REPAIR`, and `TRUNCATE` operations are not supported for subpartitions.

13.1.7.2 ALTER TABLE Examples

Begin with a table `t1` created as shown here:

```
CREATE TABLE t1 (a INTEGER, b CHAR(10));
```

To rename the table from `t1` to `t2`:

```
ALTER TABLE t1 RENAME t2;
```

To change column `a` from `INTEGER` to `TINYINT NOT NULL` (leaving the name the same), and to change column `b` from `CHAR(10)` to `CHAR(20)` as well as renaming it from `b` to `c`:

```
ALTER TABLE t2 MODIFY a TINYINT NOT NULL, CHANGE b c CHAR(20);
```

To add a new `TIMESTAMP` column named `d`:

```
ALTER TABLE t2 ADD d TIMESTAMP;
```

To add an index on column `d` and a `UNIQUE` index on column `a`:

```
ALTER TABLE t2 ADD INDEX (d), ADD UNIQUE (a);
```

To remove column `c`:

```
ALTER TABLE t2 DROP COLUMN c;
```

To add a new `AUTO_INCREMENT` integer column named `c`:

```
ALTER TABLE t2 ADD c INT UNSIGNED NOT NULL AUTO_INCREMENT,
  ADD PRIMARY KEY (c);
```

We indexed `c` (as a `PRIMARY KEY`) because `AUTO_INCREMENT` columns must be indexed, and we declare `c` as `NOT NULL` because primary key columns cannot be `NULL`.

For `NDB` tables, it is also possible to change the storage type used for a table or column. For example, consider an `NDB` table created as shown here:

```
mysql> CREATE TABLE t1 (c1 INT) TABLESPACE ts_1 ENGINE NDB;
Query OK, 0 rows affected (1.27 sec)
```

To convert this table to disk-based storage, you can use the following `ALTER TABLE` statement:

```
mysql> ALTER TABLE t1 TABLESPACE ts_1 STORAGE DISK;
Query OK, 0 rows affected (2.99 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `c1` int(11) DEFAULT NULL
) /*!50100 TABLESPACE ts_1 STORAGE DISK */
```

```
ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.01 sec)
```

It is not necessary that the tablespace was referenced when the table was originally created; however, the tablespace must be referenced by the `ALTER TABLE`:

```
mysql> CREATE TABLE t2 (c1 INT) ts_1 ENGINE NDB;
Query OK, 0 rows affected (1.00 sec)

mysql> ALTER TABLE t2 STORAGE DISK;
ERROR 1005 (HY000): Can't create table 'c.#sql-1750_3' (errno: 140)
mysql> ALTER TABLE t2 TABLESPACE ts_1 STORAGE DISK;
Query OK, 0 rows affected (3.42 sec)
Records: 0  Duplicates: 0  Warnings: 0
mysql> SHOW CREATE TABLE t2\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t2` (
  `c1` int(11) DEFAULT NULL
) /*!50100 TABLESPACE ts_1 STORAGE DISK */
ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.01 sec)
```

To change the storage type of an individual column, you can use `ALTER TABLE ... MODIFY [COLUMN]`. For example, suppose you create an NDB Cluster Disk Data table with two columns, using this `CREATE TABLE` statement:

```
mysql> CREATE TABLE t3 (c1 INT, c2 INT)
    ->     TABLESPACE ts_1 STORAGE DISK ENGINE NDB;
Query OK, 0 rows affected (1.34 sec)
```

To change column `c2` from disk-based to in-memory storage, include a STORAGE MEMORY clause in the column definition used by the ALTER TABLE statement, as shown here:

```
mysql> ALTER TABLE t3 MODIFY c2 INT STORAGE MEMORY;
Query OK, 0 rows affected (3.14 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

You can make an in-memory column into a disk-based column by using `STORAGE DISK` in a similar fashion.

Column `c1` uses disk-based storage, since this is the default for the table (determined by the table-level `STORAGE DISK` clause in the `CREATE TABLE` statement). However, column `c2` uses in-memory storage, as can be seen here in the output of SHOW `CREATE TABLE`:

```
mysql> SHOW CREATE TABLE t3\G
*************************** 1. row ***************************
       Table: t3
Create Table: CREATE TABLE `t3` (
  `c1` int(11) DEFAULT NULL,
  `c2` int(11) /*!50120 STORAGE MEMORY */ DEFAULT NULL
) /*!50100 TABLESPACE ts_1 STORAGE DISK */ ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.02 sec)
```

When you add an `AUTO_INCREMENT` column, column values are filled in with sequence numbers automatically. For `MyISAM` tables, you can set the first sequence number by executing `SET INSERT_ID=value` before `ALTER TABLE` or by using the `AUTO_INCREMENT=value` table option.

With `MyISAM` tables, if you do not change the `AUTO_INCREMENT` column, the sequence number is not affected. If you drop an `AUTO_INCREMENT` column and then add another `AUTO_INCREMENT` column, the numbers are resequenced beginning with 1.

When replication is used, adding an `AUTO_INCREMENT` column to a table might not produce the same ordering of the rows on the replica and the source. This occurs because the order in which the rows are numbered depends on the specific storage engine used for the table and the order in which the rows were inserted. If it is important to have the same order on the source and replica, the rows must be ordered before assigning an `AUTO_INCREMENT` number. Assuming that you want to add an

`AUTO_INCREMENT` column to the table `t1`, the following statements produce a new table `t2` identical to `t1` but with an `AUTO_INCREMENT` column:

```
CREATE TABLE t2 (id INT AUTO_INCREMENT PRIMARY KEY)
SELECT * FROM t1 ORDER BY col1, col2;
```

This assumes that the table `t1` has columns `col1` and `col2`.

This set of statements also produces a new table `t2` identical to `t1`, with the addition of an `AUTO_INCREMENT` column:

```
CREATE TABLE t2 LIKE t1;
ALTER TABLE t2 ADD id INT AUTO_INCREMENT PRIMARY KEY;
INSERT INTO t2 SELECT * FROM t1 ORDER BY col1, col2;
```

Important

To guarantee the same ordering on both source and replica, *all* columns of `t1` must be referenced in the `ORDER BY` clause.

Regardless of the method used to create and populate the copy having the `AUTO_INCREMENT` column, the final step is to drop the original table and then rename the copy:

```
DROP TABLE t1;
ALTER TABLE t2 RENAME t1;
```

13.1.8 ALTER TABLESPACE Statement

```
ALTER TABLESPACE tablespace_name
    {ADD | DROP} DATAFILE 'file_name'
    [INITIAL_SIZE [=] size]
    [WAIT]
    ENGINE [=] engine_name
```

This statement is used either to add a new data file, or to drop a data file from a tablespace.

The `ADD DATAFILE` variant enables you to specify an initial size using an `INITIAL_SIZE` clause, where *`size`* is measured in bytes; the default value is 134217728 (128 MB). Prior to MySQL NDB Cluster 7.3.2, this value was required to be specified using digits (Bug #13116514, Bug #16104705, Bug #62858); in MySQL NDB Cluster 7.3.2 and later, you may optionally follow *`size`* with a one-letter abbreviation for an order of magnitude, similar to those used in `my.cnf`. Generally, this is one of the letters `M` (megabytes) or `G` (gigabytes).

Note

All NDB Cluster Disk Data objects share the same namespace. This means that *each Disk Data object* must be uniquely named (and not merely each Disk Data object of a given type). For example, you cannot have a tablespace and a data file with the same name, or an undo log file and a tablespace with the same name.

On 32-bit systems, the maximum supported value for `INITIAL_SIZE` is 4294967296 (4 GB). (Bug #29186)

`INITIAL_SIZE` is rounded, explicitly, as for `CREATE TABLESPACE`.

Once a data file has been created, its size cannot be changed; however, you can add more data files to the tablespace using additional `ALTER TABLESPACE ... ADD DATAFILE` statements.

Using `DROP DATAFILE` with `ALTER TABLESPACE` drops the data file '*`file_name`*' from the tablespace. You cannot drop a data file from a tablespace which is in use by any table; in other words, the data file must be empty (no extents used). See Section 18.5.10.1, “NDB Cluster Disk Data Objects”. In addition, any data file to be dropped must previously have been added to the tablespace with `CREATE TABLESPACE` or `ALTER TABLESPACE`.

Both `ALTER TABLESPACE ... ADD DATAFILE` and `ALTER TABLESPACE ... DROP DATAFILE` require an `ENGINE` clause which specifies the storage engine used by the tablespace. Currently, the only accepted values for *`engine_name`* are `NDB` and `NDBCLUSTER`.

`WAIT` is parsed but otherwise ignored, and so has no effect in MySQL 5.6. It is intended for future expansion.

When `ALTER TABLESPACE ... ADD DATAFILE` is used with `ENGINE = NDB`, a data file is created on each Cluster data node. You can verify that the data files were created and obtain information about them by querying the `INFORMATION_SCHEMA.FILES` table. For example, the following query shows all data files belonging to the tablespace named `newts`:

```
mysql> SELECT LOGFILE_GROUP_NAME, FILE_NAME, EXTRA
    -> FROM INFORMATION_SCHEMA.FILES
    -> WHERE TABLESPACE_NAME = 'newts' AND FILE_TYPE = 'DATAFILE';
+--------------------+--------------+----------------+
| LOGFILE_GROUP_NAME | FILE_NAME    | EXTRA          |
+--------------------+--------------+----------------+
| lg_3               | newdata.dat  | CLUSTER_NODE=3 |
| lg_3               | newdata.dat  | CLUSTER_NODE=4 |
| lg_3               | newdata2.dat | CLUSTER_NODE=3 |
| lg_3               | newdata2.dat | CLUSTER_NODE=4 |
+--------------------+--------------+----------------+
2 rows in set (0.03 sec)
```

See Section 21.31.1, “The INFORMATION_SCHEMA FILES Table”.

`ALTER TABLESPACE` is useful only with Disk Data storage for NDB Cluster. See Section 18.5.10, “NDB Cluster Disk Data Tables”.

13.1.9 ALTER VIEW Statement

```
ALTER
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    [DEFINER = user]
    [SQL SECURITY { DEFINER | INVOKER }]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

This statement changes the definition of a view, which must exist. The syntax is similar to that for `CREATE VIEW` see Section 13.1.20, “CREATE VIEW Statement”). This statement requires the `CREATE VIEW` and `DROP` privileges for the view, and some privilege for each column referred to in the `SELECT` statement. `ALTER VIEW` is permitted only to the definer or users with the `SUPER` privilege.

13.1.10 CREATE DATABASE Statement

```
CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
    [create_option] ...

create_option: [DEFAULT] {
    CHARACTER SET [=] charset_name
  | COLLATE [=] collation_name
}
```

`CREATE DATABASE` creates a database with the given name. To use this statement, you need the `CREATE` privilege for the database. `CREATE SCHEMA` is a synonym for `CREATE DATABASE`.

An error occurs if the database exists and you did not specify `IF NOT EXISTS`.

`CREATE DATABASE` is not permitted within a session that has an active `LOCK TABLES` statement.

Each *`create_option`* specifies a database characteristic. Database characteristics are stored in the `db.opt` file in the database directory. The `CHARACTER SET` option specifies the default database character set. The `COLLATE` option specifies the default database collation. For information about character set and collation names, see Chapter 10, *Character Sets, Collations, Unicode*.

To see the available character sets and collations, use the `SHOW CHARACTER SET` and `SHOW COLLATION` statements, respectively. See Section 13.7.5.4, “SHOW CHARACTER SET Statement”, and Section 13.7.5.5, “SHOW COLLATION Statement”.

A database in MySQL is implemented as a directory containing files that correspond to tables in the database. Because there are no tables in a database when it is initially created, the `CREATE DATABASE` statement creates only a directory under the MySQL data directory and the `db.opt` file. Rules for permissible database names are given in Section 9.2, “Schema Object Names”. If a database name contains special characters, the name for the database directory contains encoded versions of those characters as described in Section 9.2.4, “Mapping of Identifiers to File Names”.

If you manually create a directory under the data directory (for example, with `mkdir`), the server considers it a database directory and it shows up in the output of `SHOW DATABASES`.

When you create a database, let the server manage the directory and the files in it. Manipulating database directories and files directly can cause inconsistencies and unexpected results.

MySQL has no limit on the number of databases. The underlying file system may have a limit on the number of directories.

You can also use the `mysqladmin` program to create databases. See Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”.

13.1.11 CREATE EVENT Statement

```
CREATE
    [DEFINER = user]
    EVENT
    [IF NOT EXISTS]
    event_name
    ON SCHEDULE schedule
    [ON COMPLETION [NOT] PRESERVE]
    [ENABLE | DISABLE | DISABLE ON SLAVE]
    [COMMENT 'string']
    DO event_body;

schedule: {
    AT timestamp [+ INTERVAL interval] ...
  | EVERY interval
    [STARTS timestamp [+ INTERVAL interval] ...]
    [ENDS timestamp [+ INTERVAL interval] ...]
}

interval:
    quantity {YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE |
              WEEK | SECOND | YEAR_MONTH | DAY_HOUR | DAY_MINUTE |
              DAY_SECOND | HOUR_MINUTE | HOUR_SECOND | MINUTE_SECOND}
```

This statement creates and schedules a new event. The event does not run unless the Event Scheduler is enabled. For information about checking Event Scheduler status and enabling it if necessary, see Section 20.4.2, “Event Scheduler Configuration”.

`CREATE EVENT` requires the `EVENT` privilege for the schema in which the event is to be created. If the `DEFINER` clause is present, the privileges required depend on the *`user`* value, as discussed in Section 20.6, “Stored Object Access Control”.

The minimum requirements for a valid `CREATE EVENT` statement are as follows:

- The keywords `CREATE EVENT` plus an event name, which uniquely identifies the event in a database schema.
- An `ON SCHEDULE` clause, which determines when and how often the event executes.
- A `DO` clause, which contains the SQL statement to be executed by an event.

This is an example of a minimal `CREATE EVENT` statement:

```
CREATE EVENT myevent
    ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 1 HOUR
    DO
      UPDATE myschema.mytable SET mycol = mycol + 1;
```

The previous statement creates an event named `myevent`. This event executes once—one hour following its creation—by running an SQL statement that increments the value of the `myschema.mytable` table's `mycol` column by 1.

The *`event_name`* must be a valid MySQL identifier with a maximum length of 64 characters. Event names are not case-sensitive, so you cannot have two events named `myevent` and `MyEvent` in the same schema. In general, the rules governing event names are the same as those for names of stored routines. See Section 9.2, “Schema Object Names”.

An event is associated with a schema. If no schema is indicated as part of *`event_name`*, the default (current) schema is assumed. To create an event in a specific schema, qualify the event name with a schema using *`schema_name.event_name`* syntax.

The `DEFINER` clause specifies the MySQL account to be used when checking access privileges at event execution time. If the `DEFINER` clause is present, the *`user`* value should be a MySQL account specified as *`'user_name'@'host_name'`*, `CURRENT_USER`, or `CURRENT_USER()`. The permitted *`user`* values depend on the privileges you hold, as discussed in Section 20.6, “Stored Object Access Control”. Also see that section for additional information about event security.

If the `DEFINER` clause is omitted, the default definer is the user who executes the `CREATE EVENT` statement. This is the same as specifying `DEFINER = CURRENT_USER` explicitly.

Within an event body, the `CURRENT_USER` function returns the account used to check privileges at event execution time, which is the `DEFINER` user. For information about user auditing within events, see Section 6.2.15, “SQL-Based Account Activity Auditing”.

`IF NOT EXISTS` has the same meaning for `CREATE EVENT` as for `CREATE TABLE`: If an event named *`event_name`* already exists in the same schema, no action is taken, and no error results. (However, a warning is generated in such cases.)

The `ON SCHEDULE` clause determines when, how often, and for how long the *`event_body`* defined for the event repeats. This clause takes one of two forms:

- `AT` *`timestamp`* is used for a one-time event. It specifies that the event executes one time only at the date and time given by *`timestamp`*, which must include both the date and time, or must be an expression that resolves to a datetime value. You may use a value of either the `DATETIME` or `TIMESTAMP` type for this purpose. If the date is in the past, a warning occurs, as shown here:

  ```
  mysql> SELECT NOW();
  +---------------------+
  | NOW()               |
  +---------------------+
  | 2006-02-10 23:59:01 |
  +---------------------+
  1 row in set (0.04 sec)

  mysql> CREATE EVENT e_totals
      ->     ON SCHEDULE AT '2006-02-10 23:59:00'
      ->     DO INSERT INTO test.totals VALUES (NOW());
  Query OK, 0 rows affected, 1 warning (0.00 sec)

  mysql> SHOW WARNINGS\G
  *************************** 1. row ***************************
    Level: Note
     Code: 1588
  Message: Event execution time is in the past and ON COMPLETION NOT
           PRESERVE is set. The event was dropped immediately after
           creation.
  ```

  `CREATE EVENT` statements which are themselves invalid—for whatever reason—fail with an error.

You may use CURRENT_TIMESTAMP to specify the current date and time. In such a case, the event acts as soon as it is created.

To create an event which occurs at some point in the future relative to the current date and time—such as that expressed by the phrase “three weeks from now”—you can use the optional clause + INTERVAL *interval*. The *interval* portion consists of two parts, a quantity and a unit of time, and follows the syntax rules described in Temporal Intervals, except that you cannot use any units keywords that involving microseconds when defining an event. With some interval types, complex time units may be used. For example, “two minutes and ten seconds” can be expressed as + INTERVAL '2:10' MINUTE_SECOND.

You can also combine intervals. For example, AT CURRENT_TIMESTAMP + INTERVAL 3 WEEK + INTERVAL 2 DAY is equivalent to “three weeks and two days from now”. Each portion of such a clause must begin with + INTERVAL.

- To repeat actions at a regular interval, use an EVERY clause. The EVERY keyword is followed by an *interval* as described in the previous discussion of the AT keyword. (+ INTERVAL is *not* used with EVERY.) For example, EVERY 6 WEEK means “every six weeks”.

  Although + INTERVAL clauses are not permitted in an EVERY clause, you can use the same complex time units permitted in a + INTERVAL.

  An EVERY clause may contain an optional STARTS clause. STARTS is followed by a *timestamp* value that indicates when the action should begin repeating, and may also use + INTERVAL *interval* to specify an amount of time “from now”. For example, EVERY 3 MONTH STARTS CURRENT_TIMESTAMP + INTERVAL 1 WEEK means “every three months, beginning one week from now”. Similarly, you can express “every two weeks, beginning six hours and fifteen minutes from now” as EVERY 2 WEEK STARTS CURRENT_TIMESTAMP + INTERVAL '6:15' HOUR_MINUTE. Not specifying STARTS is the same as using STARTS CURRENT_TIMESTAMP—that is, the action specified for the event begins repeating immediately upon creation of the event.

  An EVERY clause may contain an optional ENDS clause. The ENDS keyword is followed by a *timestamp* value that tells MySQL when the event should stop repeating. You may also use + INTERVAL *interval* with ENDS; for instance, EVERY 12 HOUR STARTS CURRENT_TIMESTAMP + INTERVAL 30 MINUTE ENDS CURRENT_TIMESTAMP + INTERVAL 4 WEEK is equivalent to “every twelve hours, beginning thirty minutes from now, and ending four weeks from now”. Not using ENDS means that the event continues executing indefinitely.

  ENDS supports the same syntax for complex time units as STARTS does.

  You may use STARTS, ENDS, both, or neither in an EVERY clause.

  If a repeating event does not terminate within its scheduling interval, the result may be multiple instances of the event executing simultaneously. If this is undesirable, you should institute a mechanism to prevent simultaneous instances. For example, you could use the GET_LOCK() function, or row or table locking.

The ON SCHEDULE clause may use expressions involving built-in MySQL functions and user variables to obtain any of the *timestamp* or *interval* values which it contains. You may not use stored functions or user-defined functions in such expressions, nor may you use any table references; however, you may use SELECT FROM DUAL. This is true for both CREATE EVENT and ALTER EVENT statements. References to stored functions, user-defined functions, and tables in such cases are specifically not permitted, and fail with an error (see Bug #22830).

Times in the ON SCHEDULE clause are interpreted using the current session time_zone value. This becomes the event time zone; that is, the time zone that is used for event scheduling and is in effect within the event as it executes. These times are converted to UTC and stored along with the event time zone in the mysql.event table. This enables event execution to proceed as defined regardless of any subsequent changes to the server time zone or daylight saving time effects. For additional information

about representation of event times, see Section 20.4.4, “Event Metadata”. See also Section 13.7.5.19, “SHOW EVENTS Statement”, and Section 21.8, “The INFORMATION_SCHEMA EVENTS Table”.

Normally, once an event has expired, it is immediately dropped. You can override this behavior by specifying `ON COMPLETION PRESERVE`. Using `ON COMPLETION NOT PRESERVE` merely makes the default nonpersistent behavior explicit.

You can create an event but prevent it from being active using the `DISABLE` keyword. Alternatively, you can use `ENABLE` to make explicit the default status, which is active. This is most useful in conjunction with `ALTER EVENT` (see Section 13.1.2, “ALTER EVENT Statement”).

A third value may also appear in place of `ENABLE` or `DISABLE`; `DISABLE ON SLAVE` is set for the status of an event on a replica to indicate that the event was created on the source and replicated to the replica, but is not executed on the replica. See Section 17.4.1.16, “Replication of Invoked Features”.

You may supply a comment for an event using a `COMMENT` clause. *`comment`* may be any string of up to 64 characters that you wish to use for describing the event. The comment text, being a string literal, must be surrounded by quotation marks.

The `DO` clause specifies an action carried by the event, and consists of an SQL statement. Nearly any valid MySQL statement that can be used in a stored routine can also be used as the action statement for a scheduled event. (See Section 20.8, “Restrictions on Stored Programs”.) For example, the following event `e_hourly` deletes all rows from the `sessions` table once per hour, where this table is part of the `site_activity` schema:

```
CREATE EVENT e_hourly
    ON SCHEDULE
      EVERY 1 HOUR
    COMMENT 'Clears out sessions table each hour.'
    DO
      DELETE FROM site_activity.sessions;
```

MySQL stores the `sql_mode` system variable setting in effect when an event is created or altered, and always executes the event with this setting in force, *regardless of the current server SQL mode when the event begins executing*.

A `CREATE EVENT` statement that contains an `ALTER EVENT` statement in its `DO` clause appears to succeed; however, when the server attempts to execute the resulting scheduled event, the execution fails with an error.

Note

Statements such as `SELECT` or `SHOW` that merely return a result set have no effect when used in an event; the output from these is not sent to the MySQL Monitor, nor is it stored anywhere. However, you can use statements such as `SELECT ... INTO` and `INSERT INTO ... SELECT` that store a result. (See the next example in this section for an instance of the latter.)

The schema to which an event belongs is the default schema for table references in the `DO` clause. Any references to tables in other schemas must be qualified with the proper schema name.

As with stored routines, you can use compound-statement syntax in the `DO` clause by using the `BEGIN` and `END` keywords, as shown here:

```
delimiter |

CREATE EVENT e_daily
    ON SCHEDULE
      EVERY 1 DAY
    COMMENT 'Saves total number of sessions then clears the table each day'
    DO
      BEGIN
        INSERT INTO site_activity.totals (time, total)
          SELECT CURRENT_TIMESTAMP, COUNT(*)
```

```
            FROM site_activity.sessions;
        DELETE FROM site_activity.sessions;
      END |

delimiter ;
```

This example uses the `delimiter` command to change the statement delimiter. See Section 20.1, “Defining Stored Programs”.

More complex compound statements, such as those used in stored routines, are possible in an event. This example uses local variables, an error handler, and a flow control construct:

```
delimiter |

CREATE EVENT e
    ON SCHEDULE
      EVERY 5 SECOND
    DO
      BEGIN
        DECLARE v INTEGER;
        DECLARE CONTINUE HANDLER FOR SQLEXCEPTION BEGIN END;

        SET v = 0;

        WHILE v < 5 DO
          INSERT INTO t1 VALUES (0);
          UPDATE t2 SET s1 = s1 + 1;
          SET v = v + 1;
        END WHILE;
    END |

delimiter ;
```

There is no way to pass parameters directly to or from events; however, it is possible to invoke a stored routine with parameters within an event:

```
CREATE EVENT e_call_myproc
    ON SCHEDULE
      AT CURRENT_TIMESTAMP + INTERVAL 1 DAY
    DO CALL myproc(5, 27);
```

If an event's definer has privileges sufficient to set global system variables (see Section 5.1.8.1, “System Variable Privileges”), the event can read and write global variables. As granting such privileges entails a potential for abuse, extreme care must be taken in doing so.

Generally, any statements that are valid in stored routines may be used for action statements executed by events. For more information about statements permitted within stored routines, see Section 20.2.1, “Stored Routine Syntax”. You can create an event as part of a stored routine, but an event cannot be created by another event.

13.1.12 CREATE FUNCTION Statement

The `CREATE FUNCTION` statement is used to create stored functions and user-defined functions (UDFs):

- For information about creating stored functions, see Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”.
- For information about creating user-defined functions, see Section 13.7.3.1, “CREATE FUNCTION Statement for User-Defined Functions”.

13.1.13 CREATE INDEX Statement

```
CREATE [ONLINE | OFFLINE] [UNIQUE | FULLTEXT | SPATIAL] INDEX index_name
    [index_type]
    ON tbl_name (key_part,...)
    [index_option]
    [algorithm_option | lock_option] ...
```

```
key_part:
    col_name [(length)] [ASC | DESC]

index_option: {
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
}

index_type:
    USING {BTREE | HASH}

algorithm_option:
    ALGORITHM [=] {DEFAULT | INPLACE | COPY}

lock_option:
    LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
```

Normally, you create all indexes on a table at the time the table itself is created with `CREATE TABLE`. See Section 13.1.17, “CREATE TABLE Statement”. This guideline is especially important for `InnoDB` tables, where the primary key determines the physical layout of rows in the data file. `CREATE INDEX` enables you to add indexes to existing tables.

`CREATE INDEX` is mapped to an `ALTER TABLE` statement to create indexes. See Section 13.1.7, “ALTER TABLE Statement”. `CREATE INDEX` cannot be used to create a `PRIMARY KEY`; use `ALTER TABLE` instead. For more information about indexes, see Section 8.3.1, “How MySQL Uses Indexes”.

When the `innodb_stats_persistent` setting is enabled, run the `ANALYZE TABLE` statement for an `InnoDB` table after creating an index on that table.

An index specification of the form `(key_part1, key_part2, ...)` creates an index with multiple key parts. Index key values are formed by concatenating the values of the given key parts. For example `(col1, col2, col3)` specifies a multiple-column index with index keys consisting of values from `col1`, `col2`, and `col3`.

A *`key_part`* specification can end with `ASC` or `DESC`. These keywords are permitted for future extensions for specifying ascending or descending index value storage. Currently, they are parsed but ignored; index values are always stored in ascending order.

The following sections describe different aspects of the `CREATE INDEX` statement:

- Column Prefix Key Parts
- Unique Indexes
- Full-Text Indexes
- Spatial Indexes
- Index Options
- Table Copying and Locking Options

Column Prefix Key Parts

For string columns, indexes can be created that use only the leading part of column values, using *`col_name(length)`* syntax to specify an index prefix length:

- Prefixes can be specified for `CHAR`, `VARCHAR`, `BINARY`, and `VARBINARY` key parts.
- Prefixes *must* be specified for `BLOB` and `TEXT` key parts. Additionally, `BLOB` and `TEXT` columns can be indexed only for `InnoDB`, `MyISAM`, and `BLACKHOLE` tables.
- Prefix *limits* are measured in bytes. However, prefix *lengths* for index specifications in `CREATE TABLE`, `ALTER TABLE`, and `CREATE INDEX` statements are interpreted as number of characters for

nonbinary string types (CHAR, VARCHAR, TEXT) and number of bytes for binary string types (BINARY, VARBINARY, BLOB). Take this into account when specifying a prefix length for a nonbinary string column that uses a multibyte character set.

Prefix support and lengths of prefixes (where supported) are storage engine dependent. For example, a prefix can be up to 767 bytes long for InnoDB tables or 3072 bytes if the innodb_large_prefix option is enabled. For MyISAM tables, the prefix length limit is 1000 bytes. The NDB storage engine does not support prefixes (see Section 18.1.7.6, "Unsupported or Missing Features in NDB Cluster").

The statement shown here creates an index using the first 10 characters of the name column (assuming that name has a nonbinary string type):

```
CREATE INDEX part_of_name ON customer (name(10));
```

If names in the column usually differ in the first 10 characters, lookups performed using this index should not be much slower than using an index created from the entire name column. Also, using column prefixes for indexes can make the index file much smaller, which could save a lot of disk space and might also speed up INSERT operations.

Unique Indexes

A UNIQUE index creates a constraint such that all values in the index must be distinct. An error occurs if you try to add a new row with a key value that matches an existing row. If you specify a prefix value for a column in a UNIQUE index, the column values must be unique within the prefix length. A UNIQUE index permits multiple NULL values for columns that can contain NULL.

If a table has a PRIMARY KEY or UNIQUE NOT NULL index that consists of a single column that has an integer type, you can use _rowid to refer to the indexed column in SELECT statements, as follows:

- _rowid refers to the PRIMARY KEY column if there is a PRIMARY KEY consisting of a single integer column. If there is a PRIMARY KEY but it does not consist of a single integer column, _rowid cannot be used.
- Otherwise, _rowid refers to the column in the first UNIQUE NOT NULL index if that index consists of a single integer column. If the first UNIQUE NOT NULL index does not consist of a single integer column, _rowid cannot be used.

Full-Text Indexes

FULLTEXT indexes are supported only for InnoDB and MyISAM tables and can include only CHAR, VARCHAR, and TEXT columns. Indexing always happens over the entire column; column prefix indexing is not supported and any prefix length is ignored if specified. See Section 12.10, "Full-Text Search Functions", for details of operation.

Spatial Indexes

The MyISAM, InnoDB, NDB, and ARCHIVE storage engines support spatial columns such as POINT and GEOMETRY. (Section 11.4, "Spatial Data Types", describes the spatial data types.) However, support for spatial column indexing varies among engines. Spatial and nonspatial indexes on spatial columns are available according to the following rules.

Spatial indexes on spatial columns (created using SPATIAL INDEX) have these characteristics:

- Available only for MyISAM tables. Specifying SPATIAL INDEX for other storage engines results in an error.
- Indexed columns must be NOT NULL.
- Column prefix lengths are prohibited. The full width of each column is indexed.

Nonspatial indexes on spatial columns (created with INDEX, UNIQUE, or PRIMARY KEY) have these characteristics:

- Permitted for any storage engine that supports spatial columns except `ARCHIVE`.
- Columns can be `NULL` unless the index is a primary key.
- For each spatial column in a non-`SPATIAL` index except `POINT` columns, a column prefix length must be specified. (This is the same requirement as for indexed `BLOB` columns.) The prefix length is given in bytes.
- The index type for a non-`SPATIAL` index depends on the storage engine. Currently, B-tree is used.
- Permitted for a column that can have `NULL` values only for `InnoDB`, `MyISAM`, and `MEMORY` tables.

Index Options

Following the key part list, index options can be given. An *`index_option`* value can be any of the following:

- `KEY_BLOCK_SIZE [=]` *`value`*

  For `MyISAM` tables, `KEY_BLOCK_SIZE` optionally specifies the size in bytes to use for index key blocks. The value is treated as a hint; a different size could be used if necessary. A `KEY_BLOCK_SIZE` value specified for an individual index definition overrides a table-level `KEY_BLOCK_SIZE` value.

  `KEY_BLOCK_SIZE` is not supported at the index level for `InnoDB` tables. See Section 13.1.17, “CREATE TABLE Statement”.

- *`index_type`*

  Some storage engines permit you to specify an index type when creating an index. For example:

```
CREATE TABLE lookup (id INT) ENGINE = MEMORY;
CREATE INDEX id_index ON lookup (id) USING BTREE;
```

  Table 13.1, “Index Types Per Storage Engine” shows the permissible index type values supported by different storage engines. Where multiple index types are listed, the first one is the default when no index type specifier is given. Storage engines not listed in the table do not support an *`index_type`* clause in index definitions.

Table 13.1 Index Types Per Storage Engine

Storage Engine	Permissible Index Types
`InnoDB`	`BTREE`
`MyISAM`	`BTREE`
`MEMORY`/`HEAP`	`HASH`, `BTREE`
`NDB`	`HASH`, `BTREE` (see note in text)

  The *`index_type`* clause cannot be used for `FULLTEXT INDEX` or `SPATIAL INDEX` specifications. Full-text index implementation is storage engine dependent. Spatial indexes are implemented as R-tree indexes.

  `BTREE` indexes are implemented by the `NDB` storage engine as T-tree indexes.

Note

For indexes on `NDB` table columns, the `USING` option can be specified only for a unique index or primary key. `USING HASH` prevents the creation of an ordered index; otherwise, creating a unique index or primary key on an `NDB` table automatically results in the creation of both an ordered index and a hash index, each of which indexes the same set of columns.

For unique indexes that include one or more `NULL` columns of an `NDB` table, the hash index can be used only to look up literal values, which means that `IS [NOT] NULL` conditions require a full scan of the table. One workaround is to make sure that a unique index using one or more `NULL` columns on such a table is always created in such a way that it includes the ordered index; that is, avoid employing `USING HASH` when creating the index.

If you specify an index type that is not valid for a given storage engine, but another index type is available that the engine can use without affecting query results, the engine uses the available type. The parser recognizes `RTREE` as a type name, but currently this cannot be specified for any storage engine.

Note

Use of the *`index_type`* option before the `ON tbl_name` clause is deprecated; expect support for use of the option in this position to be removed in a future MySQL release. If an *`index_type`* option is given in both the earlier and later positions, the final option applies.

`TYPE` *`type_name`* is recognized as a synonym for `USING` *`type_name`*. However, `USING` is the preferred form.

The following tables show index characteristics for the storage engines that support the *`index_type`* option.

Table 13.2 InnoDB Storage Engine Index Characteristics

Index Class	Index Type	Stores NULL VALUES	Permits Multiple NULL Values	IS NULL Scan Type	IS NOT NULL Scan Type
Primary key	`BTREE`	No	No	N/A	N/A
Unique	`BTREE`	Yes	Yes	Index	Index
Key	`BTREE`	Yes	Yes	Index	Index
`FULLTEXT`	N/A	Yes	Yes	Table	Table

Table 13.3 MyISAM Storage Engine Index Characteristics

Index Class	Index Type	Stores NULL VALUES	Permits Multiple NULL Values	IS NULL Scan Type	IS NOT NULL Scan Type
Primary key	`BTREE`	No	No	N/A	N/A
Unique	`BTREE`	Yes	Yes	Index	Index
Key	`BTREE`	Yes	Yes	Index	Index
`FULLTEXT`	N/A	Yes	Yes	Table	Table
`SPATIAL`	N/A	No	No	N/A	N/A

Table 13.4 MEMORY Storage Engine Index Characteristics

Index Class	Index Type	Stores NULL VALUES	Permits Multiple NULL Values	IS NULL Scan Type	IS NOT NULL Scan Type
Primary key	`BTREE`	No	No	N/A	N/A
Unique	`BTREE`	Yes	Yes	Index	Index
Key	`BTREE`	Yes	Yes	Index	Index
Primary key	`HASH`	No	No	N/A	N/A
Unique	`HASH`	Yes	Yes	Index	Index

Index Class	Index Type	Stores NULL VALUES	Permits Multiple NULL Values	IS NULL Scan Type	IS NOT NULL Scan Type
Key	HASH	Yes	Yes	Index	Index

Table 13.5 NDB Storage Engine Index Characteristics

Index Class	Index Type	Stores NULL VALUES	Permits Multiple NULL Values	IS NULL Scan Type	IS NOT NULL Scan Type
Primary key	BTREE	No	No	Index	Index
Unique	BTREE	Yes	Yes	Index	Index
Key	BTREE	Yes	Yes	Index	Index
Primary key	HASH	No	No	Table (see note 1)	Table (see note 1)
Unique	HASH	Yes	Yes	Table (see note 1)	Table (see note 1)
Key	HASH	Yes	Yes	Table (see note 1)	Table (see note 1)

Table note:

1. If `USING HASH` is specified that prevents creation of an implicit ordered index.

- `WITH PARSER parser_name`

  This option can be used only with `FULLTEXT` indexes. It associates a parser plugin with the index if full-text indexing and searching operations need special handling. See The MySQL Plugin API, for details on creating plugins.

- `COMMENT 'string'`

  Index definitions can include an optional comment of up to 1024 characters.

Table Copying and Locking Options

`ALGORITHM` and `LOCK` clauses may be given to influence the table copying method and level of concurrency for reading and writing the table while its indexes are being modified. They have the same meaning as for the `ALTER TABLE` statement. For more information, see Section 13.1.7, “ALTER TABLE Statement”

Indexes on variable-width columns of `NDBCLUSTER` tables are created online; that is, without any table copying. The table is not locked against access from other NDB Cluster API nodes, although it is locked against other operations on the *same* API node for the duration of the operation. This is done automatically by the server whenever it determines that it is possible to do so; you do not have to use any special SQL syntax or server options to cause it to happen.

In standard MySQL 5.6 releases, it is not possible to override the server when it determines that an index is to be created without table copying. In NDB Cluster, you can create indexes offline (which causes the table to be locked to all API nodes in the cluster) using the `OFFLINE` keyword. The rules and limitations governing `CREATE OFFLINE INDEX` and `CREATE ONLINE INDEX` are the same as for `ALTER OFFLINE TABLE ... ADD INDEX` and `ALTER ONLINE TABLE ... ADD INDEX`. You cannot cause the noncopying creation of an index that would normally be created offline by using the `ONLINE` keyword: If it is not possible to perform the `CREATE INDEX` operation without table copying, the server ignores the `ONLINE` keyword. For more information, see Section 18.5.11, “Online Operations with ALTER TABLE in NDB Cluster”.

The `ONLINE` and `OFFLINE` keywords are available only in NDB Cluster; attempting to use these keywords in standard MySQL Server 5.6 releases results in a syntax error. The `ONLINE` and `OFFLINE` keywords are deprecated in MySQL NDB Cluster 7.3; they continue to be supported in MySQL NDB Cluster 7.4, but they are subject to removal in a future NDB Cluster release.

13.1.14 CREATE LOGFILE GROUP Statement

```
CREATE LOGFILE GROUP logfile_group
    ADD UNDOFILE 'undo_file'
    [INITIAL_SIZE [=] initial_size]
    [UNDO_BUFFER_SIZE [=] undo_buffer_size]
    [REDO_BUFFER_SIZE [=] redo_buffer_size]
    [NODEGROUP [=] nodegroup_id]
    [WAIT]
    [COMMENT [=] 'string']
    ENGINE [=] engine_name
```

This statement creates a new log file group named *`logfile_group`* having a single `UNDO` file named '*`undo_file`*'. A `CREATE LOGFILE GROUP` statement has one and only one `ADD UNDOFILE` clause. For rules covering the naming of log file groups, see Section 9.2, "Schema Object Names".

Note

All NDB Cluster Disk Data objects share the same namespace. This means that *each Disk Data object* must be uniquely named (and not merely each Disk Data object of a given type). For example, you cannot have a tablespace and a log file group with the same name, or a tablespace and a data file with the same name.

In MySQL NDB Cluster 7.3 and later, you can have only one log file group per Cluster at any given time. (See Bug #16386)

The optional `INITIAL_SIZE` parameter sets the `UNDO` file's initial size; if not specified, it defaults to `128M` (128 megabytes). The optional `UNDO_BUFFER_SIZE` parameter sets the size used by the `UNDO` buffer for the log file group; The default value for `UNDO_BUFFER_SIZE` is `8M` (eight megabytes); this value cannot exceed the amount of system memory available. Both of these parameters are specified in bytes. In MySQL NDB Cluster 7.3.2 and later, you may optionally follow either or both of these with a one-letter abbreviation for an order of magnitude, similar to those used in `my.cnf`. Generally, this is one of the letters `M` (for megabytes) or `G` (for gigabytes). Prior to MySQL NDB Cluster 7.3.2, the values for these options could only be specified using digits. (Bug #13116514, Bug #16104705, Bug #62858)

Memory used for `UNDO_BUFFER_SIZE` comes from the global pool whose size is determined by the value of the `SharedGlobalMemory` data node configuration parameter. This includes any default value implied for this option by the setting of the `InitialLogFileGroup` data node configuration parameter.

The maximum permitted for `UNDO_BUFFER_SIZE` is 629145600 (600 MB).

On 32-bit systems, the maximum supported value for `INITIAL_SIZE` is 4294967296 (4 GB). (Bug #29186)

The minimum allowed value for `INITIAL_SIZE` is 1048576 (1 MB).

The `ENGINE` option determines the storage engine to be used by this log file group, with *`engine_name`* being the name of the storage engine. In MySQL 5.6, this must be `NDB` (or `NDBCLUSTER`). If `ENGINE` is not set, MySQL tries to use the engine specified by the `default_storage_engine` server system variable (formerly `storage_engine`). In any case, if the engine is not specified as `NDB` or `NDBCLUSTER`, the `CREATE LOGFILE GROUP` statement appears to succeed but actually fails to create the log file group, as shown here:

```
mysql> CREATE LOGFILE GROUP lg1
    ->     ADD UNDOFILE 'undo.dat' INITIAL_SIZE = 10M;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-------+------+----------------------------------------------------------------------------------------------
| Level | Code | Message
+-------+------+----------------------------------------------------------------------------------------------
| Error | 1478 | Table storage engine 'InnoDB' does not support the create option 'TABLESPACE or LOGFILE GR
+-------+------+----------------------------------------------------------------------------------------------
1 row in set (0.00 sec)
```

```
mysql> DROP LOGFILE GROUP lg1 ENGINE = NDB;
ERROR 1529 (HY000): Failed to drop LOGFILE GROUP

mysql> CREATE LOGFILE GROUP lg1
    ->     ADD UNDOFILE 'undo.dat' INITIAL_SIZE = 10M
    ->     ENGINE = NDB;
Query OK, 0 rows affected (2.97 sec)
```

The fact that the `CREATE LOGFILE GROUP` statement does not actually return an error when a non-`NDB` storage engine is named, but rather appears to succeed, is a known issue which we hope to address in a future release of NDB Cluster.

`REDO_BUFFER_SIZE`, `NODEGROUP`, `WAIT`, and `COMMENT` are parsed but ignored, and so have no effect in MySQL 5.6. These options are intended for future expansion.

When used with `ENGINE [=] NDB`, a log file group and associated `UNDO` log file are created on each Cluster data node. You can verify that the `UNDO` files were created and obtain information about them by querying the `INFORMATION_SCHEMA.FILES` table. For example:

```
mysql> SELECT LOGFILE_GROUP_NAME, LOGFILE_GROUP_NUMBER, EXTRA
    -> FROM INFORMATION_SCHEMA.FILES
    -> WHERE FILE_NAME = 'undo_10.dat';
+--------------------+----------------------+----------------+
| LOGFILE_GROUP_NAME | LOGFILE_GROUP_NUMBER | EXTRA          |
+--------------------+----------------------+----------------+
| lg_3               |                   11 | CLUSTER_NODE=3 |
| lg_3               |                   11 | CLUSTER_NODE=4 |
+--------------------+----------------------+----------------+
2 rows in set (0.06 sec)
```

`CREATE LOGFILE GROUP` is useful only with Disk Data storage for NDB Cluster. See Section 18.5.10, “NDB Cluster Disk Data Tables”.

13.1.15 CREATE PROCEDURE and CREATE FUNCTION Statements

```
CREATE
    [DEFINER = user]
    PROCEDURE sp_name ([proc_parameter[,...]])
    [characteristic ...] routine_body

CREATE
    [DEFINER = user]
    FUNCTION sp_name ([func_parameter[,...]])
    RETURNS type
    [characteristic ...] routine_body

proc_parameter:
    [ IN | OUT | INOUT ] param_name type

func_parameter:
    param_name type

type:
    Any valid MySQL data type

characteristic: {
    COMMENT 'string'
  | LANGUAGE SQL
  | [NOT] DETERMINISTIC
  | { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
}

routine_body:
    Valid SQL routine statement
```

These statements are used to create a stored routine (a stored procedure or function). That is, the specified routine becomes known to the server. By default, a stored routine is associated with the

default database. To associate the routine explicitly with a given database, specify the name as *db_name.sp_name* when you create it.

The CREATE FUNCTION statement is also used in MySQL to support UDFs (user-defined functions). See Section 13.7.3.1, "CREATE FUNCTION Statement for User-Defined Functions". A UDF can be regarded as an external stored function. Stored functions share their namespace with UDFs. See Section 9.2.5, "Function Name Parsing and Resolution", for the rules describing how the server interprets references to different kinds of functions.

To invoke a stored procedure, use the CALL statement (see Section 13.2.1, "CALL Statement"). To invoke a stored function, refer to it in an expression. The function returns a value during expression evaluation.

CREATE PROCEDURE and CREATE FUNCTION require the CREATE ROUTINE privilege. If the DEFINER clause is present, the privileges required depend on the *user* value, as discussed in Section 20.6, "Stored Object Access Control". If binary logging is enabled, CREATE FUNCTION might require the SUPER privilege, as discussed in Section 20.7, "Stored Program Binary Logging".

By default, MySQL automatically grants the ALTER ROUTINE and EXECUTE privileges to the routine creator. This behavior can be changed by disabling the automatic_sp_privileges system variable. See Section 20.2.2, "Stored Routines and MySQL Privileges".

The DEFINER and SQL SECURITY clauses specify the security context to be used when checking access privileges at routine execution time, as described later in this section.

If the routine name is the same as the name of a built-in SQL function, a syntax error occurs unless you use a space between the name and the following parenthesis when defining the routine or invoking it later. For this reason, avoid using the names of existing SQL functions for your own stored routines.

The IGNORE_SPACE SQL mode applies to built-in functions, not to stored routines. It is always permissible to have spaces after a stored routine name, regardless of whether IGNORE_SPACE is enabled.

The parameter list enclosed within parentheses must always be present. If there are no parameters, an empty parameter list of () should be used. Parameter names are not case-sensitive.

Each parameter is an IN parameter by default. To specify otherwise for a parameter, use the keyword OUT or INOUT before the parameter name.

Note

Specifying a parameter as IN, OUT, or INOUT is valid only for a PROCEDURE. For a FUNCTION, parameters are always regarded as IN parameters.

An IN parameter passes a value into a procedure. The procedure might modify the value, but the modification is not visible to the caller when the procedure returns. An OUT parameter passes a value from the procedure back to the caller. Its initial value is NULL within the procedure, and its value is visible to the caller when the procedure returns. An INOUT parameter is initialized by the caller, can be modified by the procedure, and any change made by the procedure is visible to the caller when the procedure returns.

For each OUT or INOUT parameter, pass a user-defined variable in the CALL statement that invokes the procedure so that you can obtain its value when the procedure returns. If you are calling the procedure from within another stored procedure or function, you can also pass a routine parameter or local routine variable as an OUT or INOUT parameter. If you are calling the procedure from within a trigger, you can also pass NEW.*col_name* as an OUT or INOUT parameter.

For information about the effect of unhandled conditions on procedure parameters, see Section 13.6.7.8, "Condition Handling and OUT or INOUT Parameters".

Routine parameters cannot be referenced in statements prepared within the routine; see Section 20.8, "Restrictions on Stored Programs".

The following example shows a simple stored procedure that, given a country code, counts the number of cities for that country that appear in the `city` table of the `world` database. The country code is passed using an `IN` parameter, and the city count is returned using an `OUT` parameter:

```
mysql> delimiter //

mysql> CREATE PROCEDURE citycount (IN country CHAR(3), OUT cities INT)
       BEGIN
         SELECT COUNT(*) INTO cities FROM world.city
         WHERE CountryCode = country;
       END//
Query OK, 0 rows affected (0.01 sec)

mysql> delimiter ;

mysql> CALL citycount('JPN', @cities); -- cities in Japan
Query OK, 1 row affected (0.00 sec)

mysql> SELECT @cities;
+---------+
| @cities |
+---------+
|     248 |
+---------+
1 row in set (0.00 sec)

mysql> CALL citycount('FRA', @cities); -- cities in France
Query OK, 1 row affected (0.00 sec)

mysql> SELECT @cities;
+---------+
| @cities |
+---------+
|      40 |
+---------+
1 row in set (0.00 sec)
```

The example uses the `mysql` client `delimiter` command to change the statement delimiter from `;` to `//` while the procedure is being defined. This enables the `;` delimiter used in the procedure body to be passed through to the server rather than being interpreted by `mysql` itself. See Section 20.1, “Defining Stored Programs”.

The `RETURNS` clause may be specified only for a `FUNCTION`, for which it is mandatory. It indicates the return type of the function, and the function body must contain a `RETURN` *`value`* statement. If the `RETURN` statement returns a value of a different type, the value is coerced to the proper type. For example, if a function specifies an `ENUM` or `SET` value in the `RETURNS` clause, but the `RETURN` statement returns an integer, the value returned from the function is the string for the corresponding `ENUM` member of set of `SET` members.

The following example function takes a parameter, performs an operation using an SQL function, and returns the result. In this case, it is unnecessary to use `delimiter` because the function definition contains no internal `;` statement delimiters:

```
mysql> CREATE FUNCTION hello (s CHAR(20))
mysql> RETURNS CHAR(50) DETERMINISTIC
       RETURN CONCAT('Hello, ',s,'!');
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT hello('world');
+----------------+
| hello('world') |
+----------------+
| Hello, world!  |
+----------------+
1 row in set (0.00 sec)
```

Parameter types and function return types can be declared to use any valid data type. The `COLLATE` attribute can be used if preceded by a `CHARACTER SET` specification.

The *`routine_body`* consists of a valid SQL routine statement. This can be a simple statement such as `SELECT` or `INSERT`, or a compound statement written using `BEGIN` and `END`. Compound statements can contain declarations, loops, and other control structure statements. The syntax for these statements is described in Section 13.6, “Compound Statements”. In practice, stored functions tend to use compound statements, unless the body consists of a single `RETURN` statement.

MySQL permits routines to contain DDL statements, such as `CREATE` and `DROP`. MySQL also permits stored procedures (but not stored functions) to contain SQL transaction statements such as `COMMIT`. Stored functions may not contain statements that perform explicit or implicit commit or rollback. Support for these statements is not required by the SQL standard, which states that each DBMS vendor may decide whether to permit them.

Statements that return a result set can be used within a stored procedure but not within a stored function. This prohibition includes `SELECT` statements that do not have an `INTO` *`var_list`* clause and other statements such as `SHOW`, `EXPLAIN`, and `CHECK TABLE`. For statements that can be determined at function definition time to return a result set, a `Not allowed to return a result set from a function` error occurs (`ER_SP_NO_RETSET`). For statements that can be determined only at runtime to return a result set, a `PROCEDURE %s can't return a result set in the given context` error occurs (`ER_SP_BADSELECT`).

`USE` statements within stored routines are not permitted. When a routine is invoked, an implicit `USE` *`db_name`* is performed (and undone when the routine terminates). The causes the routine to have the given default database while it executes. References to objects in databases other than the routine default database should be qualified with the appropriate database name.

For additional information about statements that are not permitted in stored routines, see Section 20.8, “Restrictions on Stored Programs”.

For information about invoking stored procedures from within programs written in a language that has a MySQL interface, see Section 13.2.1, “CALL Statement”.

MySQL stores the `sql_mode` system variable setting in effect when a routine is created or altered, and always executes the routine with this setting in force, *regardless of the current server SQL mode when the routine begins executing*.

The switch from the SQL mode of the invoker to that of the routine occurs after evaluation of arguments and assignment of the resulting values to routine parameters. If you define a routine in strict SQL mode but invoke it in nonstrict mode, assignment of arguments to routine parameters does not take place in strict mode. If you require that expressions passed to a routine be assigned in strict SQL mode, you should invoke the routine with strict mode in effect.

The `COMMENT` characteristic is a MySQL extension, and may be used to describe the stored routine. This information is displayed by the `SHOW CREATE PROCEDURE` and `SHOW CREATE FUNCTION` statements.

The `LANGUAGE` characteristic indicates the language in which the routine is written. The server ignores this characteristic; only SQL routines are supported.

A routine is considered “deterministic” if it always produces the same result for the same input parameters, and “not deterministic” otherwise. If neither `DETERMINISTIC` nor `NOT DETERMINISTIC` is given in the routine definition, the default is `NOT DETERMINISTIC`. To declare that a function is deterministic, you must specify `DETERMINISTIC` explicitly.

Assessment of the nature of a routine is based on the “honesty” of the creator: MySQL does not check that a routine declared `DETERMINISTIC` is free of statements that produce nondeterministic results. However, misdeclaring a routine might affect results or affect performance. Declaring a nondeterministic routine as `DETERMINISTIC` might lead to unexpected results by causing the optimizer to make incorrect execution plan choices. Declaring a deterministic routine as `NONDETERMINISTIC` might diminish performance by causing available optimizations not to be used.

If binary logging is enabled, the `DETERMINISTIC` characteristic affects which routine definitions MySQL accepts. See Section 20.7, “Stored Program Binary Logging”.

A routine that contains the `NOW()` function (or its synonyms) or `RAND()` is nondeterministic, but it might still be replication-safe. For `NOW()`, the binary log includes the timestamp and replicates correctly. `RAND()` also replicates correctly as long as it is called only a single time during the execution of a routine. (You can consider the routine execution timestamp and random number seed as implicit inputs that are identical on the source and replica.)

Several characteristics provide information about the nature of data use by the routine. In MySQL, these characteristics are advisory only. The server does not use them to constrain what kinds of statements a routine is permitted to execute.

- `CONTAINS SQL` indicates that the routine does not contain statements that read or write data. This is the default if none of these characteristics is given explicitly. Examples of such statements are `SET @x = 1` or `DO RELEASE_LOCK('abc')`, which execute but neither read nor write data.
- `NO SQL` indicates that the routine contains no SQL statements.
- `READS SQL DATA` indicates that the routine contains statements that read data (for example, `SELECT`), but not statements that write data.
- `MODIFIES SQL DATA` indicates that the routine contains statements that may write data (for example, `INSERT` or `DELETE`).

The `SQL SECURITY` characteristic can be `DEFINER` or `INVOKER` to specify the security context; that is, whether the routine executes using the privileges of the account named in the routine `DEFINER` clause or the user who invokes it. This account must have permission to access the database with which the routine is associated. The default value is `DEFINER`. The user who invokes the routine must have the `EXECUTE` privilege for it, as must the `DEFINER` account if the routine executes in definer security context.

The `DEFINER` clause specifies the MySQL account to be used when checking access privileges at routine execution time for routines that have the `SQL SECURITY DEFINER` characteristic.

If the `DEFINER` clause is present, the *`user`* value should be a MySQL account specified as *`'user_name'@'host_name'`*, `CURRENT_USER`, or `CURRENT_USER()`. The permitted *`user`* values depend on the privileges you hold, as discussed in Section 20.6, “Stored Object Access Control”. Also see that section for additional information about stored routine security.

If the `DEFINER` clause is omitted, the default definer is the user who executes the `CREATE PROCEDURE` or `CREATE FUNCTION` statement. This is the same as specifying `DEFINER = CURRENT_USER` explicitly.

Within the body of a stored routine that is defined with the `SQL SECURITY DEFINER` characteristic, the `CURRENT_USER` function returns the routine's `DEFINER` value. For information about user auditing within stored routines, see Section 6.2.15, “SQL-Based Account Activity Auditing”.

Consider the following procedure, which displays a count of the number of MySQL accounts listed in the `mysql.user` system table:

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE account_count()
BEGIN
  SELECT 'Number of accounts:', COUNT(*) FROM mysql.user;
END;
```

The procedure is assigned a `DEFINER` account of `'admin'@'localhost'` no matter which user defines it. It executes with the privileges of that account no matter which user invokes it (because the default security characteristic is `DEFINER`). The procedure succeeds or fails depending on whether invoker has the `EXECUTE` privilege for it and `'admin'@'localhost'` has the `SELECT` privilege for the `mysql.user` system table.

Now suppose that the procedure is defined with the `SQL SECURITY INVOKER` characteristic:

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE account_count()
SQL SECURITY INVOKER
```

```
BEGIN
  SELECT 'Number of accounts:', COUNT(*) FROM mysql.user;
END;
```

The procedure still has a `DEFINER` of `'admin'@'localhost'`, but in this case, it executes with the privileges of the invoking user. Thus, the procedure succeeds or fails depending on whether the invoker has the `EXECUTE` privilege for it and the `SELECT` privilege for the `mysql.user` table.

The server handles the data type of a routine parameter, local routine variable created with `DECLARE`, or function return value as follows:

- Assignments are checked for data type mismatches and overflow. Conversion and overflow problems result in warnings, or errors in strict SQL mode.
- Only scalar values can be assigned. For example, a statement such as `SET x = (SELECT 1, 2)` is invalid.
- For character data types, if `CHARACTER SET` is includedd in the declaration, the specified character set and its default collation is used. If the `COLLATE` attribute is also present, that collation is used rather than the default collation.

  If `CHARACTER SET` and `COLLATE` are not present, the database character set and collation in effect at routine creation time are used. To avoid having the server use the database character set and collation, provide an explicit `CHARACTER SET` and a `COLLATE` attribute for character data parameters.

  If you alter the database default character set or collation, stored routines that are to use the new database defaults must be dropped and recreated.

  The database character set and collation are given by the value of the `character_set_database` and `collation_database` system variables. For more information, see Section 10.3.3, “Database Character Set and Collation”.

13.1.16 CREATE SERVER Statement

```
CREATE SERVER server_name
    FOREIGN DATA WRAPPER wrapper_name
    OPTIONS (option [, option] ...)

option: {
    HOST character-literal
  | DATABASE character-literal
  | USER character-literal
  | PASSWORD character-literal
  | SOCKET character-literal
  | OWNER character-literal
  | PORT numeric-literal
}
```

This statement creates the definition of a server for use with the `FEDERATED` storage engine. The `CREATE SERVER` statement creates a new row in the `servers` table in the `mysql` database. This statement requires the `SUPER` privilege.

The *`server_name`* should be a unique reference to the server. Server definitions are global within the scope of the server, it is not possible to qualify the server definition to a specific database. *`server_name`* has a maximum length of 64 characters (names longer than 64 characters are silently truncated), and is case-insensitive. You may specify the name as a quoted string.

The *`wrapper_name`* is an identifier and may be quoted with single quotation marks.

For each *`option`* you must specify either a character literal or numeric literal. Character literals are UTF-8, support a maximum length of 64 characters and default to a blank (empty) string. String literals are silently truncated to 64 characters. Numeric literals must be a number between 0 and 9999, default value is 0.

Note

The `OWNER` option is currently not applied, and has no effect on the ownership or operation of the server connection that is created.

The `CREATE SERVER` statement creates an entry in the `mysql.servers` table that can later be used with the `CREATE TABLE` statement when creating a `FEDERATED` table. The options that you specify are used to populate the columns in the `mysql.servers` table. The table columns are `Server_name`, `Host`, `Db`, `Username`, `Password`, `Port` and `Socket`.

For example:

```
CREATE SERVER s
FOREIGN DATA WRAPPER mysql
OPTIONS (USER 'Remote', HOST '198.51.100.106', DATABASE 'test');
```

Be sure to specify all options necessary to establish a connection to the server. The user name, host name, and database name are mandatory. Other options might be required as well, such as password.

The data stored in the table can be used when creating a connection to a `FEDERATED` table:

```
CREATE TABLE t (s1 INT) ENGINE=FEDERATED CONNECTION='s';
```

For more information, see Section 15.8, “The FEDERATED Storage Engine”.

`CREATE SERVER` causes an implicit commit. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

`CREATE SERVER` is not written to the binary log, regardless of the logging format that is in use.

In MySQL 5.6.11 only, `gtid_next` must be set to `AUTOMATIC` before issuing this statement. (Bug #16062608, Bug #16715809, Bug #69045)

13.1.17 CREATE TABLE Statement

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    (create_definition,...)
    [table_options]
    [partition_options]

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    [(create_definition,...)]
    [table_options]
    [partition_options]
    [IGNORE | REPLACE]
    [AS] query_expression

CREATE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    { LIKE old_tbl_name | (LIKE old_tbl_name) }

create_definition: {
    col_name column_definition
  | {INDEX | KEY} [index_name] [index_type] (key_part,...)
      [index_option] ...
  | {FULLTEXT | SPATIAL} [INDEX | KEY] [index_name] (key_part,...)
      [index_option] ...
  | [CONSTRAINT [symbol]] PRIMARY KEY
      [index_type] (key_part,...)
      [index_option] ...
  | [CONSTRAINT [symbol]] UNIQUE [INDEX | KEY]
      [index_name] [index_type] (key_part,...)
      [index_option] ...
  | [CONSTRAINT [symbol]] FOREIGN KEY
      [index_name] (col_name,...)
      reference_definition
  | CHECK (expr)
}

column_definition:
```

```
    data_type [NOT NULL | NULL] [DEFAULT default_value]
      [AUTO_INCREMENT] [UNIQUE [KEY]] [[PRIMARY] KEY]
      [COMMENT 'string']
      [COLLATE collation_name]
      [COLUMN_FORMAT {FIXED | DYNAMIC | DEFAULT}]
      [STORAGE {DISK | MEMORY}]
      [reference_definition]

data_type:
    (see Chapter 11, Data Types)

key_part:
    col_name [(length)] [ASC | DESC]

index_type:
    USING {BTREE | HASH}

index_option: {
    KEY_BLOCK_SIZE [=] value
  | index_type
  | WITH PARSER parser_name
  | COMMENT 'string'
}

reference_definition:
    REFERENCES tbl_name (key_part,...)
      [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
      [ON DELETE reference_option]
      [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT

table_options:
    table_option [[,] table_option] ...

table_option: {
    AUTO_INCREMENT [=] value
  | AVG_ROW_LENGTH [=] value
  | [DEFAULT] CHARACTER SET [=] charset_name
  | CHECKSUM [=] {0 | 1}
  | [DEFAULT] COLLATE [=] collation_name
  | COMMENT [=] 'string'
  | CONNECTION [=] 'connect_string'
  | {DATA | INDEX} DIRECTORY [=] 'absolute path to directory'
  | DELAY_KEY_WRITE [=] {0 | 1}
  | ENGINE [=] engine_name
  | INSERT_METHOD [=] { NO | FIRST | LAST }
  | KEY_BLOCK_SIZE [=] value
  | MAX_ROWS [=] value
  | MIN_ROWS [=] value
  | PACK_KEYS [=] {0 | 1 | DEFAULT}
  | PASSWORD [=] 'string'
  | ROW_FORMAT [=] {DEFAULT | DYNAMIC | FIXED | COMPRESSED | REDUNDANT | COMPACT}
  | STATS_AUTO_RECALC [=] {DEFAULT | 0 | 1}
  | STATS_PERSISTENT [=] {DEFAULT | 0 | 1}
  | STATS_SAMPLE_PAGES [=] value
  | TABLESPACE tablespace_name [STORAGE {DISK | MEMORY}]
  | UNION [=] (tbl_name[,tbl_name]...)
}

partition_options:
    PARTITION BY
        { [LINEAR] HASH(expr)
        | [LINEAR] KEY [ALGORITHM={1 | 2}] (column_list)
        | RANGE{(expr) | COLUMNS(column_list)}
        | LIST{(expr) | COLUMNS(column_list)} }
    [PARTITIONS num]
    [SUBPARTITION BY
        { [LINEAR] HASH(expr)
        | [LINEAR] KEY [ALGORITHM={1 | 2}] (column_list) }
      [SUBPARTITIONS num]
```

```
    ]
    [(partition_definition [, partition_definition] ...)]

partition_definition:
    PARTITION partition_name
        [VALUES
            {LESS THAN {(expr | value_list) | MAXVALUE}
            |
            IN (value_list)}]
        [[STORAGE] ENGINE [=] engine_name]
        [COMMENT [=] 'string' ]
        [DATA DIRECTORY [=] 'data_dir']
        [INDEX DIRECTORY [=] 'index_dir']
        [MAX_ROWS [=] max_number_of_rows]
        [MIN_ROWS [=] min_number_of_rows]
        [TABLESPACE [=] tablespace_name]
        [NODEGROUP [=] node_group_id]
        [(subpartition_definition [, subpartition_definition] ...)]

subpartition_definition:
    SUBPARTITION logical_name
        [[STORAGE] ENGINE [=] engine_name]
        [COMMENT [=] 'string' ]
        [DATA DIRECTORY [=] 'data_dir']
        [INDEX DIRECTORY [=] 'index_dir']
        [MAX_ROWS [=] max_number_of_rows]
        [MIN_ROWS [=] min_number_of_rows]
        [TABLESPACE [=] tablespace_name]
        [NODEGROUP [=] node_group_id]

query_expression:
    SELECT ...   (Some valid select or union statement)
```

`CREATE TABLE` creates a table with the given name. You must have the `CREATE` privilege for the table.

By default, tables are created in the default database, using the `InnoDB` storage engine. An error occurs if the table exists, if there is no default database, or if the database does not exist.

MySQL has no limit on the number of tables. The underlying file system may have a limit on the number of files that represent tables. Individual storage engines may impose engine-specific constraints. `InnoDB` permits up to 4 billion tables.

For information about the physical representation of a table, see Section 13.1.17.1, “Files Created by CREATE TABLE”.

There are several aspects to the `CREATE TABLE` statement, described under the following topics in this section:

- Table Name
- Temporary Tables
- Table Cloning and Copying
- Column Data Types and Attributes
- Indexes and Foreign Keys
- Table Options
- Table Partitioning

Table Name

- *`tbl_name`*

  The table name can be specified as *`db_name.tbl_name`* to create the table in a specific database. This works regardless of whether there is a default database, assuming that the database exists.

If you use quoted identifiers, quote the database and table names separately. For example, write `` `mydb`.`mytbl` ``, not `` `mydb.mytbl` ``.

Rules for permissible table names are given in Section 9.2, “Schema Object Names”.

- `IF NOT EXISTS`

  Prevents an error from occurring if the table exists. However, there is no verification that the existing table has a structure identical to that indicated by the `CREATE TABLE` statement.

Temporary Tables

You can use the `TEMPORARY` keyword when creating a table. A `TEMPORARY` table is visible only within the current session, and is dropped automatically when the session is closed. For more information, see Section 13.1.17.2, “CREATE TEMPORARY TABLE Statement”.

Table Cloning and Copying

- `LIKE`

  Use `CREATE TABLE ... LIKE` to create an empty table based on the definition of another table, including any column attributes and indexes defined in the original table:

  ```
  CREATE TABLE new_tbl LIKE orig_tbl;
  ```

  For more information, see Section 13.1.17.3, “CREATE TABLE ... LIKE Statement”.

- `[AS] query_expression`

  To create one table from another, add a `SELECT` statement at the end of the `CREATE TABLE` statement:

  ```
  CREATE TABLE new_tbl AS SELECT * FROM orig_tbl;
  ```

  For more information, see Section 13.1.17.4, “CREATE TABLE ... SELECT Statement”.

- `IGNORE | REPLACE`

  The `IGNORE` and `REPLACE` options indicate how to handle rows that duplicate unique key values when copying a table using a `SELECT` statement.

  For more information, see Section 13.1.17.4, “CREATE TABLE ... SELECT Statement”.

Column Data Types and Attributes

There is a hard limit of 4096 columns per table, but the effective maximum may be less for a given table and depends on the factors discussed in Section 8.4.7, “Limits on Table Column Count and Row Size”.

- *`data_type`*

  `data_type` represents the data type in a column definition. For a full description of the syntax available for specifying column data types, as well as information about the properties of each type, see Chapter 11, *Data Types*.

  - Some attributes do not apply to all data types. `AUTO_INCREMENT` applies only to integer and floating-point types. `DEFAULT` does not apply to the `BLOB` or `TEXT` types.
  - Character data types (`CHAR`, `VARCHAR`, the `TEXT` types, `ENUM`, `SET`, and any synonyms) can include `CHARACTER SET` to specify the character set for the column. `CHARSET` is a synonym for `CHARACTER SET`. A collation for the character set can be specified with the `COLLATE` attribute, along with any other attributes. For details, see Chapter 10, *Character Sets, Collations, Unicode*. Example:

```
CREATE TABLE t (c CHAR(20) CHARACTER SET utf8 COLLATE utf8_bin);
```

MySQL 5.6 interprets length specifications in character column definitions in characters. Lengths for `BINARY` and `VARBINARY` are in bytes.

- For `CHAR`, `VARCHAR`, `BINARY`, and `VARBINARY` columns, indexes can be created that use only the leading part of column values, using *`col_name(length)`* syntax to specify an index prefix length. `BLOB` and `TEXT` columns also can be indexed, but a prefix length *must* be given. Prefix lengths are given in characters for nonbinary string types and in bytes for binary string types. That is, index entries consist of the first *`length`* characters of each column value for `CHAR`, `VARCHAR`, and `TEXT` columns, and the first *`length`* bytes of each column value for `BINARY`, `VARBINARY`, and `BLOB` columns. Indexing only a prefix of column values like this can make the index file much smaller. For additional information about index prefixes, see Section 13.1.13, “CREATE INDEX Statement”.

  Only the `InnoDB` and `MyISAM` storage engines support indexing on `BLOB` and `TEXT` columns. For example:

  ```
  CREATE TABLE test (blob_col BLOB, INDEX(blob_col(10)));
  ```

- `NOT NULL | NULL`

  If neither `NULL` nor `NOT NULL` is specified, the column is treated as though `NULL` had been specified.

  In MySQL 5.6, only the `InnoDB`, `MyISAM`, and `MEMORY` storage engines support indexes on columns that can have `NULL` values. In other cases, you must declare indexed columns as `NOT NULL` or an error results.

- `DEFAULT`

  Specifies a default value for a column. For more information about default value handling, including the case that a column definition includes no explicit `DEFAULT` value, see Section 11.5, “Data Type Default Values”.

  If the `NO_ZERO_DATE` or `NO_ZERO_IN_DATE` SQL mode is enabled and a date-valued default is not correct according to that mode, `CREATE TABLE` produces a warning if strict SQL mode is not enabled and an error if strict mode is enabled. For example, with `NO_ZERO_IN_DATE` enabled, `c1 DATE DEFAULT '2010-00-00'` produces a warning.

- `AUTO_INCREMENT`

  An integer or floating-point column can have the additional attribute `AUTO_INCREMENT`. When you insert a value of `NULL` (recommended) or `0` into an indexed `AUTO_INCREMENT` column, the column is set to the next sequence value. Typically this is *`value`*`+1`, where *`value`* is the largest value for the column currently in the table. `AUTO_INCREMENT` sequences begin with `1`.

  To retrieve an `AUTO_INCREMENT` value after inserting a row, use the `LAST_INSERT_ID()` SQL function or the `mysql_insert_id()` C API function. See Section 12.16, “Information Functions”, and mysql_insert_id().

  If the `NO_AUTO_VALUE_ON_ZERO` SQL mode is enabled, you can store `0` in `AUTO_INCREMENT` columns as `0` without generating a new sequence value. See Section 5.1.10, “Server SQL Modes”.

  There can be only one `AUTO_INCREMENT` column per table, it must be indexed, and it cannot have a `DEFAULT` value. An `AUTO_INCREMENT` column works properly only if it contains only positive values. Inserting a negative number is regarded as inserting a very large positive number. This is done to

avoid precision problems when numbers “wrap” over from positive to negative and also to ensure that you do not accidentally get an AUTO_INCREMENT column that contains 0.

For MyISAM tables, you can specify an AUTO_INCREMENT secondary column in a multiple-column key. See Section 3.6.9, “Using AUTO_INCREMENT”.

To make MySQL compatible with some ODBC applications, you can find the AUTO_INCREMENT value for the last inserted row with the following query:

```
SELECT * FROM tbl_name WHERE auto_col IS NULL
```

This method requires that sql_auto_is_null variable is not set to 0. See Section 5.1.7, “Server System Variables”.

For information about InnoDB and AUTO_INCREMENT, see Section 14.6.1.6, “AUTO_INCREMENT Handling in InnoDB”. For information about AUTO_INCREMENT and MySQL Replication, see Section 17.4.1.1, “Replication and AUTO_INCREMENT”.

- COMMENT

  A comment for a column can be specified with the COMMENT option, up to 1024 characters long. The comment is displayed by the SHOW CREATE TABLE and SHOW FULL COLUMNS statements.

- COLUMN_FORMAT

  In NDB Cluster, it is also possible to specify a data storage format for individual columns of NDB tables using COLUMN_FORMAT. Permissible column formats are FIXED, DYNAMIC, and DEFAULT. FIXED is used to specify fixed-width storage, DYNAMIC permits the column to be variable-width, and DEFAULT causes the column to use fixed-width or variable-width storage as determined by the column's data type (possibly overridden by a ROW_FORMAT specifier).

  For NDB tables, the default value for COLUMN_FORMAT is DEFAULT.

  In NDB Cluster, the maximum possible offset for a column defined with COLUMN_FORMAT=FIXED is 8188 bytes. For more information and possible workarounds, see Section 18.1.7.5, “Limits Associated with Database Objects in NDB Cluster”.

  COLUMN_FORMAT currently has no effect on columns of tables using storage engines other than NDB. In MySQL 5.6 and later, COLUMN_FORMAT is silently ignored.

- STORAGE

  For NDB tables, it is possible to specify whether the column is stored on disk or in memory by using a STORAGE clause. STORAGE DISK causes the column to be stored on disk, and STORAGE MEMORY causes in-memory storage to be used. The CREATE TABLE statement used must still include a TABLESPACE clause:

```
mysql> CREATE TABLE t1 (
    ->     c1 INT STORAGE DISK,
    ->     c2 INT STORAGE MEMORY
    -> ) ENGINE NDB;
ERROR 1005 (HY000): Can't create table 'c.t1' (errno: 140)

mysql> CREATE TABLE t1 (
    ->     c1 INT STORAGE DISK,
    ->     c2 INT STORAGE MEMORY
    -> ) TABLESPACE ts_1 ENGINE NDB;
Query OK, 0 rows affected (1.06 sec)
```

  For NDB tables, STORAGE DEFAULT is equivalent to STORAGE MEMORY.

  The STORAGE clause has no effect on tables using storage engines other than NDB. The STORAGE keyword is supported only in the build of mysqld that is supplied with NDB Cluster; it is not

recognized in any other version of MySQL, where any attempt to use the `STORAGE` keyword causes a syntax error.

Indexes and Foreign Keys

Several keywords apply to creation of indexes and foreign keys. For general background in addition to the following descriptions, see Section 13.1.13, “CREATE INDEX Statement”, and Section 13.1.17.5, “FOREIGN KEY Constraints”.

- `CONSTRAINT symbol`

  The `CONSTRAINT symbol` clause may be given to name a constraint. If the clause is not given, or a *`symbol`* is not included following the `CONSTRAINT` keyword, MySQL automatically generates a constraint name. The *`symbol`* value, if used, must be unique per schema (database), per constraint type. A duplicate *`symbol`* results in an error. See also the discussion about length limits of generated constraint identifiers at Section 9.2.1, “Identifier Length Limits”.

  The SQL standard specifies that all types of constraints (primary key, unique index, foreign key, check) belong to the same namespace. In MySQL, each constraint type has its own namespace per schema. Consequently, names for each type of constraint must be unique per schema.

- `PRIMARY KEY`

  A unique index where all key columns must be defined as `NOT NULL`. If they are not explicitly declared as `NOT NULL`, MySQL declares them so implicitly (and silently). A table can have only one `PRIMARY KEY`. The name of a `PRIMARY KEY` is always `PRIMARY`, which thus cannot be used as the name for any other kind of index.

  If you do not have a `PRIMARY KEY` and an application asks for the `PRIMARY KEY` in your tables, MySQL returns the first `UNIQUE` index that has no `NULL` columns as the `PRIMARY KEY`.

  In `InnoDB` tables, keep the `PRIMARY KEY` short to minimize storage overhead for secondary indexes. Each secondary index entry contains a copy of the primary key columns for the corresponding row. (See Section 14.6.2.1, “Clustered and Secondary Indexes”.)

  In the created table, a `PRIMARY KEY` is placed first, followed by all `UNIQUE` indexes, and then the nonunique indexes. This helps the MySQL optimizer to prioritize which index to use and also more quickly to detect duplicated `UNIQUE` keys.

  A `PRIMARY KEY` can be a multiple-column index. However, you cannot create a multiple-column index using the `PRIMARY KEY` key attribute in a column specification. Doing so only marks that single column as primary. You must use a separate `PRIMARY KEY(key_part, ...)` clause.

  If a table has a `PRIMARY KEY` or `UNIQUE NOT NULL` index that consists of a single column that has an integer type, you can use `_rowid` to refer to the indexed column in `SELECT` statements, as described in Unique Indexes.

  In MySQL, the name of a `PRIMARY KEY` is `PRIMARY`. For other indexes, if you do not assign a name, the index is assigned the same name as the first indexed column, with an optional suffix (`_2`, `_3`, `...`) to make it unique. You can see index names for a table using `SHOW INDEX FROM tbl_name`. See Section 13.7.5.23, “SHOW INDEX Statement”.

- `KEY | INDEX`

  `KEY` is normally a synonym for `INDEX`. The key attribute `PRIMARY KEY` can also be specified as just `KEY` when given in a column definition. This was implemented for compatibility with other database systems.

- `UNIQUE`

A UNIQUE index creates a constraint such that all values in the index must be distinct. An error occurs if you try to add a new row with a key value that matches an existing row. For all engines, a UNIQUE index permits multiple NULL values for columns that can contain NULL. If you specify a prefix value for a column in a UNIQUE index, the column values must be unique within the prefix length.

If a table has a PRIMARY KEY or UNIQUE NOT NULL index that consists of a single column that has an integer type, you can use _rowid to refer to the indexed column in SELECT statements, as described in Unique Indexes.

- FULLTEXT

  A FULLTEXT index is a special type of index used for full-text searches. Only the InnoDB and MyISAM storage engines support FULLTEXT indexes. They can be created only from CHAR, VARCHAR, and TEXT columns. Indexing always happens over the entire column; column prefix indexing is not supported and any prefix length is ignored if specified. See Section 12.10, “Full-Text Search Functions”, for details of operation. A WITH PARSER clause can be specified as an *index_option* value to associate a parser plugin with the index if full-text indexing and searching operations need special handling. This clause is valid only for FULLTEXT indexes. See The MySQL Plugin API, for details on creating plugins.

- SPATIAL

  You can create SPATIAL indexes on spatial data types. Spatial types are supported only for MyISAM tables and indexed columns must be declared as NOT NULL. See Section 11.4, “Spatial Data Types”.

- FOREIGN KEY

  MySQL supports foreign keys, which let you cross-reference related data across tables, and foreign key constraints, which help keep this spread-out data consistent. For definition and option information, see *reference_definition*, and *reference_option*.

  Partitioned tables employing the InnoDB storage engine do not support foreign keys. See Section 19.6, “Restrictions and Limitations on Partitioning”, for more information.

- CHECK

  The CHECK clause is parsed but ignored by all storage engines.

- *key_part*

  - A *key_part* specification can end with ASC or DESC. These keywords are permitted for future extensions for specifying ascending or descending index value storage. Currently, they are parsed but ignored; index values are always stored in ascending order.

  - Prefixes, defined by the *length* attribute, can be up to 767 bytes long for InnoDB tables or 3072 bytes if the innodb_large_prefix option is enabled. For MyISAM tables, the prefix length limit is 1000 bytes.

    Prefix *limits* are measured in bytes. However, prefix *lengths* for index specifications in CREATE TABLE, ALTER TABLE, and CREATE INDEX statements are interpreted as number of characters for nonbinary string types (CHAR, VARCHAR, TEXT) and number of bytes for binary string types (BINARY, VARBINARY, BLOB). Take this into account when specifying a prefix length for a nonbinary string column that uses a multibyte character set.

- *index_type*

  Some storage engines permit you to specify an index type when creating an index. The syntax for the *index_type* specifier is USING *type_name*.

Example:

```
CREATE TABLE lookup
  (id INT, INDEX USING BTREE (id))
  ENGINE = MEMORY;
```

The preferred position for `USING` is after the index column list. It can be given before the column list, but support for use of the option in that position is deprecated; expect it to be removed in a future MySQL release.

- *`index_option`*

  `index_option` values specify additional options for an index.

  - `KEY_BLOCK_SIZE`

    For `MyISAM` tables, `KEY_BLOCK_SIZE` optionally specifies the size in bytes to use for index key blocks. The value is treated as a hint; a different size could be used if necessary. A `KEY_BLOCK_SIZE` value specified for an individual index definition overrides the table-level `KEY_BLOCK_SIZE` value.

    For information about the table-level `KEY_BLOCK_SIZE` attribute, see Table Options.

  - `WITH PARSER`

    A `WITH PARSER` clause can be specified as an *`index_option`* value to associate a parser plugin with the index if full-text indexing and searching operations need special handling. This clause is valid only for `FULLTEXT` indexes. See The MySQL Plugin API, for details on creating plugins.

  - `COMMENT`

    Index definitions can include an optional comment of up to 1024 characters.

  For more information about permissible *`index_option`* values, see Section 13.1.13, “CREATE INDEX Statement”. For more information about indexes, see Section 8.3.1, “How MySQL Uses Indexes”.

- *`reference_definition`*

  For *`reference_definition`* syntax details and examples, see Section 13.1.17.5, “FOREIGN KEY Constraints”.

  `InnoDB` and `NDB` tables support checking of foreign key constraints. The columns of the referenced table must always be explicitly named. Both `ON DELETE` and `ON UPDATE` actions on foreign keys are supported. For more detailed information and examples, see Section 13.1.17.5, “FOREIGN KEY Constraints”.

  For other storage engines, MySQL Server parses and ignores the `FOREIGN KEY` and `REFERENCES` syntax in `CREATE TABLE` statements. See Section 1.7.2.3, “FOREIGN KEY Constraint Differences”.

Important

For users familiar with the ANSI/ISO SQL Standard, please note that no storage engine, including `InnoDB`, recognizes or enforces the `MATCH` clause used in referential integrity constraint definitions. Use of an explicit `MATCH` clause does not have the specified effect, and also causes `ON DELETE` and `ON UPDATE` clauses to be ignored. For these reasons, specifying `MATCH` should be avoided.

The `MATCH` clause in the SQL standard controls how `NULL` values in a composite (multiple-column) foreign key are handled when comparing to a

> primary key. `InnoDB` essentially implements the semantics defined by `MATCH SIMPLE`, which permit a foreign key to be all or partially `NULL`. In that case, the (child table) row containing such a foreign key is permitted to be inserted, and does not match any row in the referenced (parent) table. It is possible to implement other semantics using triggers.
>
> Additionally, MySQL requires that the referenced columns be indexed for performance. However, it does not enforce any requirement that the referenced columns be declared `UNIQUE` or `NOT NULL`. The handling of foreign key references to nonunique keys or keys that contain `NULL` values is not well defined for operations such as `UPDATE` or `DELETE CASCADE`. You are advised to use foreign keys that reference only keys that are both `UNIQUE` (or `PRIMARY`) and `NOT NULL`.
>
> MySQL parses but ignores "inline `REFERENCES` specifications" (as defined in the SQL standard) where the references are defined as part of the column specification. MySQL accepts `REFERENCES` clauses only when specified as part of a separate `FOREIGN KEY` specification.

- *`reference_option`*

  For information about the `RESTRICT`, `CASCADE`, `SET NULL`, `NO ACTION`, and `SET DEFAULT` options, see Section 13.1.17.5, "FOREIGN KEY Constraints".

Table Options

Table options are used to optimize the behavior of the table. In most cases, you do not have to specify any of them. These options apply to all storage engines unless otherwise indicated. Options that do not apply to a given storage engine may be accepted and remembered as part of the table definition. Such options then apply if you later use `ALTER TABLE` to convert the table to use a different storage engine.

- `ENGINE`

  Specifies the storage engine for the table, using one of the names shown in the following table. The engine name can be unquoted or quoted. The quoted name `'DEFAULT'` is recognized but ignored.

Storage Engine	Description
`InnoDB`	Transaction-safe tables with row locking and foreign keys. The default storage engine for new tables. See Chapter 14, *The InnoDB Storage Engine*, and in particular Section 14.1, "Introduction to InnoDB" if you have MySQL experience but are new to `InnoDB`.
`MyISAM`	The binary portable storage engine that is primarily used for read-only or read-mostly workloads. See Section 15.2, "The MyISAM Storage Engine".
`MEMORY`	The data for this storage engine is stored only in memory. See Section 15.3, "The MEMORY Storage Engine".
`CSV`	Tables that store rows in comma-separated values format. See Section 15.4, "The CSV Storage Engine".
`ARCHIVE`	The archiving storage engine. See Section 15.5, "The ARCHIVE Storage Engine".
`EXAMPLE`	An example engine. See Section 15.9, "The EXAMPLE Storage Engine".
`FEDERATED`	Storage engine that accesses remote tables. See Section 15.8, "The FEDERATED Storage Engine".
`HEAP`	This is a synonym for `MEMORY`.

Storage Engine	Description
MERGE	A collection of MyISAM tables used as one table. Also known as MRG_MyISAM. See Section 15.7, “The MERGE Storage Engine”.
NDB	Clustered, fault-tolerant, memory-based tables, supporting transactions and foreign keys. Also known as NDBCLUSTER. See Chapter 18, *MySQL NDB Cluster 7.3 and NDB Cluster 7.4*.

If a storage engine is specified that is not available, MySQL uses the default engine instead. Normally, this is MyISAM. For example, if a table definition includes the ENGINE=INNODB option but the MySQL server does not support INNODB tables, the table is created as a MyISAM table. This makes it possible to have a replication setup where you have transactional tables on the source but tables created on the replica are nontransactional (to get more speed). In MySQL 5.6, a warning occurs if the storage engine specification is not honored.

Engine substitution can be controlled by the setting of the NO_ENGINE_SUBSTITUTION SQL mode, as described in Section 5.1.10, “Server SQL Modes”.

Note

The older TYPE option that was synonymous with ENGINE was removed in MySQL 5.5. *When upgrading to MySQL 5.5 or later, you must convert existing applications that rely on TYPE to use ENGINE instead*.

- AUTO_INCREMENT

  The initial AUTO_INCREMENT value for the table. In MySQL 5.6, this works for MyISAM, MEMORY, InnoDB, and ARCHIVE tables. To set the first auto-increment value for engines that do not support the AUTO_INCREMENT table option, insert a “dummy” row with a value one less than the desired value after creating the table, and then delete the dummy row.

  For engines that support the AUTO_INCREMENT table option in CREATE TABLE statements, you can also use ALTER TABLE *tbl_name* AUTO_INCREMENT = *N* to reset the AUTO_INCREMENT value. The value cannot be set lower than the maximum value currently in the column.

- AVG_ROW_LENGTH

  An approximation of the average row length for your table. You need to set this only for large tables with variable-size rows.

  When you create a MyISAM table, MySQL uses the product of the MAX_ROWS and AVG_ROW_LENGTH options to decide how big the resulting table is. If you don't specify either option, the maximum size for MyISAM data and index files is 256TB by default. (If your operating system does not support files that large, table sizes are constrained by the file size limit.) If you want to keep down the pointer sizes to make the index smaller and faster and you don't really need big files, you can decrease the default pointer size by setting the myisam_data_pointer_size system variable. (See Section 5.1.7, “Server System Variables”.) If you want all your tables to be able to grow above the default limit and are willing to have your tables slightly slower and larger than necessary, you can increase the default pointer size by setting this variable. Setting the value to 7 permits table sizes up to 65,536TB.

- [DEFAULT] CHARACTER SET

  Specifies a default character set for the table. CHARSET is a synonym for CHARACTER SET. If the character set name is DEFAULT, the database character set is used.

- CHECKSUM

  Set this to 1 if you want MySQL to maintain a live checksum for all rows (that is, a checksum that MySQL updates automatically as the table changes). This makes the table a little slower to update,

but also makes it easier to find corrupted tables. The `CHECKSUM TABLE` statement reports the checksum. (`MyISAM` only.)

- `[DEFAULT] COLLATE`

  Specifies a default collation for the table.

- `COMMENT`

  A comment for the table, up to 2048 characters long.

- `CONNECTION`

  The connection string for a `FEDERATED` table.

  Note

  Older versions of MySQL used a `COMMENT` option for the connection string.

- `DATA DIRECTORY`, `INDEX DIRECTORY`

  For `InnoDB`, the `DATA DIRECTORY='directory'` clause permits creating a table outside of the data directory. The `innodb_file_per_table` variable must be enabled to use the `DATA DIRECTORY` clause. The full directory path must be specified. For more information, see Section 14.6.1.2, "Creating Tables Externally".

  When creating `MyISAM` tables, you can use the `DATA DIRECTORY='directory'` clause, the `INDEX DIRECTORY='directory'` clause, or both. They specify where to put a `MyISAM` table's data file and index file, respectively. Unlike `InnoDB` tables, MySQL does not create subdirectories that correspond to the database name when creating a `MyISAM` table with a `DATA DIRECTORY` or `INDEX DIRECTORY` option. Files are created in the directory that is specified.

  As of MySQL 5.6.35, you must have the `FILE` privilege to use the `DATA DIRECTORY` or `INDEX DIRECTORY` table option.

  Important

  Table-level `DATA DIRECTORY` and `INDEX DIRECTORY` options are ignored for partitioned tables. (Bug #32091)

  These options work only when you are not using the `--skip-symbolic-links` option. Your operating system must also have a working, thread-safe `realpath()` call. See Section 8.12.3.2, "Using Symbolic Links for MyISAM Tables on Unix", for more complete information.

  If a `MyISAM` table is created with no `DATA DIRECTORY` option, the `.MYD` file is created in the database directory. By default, if `MyISAM` finds an existing `.MYD` file in this case, it overwrites it. The same applies to `.MYI` files for tables created with no `INDEX DIRECTORY` option. To suppress this behavior, start the server with the `--keep_files_on_create` option, in which case `MyISAM` does not overwrite existing files and returns an error instead.

  If a `MyISAM` table is created with a `DATA DIRECTORY` or `INDEX DIRECTORY` option and an existing `.MYD` or `.MYI` file is found, MyISAM always returns an error. It does not overwrite a file in the specified directory.

  Important

  You cannot use path names that contain the MySQL data directory with `DATA DIRECTORY` or `INDEX DIRECTORY`. This includes partitioned tables and individual table partitions. (See Bug #32167.)

- `DELAY_KEY_WRITE`

Set this to 1 if you want to delay key updates for the table until the table is closed. See the description of the `delay_key_write` system variable in Section 5.1.7, “Server System Variables”. (`MyISAM` only.)

- `INSERT_METHOD`

  If you want to insert data into a `MERGE` table, you must specify with `INSERT_METHOD` the table into which the row should be inserted. `INSERT_METHOD` is an option useful for `MERGE` tables only. Use a value of `FIRST` or `LAST` to have inserts go to the first or last table, or a value of `NO` to prevent inserts. See Section 15.7, “The MERGE Storage Engine”.

- `KEY_BLOCK_SIZE`

  For `MyISAM` tables, `KEY_BLOCK_SIZE` optionally specifies the size in bytes to use for index key blocks. The value is treated as a hint; a different size could be used if necessary. A `KEY_BLOCK_SIZE` value specified for an individual index definition overrides the table-level `KEY_BLOCK_SIZE` value.

  For `InnoDB` tables, `KEY_BLOCK_SIZE` specifies the page size in kilobytes to use for compressed `InnoDB` tables. The `KEY_BLOCK_SIZE` value is treated as a hint; a different size could be used by `InnoDB` if necessary. `KEY_BLOCK_SIZE` can only be less than or equal to the `innodb_page_size` value. A value of 0 represents the default compressed page size, which is half of the `innodb_page_size` value. Depending on `innodb_page_size`, possible `KEY_BLOCK_SIZE` values include 0, 1, 2, 4, 8, and 16. See InnoDB Table Compression for more information.

  Oracle recommends enabling `innodb_strict_mode` when specifying `KEY_BLOCK_SIZE` for `InnoDB` tables. When `innodb_strict_mode` is enabled, specifying an invalid `KEY_BLOCK_SIZE` value returns an error. If `innodb_strict_mode` is disabled, an invalid `KEY_BLOCK_SIZE` value results in a warning, and the `KEY_BLOCK_SIZE` option is ignored.

  The `Create_options` column in response to `SHOW TABLE STATUS` reports the originally specified `KEY_BLOCK_SIZE` option, as does `SHOW CREATE TABLE`.

  `InnoDB` only supports `KEY_BLOCK_SIZE` at the table level.

- `MAX_ROWS`

  The maximum number of rows you plan to store in the table. This is not a hard limit, but rather a hint to the storage engine that the table must be able to store at least this many rows.

  The `NDB` storage engine treats this value as a maximum. If you plan to create very large NDB Cluster tables (containing millions of rows), you should use this option to insure that `NDB` allocates sufficient number of index slots in the hash table used for storing hashes of the table's primary keys by setting `MAX_ROWS = 2 * rows`, where *`rows`* is the number of rows that you expect to insert into the table.

  The maximum `MAX_ROWS` value is 4294967295; larger values are truncated to this limit.

- `MIN_ROWS`

  The minimum number of rows you plan to store in the table. The `MEMORY` storage engine uses this option as a hint about memory use.

- `PACK_KEYS`

  Takes effect only with `MyISAM` tables. Set this option to 1 if you want to have smaller indexes. This usually makes updates slower and reads faster. Setting the option to 0 disables all packing of

keys. Setting it to `DEFAULT` tells the storage engine to pack only long `CHAR`, `VARCHAR`, `BINARY`, or `VARBINARY` columns.

If you do not use `PACK_KEYS`, the default is to pack strings, but not numbers. If you use `PACK_KEYS=1`, numbers are packed as well.

When packing binary number keys, MySQL uses prefix compression:

- Every key needs one extra byte to indicate how many bytes of the previous key are the same for the next key.
- The pointer to the row is stored in high-byte-first order directly after the key, to improve compression.

This means that if you have many equal keys on two consecutive rows, all following "same" keys usually only take two bytes (including the pointer to the row). Compare this to the ordinary case where the following keys takes `storage_size_for_key + pointer_size` (where the pointer size is usually 4). Conversely, you get a significant benefit from prefix compression only if you have many numbers that are the same. If all keys are totally different, you use one byte more per key, if the key is not a key that can have `NULL` values. (In this case, the packed key length is stored in the same byte that is used to mark if a key is `NULL`.)

- `PASSWORD`

  This option is unused. If you have a need to scramble your `.frm` files and make them unusable to any other MySQL server, please contact our sales department.

- `ROW_FORMAT`

  Defines the physical format in which the rows are stored.

  When creating a table with strict mode disabled, the storage engine's default row format is used if the specified row format is not supported. The actual row format of the table is reported in the `Row_format` column in response to `SHOW TABLE STATUS`. The `Create_options` column shows the row format that was specified in the `CREATE TABLE` statement, as does `SHOW CREATE TABLE`.

  Row format choices differ depending on the storage engine used for the table.

  For `InnoDB` tables:

  - Rows are stored in compact format (`ROW_FORMAT=COMPACT`) by default.
  - The noncompact format used in older versions of MySQL can still be requested by specifying `ROW_FORMAT=REDUNDANT`.
  - To enable compression for `InnoDB` tables, specify `ROW_FORMAT=COMPRESSED` and follow the procedures in Section 14.9, "InnoDB Table Compression".
  - For more efficient `InnoDB` storage of data types, especially `BLOB` types, specify `ROW_FORMAT=DYNAMIC` and follow the procedures in DYNAMIC Row Format. Both the `COMPRESSED` and `DYNAMIC` row formats require creating the table with the configuration settings `innodb_file_per_table=1` and `innodb_file_format=barracuda`.
  - When you specify a non-default `ROW_FORMAT` clause, consider also enabling the `innodb_strict_mode` configuration option.
  - `ROW_FORMAT=FIXED` is not supported. If `ROW_FORMAT=FIXED` is specified while `innodb_strict_mode` is disabled, `InnoDB` issues a warning and assumes

ROW_FORMAT=COMPACT. If ROW_FORMAT=FIXED is specified while innodb_strict_mode is enabled, InnoDB returns an error.

- For additional information about InnoDB row formats, see Section 14.11, "InnoDB Row Formats".

For MyISAM tables, the option value can be FIXED or DYNAMIC for static or variable-length row format. myisampack sets the type to COMPRESSED. See Section 15.2.3, "MyISAM Table Storage Formats".

- STATS_AUTO_RECALC

  Specifies whether to automatically recalculate persistent statistics for an InnoDB table. The value DEFAULT causes the persistent statistics setting for the table to be determined by the innodb_stats_auto_recalc configuration option. The value 1 causes statistics to be recalculated when 10% of the data in the table has changed. The value 0 prevents automatic recalculation for this table; with this setting, issue an ANALYZE TABLE statement to recalculate the statistics after making substantial changes to the table. For more information about the persistent statistics feature, see Section 14.8.11.1, "Configuring Persistent Optimizer Statistics Parameters".

- STATS_PERSISTENT

  Specifies whether to enable persistent statistics for an InnoDB table. The value DEFAULT causes the persistent statistics setting for the table to be determined by the innodb_stats_persistent configuration option. The value 1 enables persistent statistics for the table, while the value 0 turns off this feature. After enabling persistent statistics through a CREATE TABLE or ALTER TABLE statement, issue an ANALYZE TABLE statement to calculate the statistics, after loading representative data into the table. For more information about the persistent statistics feature, see Section 14.8.11.1, "Configuring Persistent Optimizer Statistics Parameters".

- STATS_SAMPLE_PAGES

  The number of index pages to sample when estimating cardinality and other statistics for an indexed column, such as those calculated by ANALYZE TABLE. For more information, see Section 14.8.11.1, "Configuring Persistent Optimizer Statistics Parameters".

- TABLESPACE

  The TABLESPACE and STORAGE table options are employed only with NDB tables. The tablespace named *tablespace_name* must already have been created using CREATE TABLESPACE. STORAGE determines the type of storage used (disk or memory), and can be either DISK or MEMORY.

  TABLESPACE ... STORAGE DISK assigns a table to an NDB Cluster Disk Data tablespace. See Section 18.5.10, "NDB Cluster Disk Data Tables", for more information.

  Important

  A STORAGE clause cannot be used in a CREATE TABLE statement without a TABLESPACE clause.

- UNION

  Used to access a collection of identical MyISAM tables as one. This works only with MERGE tables. See Section 15.7, "The MERGE Storage Engine".

  You must have SELECT, UPDATE, and DELETE privileges for the tables you map to a MERGE table.

  Note

  Formerly, all tables used had to be in the same database as the MERGE table itself. This restriction no longer applies.

Table Partitioning

partition_options can be used to control partitioning of the table created with CREATE TABLE.

Not all options shown in the syntax for *partition_options* at the beginning of this section are available for all partitioning types. Please see the listings for the following individual types for information specific to each type, and see Chapter 19, *Partitioning*, for more complete information about the workings of and uses for partitioning in MySQL, as well as additional examples of table creation and other statements relating to MySQL partitioning.

Partitions can be modified, merged, added to tables, and dropped from tables. For basic information about the MySQL statements to accomplish these tasks, see Section 13.1.7, "ALTER TABLE Statement". For more detailed descriptions and examples, see Section 19.3, "Partition Management".

- PARTITION BY

  If used, a *partition_options* clause begins with PARTITION BY. This clause contains the function that is used to determine the partition; the function returns an integer value ranging from 1 to *num*, where *num* is the number of partitions. (The maximum number of user-defined partitions which a table may contain is 1024; the number of subpartitions—discussed later in this section—is included in this maximum.)

  Note

  The expression (*expr*) used in a PARTITION BY clause cannot refer to any columns not in the table being created; such references are specifically not permitted and cause the statement to fail with an error. (Bug #29444)

- HASH(*expr*)

  Hashes one or more columns to create a key for placing and locating rows. *expr* is an expression using one or more table columns. This can be any valid MySQL expression (including MySQL functions) that yields a single integer value. For example, these are both valid CREATE TABLE statements using PARTITION BY HASH:

  ```
  CREATE TABLE t1 (col1 INT, col2 CHAR(5))
      PARTITION BY HASH(col1);

  CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATETIME)
      PARTITION BY HASH ( YEAR(col3) );
  ```

  You may not use either VALUES LESS THAN or VALUES IN clauses with PARTITION BY HASH.

  PARTITION BY HASH uses the remainder of *expr* divided by the number of partitions (that is, the modulus). For examples and additional information, see Section 19.2.4, "HASH Partitioning".

  The LINEAR keyword entails a somewhat different algorithm. In this case, the number of the partition in which a row is stored is calculated as the result of one or more logical AND operations. For discussion and examples of linear hashing, see Section 19.2.4.1, "LINEAR HASH Partitioning".

- KEY(*column_list*)

  This is similar to HASH, except that MySQL supplies the hashing function so as to guarantee an even data distribution. The *column_list* argument is simply a list of 1 or more table columns (maximum: 16). This example shows a simple table partitioned by key, with 4 partitions:

  ```
  CREATE TABLE tk (col1 INT, col2 CHAR(5), col3 DATE)
      PARTITION BY KEY(col3)
      PARTITIONS 4;
  ```

  For tables that are partitioned by key, you can employ linear partitioning by using the LINEAR keyword. This has the same effect as with tables that are partitioned by HASH. That is, the partition number is found using the & operator rather than the modulus (see Section 19.2.4.1, "LINEAR

HASH Partitioning", and Section 19.2.5, "KEY Partitioning", for details). This example uses linear partitioning by key to distribute data between 5 partitions:

```
CREATE TABLE tk (col1 INT, col2 CHAR(5), col3 DATE)
    PARTITION BY LINEAR KEY(col3)
    PARTITIONS 5;
```

The `ALGORITHM={1 | 2}` option is supported with `[SUB]PARTITION BY [LINEAR] KEY` beginning with MySQL 5.6.11. `ALGORITHM=1` causes the server to use the same key-hashing functions as MySQL 5.1; `ALGORITHM=2` means that the server employs the key-hashing functions implemented and used by default for new `KEY` partitioned tables in MySQL 5.5 and later. (Partitioned tables created with the key-hashing functions employed in MySQL 5.5 and later cannot be used by a MySQL 5.1 server.) Not specifying the option has the same effect as using `ALGORITHM=2`. This option is intended for use chiefly when upgrading or downgrading `[LINEAR] KEY` partitioned tables between MySQL 5.1 and later MySQL versions, or for creating tables partitioned by `KEY` or `LINEAR KEY` on a MySQL 5.5 or later server which can be used on a MySQL 5.1 server. For more information, see Section 13.1.7.1, "ALTER TABLE Partition Operations".

`mysqldump` in MySQL 5.6.11 and later writes this option encased in versioned comments, like this:

```
CREATE TABLE t1 (a INT)
/*!50100 PARTITION BY KEY */ /*!50611 ALGORITHM = 1 */ /*!50100 ()
      PARTITIONS 3 */
```

This causes MySQL 5.6.10 and earlier servers to ignore the option, which would otherwise cause a syntax error in those versions. If you plan to load a dump made on a MySQL 5.5.31 or later MySQL 5.5 server where you use tables that are partitioned or subpartitioned by `KEY` into a MySQL 5.6 server previous to version 5.6.11, be sure to consult Section 2.11.3, "Changes in MySQL 5.6", before proceeding. (The information found there also applies if you are loading a dump containing `KEY` partitioned or subpartitioned tables made from a MySQL 5.6.11 or later server into a MySQL 5.5.30 or earlier server.)

Also in MySQL 5.6.11 and later, `ALGORITHM=1` is shown when necessary in the output of `SHOW CREATE TABLE` using versioned comments in the same manner as `mysqldump`. `ALGORITHM=2` is always omitted from `SHOW CREATE TABLE` output, even if this option was specified when creating the original table.

You may not use either `VALUES LESS THAN` or `VALUES IN` clauses with `PARTITION BY KEY`.

- `RANGE(expr)`

  In this case, *`expr`* shows a range of values using a set of `VALUES LESS THAN` operators. When using range partitioning, you must define at least one partition using `VALUES LESS THAN`. You cannot use `VALUES IN` with range partitioning.

  Note

  For tables partitioned by `RANGE`, `VALUES LESS THAN` must be used with either an integer literal value or an expression that evaluates to a single integer value. In MySQL 5.6, you can overcome this limitation in a table that is defined using `PARTITION BY RANGE COLUMNS`, as described later in this section.

  Suppose that you have a table that you wish to partition on a column containing year values, according to the following scheme.

Partition Number:	Years Range:
0	1990 and earlier
1	1991 to 1994
2	1995 to 1998

Partition Number:	Years Range:
3	1999 to 2002
4	2003 to 2005
5	2006 and later

A table implementing such a partitioning scheme can be realized by the `CREATE TABLE` statement shown here:

```
CREATE TABLE t1 (
    year_col  INT,
    some_data INT
)
PARTITION BY RANGE (year_col) (
    PARTITION p0 VALUES LESS THAN (1991),
    PARTITION p1 VALUES LESS THAN (1995),
    PARTITION p2 VALUES LESS THAN (1999),
    PARTITION p3 VALUES LESS THAN (2002),
    PARTITION p4 VALUES LESS THAN (2006),
    PARTITION p5 VALUES LESS THAN MAXVALUE
);
```

`PARTITION ... VALUES LESS THAN ...` statements work in a consecutive fashion. `VALUES LESS THAN MAXVALUE` works to specify “leftover” values that are greater than the maximum value otherwise specified.

`VALUES LESS THAN` clauses work sequentially in a manner similar to that of the `case` portions of a `switch ... case` block (as found in many programming languages such as C, Java, and PHP). That is, the clauses must be arranged in such a way that the upper limit specified in each successive `VALUES LESS THAN` is greater than that of the previous one, with the one referencing `MAXVALUE` coming last of all in the list.

- `RANGE COLUMNS(`*`column_list`*`)`

  This variant on `RANGE` facilitates partition pruning for queries using range conditions on multiple columns (that is, having conditions such as `WHERE a = 1 AND b < 10` or `WHERE a = 1 AND b = 10 AND c < 10`). It enables you to specify value ranges in multiple columns by using a list of columns in the `COLUMNS` clause and a set of column values in each `PARTITION ... VALUES LESS THAN (`*`value_list`*`)` partition definition clause. (In the simplest case, this set consists of a single column.) The maximum number of columns that can be referenced in the *`column_list`* and *`value_list`* is 16.

  The *`column_list`* used in the `COLUMNS` clause may contain only names of columns; each column in the list must be one of the following MySQL data types: the integer types; the string types; and time or date column types. Columns using `BLOB`, `TEXT`, `SET`, `ENUM`, `BIT`, or spatial data types are not permitted; columns that use floating-point number types are also not permitted. You also may not use functions or arithmetic expressions in the `COLUMNS` clause.

  The `VALUES LESS THAN` clause used in a partition definition must specify a literal value for each column that appears in the `COLUMNS()` clause; that is, the list of values used for each `VALUES LESS THAN` clause must contain the same number of values as there are columns listed in the `COLUMNS` clause. An attempt to use more or fewer values in a `VALUES LESS THAN` clause than there are in the `COLUMNS` clause causes the statement to fail with the error `Inconsistency in usage of column lists for partitioning...`. You cannot use `NULL` for any value appearing in `VALUES LESS THAN`. It is possible to use `MAXVALUE` more than once for a given column other than the first, as shown in this example:

```
CREATE TABLE rc (
    a INT NOT NULL,
    b INT NOT NULL
)
PARTITION BY RANGE COLUMNS(a,b) (
    PARTITION p0 VALUES LESS THAN (10,5),
```

```
    PARTITION p1 VALUES LESS THAN (20,10),
    PARTITION p2 VALUES LESS THAN (50,MAXVALUE),
    PARTITION p3 VALUES LESS THAN (65,MAXVALUE),
    PARTITION p4 VALUES LESS THAN (MAXVALUE,MAXVALUE)
);
```

Each value used in a `VALUES LESS THAN` value list must match the type of the corresponding column exactly; no conversion is made. For example, you cannot use the string `'1'` for a value that matches a column that uses an integer type (you must use the numeral `1` instead), nor can you use the numeral `1` for a value that matches a column that uses a string type (in such a case, you must use a quoted string: `'1'`).

For more information, see Section 19.2.1, “RANGE Partitioning”, and Section 19.4, “Partition Pruning”.

- `LIST(expr)`

  This is useful when assigning partitions based on a table column with a restricted set of possible values, such as a state or country code. In such a case, all rows pertaining to a certain state or country can be assigned to a single partition, or a partition can be reserved for a certain set of states or countries. It is similar to `RANGE`, except that only `VALUES IN` may be used to specify permissible values for each partition.

  `VALUES IN` is used with a list of values to be matched. For instance, you could create a partitioning scheme such as the following:

```
CREATE TABLE client_firms (
    id   INT,
    name VARCHAR(35)
)
PARTITION BY LIST (id) (
    PARTITION r0 VALUES IN (1, 5, 9, 13, 17, 21),
    PARTITION r1 VALUES IN (2, 6, 10, 14, 18, 22),
    PARTITION r2 VALUES IN (3, 7, 11, 15, 19, 23),
    PARTITION r3 VALUES IN (4, 8, 12, 16, 20, 24)
);
```

  When using list partitioning, you must define at least one partition using `VALUES IN`. You cannot use `VALUES LESS THAN` with `PARTITION BY LIST`.

  Note

  For tables partitioned by `LIST`, the value list used with `VALUES IN` must consist of integer values only. In MySQL 5.6, you can overcome this limitation using partitioning by `LIST COLUMNS`, which is described later in this section.

- `LIST COLUMNS(column_list)`

  This variant on `LIST` facilitates partition pruning for queries using comparison conditions on multiple columns (that is, having conditions such as `WHERE a = 5 AND b = 5` or `WHERE a = 1 AND b = 10 AND c = 5`). It enables you to specify values in multiple columns by using a list of columns in the `COLUMNS` clause and a set of column values in each `PARTITION ... VALUES IN (value_list)` partition definition clause.

  The rules governing regarding data types for the column list used in `LIST COLUMNS(column_list)` and the value list used in `VALUES IN(value_list)` are the same as those for the column list used in `RANGE COLUMNS(column_list)` and the value list used in `VALUES LESS THAN(value_list)`, respectively, except that in the `VALUES IN` clause, `MAXVALUE` is not permitted, and you may use `NULL`.

  There is one important difference between the list of values used for `VALUES IN` with `PARTITION BY LIST COLUMNS` as opposed to when it is used with `PARTITION BY LIST`. When used with `PARTITION BY LIST COLUMNS`, each element in the `VALUES IN` clause must be a *set* of column

values; the number of values in each set must be the same as the number of columns used in the `COLUMNS` clause, and the data types of these values must match those of the columns (and occur in the same order). In the simplest case, the set consists of a single column. The maximum number of columns that can be used in the *`column_list`* and in the elements making up the *`value_list`* is 16.

The table defined by the following `CREATE TABLE` statement provides an example of a table using `LIST COLUMNS` partitioning:

```
CREATE TABLE lc (
    a INT NULL,
    b INT NULL
)
PARTITION BY LIST COLUMNS(a,b) (
    PARTITION p0 VALUES IN( (0,0), (NULL,NULL) ),
    PARTITION p1 VALUES IN( (0,1), (0,2), (0,3), (1,1), (1,2) ),
    PARTITION p2 VALUES IN( (1,0), (2,0), (2,1), (3,0), (3,1) ),
    PARTITION p3 VALUES IN( (1,3), (2,2), (2,3), (3,2), (3,3) )
);
```

- `PARTITIONS` *`num`*

  The number of partitions may optionally be specified with a `PARTITIONS` *`num`* clause, where *`num`* is the number of partitions. If both this clause *and* any `PARTITION` clauses are used, *`num`* must be equal to the total number of any partitions that are declared using `PARTITION` clauses.

  Note

  Whether or not you use a `PARTITIONS` clause in creating a table that is partitioned by `RANGE` or `LIST`, you must still include at least one `PARTITION VALUES` clause in the table definition (see below).

- `SUBPARTITION BY`

  A partition may optionally be divided into a number of subpartitions. This can be indicated by using the optional `SUBPARTITION BY` clause. Subpartitioning may be done by `HASH` or `KEY`. Either of these may be `LINEAR`. These work in the same way as previously described for the equivalent partitioning types. (It is not possible to subpartition by `LIST` or `RANGE`.)

  The number of subpartitions can be indicated using the `SUBPARTITIONS` keyword followed by an integer value.

- Rigorous checking of the value used in `PARTITIONS` or `SUBPARTITIONS` clauses is applied and this value must adhere to the following rules:

  - The value must be a positive, nonzero integer.
  - No leading zeros are permitted.
  - The value must be an integer literal, and cannot not be an expression. For example, `PARTITIONS 0.2E+01` is not permitted, even though `0.2E+01` evaluates to `2`. (Bug #15890)

- *partition_definition*

  Each partition may be individually defined using a *partition_definition* clause. The individual parts making up this clause are as follows:

  - PARTITION *partition_name*

    Specifies a logical name for the partition.

  - VALUES

    For range partitioning, each partition must include a VALUES LESS THAN clause; for list partitioning, you must specify a VALUES IN clause for each partition. This is used to determine which rows are to be stored in this partition. See the discussions of partitioning types in Chapter 19, *Partitioning*, for syntax examples.

  - [STORAGE] ENGINE

    The partitioning handler accepts a [STORAGE] ENGINE option for both PARTITION and SUBPARTITION. Currently, the only way in which this can be used is to set all partitions or all subpartitions to the same storage engine, and an attempt to set different storage engines for partitions or subpartitions in the same table raises the error ERROR 1469 (HY000): The mix of handlers in the partitions is not permitted in this version of MySQL. We expect to lift this restriction on partitioning in a future MySQL release.

  - COMMENT

    An optional COMMENT clause may be used to specify a string that describes the partition. Example:

    ```
    COMMENT = 'Data for the years previous to 1999'
    ```

    The maximum length for a partition comment is 1024 characters.

  - DATA DIRECTORY and INDEX DIRECTORY

    DATA DIRECTORY and INDEX DIRECTORY may be used to indicate the directory where, respectively, the data and indexes for this partition are to be stored. Both the *data_dir* and the *index_dir* must be absolute system path names.

    As of MySQL 5.6.35, you must have the FILE privilege to use the DATA DIRECTORY or INDEX DIRECTORY partition option.

    Example:

    ```
    CREATE TABLE th (id INT, name VARCHAR(30), adate DATE)
    PARTITION BY LIST(YEAR(adate))
    (
      PARTITION p1999 VALUES IN (1995, 1999, 2003)
        DATA DIRECTORY = '/var/appdata/95/data'
        INDEX DIRECTORY = '/var/appdata/95/idx',
      PARTITION p2000 VALUES IN (1996, 2000, 2004)
        DATA DIRECTORY = '/var/appdata/96/data'
        INDEX DIRECTORY = '/var/appdata/96/idx',
      PARTITION p2001 VALUES IN (1997, 2001, 2005)
        DATA DIRECTORY = '/var/appdata/97/data'
        INDEX DIRECTORY = '/var/appdata/97/idx',
      PARTITION p2002 VALUES IN (1998, 2002, 2006)
        DATA DIRECTORY = '/var/appdata/98/data'
        INDEX DIRECTORY = '/var/appdata/98/idx'
    ```

```
);
```

`DATA DIRECTORY` and `INDEX DIRECTORY` behave in the same way as in the `CREATE TABLE` statement's *`table_option`* clause as used for `MyISAM` tables.

One data directory and one index directory may be specified per partition. If left unspecified, the data and indexes are stored by default in the table's database directory.

On Windows, the `DATA DIRECTORY` and `INDEX DIRECTORY` options are not supported for individual partitions or subpartitions of `MyISAM` tables, and the `INDEX DIRECTORY` option is not supported for individual partitions or subpartitions of `InnoDB` tables. These options are ignored on Windows, except that a warning is generated. (Bug #30459)

Note

The `DATA DIRECTORY` and `INDEX DIRECTORY` options are ignored for creating partitioned tables if `NO_DIR_IN_CREATE` is in effect. (Bug #24633)

- `MAX_ROWS` and `MIN_ROWS`

  May be used to specify, respectively, the maximum and minimum number of rows to be stored in the partition. The values for *`max_number_of_rows`* and *`min_number_of_rows`* must be positive integers. As with the table-level options with the same names, these act only as “suggestions” to the server and are not hard limits.

- `TABLESPACE`

  May be used to designate a tablespace for the partition. Used for NDB Cluster only.

- *`subpartition_definition`*

  The partition definition may optionally contain one or more *`subpartition_definition`* clauses. Each of these consists at a minimum of the `SUBPARTITION` *`name`*, where *`name`* is an identifier for the subpartition. Except for the replacement of the `PARTITION` keyword with `SUBPARTITION`, the syntax for a subpartition definition is identical to that for a partition definition.

  Subpartitioning must be done by `HASH` or `KEY`, and can be done only on `RANGE` or `LIST` partitions. See Section 19.2.6, “Subpartitioning”.

13.1.17.1 Files Created by CREATE TABLE

MySQL represents each table by an `.frm` table format (definition) file in the database directory. The storage engine for the table might create other files as well.

For `InnoDB` tables, the file storage is controlled by the `innodb_file_per_table` configuration option. For each `InnoDB` table created when this option is turned on, the table data and all associated indexes are stored in a .ibd file located inside the database directory. When this option is turned off, all `InnoDB` tables and indexes are stored in the system tablespace, represented by one or more ibdata* files.

For `MyISAM` tables, the storage engine creates data and index files. Thus, for each `MyISAM` table *`tbl_name`*, there are three disk files.

File	Purpose
`tbl_name``.frm`	Table format (definition) file
`tbl_name``.MYD`	Data file
`tbl_name``.MYI`	Index file

Chapter 15, *Alternative Storage Engines*, describes what files each storage engine creates to represent tables. If a table name contains special characters, the names for the table files contain encoded versions of those characters as described in Section 9.2.4, “Mapping of Identifiers to File Names”.

Limits Imposed by .frm File Structure

As described previously, each table has an `.frm` file that contains the table definition. The server uses the following expression to check some of the table information stored in the file against an upper limit of 64KB:

```
if (info_length+(ulong) create_fields.elements*FCOMP+288+
    n_length+int_length+com_length > 65535L || int_count > 255)
```

The portion of the information stored in the `.frm` file that is checked against the expression cannot grow beyond the 64KB limit, so if the table definition reaches this size, no more columns can be added.

The relevant factors in the expression are:

- `info_length` is space needed for “screens.” This is related to MySQL's Unireg heritage.
- `create_fields.elements` is the number of columns.
- `FCOMP` is 17.
- `n_length` is the total length of all column names, including one byte per name as a separator.
- `int_length` is related to the list of values for `ENUM` and `SET` columns. In this context, “int” does not mean “integer.” It means “interval,” a term that refers collectively to `ENUM` and `SET` columns.
- `int_count` is the number of unique `ENUM` and `SET` definitions.
- `com_length` is the total length of column comments.

The expression just described has several implications for permitted table definitions:

- Using long column names can reduce the maximum number of columns, as can the inclusion of `ENUM` or `SET` columns, or use of column comments.
- A table can have no more than 255 unique `ENUM` and `SET` definitions. Columns with identical element lists are considered the same against this limt. For example, if a table contains these two columns, they count as one (not two) toward this limit because the definitions are identical:

  ```
  e1 ENUM('a','b','c')
  e2 ENUM('a','b','c')
  ```

- The sum of the length of element names in the unique `ENUM` and `SET` definitions counts toward the 64KB limit, so although the theoretical limit on number of elements in a given `ENUM` column is 65,535, the practical limit is less than 3000.

13.1.17.2 CREATE TEMPORARY TABLE Statement

You can use the `TEMPORARY` keyword when creating a table. A `TEMPORARY` table is visible only within the current session, and is dropped automatically when the session is closed. This means that two different sessions can use the same temporary table name without conflicting with each other or with an existing non-`TEMPORARY` table of the same name. (The existing table is hidden until the temporary table is dropped.)

`CREATE TABLE` causes an implicit commit, except when used with the `TEMPORARY` keyword. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

`TEMPORARY` tables have a very loose relationship with databases (schemas). Dropping a database does not automatically drop any `TEMPORARY` tables created within that database. Also, you can create

a TEMPORARY table in a nonexistent database if you qualify the table name with the database name in the CREATE TABLE statement. In this case, all subsequent references to the table must be qualified with the database name.

To create a temporary table, you must have the CREATE TEMPORARY TABLES privilege. After a session has created a temporary table, the server performs no further privilege checks on the table. The creating session can perform any operation on the table, such as DROP TABLE, INSERT, UPDATE, or SELECT.

One implication of this behavior is that a session can manipulate its temporary tables even if the current user has no privilege to create them. Suppose that the current user does not have the CREATE TEMPORARY TABLES privilege but is able to execute a definer-context stored procedure that executes with the privileges of a user who does have CREATE TEMPORARY TABLES and that creates a temporary table. While the procedure executes, the session uses the privileges of the defining user. After the procedure returns, the effective privileges revert to those of the current user, which can still see the temporary table and perform any operation on it.

13.1.17.3 CREATE TABLE ... LIKE Statement

Use CREATE TABLE ... LIKE to create an empty table based on the definition of another table, including any column attributes and indexes defined in the original table:

```
CREATE TABLE new_tbl LIKE orig_tbl;
```

The copy is created using the same version of the table storage format as the original table. The SELECT privilege is required on the original table.

LIKE works only for base tables, not for views.

Important

You cannot execute CREATE TABLE or CREATE TABLE ... LIKE while a LOCK TABLES statement is in effect.

CREATE TABLE ... LIKE makes the same checks as CREATE TABLE and does not just copy the .frm file. This means that if the current SQL mode is different from the mode in effect when the original table was created, the table definition might be considered invalid for the new mode and the statement fails.

CREATE TABLE ... LIKE does not preserve any DATA DIRECTORY or INDEX DIRECTORY table options that were specified for the original table, or any foreign key definitions.

If the original table is a TEMPORARY table, CREATE TABLE ... LIKE does not preserve TEMPORARY. To create a TEMPORARY destination table, use CREATE TEMPORARY TABLE ... LIKE.

13.1.17.4 CREATE TABLE ... SELECT Statement

You can create one table from another by adding a SELECT statement at the end of the CREATE TABLE statement:

```
CREATE TABLE new_tbl [AS] SELECT * FROM orig_tbl;
```

MySQL creates new columns for all elements in the SELECT. For example:

```
mysql> CREATE TABLE test (a INT NOT NULL AUTO_INCREMENT,
    ->        PRIMARY KEY (a), KEY(b))
    ->        ENGINE=MyISAM SELECT b,c FROM test2;
```

This creates a MyISAM table with three columns, a, b, and c. The ENGINE option is part of the CREATE TABLE statement, and should not be used following the SELECT; this would result in a syntax error. The same is true for other CREATE TABLE options such as CHARSET.

Notice that the columns from the SELECT statement are appended to the right side of the table, not overlapped onto it. Take the following example:

```
mysql> SELECT * FROM foo;
+---+
| n |
+---+
| 1 |
+---+

mysql> CREATE TABLE bar (m INT) SELECT n FROM foo;
Query OK, 1 row affected (0.02 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM bar;
+------+---+
| m    | n |
+------+---+
| NULL | 1 |
+------+---+
1 row in set (0.00 sec)
```

For each row in table foo, a row is inserted in bar with the values from foo and default values for the new columns.

In a table resulting from CREATE TABLE ... SELECT, columns named only in the CREATE TABLE part come first. Columns named in both parts or only in the SELECT part come after that. The data type of SELECT columns can be overridden by also specifying the column in the CREATE TABLE part.

If any errors occur while copying the data to the table, it is automatically dropped and not created.

You can precede the SELECT by IGNORE or REPLACE to indicate how to handle rows that duplicate unique key values. With IGNORE, rows that duplicate an existing row on a unique key value are discarded. With REPLACE, new rows replace rows that have the same unique key value. If neither IGNORE nor REPLACE is specified, duplicate unique key values result in an error.

Because the ordering of the rows in the underlying SELECT statements cannot always be determined, CREATE TABLE ... IGNORE SELECT and CREATE TABLE ... REPLACE SELECT statements are flagged as unsafe for statement-based replication. Such statements produce a warning in the error log when using statement-based mode and are written to the binary log using the row-based format when using MIXED mode. See also Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”.

CREATE TABLE ... SELECT does not automatically create any indexes for you. This is done intentionally to make the statement as flexible as possible. If you want to have indexes in the created table, you should specify these before the SELECT statement:

```
mysql> CREATE TABLE bar (UNIQUE (n)) SELECT n FROM foo;
```

Some conversion of data types might occur. For example, the AUTO_INCREMENT attribute is not preserved, and VARCHAR columns can become CHAR columns. Retrained attributes are NULL (or NOT NULL) and, for those columns that have them, CHARACTER SET, COLLATION, COMMENT, and the DEFAULT clause.

When creating a table with CREATE TABLE ... SELECT, make sure to alias any function calls or expressions in the query. If you do not, the CREATE statement might fail or result in undesirable column names.

```
CREATE TABLE artists_and_works
  SELECT artist.name, COUNT(work.artist_id) AS number_of_works
  FROM artist LEFT JOIN work ON artist.id = work.artist_id
  GROUP BY artist.id;
```

You can also explicitly specify the data type for a column in the created table:

```
CREATE TABLE foo (a TINYINT NOT NULL) SELECT b+1 AS a FROM bar;
```

For `CREATE TABLE ... SELECT`, if `IF NOT EXISTS` is given and the destination table already exists, the result is version dependent. Before MySQL 5.5.6, MySQL handles the statement as follows:

- The table definition given in the `CREATE TABLE` part is ignored. No error occurs, even if the definition does not match that of the existing table. MySQL attempts to insert the rows from the `SELECT` part anyway.
- If there is a mismatch between the number of columns in the table and the number of columns produced by the `SELECT` part, the selected values are assigned to the rightmost columns. For example, if the table contains n columns and the `SELECT` produces m columns, where $m < n$, the selected values are assigned to the m rightmost columns in the table. Each of the initial $n - m$ columns is assigned its default value, either that specified explicitly in the column definition or the implicit column data type default if the definition contains no default. If the `SELECT` part produces too many columns ($m > n$), an error occurs.
- If strict SQL mode is enabled and any of these initial columns do not have an explicit default value, the statement fails with an error.

The following example illustrates `IF NOT EXISTS` handling:

```
mysql> CREATE TABLE t1 (i1 INT DEFAULT 0, i2 INT, i3 INT, i4 INT);
Query OK, 0 rows affected (0.05 sec)

mysql> CREATE TABLE IF NOT EXISTS t1 (c1 CHAR(10)) SELECT 1, 2;
Query OK, 1 row affected, 1 warning (0.01 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM t1;
+------+------+------+------+
| i1   | i2   | i3   | i4   |
+------+------+------+------+
|    0 | NULL |    1 |    2 |
+------+------+------+------+
1 row in set (0.00 sec)
```

As of MySQL 5.5.6, handling of `CREATE TABLE IF NOT EXISTS ... SELECT` statements was changed for the case that the destination table already exists. This change also involves a change in MySQL 5.1 beginning with 5.1.51.

- Previously, for `CREATE TABLE IF NOT EXISTS ... SELECT`, MySQL produced a warning that the table exists, but inserted the rows and wrote the statement to the binary log anyway. By contrast, `CREATE TABLE ... SELECT` (without `IF NOT EXISTS`) failed with an error, but MySQL inserted no rows and did not write the statement to the binary log.
- MySQL now handles both statements the same way when the destination table exists, in that neither statement inserts rows or is written to the binary log. The difference between them is that MySQL produces a warning when `IF NOT EXISTS` is present and an error when it is not.

This change means that, for the preceding example, the `CREATE TABLE IF NOT EXISTS ... SELECT` statement inserts nothing into the destination table as of MySQL 5.5.6.

This change in handling of `IF NOT EXISTS` results in an incompatibility for statement-based replication from a MySQL 5.1 source with the original behavior and a MySQL 5.5 replica with the new behavior. Suppose that `CREATE TABLE IF NOT EXISTS ... SELECT` is executed on the source and the destination table exists. The result is that rows are inserted on the source but not on the replica. (Row-based replication does not have this problem.)

To address this issue, statement-based binary logging for `CREATE TABLE IF NOT EXISTS ... SELECT` is changed in MySQL 5.1 as of 5.1.51:

- If the destination table does not exist, there is no change: The statement is logged as is.
- If the destination table does exist, the statement is logged as the equivalent pair of `CREATE TABLE IF NOT EXISTS` and `INSERT ... SELECT` statements. (If the `SELECT` in the original statement

is preceded by `IGNORE` or `REPLACE`, the `INSERT` becomes `INSERT IGNORE` or `REPLACE`, respectively.)

This change provides forward compatibility for statement-based replication from MySQL 5.1 to 5.5 because when the destination table exists, the rows are inserted on both the source and replica. To take advantage of this compatibility measure, the 5.1 server must be at least 5.1.51 and the 5.5 server must be at least 5.5.6.

To upgrade an existing 5.1-to-5.5 replication scenario, upgrade the source first to 5.1.51 or higher. Note that this differs from the usual replication upgrade advice of upgrading the replica first.

A workaround for applications that wish to achieve the original effect (rows inserted regardless of whether the destination table exists) is to use `CREATE TABLE IF NOT EXISTS` and `INSERT ... SELECT` statements rather than `CREATE TABLE IF NOT EXISTS ... SELECT` statements.

Along with the change just described, the following related change was made: Previously, if an existing view was named as the destination table for `CREATE TABLE IF NOT EXISTS ... SELECT`, rows were inserted into the underlying base table and the statement was written to the binary log. As of MySQL 5.1.51 and 5.5.6, nothing is inserted or logged.

To ensure that the binary log can be used to re-create the original tables, MySQL does not permit concurrent inserts during `CREATE TABLE ... SELECT`.

Important

You cannot use `FOR UPDATE` as part of the `SELECT` in a statement such as `CREATE TABLE new_table SELECT ... FROM old_table ...`. If you attempt to do so, the statement fails. This represents a change in behavior from MySQL 5.5 and earlier, which permitted `CREATE TABLE ... SELECT` statements to make changes in tables other than the table being created.

This change can also have implications for statement-based replication from an older source to a MySQL 5.6 or higher replica. See Section 17.4.1.7, “Replication of CREATE TABLE ... SELECT Statements”, for more information.

13.1.17.5 FOREIGN KEY Constraints

MySQL supports foreign keys, which permit cross-referencing related data across tables, and foreign key constraints, which help keep the related data consistent.

A foreign key relationship involves a parent table that holds the initial column values, and a child table with column values that reference the parent column values. A foreign key constraint is defined on the child table.

The essential syntax for a defining a foreign key constraint in a `CREATE TABLE` or `ALTER TABLE` statement includes the following:

```
[CONSTRAINT [symbol]] FOREIGN KEY
    [index_name] (col_name, ...)
    REFERENCES tbl_name (col_name,...)
    [ON DELETE reference_option]
    [ON UPDATE reference_option]

reference_option:
    RESTRICT | CASCADE | SET NULL | NO ACTION | SET DEFAULT
```

Foreign key constraint usage is described under the following topics in this section:

- Identifiers
- Conditions and Restrictions
- Referential Actions

- Foreign Key Constraint Examples
- Adding Foreign Key Constraints
- Dropping Foreign Key Constraints
- Foreign Key Checks
- Foreign Key Definitions and Metadata
- Foreign Key Errors

Identifiers

Foreign key constraint naming is governed by the following rules:

- The `CONSTRAINT` *`symbol`* value is used, if defined.
- If the `CONSTRAINT` *`symbol`* clause is not defined, or a symbol is not included following the `CONSTRAINT` keyword:
  - For `InnoDB` tables, a constraint name is generated automatically.
  - For `NDB` tables, the `FOREIGN KEY` *`index_name`* value is used, if defined. Otherwise, a constraint name is generated automatically.
- The `CONSTRAINT` *`symbol`* value, if defined, must be unique in the database. A duplicate *`symbol`* results in an error similar to: `ERROR 1005 (HY000): Can't create table 'test.fk1' (errno: 121)`.

Table and column identifiers in a `FOREIGN KEY ... REFERENCES` clause can be quoted within backticks (`` ` ``). Alternatively, double quotation marks (`"`) can be used if the `ANSI_QUOTES` SQL mode is enabled. The `lower_case_table_names` system variable setting is also taken into account.

Conditions and Restrictions

Foreign key constraints are subject to the following conditions and restrictions:

- Parent and child tables must use the same storage engine, and they cannot be defined as temporary tables.
- Creating a foreign key constraint requires at least one of the `SELECT`, `INSERT`, `UPDATE`, `DELETE`, or `REFERENCES` privileges on the parent table as of 5.6.22.
- Corresponding columns in the foreign key and the referenced key must have similar data types. *The size and sign of fixed precision types such as* *`INTEGER`* *and* *`DECIMAL`* *must be the same*. The length of string types need not be the same. For nonbinary (character) string columns, the character set and collation must be the same.
- MySQL supports foreign key references between one column and another within a table. (A column cannot have a foreign key reference to itself.) In these cases, a “child table record” refers to a dependent record within the same table.
- MySQL requires indexes on foreign keys and referenced keys so that foreign key checks can be fast and not require a table scan. In the referencing table, there must be an index where the foreign key columns are listed as the *first* columns in the same order. Such an index is created on the referencing table automatically if it does not exist. This index might be silently dropped later if you create another index that can be used to enforce the foreign key constraint. *`index_name`*, if given, is used as described previously.
- `InnoDB` permits a foreign key to reference any index column or group of columns. However, in the referenced table, there must be an index where the referenced columns are the *first* columns

in the same order. Hidden columns that `InnoDB` adds to an index are also considered (see Section 14.6.2.1, “Clustered and Secondary Indexes”).

`NDB` requires an explicit unique key (or primary key) on any column referenced as a foreign key. `InnoDB` does not, which is an extension of standard SQL.

- Index prefixes on foreign key columns are not supported. Consequently, `BLOB` and `TEXT` columns cannot be included in a foreign key because indexes on those columns must always include a prefix length.
- `InnoDB` does not currently support foreign keys for tables with user-defined partitioning. This includes both parent and child tables.

  This restriction does not apply for `NDB` tables that are partitioned by `KEY` or `LINEAR KEY` (the only user partitioning types supported by the `NDB` storage engine); these may have foreign key references or be the targets of such references.
- A table in a foreign key relationship cannot be altered to use another storage engine. To change the storage engine, you must drop any foreign key constraints first.

For information about how the MySQL implementation of foreign key constraints differs from the SQL standard, see Section 1.7.2.3, “FOREIGN KEY Constraint Differences”.

Referential Actions

When an `UPDATE` or `DELETE` operation affects a key value in the parent table that has matching rows in the child table, the result depends on the *referential action* specified by `ON UPDATE` and `ON DELETE` subclauses of the `FOREIGN KEY` clause. Referential actions include:

- `CASCADE`: Delete or update the row from the parent table and automatically delete or update the matching rows in the child table. Both `ON DELETE CASCADE` and `ON UPDATE CASCADE` are supported. Between two tables, do not define several `ON UPDATE CASCADE` clauses that act on the same column in the parent table or in the child table.

  Note

  Cascaded foreign key actions do not activate triggers.
- `SET NULL`: Delete or update the row from the parent table and set the foreign key column or columns in the child table to `NULL`. Both `ON DELETE SET NULL` and `ON UPDATE SET NULL` clauses are supported.

  If you specify a `SET NULL` action, *make sure that you have not declared the columns in the child table as `NOT NULL`*.
- `RESTRICT`: Rejects the delete or update operation for the parent table. Specifying `RESTRICT` (or `NO ACTION`) is the same as omitting the `ON DELETE` or `ON UPDATE` clause.
- `NO ACTION`: A keyword from standard SQL. In MySQL, equivalent to `RESTRICT`. The MySQL Server rejects the delete or update operation for the parent table if there is a related foreign key value in the referenced table. Some database systems have deferred checks, and `NO ACTION` is a deferred check. In MySQL, foreign key constraints are checked immediately, so `NO ACTION` is the same as `RESTRICT`.
- `SET DEFAULT`: This action is recognized by the MySQL parser, but both `InnoDB` and `NDB` reject table definitions containing `ON DELETE SET DEFAULT` or `ON UPDATE SET DEFAULT` clauses.

For storage engines that support foreign keys, MySQL rejects any `INSERT` or `UPDATE` operation that attempts to create a foreign key value in a child table if there is no matching candidate key value in the parent table.

For an `ON DELETE` or `ON UPDATE` that is not specified, the default action is always `RESTRICT`.

For `NDB` tables, `ON UPDATE CASCADE` is not supported where the reference is to the parent table's primary key.

As of NDB 7.3.25 and NDB 7.4.24: For `NDB` tables, `ON DELETE CASCADE` is not supported where the child table contains one or more columns of any of the `TEXT` or `BLOB` types. (Bug #89511, Bug #27484882)

`InnoDB` performs cascading operations using a depth-first search algorithm on the records of the index that corresponds to the foreign key constraint.

Foreign Key Constraint Examples

This simple example relates `parent` and `child` tables through a single-column foreign key:

```
CREATE TABLE parent (
    id INT NOT NULL,
    PRIMARY KEY (id)
) ENGINE=INNODB;

CREATE TABLE child (
    id INT,
    parent_id INT,
    INDEX par_ind (parent_id),
    FOREIGN KEY (parent_id)
        REFERENCES parent(id)
        ON DELETE CASCADE
) ENGINE=INNODB;
```

This is a more complex example in which a `product_order` table has foreign keys for two other tables. One foreign key references a two-column index in the `product` table. The other references a single-column index in the `customer` table:

```
CREATE TABLE product (
    category INT NOT NULL, id INT NOT NULL,
    price DECIMAL,
    PRIMARY KEY(category, id)
)   ENGINE=INNODB;

CREATE TABLE customer (
    id INT NOT NULL,
    PRIMARY KEY (id)
)   ENGINE=INNODB;

CREATE TABLE product_order (
    no INT NOT NULL AUTO_INCREMENT,
    product_category INT NOT NULL,
    product_id INT NOT NULL,
    customer_id INT NOT NULL,

    PRIMARY KEY(no),
    INDEX (product_category, product_id),
    INDEX (customer_id),

    FOREIGN KEY (product_category, product_id)
      REFERENCES product(category, id)
      ON UPDATE CASCADE ON DELETE RESTRICT,

    FOREIGN KEY (customer_id)
      REFERENCES customer(id)
)   ENGINE=INNODB;
```

Adding Foreign Key Constraints

You can add a foreign key constraint to an existing table using the following `ALTER TABLE` syntax:

```
ALTER TABLE tbl_name
```

```
    ADD [CONSTRAINT [symbol]] FOREIGN KEY
    [index_name] (col_name, ...)
    REFERENCES tbl_name (col_name,...)
    [ON DELETE reference_option]
    [ON UPDATE reference_option]
```

The foreign key can be self referential (referring to the same table). When you add a foreign key constraint to a table using `ALTER TABLE`, *remember to first create an index on the column(s) referenced by the foreign key.*

Dropping Foreign Key Constraints

You can drop a foreign key constraint using the following `ALTER TABLE` syntax:

```
ALTER TABLE tbl_name DROP FOREIGN KEY fk_symbol;
```

If the `FOREIGN KEY` clause defined a `CONSTRAINT` name when you created the constraint, you can refer to that name to drop the foreign key constraint. Otherwise, a constraint name was generated internally, and you must use that value. To determine the foreign key constraint name, use `SHOW CREATE TABLE`:

```
mysql> SHOW CREATE TABLE child\G
*************************** 1. row ***************************
       Table: child
Create Table: CREATE TABLE `child` (
  `id` int(11) DEFAULT NULL,
  `parent_id` int(11) DEFAULT NULL,
  KEY `par_ind` (`parent_id`),
  CONSTRAINT `child_ibfk_1` FOREIGN KEY (`parent_id`)
  REFERENCES `parent` (`id`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1

mysql> ALTER TABLE child DROP FOREIGN KEY `child_ibfk_1`;
```

Adding and dropping a foreign key in the same `ALTER TABLE` statement is supported for `ALTER TABLE ... ALGORITHM=INPLACE`. It is not supported for `ALTER TABLE ... ALGORITHM=COPY`.

Foreign Key Checks

Foreign key checking is controlled by the `foreign_key_checks` variable, which is enabled by default. Typically, you leave this variable enabled during normal operation to enforce referential integrity. Beginning with MySQL NDB Cluster 7.3.2, the `foreign_key_checks` variable has the same effect on `NDB` tables as it does for `InnoDB` tables. Previously, the setting was ignored for NDB tables and all such checks were enforced (Bug #14095855).

The `foreign_key_checks` variable is dynamic and supports both global and session scopes. For information about using system variables, see Section 5.1.8, “Using System Variables”.

Disabling foreign key checking is useful when:

- Dropping a table that is referenced by a foreign key constraint. A referenced table can only be dropped after `foreign_key_checks` is disabled. When you drop a table, constraints defined on the table are also dropped.
- Reloading tables in different order than required by their foreign key relationships. For example, `mysqldump` produces correct definitions of tables in the dump file, including foreign key constraints for child tables. To make it easier to reload dump files for tables with foreign key relationships, `mysqldump` automatically includes a statement in the dump output that disables `foreign_key_checks`. This enables you to import the tables in any order in case the dump file contains tables that are not correctly ordered for foreign keys. Disabling `foreign_key_checks` also speeds up the import operation by avoiding foreign key checks.
- Executing `LOAD DATA` operations, to avoid foreign key checking.

- Performing an `ALTER TABLE` operation on a table that has a foreign key relationship.

When `foreign_key_checks` is disabled, foreign key constraints are ignored, with the following exceptions:

- Recreating a table that was previously dropped returns an error if the table definition does not conform to the foreign key constraints that reference the table. The table must have the correct column names and types. It must also have indexes on the referenced keys. If these requirements are not satisfied, MySQL returns Error 1005 that refers to errno: 150 in the error message, which means that a foreign key constraint was not correctly formed.
- Altering a table returns an error (errno: 150) if a foreign key definition is incorrectly formed for the altered table.
- Dropping an index required by a foreign key constraint. The foreign key constraint must be removed before dropping the index.
- Creating a foreign key constraint where a column references a nonmatching column type.

Disabling `foreign_key_checks` has these additional implications:

- It is permitted to drop a database that contains tables with foreign keys that are referenced by tables outside the database.
- It is permitted to drop a table with foreign keys referenced by other tables.
- Enabling `foreign_key_checks` does not trigger a scan of table data, which means that rows added to a table while `foreign_key_checks` is disabled are not checked for consistency when `foreign_key_checks` is re-enabled.

Foreign Key Definitions and Metadata

To view a foreign key definition, use `SHOW CREATE TABLE`:

```
mysql> SHOW CREATE TABLE child\G
*************************** 1. row ***************************
       Table: child
Create Table: CREATE TABLE `child` (
  `id` int(11) DEFAULT NULL,
  `parent_id` int(11) DEFAULT NULL,
  KEY `par_ind` (`parent_id`),
  CONSTRAINT `child_ibfk_1` FOREIGN KEY (`parent_id`)
  REFERENCES `parent` (`id`) ON DELETE CASCADE
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

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

```
mysql> SELECT TABLE_SCHEMA, TABLE_NAME, COLUMN_NAME, CONSTRAINT_NAME
       FROM INFORMATION_SCHEMA.KEY_COLUMN_USAGE
       WHERE REFERENCED_TABLE_SCHEMA IS NOT NULL;
+-------------+-----------+-------------+-----------------+
| TABLE_SCHEMA | TABLE_NAME | COLUMN_NAME | CONSTRAINT_NAME |
+-------------+-----------+-------------+-----------------+
| test         | child      | parent_id   | child_ibfk_1    |
+-------------+-----------+-------------+-----------------+
```

You can obtain information specific to `InnoDB` foreign keys from the `INNODB_SYS_FOREIGN` and `INNODB_SYS_FOREIGN_COLS` tables. Example queries are show here:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_FOREIGN \G
*************************** 1. row ***************************
      ID: test/child_ibfk_1
FOR_NAME: test/child
REF_NAME: test/parent
```

```
  N_COLS: 1
    TYPE: 1

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_FOREIGN_COLS \G
*************************** 1. row ***************************
          ID: test/child_ibfk_1
FOR_COL_NAME: parent_id
REF_COL_NAME: id
         POS: 0
```

Foreign Key Errors

In the event of a foreign key error involving `InnoDB` tables (usually Error 150 in the MySQL Server), information about the latest foreign key error can be obtained by checking `SHOW ENGINE INNODB STATUS` output.

```
mysql> SHOW ENGINE INNODB STATUS\G
...
------------------------
LATEST FOREIGN KEY ERROR
------------------------
2014-10-17 09:51:31 7f47bcde6700 Transaction:
TRANSACTION 436786, ACTIVE 0 sec inserting
mysql tables in use 1, locked 1
4 lock struct(s), heap size 1184, 3 row lock(s), undo log entries 3
MySQL thread id 1, OS thread handle 0x7f47bcde6700, query id 96 localhost
root update
INSERT INTO child VALUES
    (NULL, 1)
    , (NULL, 2)
    , (NULL, 3)
    , (NULL, 4)
    , (NULL, 5)
    , (NULL, 6)
Foreign key constraint fails for table `mysql`.`child`:
,
  CONSTRAINT `child_ibfk_1` FOREIGN KEY (`parent_id`) REFERENCES `parent`
  (`id`)
 ON DELETE CASCADE ON UPDATE CASCADE
Trying to add in child table, in index `par_ind` tuple:
DATA TUPLE: 2 fields;
 0: len 4; hex 80000003; asc     ;;
 1: len 4; hex 80000003; asc     ;;

But in parent table `mysql`.`parent`, in index `PRIMARY`,
the closest match we can find is record:
PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 80000004; asc     ;;
 1: len 6; hex 00000006aa26; asc      &;;
 2: len 7; hex 9d000001610137; asc     a 7;;
------------------------
LATEST DETECTED DEADLOCK
------------------------
2014-10-17 09:52:38 7f47bcde6700
*** (1) TRANSACTION:
TRANSACTION 436801, ACTIVE 12 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 360, 1 row lock(s)
MySQL thread id 2, OS thread handle 0x7f47bcda5700, query id 102 localhost
root updating
DELETE FROM t WHERE i = 1
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 3693 page no 3 n bits 72 index `GEN_CLUST_INDEX` of
table `mysql`.`t` trx id 436801 lock_mode X waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 4; compact format; info
bits 0
 0: len 6; hex 000000003a00; asc     : ;;
 1: len 6; hex 00000006aa3f; asc      ?;;
 2: len 7; hex ad0000021d0110; asc        ;;
 3: len 4; hex 80000001; asc     ;;
...
```

Warning

`ER_NO_REFERENCED_ROW_2` and `ER_ROW_IS_REFERENCED_2` error messages for foreign key operations expose information about parent tables, even if the user has no parent table access privileges. To hide information about parent tables, include the appropriate condition handlers in application code and stored programs.

13.1.17.6 Silent Column Specification Changes

In some cases, MySQL silently changes column specifications from those given in a `CREATE TABLE` or `ALTER TABLE` statement. These might be changes to a data type, to attributes associated with a data type, or to an index specification.

All changes are subject to the internal row-size limit of 65,535 bytes, which may cause some attempts at data type changes to fail. See Section 8.4.7, “Limits on Table Column Count and Row Size”.

- Columns that are part of a `PRIMARY KEY` are made `NOT NULL` even if not declared that way.
- Trailing spaces are automatically deleted from `ENUM` and `SET` member values when the table is created.
- MySQL maps certain data types used by other SQL database vendors to MySQL types. See Section 11.8, “Using Data Types from Other Database Engines”.
- If you include a `USING` clause to specify an index type that is not permitted for a given storage engine, but there is another index type available that the engine can use without affecting query results, the engine uses the available type.
- If strict SQL mode is not enabled, a `VARCHAR` column with a length specification greater than 65535 is converted to `TEXT`, and a `VARBINARY` column with a length specification greater than 65535 is converted to `BLOB`. Otherwise, an error occurs in either of these cases.
- Specifying the `CHARACTER SET binary` attribute for a character data type causes the column to be created as the corresponding binary data type: `CHAR` becomes `BINARY`, `VARCHAR` becomes `VARBINARY`, and `TEXT` becomes `BLOB`. For the `ENUM` and `SET` data types, this does not occur; they are created as declared. Suppose that you specify a table using this definition:

```
CREATE TABLE t
(
  c1 VARCHAR(10) CHARACTER SET binary,
  c2 TEXT CHARACTER SET binary,
  c3 ENUM('a','b','c') CHARACTER SET binary
);
```

The resulting table has this definition:

```
CREATE TABLE t
(
  c1 VARBINARY(10),
  c2 BLOB,
  c3 ENUM('a','b','c') CHARACTER SET binary
);
```

To see whether MySQL used a data type other than the one you specified, issue a `DESCRIBE` or `SHOW CREATE TABLE` statement after creating or altering the table.

Certain other data type changes can occur if you compress a table using `myisampack`. See Section 15.2.3.3, “Compressed Table Characteristics”.

13.1.18 CREATE TABLESPACE Statement

```
CREATE TABLESPACE tablespace_name
    ADD DATAFILE 'file_name'
```

```
    USE LOGFILE GROUP logfile_group
    [EXTENT_SIZE [=] extent_size]
    [INITIAL_SIZE [=] initial_size]
    [AUTOEXTEND_SIZE [=] autoextend_size]
    [MAX_SIZE [=] max_size]
    [NODEGROUP [=] nodegroup_id]
    [WAIT]
    [COMMENT [=] comment_text]
    ENGINE [=] engine_name
```

This statement is used to create a tablespace, which can contain one or more data files, providing storage space for tables. One data file is created and added to the tablespace using this statement. Additional data files may be added to the tablespace by using the `ALTER TABLESPACE` statement (see Section 13.1.8, "ALTER TABLESPACE Statement"). For rules covering the naming of tablespaces, see Section 9.2, "Schema Object Names".

Note

All NDB Cluster Disk Data objects share the same namespace. This means that *each Disk Data object* must be uniquely named (and not merely each Disk Data object of a given type). For example, you cannot have a tablespace and a log file group with the same name, or a tablespace and a data file with the same name.

A log file group of one or more `UNDO` log files must be assigned to the tablespace to be created with the `USE LOGFILE GROUP` clause. *`logfile_group`* must be an existing log file group created with `CREATE LOGFILE GROUP` (see Section 13.1.14, "CREATE LOGFILE GROUP Statement"). Multiple tablespaces may use the same log file group for `UNDO` logging.

The `EXTENT_SIZE` sets the size, in bytes, of the extents used by any files belonging to the tablespace. The default value is 1M. The minimum size is 32K, and theoretical maximum is 2G, although the practical maximum size depends on a number of factors. In most cases, changing the extent size does not have any measurable effect on performance, and the default value is recommended for all but the most unusual situations.

An *extent* is a unit of disk space allocation. One extent is filled with as much data as that extent can contain before another extent is used. In theory, up to 65,535 (64K) extents may used per data file; however, the recommended maximum is 32,768 (32K). The recommended maximum size for a single data file is 32G—that is, 32K extents × 1 MB per extent. In addition, once an extent is allocated to a given partition, it cannot be used to store data from a different partition; an extent cannot store data from more than one partition. This means, for example that a tablespace having a single datafile whose `INITIAL_SIZE` is 256 MB and whose `EXTENT_SIZE` is 128M has just two extents, and so can be used to store data from at most two different disk data table partitions.

You can see how many extents remain free in a given data file by querying the `INFORMATION_SCHEMA.FILES` table, and so derive an estimate for how much space remains free in the file. For further discussion and examples, see Section 21.31.1, "The INFORMATION_SCHEMA FILES Table".

The `INITIAL_SIZE` parameter sets the data file's total size in bytes. Once the file has been created, its size cannot be changed; however, you can add more data files to the tablespace using `ALTER TABLESPACE ... ADD DATAFILE`. See Section 13.1.8, "ALTER TABLESPACE Statement".

`INITIAL_SIZE` is optional; its default value is 134217728 (128 MB).

On 32-bit systems, the maximum supported value for `INITIAL_SIZE` is 4294967296 (4 GB). (Bug #29186)

When setting `EXTENT_SIZE`, you may optionally follow the number with a one-letter abbreviation for an order of magnitude, similar to those used in `my.cnf`. Generally, this is one of the letters `M` (for megabytes) or `G` (for gigabytes). In MySQL NDB Cluster 7.3.2 and later, these abbreviations are also supported when specifying `INITIAL_SIZE` as well. (Bug #13116514, Bug #16104705, Bug #62858)

INITIAL_SIZE, EXTENT_SIZE, and UNDO_BUFFER_SIZE are subject to rounding as follows:

- EXTENT_SIZE and UNDO_BUFFER_SIZE are each rounded up to the nearest whole multiple of 32K.
- INITIAL_SIZE is rounded *down* to the nearest whole multiple of 32K.

  For data files, *INITIAL_SIZE* is subject to further rounding; the result just obtained is rounded up to the nearest whole multiple of EXTENT_SIZE (after any rounding).

The rounding just described is done explicitly, and a warning is issued by the MySQL Server when any such rounding is performed. The rounded values are also used by the NDB kernel for calculating INFORMATION_SCHEMA.FILES column values and other purposes. However, to avoid an unexpected result, we suggest that you always use whole multiples of 32K in specifying these options.

AUTOEXTEND_SIZE, MAX_SIZE, NODEGROUP, WAIT, and COMMENT are parsed but ignored, and so currently have no effect. These options are intended for future expansion.

The ENGINE parameter determines the storage engine which uses this tablespace, with *engine_name* being the name of the storage engine. Currently, *engine_name* must be one of the values NDB or NDBCLUSTER.

When CREATE TABLESPACE is used with ENGINE = NDB, a tablespace and associated data file are created on each Cluster data node. You can verify that the data files were created and obtain information about them by querying the INFORMATION_SCHEMA.FILES table. For example:

```
mysql> SELECT LOGFILE_GROUP_NAME, FILE_NAME, EXTRA
    -> FROM INFORMATION_SCHEMA.FILES
    -> WHERE TABLESPACE_NAME = 'newts' AND FILE_TYPE = 'DATAFILE';
+--------------------+-------------+----------------+
| LOGFILE_GROUP_NAME | FILE_NAME   | EXTRA          |
+--------------------+-------------+----------------+
| lg_3               | newdata.dat | CLUSTER_NODE=3 |
| lg_3               | newdata.dat | CLUSTER_NODE=4 |
+--------------------+-------------+----------------+
2 rows in set (0.01 sec)
```

(See Section 21.31.1, “The INFORMATION_SCHEMA FILES Table”.)

CREATE TABLESPACE is useful only with Disk Data storage for NDB Cluster. See Section 18.5.10, “NDB Cluster Disk Data Tables”.

13.1.19 CREATE TRIGGER Statement

```
CREATE
    [DEFINER = user]
    TRIGGER trigger_name
    trigger_time trigger_event
    ON tbl_name FOR EACH ROW
    trigger_body

trigger_time: { BEFORE | AFTER }

trigger_event: { INSERT | UPDATE | DELETE }
```

This statement creates a new trigger. A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. The trigger becomes associated with the table named *tbl_name*, which must refer to a permanent table. You cannot associate a trigger with a TEMPORARY table or a view.

Trigger names exist in the schema namespace, meaning that all triggers must have unique names within a schema. Triggers in different schemas can have the same name.

This section describes CREATE TRIGGER syntax. For additional discussion, see Section 20.3.1, “Trigger Syntax and Examples”.

`CREATE TRIGGER` requires the `TRIGGER` privilege for the table associated with the trigger. If the `DEFINER` clause is present, the privileges required depend on the *`user`* value, as discussed in Section 20.6, “Stored Object Access Control”. If binary logging is enabled, `CREATE TRIGGER` might require the `SUPER` privilege, as discussed in Section 20.7, “Stored Program Binary Logging”.

The `DEFINER` clause determines the security context to be used when checking access privileges at trigger activation time, as described later in this section.

`trigger_time` is the trigger action time. It can be `BEFORE` or `AFTER` to indicate that the trigger activates before or after each row to be modified.

Basic column value checks occur prior to trigger activation, so you cannot use `BEFORE` triggers to convert values inappropriate for the column type to valid values.

`trigger_event` indicates the kind of operation that activates the trigger. These *`trigger_event`* values are permitted:

- `INSERT`: The trigger activates whenever a new row is inserted into the table (for example, through `INSERT`, `LOAD DATA`, and `REPLACE` statements).
- `UPDATE`: The trigger activates whenever a row is modified (for example, through `UPDATE` statements).
- `DELETE`: The trigger activates whenever a row is deleted from the table (for example, through `DELETE` and `REPLACE` statements). `DROP TABLE` and `TRUNCATE TABLE` statements on the table do *not* activate this trigger, because they do not use `DELETE`. Dropping a partition does not activate `DELETE` triggers, either.

The *`trigger_event`* does not represent a literal type of SQL statement that activates the trigger so much as it represents a type of table operation. For example, an `INSERT` trigger activates not only for `INSERT` statements but also `LOAD DATA` statements because both statements insert rows into a table.

A potentially confusing example of this is the `INSERT INTO ... ON DUPLICATE KEY UPDATE ...` syntax: a `BEFORE INSERT` trigger activates for every row, followed by either an `AFTER INSERT` trigger or both the `BEFORE UPDATE` and `AFTER UPDATE` triggers, depending on whether there was a duplicate key for the row.

Note

Cascaded foreign key actions do not activate triggers.

There cannot be multiple triggers for a given table that have the same trigger event and action time. For example, you cannot have two `BEFORE UPDATE` triggers for a table. But you can have a `BEFORE UPDATE` and a `BEFORE INSERT` trigger, or a `BEFORE UPDATE` and an `AFTER UPDATE` trigger.

`trigger_body` is the statement to execute when the trigger activates. To execute multiple statements, use the `BEGIN ... END` compound statement construct. This also enables you to use the same statements that are permissible within stored routines. See Section 13.6.1, “BEGIN ... END Compound Statement”. Some statements are not permitted in triggers; see Section 20.8, “Restrictions on Stored Programs”.

Within the trigger body, you can refer to columns in the subject table (the table associated with the trigger) by using the aliases `OLD` and `NEW`. `OLD.`*`col_name`* refers to a column of an existing row before it is updated or deleted. `NEW.`*`col_name`* refers to the column of a new row to be inserted or an existing row after it is updated.

MySQL stores the `sql_mode` system variable setting in effect when a trigger is created, and always executes the trigger body with this setting in force, *regardless of the current server SQL mode when the trigger begins executing*.

The DEFINER clause specifies the MySQL account to be used when checking access privileges at trigger activation time. If the DEFINER clause is present, the *user* value should be a MySQL account specified as *'user_name'@'host_name'*, CURRENT_USER, or CURRENT_USER(). The permitted *user* values depend on the privileges you hold, as discussed in Section 20.6, “Stored Object Access Control”. Also see that section for additional information about trigger security.

If the DEFINER clause is omitted, the default definer is the user who executes the CREATE TRIGGER statement. This is the same as specifying DEFINER = CURRENT_USER explicitly.

MySQL takes the DEFINER user into account when checking trigger privileges as follows:

- At CREATE TRIGGER time, the user who issues the statement must have the TRIGGER privilege.
- At trigger activation time, privileges are checked against the DEFINER user. This user must have these privileges:
  - The TRIGGER privilege for the subject table.
  - The SELECT privilege for the subject table if references to table columns occur using OLD.*col_name* or NEW.*col_name* in the trigger body.
  - The UPDATE privilege for the subject table if table columns are targets of SET NEW.*col_name* = *value* assignments in the trigger body.
  - Whatever other privileges normally are required for the statements executed by the trigger.

Within a trigger body, the CURRENT_USER function returns the account used to check privileges at trigger activation time. This is the DEFINER user, not the user whose actions caused the trigger to be activated. For information about user auditing within triggers, see Section 6.2.15, “SQL-Based Account Activity Auditing”.

If you use LOCK TABLES to lock a table that has triggers, the tables used within the trigger are also locked, as described in LOCK TABLES and Triggers.

For additional discussion of trigger use, see Section 20.3.1, “Trigger Syntax and Examples”.

13.1.20 CREATE VIEW Statement

```
CREATE
    [OR REPLACE]
    [ALGORITHM = {UNDEFINED | MERGE | TEMPTABLE}]
    [DEFINER = user]
    [SQL SECURITY { DEFINER | INVOKER }]
    VIEW view_name [(column_list)]
    AS select_statement
    [WITH [CASCADED | LOCAL] CHECK OPTION]
```

The CREATE VIEW statement creates a new view, or replaces an existing view if the OR REPLACE clause is given. If the view does not exist, CREATE OR REPLACE VIEW is the same as CREATE VIEW. If the view does exist, CREATE OR REPLACE VIEW replaces it.

For information about restrictions on view use, see Section 20.9, “Restrictions on Views”.

The *select_statement* is a SELECT statement that provides the definition of the view. (Selecting from the view selects, in effect, using the SELECT statement.) The *select_statement* can select from base tables or other views.

The view definition is “frozen” at creation time and is not affected by subsequent changes to the definitions of the underlying tables. For example, if a view is defined as SELECT * on a table, new columns added to the table later do not become part of the view, and columns dropped from the table result in an error when selecting from the view.

The ALGORITHM clause affects how MySQL processes the view. The DEFINER and SQL SECURITY clauses specify the security context to be used when checking access privileges at view invocation time. The WITH CHECK OPTION clause can be given to constrain inserts or updates to rows in tables referenced by the view. These clauses are described later in this section.

The CREATE VIEW statement requires the CREATE VIEW privilege for the view, and some privilege for each column selected by the SELECT statement. For columns used elsewhere in the SELECT statement, you must have the SELECT privilege. If the OR REPLACE clause is present, you must also have the DROP privilege for the view. If the DEFINER clause is present, the privileges required depend on the *user* value, as discussed in Section 20.6, “Stored Object Access Control”.

When a view is referenced, privilege checking occurs as described later in this section.

A view belongs to a database. By default, a new view is created in the default database. To create the view explicitly in a given database, use *db_name.view_name* syntax to qualify the view name with the database name:

```
CREATE VIEW test.v AS SELECT * FROM t;
```

Unqualified table or view names in the SELECT statement are also interpreted with respect to the default database. A view can refer to tables or views in other databases by qualifying the table or view name with the appropriate database name.

Within a database, base tables and views share the same namespace, so a base table and a view cannot have the same name.

Columns retrieved by the SELECT statement can be simple references to table columns, or expressions that use functions, constant values, operators, and so forth.

A view must have unique column names with no duplicates, just like a base table. By default, the names of the columns retrieved by the SELECT statement are used for the view column names. To define explicit names for the view columns, specify the optional *column_list* clause as a list of comma-separated identifiers. The number of names in *column_list* must be the same as the number of columns retrieved by the SELECT statement.

A view can be created from many kinds of SELECT statements. It can refer to base tables or other views. It can use joins, UNION, and subqueries. The SELECT need not even refer to any tables:

```
CREATE VIEW v_today (today) AS SELECT CURRENT_DATE;
```

The following example defines a view that selects two columns from another table as well as an expression calculated from those columns:

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
mysql> SELECT * FROM v;
+------+-------+-------+
| qty  | price | value |
+------+-------+-------+
|    3 |    50 |   150 |
+------+-------+-------+
```

A view definition is subject to the following restrictions:

- The SELECT statement cannot contain a subquery in the FROM clause.
- The SELECT statement cannot refer to system variables or user-defined variables.
- Within a stored program, the SELECT statement cannot refer to program parameters or local variables.
- The SELECT statement cannot refer to prepared statement parameters.

- Any table or view referred to in the definition must exist. If, after the view has been created, a table or view that the definition refers to is dropped, use of the view results in an error. To check a view definition for problems of this kind, use the `CHECK TABLE` statement.
- The definition cannot refer to a `TEMPORARY` table, and you cannot create a `TEMPORARY` view.
- You cannot associate a trigger with a view.
- Aliases for column names in the `SELECT` statement are checked against the maximum column length of 64 characters (not the maximum alias length of 256 characters).

`ORDER BY` is permitted in a view definition, but it is ignored if you select from a view using a statement that has its own `ORDER BY` or filtering or grouping. When `ORDER BY` is combined with `LIMIT` or `OFFSET` in a view definition, the ordering is always enforced before the query result is used by the outer query, but it does not guarantee that the same ordering is used in the end result. As a workaround, add an `ORDER BY` clause to the outer query.

For other options or clauses in the definition, they are added to the options or clauses of the statement that references the view, but the effect is undefined. For example, if a view definition includes a `LIMIT` clause, and you select from the view using a statement that has its own `LIMIT` clause, it is undefined which limit applies. This same principle applies to options such as `ALL`, `DISTINCT`, or `SQL_SMALL_RESULT` that follow the `SELECT` keyword, and to clauses such as `INTO`, `FOR UPDATE`, `LOCK IN SHARE MODE`, and `PROCEDURE`.

The results obtained from a view may be affected if you change the query processing environment by changing system variables:

```
mysql> CREATE VIEW v (mycol) AS SELECT 'abc';
Query OK, 0 rows affected (0.01 sec)

mysql> SET sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT "mycol" FROM v;
+-------+
| mycol |
+-------+
| mycol |
+-------+
1 row in set (0.01 sec)

mysql> SET sql_mode = 'ANSI_QUOTES';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT "mycol" FROM v;
+-------+
| mycol |
+-------+
| abc   |
+-------+
1 row in set (0.00 sec)
```

The `DEFINER` and `SQL SECURITY` clauses determine which MySQL account to use when checking access privileges for the view when a statement is executed that references the view. The valid `SQL SECURITY` characteristic values are `DEFINER` (the default) and `INVOKER`. These indicate that the required privileges must be held by the user who defined or invoked the view, respectively.

If the `DEFINER` clause is present, the *`user`* value should be a MySQL account specified as *`'user_name'@'host_name'`*, `CURRENT_USER`, or `CURRENT_USER()`. The permitted *`user`* values depend on the privileges you hold, as discussed in Section 20.6, “Stored Object Access Control”. Also see that section for additional information about view security.

If the `DEFINER` clause is omitted, the default definer is the user who executes the `CREATE VIEW` statement. This is the same as specifying `DEFINER = CURRENT_USER` explicitly.

Within a view definition, the CURRENT_USER function returns the view's DEFINER value by default. For views defined with the SQL SECURITY INVOKER characteristic, CURRENT_USER returns the account for the view's invoker. For information about user auditing within views, see Section 6.2.15, “SQL-Based Account Activity Auditing”.

Within a stored routine that is defined with the SQL SECURITY DEFINER characteristic, CURRENT_USER returns the routine's DEFINER value. This also affects a view defined within such a routine, if the view definition contains a DEFINER value of CURRENT_USER.

MySQL checks view privileges like this:

- At view definition time, the view creator must have the privileges needed to use the top-level objects accessed by the view. For example, if the view definition refers to table columns, the creator must have some privilege for each column in the select list of the definition, and the SELECT privilege for each column used elsewhere in the definition. If the definition refers to a stored function, only the privileges needed to invoke the function can be checked. The privileges required at function invocation time can be checked only as it executes: For different invocations, different execution paths within the function might be taken.
- The user who references a view must have appropriate privileges to access it (SELECT to select from it, INSERT to insert into it, and so forth.)
- When a view has been referenced, privileges for objects accessed by the view are checked against the privileges held by the view DEFINER account or invoker, depending on whether the SQL SECURITY characteristic is DEFINER or INVOKER, respectively.
- If reference to a view causes execution of a stored function, privilege checking for statements executed within the function depend on whether the function SQL SECURITY characteristic is DEFINER or INVOKER. If the security characteristic is DEFINER, the function runs with the privileges of the DEFINER account. If the characteristic is INVOKER, the function runs with the privileges determined by the view's SQL SECURITY characteristic.

Example: A view might depend on a stored function, and that function might invoke other stored routines. For example, the following view invokes a stored function f():

```
CREATE VIEW v AS SELECT * FROM t WHERE t.id = f(t.name);
```

Suppose that f() contains a statement such as this:

```
IF name IS NULL then
  CALL p1();
ELSE
  CALL p2();
END IF;
```

The privileges required for executing statements within f() need to be checked when f() executes. This might mean that privileges are needed for p1() or p2(), depending on the execution path within f(). Those privileges must be checked at runtime, and the user who must possess the privileges is determined by the SQL SECURITY values of the view v and the function f().

The DEFINER and SQL SECURITY clauses for views are extensions to standard SQL. In standard SQL, views are handled using the rules for SQL SECURITY DEFINER. The standard says that the definer of the view, which is the same as the owner of the view's schema, gets applicable privileges on the view (for example, SELECT) and may grant them. MySQL has no concept of a schema “owner”, so MySQL adds a clause to identify the definer. The DEFINER clause is an extension where the intent is to have what the standard has; that is, a permanent record of who defined the view. This is why the default DEFINER value is the account of the view creator.

The optional ALGORITHM clause is a MySQL extension to standard SQL. It affects how MySQL processes the view. ALGORITHM takes three values: MERGE, TEMPTABLE, or UNDEFINED. The default algorithm is UNDEFINED if no ALGORITHM clause is present. For more information, see Section 20.5.2, “View Processing Algorithms”, as well as Section 8.2.2.4, “Optimizing Derived Tables”.

Some views are updatable. That is, you can use them in statements such as `UPDATE`, `DELETE`, or `INSERT` to update the contents of the underlying table. For a view to be updatable, there must be a one-to-one relationship between the rows in the view and the rows in the underlying table. There are also certain other constructs that make a view nonupdatable.

The `WITH CHECK OPTION` clause can be given for an updatable view to prevent inserts or updates to rows except those for which the `WHERE` clause in the *`select_statement`* is true.

In a `WITH CHECK OPTION` clause for an updatable view, the `LOCAL` and `CASCADED` keywords determine the scope of check testing when the view is defined in terms of another view. The `LOCAL` keyword restricts the `CHECK OPTION` only to the view being defined. `CASCADED` causes the checks for underlying views to be evaluated as well. When neither keyword is given, the default is `CASCADED`.

For more information about updatable views and the `WITH CHECK OPTION` clause, see Section 20.5.3, “Updatable and Insertable Views”, and Section 20.5.4, “The View WITH CHECK OPTION Clause”.

13.1.21 DROP DATABASE Statement

```
DROP {DATABASE | SCHEMA} [IF EXISTS] db_name
```

`DROP DATABASE` drops all tables in the database and deletes the database. Be *very* careful with this statement! To use `DROP DATABASE`, you need the `DROP` privilege on the database. `DROP SCHEMA` is a synonym for `DROP DATABASE`.

Important

When a database is dropped, privileges granted specifically for the database are *not* automatically dropped. They must be dropped manually. See Section 13.7.1.4, “GRANT Statement”.

`IF EXISTS` is used to prevent an error from occurring if the database does not exist.

If the default database is dropped, the default database is unset (the `DATABASE()` function returns `NULL`).

If you use `DROP DATABASE` on a symbolically linked database, both the link and the original database are deleted.

`DROP DATABASE` returns the number of tables that were removed. This corresponds to the number of `.frm` files removed.

The `DROP DATABASE` statement removes from the given database directory those files and directories that MySQL itself may create during normal operation:

- All files with the following extensions:
  - `.frm`
  - `.BAK`
  - `.DAT`
  - `.HSH`
  - `.MRG`
  - `.MYD`
  - `.MYI`
  - `.TRG`

- `.TRN`
- `.db`
- `.ibd`
- `.ndb`
- `.par`

- The `db.opt` file, if it exists.

If other files or directories remain in the database directory after MySQL removes those just listed, the database directory cannot be removed. In this case, you must remove any remaining files or directories manually and issue the `DROP DATABASE` statement again.

Dropping a database does not remove any `TEMPORARY` tables that were created in that database. `TEMPORARY` tables are automatically removed when the session that created them ends. See Section 13.1.17.2, “CREATE TEMPORARY TABLE Statement”.

You can also drop databases with `mysqladmin`. See Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”.

13.1.22 DROP EVENT Statement

```
DROP EVENT [IF EXISTS] event_name
```

This statement drops the event named *`event_name`*. The event immediately ceases being active, and is deleted completely from the server.

If the event does not exist, the error `ERROR 1517 (HY000): Unknown event 'event_name'` results. You can override this and cause the statement to generate a warning for nonexistent events instead using `IF EXISTS`.

This statement requires the `EVENT` privilege for the schema to which the event to be dropped belongs.

13.1.23 DROP FUNCTION Statement

The `DROP FUNCTION` statement is used to drop stored functions and user-defined functions (UDFs):

- For information about dropping stored functions, see Section 13.1.26, “DROP PROCEDURE and DROP FUNCTION Statements”.
- For information about dropping user-defined functions, see Section 13.7.3.2, “DROP FUNCTION Statement for User-Defined Functions”.

13.1.24 DROP INDEX Statement

```
DROP INDEX [ONLINE | OFFLINE] index_name ON tbl_name
    [algorithm_option | lock_option] ...

algorithm_option:
    ALGORITHM [=] {DEFAULT | INPLACE | COPY}

lock_option:
    LOCK [=] {DEFAULT | NONE | SHARED | EXCLUSIVE}
```

`DROP INDEX` drops the index named *`index_name`* from the table *`tbl_name`*. This statement is mapped to an `ALTER TABLE` statement to drop the index. See Section 13.1.7, “ALTER TABLE Statement”.

To drop a primary key, the index name is always PRIMARY, which must be specified as a quoted identifier because PRIMARY is a reserved word:

```
DROP INDEX `PRIMARY` ON t;
```

Indexes on variable-width columns of NDB tables are dropped online; that is, without any table copying. The table is not locked against access from other NDB Cluster API nodes, although it is locked against other operations on the *same* API node for the duration of the operation. This is done automatically by the server whenever it determines that it is possible to do so; you do not have to use any special SQL syntax or server options to cause it to happen.

In NDB Cluster, you can drop indexes offline (which causes the table to be locked for all API nodes in the cluster) using the OFFLINE keyword. The rules and limitations governing DROP OFFLINE INDEX and DROP ONLINE INDEX are the same as for ALTER OFFLINE TABLE ... DROP INDEX and ALTER ONLINE TABLE ... DROP INDEX. You cannot cause the noncopying dropping of an index that would normally be dropped offline by using the ONLINE keyword: If it is not possible to perform the DROP operation without table copying, the server ignores the ONLINE keyword. For more information, see Section 18.5.11, “Online Operations with ALTER TABLE in NDB Cluster”.

The ONLINE and OFFLINE keywords are available only in NDB Cluster; attempting to use these keywords in standard MySQL Server 5.6 releases results in a syntax error. The ONLINE and OFFLINE keywords are deprecated in MySQL NDB Cluster 7.3; they continue to be supported in MySQL NDB Cluster 7.4, but are scheduled for removal in a future NDB Cluster release.

ALGORITHM and LOCK clauses may be given to influence the table copying method and level of concurrency for reading and writing the table while its indexes are being modified. They have the same meaning as for the ALTER TABLE statement. For more information, see Section 13.1.7, “ALTER TABLE Statement”

13.1.25 DROP LOGFILE GROUP Statement

```
DROP LOGFILE GROUP logfile_group
    ENGINE [=] engine_name
```

This statement drops the log file group named *logfile_group*. The log file group must already exist or an error results. (For information on creating log file groups, see Section 13.1.14, “CREATE LOGFILE GROUP Statement”.)

Important

Before dropping a log file group, you must drop all tablespaces that use that log file group for UNDO logging.

The required ENGINE clause provides the name of the storage engine used by the log file group to be dropped. Currently, the only permitted values for *engine_name* are NDB and NDBCLUSTER.

DROP LOGFILE GROUP is useful only with Disk Data storage for NDB Cluster. See Section 18.5.10, “NDB Cluster Disk Data Tables”.

13.1.26 DROP PROCEDURE and DROP FUNCTION Statements

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name
```

These statements are used to drop a stored routine (a stored procedure or function). That is, the specified routine is removed from the server. (DROP FUNCTION is also used to drop user-defined functions; see Section 13.7.3.2, “DROP FUNCTION Statement for User-Defined Functions”.)

To drop a stored routine, you must have the ALTER ROUTINE privilege for it. (If the automatic_sp_privileges system variable is enabled, that privilege and EXECUTE are granted automatically to the routine creator when the routine is created and dropped from the creator when the routine is dropped. See Section 20.2.2, “Stored Routines and MySQL Privileges”.)

The `IF EXISTS` clause is a MySQL extension. It prevents an error from occurring if the procedure or function does not exist. A warning is produced that can be viewed with `SHOW WARNINGS`.

`DROP FUNCTION` is also used to drop user-defined functions (see Section 13.7.3.2, "DROP FUNCTION Statement for User-Defined Functions").

13.1.27 DROP SERVER Statement

```
DROP SERVER [ IF EXISTS ] server_name
```

Drops the server definition for the server named *`server_name`*. The corresponding row in the `mysql.servers` table is deleted. This statement requires the `SUPER` privilege.

Dropping a server for a table does not affect any `FEDERATED` tables that used this connection information when they were created. See Section 13.1.16, "CREATE SERVER Statement".

`DROP SERVER` causes an implicit commit. See Section 13.3.3, "Statements That Cause an Implicit Commit".

`DROP SERVER` is not written to the binary log, regardless of the logging format that is in use.

In MySQL 5.6.11 only, `gtid_next` must be set to `AUTOMATIC` before issuing this statement. (Bug #16062608, Bug #16715809, Bug #69045)

13.1.28 DROP TABLE Statement

```
DROP [TEMPORARY] TABLE [IF EXISTS]
    tbl_name [, tbl_name] ...
    [RESTRICT | CASCADE]
```

`DROP TABLE` removes one or more tables. You must have the `DROP` privilege for each table.

Be careful with this statement! For each table, it removes the table definition and all table data. If the table is partitioned, the statement removes the table definition, all its partitions, all data stored in those partitions, and all partition definitions associated with the dropped table.

Dropping a table also drops any triggers for the table.

`DROP TABLE` causes an implicit commit, except when used with the `TEMPORARY` keyword. See Section 13.3.3, "Statements That Cause an Implicit Commit".

Important

When a table is dropped, privileges granted specifically for the table are *not* automatically dropped. They must be dropped manually. See Section 13.7.1.4, "GRANT Statement".

If any tables named in the argument list do not exist, `DROP TABLE` behavior depends on whether the `IF EXISTS` clause is given:

- Without `IF EXISTS`, the statement drops all named tables that do exist, and returns an error indicating which nonexisting tables it was unable to drop.
- With `IF EXISTS`, no error occurs for nonexisting tables. The statement drops all named tables that do exist, and generates a `NOTE` diagnostic for each nonexistent table. These notes can be displayed with `SHOW WARNINGS`. See Section 13.7.5.41, "SHOW WARNINGS Statement".

`IF EXISTS` can also be useful for dropping tables in unusual circumstances under which there is an `.frm` file but no table managed by the storage engine. (For example, if an abnormal server exit occurs after removal of the table from the storage engine but before `.frm` file removal.)

The `TEMPORARY` keyword has the following effects:

- The statement drops only `TEMPORARY` tables.
- The statement does not cause an implicit commit.
- No access rights are checked. A `TEMPORARY` table is visible only with the session that created it, so no check is necessary.

Including the `TEMPORARY` keyword is a good way to prevent accidentally dropping non-`TEMPORARY` tables.

The `RESTRICT` and `CASCADE` keywords do nothing. They are permitted to make porting easier from other database systems.

`DROP TABLE` is not supported with all `innodb_force_recovery` settings. See Section 14.21.2, "Forcing InnoDB Recovery".

13.1.29 DROP TABLESPACE Statement

```
DROP TABLESPACE tablespace_name
    ENGINE [=] engine_name
```

This statement drops a tablespace that was previously created using `CREATE TABLESPACE` (see Section 13.1.18, "CREATE TABLESPACE Statement").

Important

The tablespace to be dropped must not contain any data files; in other words, before you can drop a tablespace, you must first drop each of its data files using `ALTER TABLESPACE ... DROP DATAFILE` (see Section 13.1.8, "ALTER TABLESPACE Statement").

The `ENGINE` clause (required) specifies the storage engine used by the tablespace. Currently, the only accepted values for *`engine_name`* are `NDB` and `NDBCLUSTER`.

`DROP TABLESPACE` is useful only with Disk Data storage for NDB Cluster. See Section 18.5.10, "NDB Cluster Disk Data Tables".

13.1.30 DROP TRIGGER Statement

```
DROP TRIGGER [IF EXISTS] [schema_name.]trigger_name
```

This statement drops a trigger. The schema (database) name is optional. If the schema is omitted, the trigger is dropped from the default schema. `DROP TRIGGER` requires the `TRIGGER` privilege for the table associated with the trigger.

Use `IF EXISTS` to prevent an error from occurring for a trigger that does not exist. A `NOTE` is generated for a nonexistent trigger when using `IF EXISTS`. See Section 13.7.5.41, "SHOW WARNINGS Statement".

Triggers for a table are also dropped if you drop the table.

13.1.31 DROP VIEW Statement

```
DROP VIEW [IF EXISTS]
    view_name [, view_name] ...
    [RESTRICT | CASCADE]
```

`DROP VIEW` removes one or more views. You must have the `DROP` privilege for each view.

If any views named in the argument list do not exist, the statement returns an error indicating by name which nonexisting views it was unable to drop, but also drops all views in the list that do exist.

The `IF EXISTS` clause prevents an error from occurring for views that don't exist. When this clause is given, a `NOTE` is generated for each nonexistent view. See Section 13.7.5.41, “SHOW WARNINGS Statement”.

`RESTRICT` and `CASCADE`, if given, are parsed and ignored.

13.1.32 RENAME TABLE Statement

```
RENAME TABLE
    tbl_name TO new_tbl_name
    [, tbl_name2 TO new_tbl_name2] ...
```

`RENAME TABLE` renames one or more tables. You must have `ALTER` and `DROP` privileges for the original table, and `CREATE` and `INSERT` privileges for the new table.

For example, to rename a table named `old_table` to `new_table`, use this statement:

```
RENAME TABLE old_table TO new_table;
```

That statement is equivalent to the following `ALTER TABLE` statement:

```
ALTER TABLE old_table RENAME new_table;
```

`RENAME TABLE`, unlike `ALTER TABLE`, can rename multiple tables within a single statement:

```
RENAME TABLE old_table1 TO new_table1,
             old_table2 TO new_table2,
             old_table3 TO new_table3;
```

Renaming operations are performed left to right. Thus, to swap two table names, do this (assuming that a table with the intermediary name `tmp_table` does not already exist):

```
RENAME TABLE old_table TO tmp_table,
             new_table TO old_table,
             tmp_table TO new_table;
```

Metadata locks on tables are acquired in name order, which in some cases can make a difference in operation outcome when multiple transactions execute concurrently. See Section 8.11.4, “Metadata Locking”.

To execute `RENAME TABLE`, there must be no active transactions or tables locked with `LOCK TABLES`. With the transaction table locking conditions satisfied, the rename operation is done atomically; no other session can access any of the tables while the rename is in progress.

If any errors occur during a `RENAME TABLE`, the statement fails and no changes are made.

You can use `RENAME TABLE` to move a table from one database to another:

```
RENAME TABLE current_db.tbl_name TO other_db.tbl_name;
```

Using this method to move all tables from one database to a different one in effect renames the database (an operation for which MySQL has no single statement), except that the original database continues to exist, albeit with no tables.

Like `RENAME TABLE`, `ALTER TABLE ... RENAME` can also be used to move a table to a different database. Regardless of the statement used, if the rename operation would move the table to a database located on a different file system, the success of the outcome is platform specific and depends on the underlying operating system calls used to move table files.

If a table has triggers, attempts to rename the table into a different database fail with a `Trigger in wrong schema` (`ER_TRG_IN_WRONG_SCHEMA`) error.

To rename TEMPORARY tables, RENAME TABLE does not work. Use ALTER TABLE instead.

RENAME TABLE works for views, except that views cannot be renamed into a different database.

Any privileges granted specifically for a renamed table or view are not migrated to the new name. They must be changed manually.

RENAME TABLE *tbl_name* TO *new_tbl_name* changes internally generated foreign key constraint names and user-defined foreign key constraint names that begin with the string "*tbl_name*_ibfk_" to reflect the new table name. InnoDB interprets foreign key constraint names that begin with the string "*tbl_name*_ibfk_" as internally generated names.

Foreign key constraint names that point to the renamed table are automatically updated unless there is a conflict, in which case the statement fails with an error. A conflict occurs if the renamed constraint name already exists. In such cases, you must drop and re-create the foreign keys for them to function properly.

13.1.33 TRUNCATE TABLE Statement

```
TRUNCATE [TABLE] tbl_name
```

TRUNCATE TABLE empties a table completely. It requires the DROP privilege.

Logically, TRUNCATE TABLE is similar to a DELETE statement that deletes all rows, or a sequence of DROP TABLE and CREATE TABLE statements. To achieve high performance, it bypasses the DML method of deleting data. Thus, it cannot be rolled back, it does not cause ON DELETE triggers to fire, and it cannot be performed for InnoDB tables with parent-child foreign key relationships.

Although TRUNCATE TABLE is similar to DELETE, it is classified as a DDL statement rather than a DML statement. It differs from DELETE in the following ways:

- Truncate operations drop and re-create the table, which is much faster than deleting rows one by one, particularly for large tables.
- Truncate operations cause an implicit commit, and so cannot be rolled back. See Section 13.3.3, "Statements That Cause an Implicit Commit".
- Truncation operations cannot be performed if the session holds an active table lock.
- TRUNCATE TABLE fails for an InnoDB table or NDB table if there are any FOREIGN KEY constraints from other tables that reference the table. Foreign key constraints between columns of the same table are permitted.
- Truncation operations do not return a meaningful value for the number of deleted rows. The usual result is "0 rows affected," which should be interpreted as "no information."
- As long as the table format file *tbl_name*.frm is valid, the table can be re-created as an empty table with TRUNCATE TABLE, even if the data or index files have become corrupted.
- Any AUTO_INCREMENT value is reset to its start value. This is true even for MyISAM and InnoDB, which normally do not reuse sequence values.
- When used with partitioned tables, TRUNCATE TABLE preserves the partitioning; that is, the data and index files are dropped and re-created, while the partition definitions (.par) file is unaffected.
- The TRUNCATE TABLE statement does not invoke ON DELETE triggers.

TRUNCATE TABLE for a table closes all handlers for the table that were opened with HANDLER OPEN.

TRUNCATE TABLE is treated for purposes of binary logging and replication as DROP TABLE followed by CREATE TABLE—that is, as DDL rather than DML. This is due to the fact that, when using

`InnoDB` and other transactional storage engines where the transaction isolation level does not permit statement-based logging (`READ COMMITTED` or `READ UNCOMMITTED`), the statement was not logged and replicated when using `STATEMENT` or `MIXED` logging mode. (Bug #36763) However, it is still applied on replicas using `InnoDB` in the manner described previously.

On a system with a large `InnoDB` buffer pool and `innodb_adaptive_hash_index` enabled, `TRUNCATE TABLE` operations may cause a temporary drop in system performance due to an LRU scan that occurs when removing an `InnoDB` table's adaptive hash index entries. The problem was addressed for `DROP TABLE` in MySQL 5.5.23 (Bug #13704145, Bug #64284) but remains a known issue for `TRUNCATE TABLE` (Bug #68184).

`TRUNCATE TABLE` can be used with Performance Schema summary tables, but the effect is to reset the summary columns to 0 or `NULL`, not to remove rows. See Section 22.12.9, “Performance Schema Summary Tables”.

13.2 Data Manipulation Statements

13.2.1 CALL Statement

```
CALL sp_name([parameter[,...]])
CALL sp_name[()]
```

The `CALL` statement invokes a stored procedure that was defined previously with `CREATE PROCEDURE`.

Stored procedures that take no arguments can be invoked without parentheses. That is, `CALL p()` and `CALL p` are equivalent.

`CALL` can pass back values to its caller using parameters that are declared as `OUT` or `INOUT` parameters. When the procedure returns, a client program can also obtain the number of rows affected for the final statement executed within the routine: At the SQL level, call the `ROW_COUNT()` function; from the C API, call the `mysql_affected_rows()` function.

For information about the effect of unhandled conditions on procedure parameters, see Section 13.6.7.8, “Condition Handling and OUT or INOUT Parameters”.

To get back a value from a procedure using an `OUT` or `INOUT` parameter, pass the parameter by means of a user variable, and then check the value of the variable after the procedure returns. (If you are calling the procedure from within another stored procedure or function, you can also pass a routine parameter or local routine variable as an `IN` or `INOUT` parameter.) For an `INOUT` parameter, initialize its value before passing it to the procedure. The following procedure has an `OUT` parameter that the procedure sets to the current server version, and an `INOUT` value that the procedure increments by one from its current value:

```
CREATE PROCEDURE p (OUT ver_param VARCHAR(25), INOUT incr_param INT)
BEGIN
  # Set value of OUT parameter
  SELECT VERSION() INTO ver_param;
  # Increment value of INOUT parameter
  SET incr_param = incr_param + 1;
END;
```

Before calling the procedure, initialize the variable to be passed as the `INOUT` parameter. After calling the procedure, the values of the two variables have been set or modified:

```
mysql> SET @increment = 10;
mysql> CALL p(@version, @increment);
mysql> SELECT @version, @increment;
+------------------+------------+
| @version         | @increment |
+------------------+------------+
| 5.6.38-debug-log |         11 |
```

```
+------------------+------------+
```

In prepared `CALL` statements used with `PREPARE` and `EXECUTE`, placeholders can be used for `IN` parameters, `OUT`, and `INOUT` parameters. These types of parameters can be used as follows:

```
mysql> SET @increment = 10;
mysql> PREPARE s FROM 'CALL p(?, ?)';
mysql> EXECUTE s USING @version, @increment;
mysql> SELECT @version, @increment;
+------------------+------------+
| @version         | @increment |
+------------------+------------+
| 5.6.38-debug-log |         11 |
+------------------+------------+
```

To write C programs that use the `CALL` SQL statement to execute stored procedures that produce result sets, the `CLIENT_MULTI_RESULTS` flag must be enabled. This is because each `CALL` returns a result to indicate the call status, in addition to any result sets that might be returned by statements executed within the procedure. `CLIENT_MULTI_RESULTS` must also be enabled if `CALL` is used to execute any stored procedure that contains prepared statements. It cannot be determined when such a procedure is loaded whether those statements produce result sets, so it is necessary to assume that they do so.

`CLIENT_MULTI_RESULTS` can be enabled when you call `mysql_real_connect()`, either explicitly by passing the `CLIENT_MULTI_RESULTS` flag itself, or implicitly by passing `CLIENT_MULTI_STATEMENTS` (which also enables `CLIENT_MULTI_RESULTS`). `CLIENT_MULTI_RESULTS` is enabled by default.

To process the result of a `CALL` statement executed using `mysql_query()` or `mysql_real_query()`, use a loop that calls `mysql_next_result()` to determine whether there are more results. For an example, see C API Multiple Statement Execution Support.

C programs can use the prepared-statement interface to execute `CALL` statements and access `OUT` and `INOUT` parameters. This is done by processing the result of a `CALL` statement using a loop that calls `mysql_stmt_next_result()` to determine whether there are more results. For an example, see C API Prepared CALL Statement Support. Languages that provide a MySQL interface can use prepared `CALL` statements to directly retrieve `OUT` and `INOUT` procedure parameters.

Metadata changes to objects referred to by stored programs are detected and cause automatic reparsing of the affected statements when the program is next executed. For more information, see Section 8.10.4, “Caching of Prepared Statements and Stored Programs”.

13.2.2 DELETE Statement

`DELETE` is a DML statement that removes rows from a table.

Single-Table Syntax

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]
```

The `DELETE` statement deletes rows from *`tbl_name`* and returns the number of deleted rows. To check the number of deleted rows, call the `ROW_COUNT()` function described in Section 12.16, “Information Functions”.

Main Clauses

The conditions in the optional `WHERE` clause identify which rows to delete. With no `WHERE` clause, all rows are deleted.

where_condition is an expression that evaluates to true for each row to be deleted. It is specified as described in Section 13.2.9, “SELECT Statement”.

If the ORDER BY clause is specified, the rows are deleted in the order that is specified. The LIMIT clause places a limit on the number of rows that can be deleted. These clauses apply to single-table deletes, but not multi-table deletes.

Multiple-Table Syntax

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    tbl_name[.*] [, tbl_name[.*]] ...
    FROM table_references
    [WHERE where_condition]

DELETE [LOW_PRIORITY] [QUICK] [IGNORE]
    FROM tbl_name[.*] [, tbl_name[.*]] ...
    USING table_references
    [WHERE where_condition]
```

Privileges

You need the DELETE privilege on a table to delete rows from it. You need only the SELECT privilege for any columns that are only read, such as those named in the WHERE clause.

Performance

When you do not need to know the number of deleted rows, the TRUNCATE TABLE statement is a faster way to empty a table than a DELETE statement with no WHERE clause. Unlike DELETE, TRUNCATE TABLE cannot be used within a transaction or if you have a lock on the table. See Section 13.1.33, “TRUNCATE TABLE Statement” and Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”.

The speed of delete operations may also be affected by factors discussed in Section 8.2.4.3, “Optimizing DELETE Statements”.

To ensure that a given DELETE statement does not take too much time, the MySQL-specific LIMIT *row_count* clause for DELETE specifies the maximum number of rows to be deleted. If the number of rows to delete is larger than the limit, repeat the DELETE statement until the number of affected rows is less than the LIMIT value.

Subqueries

You cannot delete from a table and select from the same table in a subquery.

Partitioned Table Support

DELETE supports explicit partition selection using the PARTITION option, which takes a list of the comma-separated names of one or more partitions or subpartitions (or both) from which to select rows to be dropped. Partitions not included in the list are ignored. Given a partitioned table t with a partition named p0, executing the statement DELETE FROM t PARTITION (p0) has the same effect on the table as executing ALTER TABLE t TRUNCATE PARTITION (p0); in both cases, all rows in partition p0 are dropped.

PARTITION can be used along with a WHERE condition, in which case the condition is tested only on rows in the listed partitions. For example, DELETE FROM t PARTITION (p0) WHERE c < 5 deletes rows only from partition p0 for which the condition c < 5 is true; rows in any other partitions are not checked and thus not affected by the DELETE.

The PARTITION option can also be used in multiple-table DELETE statements. You can use up to one such option per table named in the FROM option.

For more information and examples, see Section 19.5, “Partition Selection”.

Auto-Increment Columns

If you delete the row containing the maximum value for an `AUTO_INCREMENT` column, the value is not reused for a `MyISAM` or `InnoDB` table. If you delete all rows in the table with `DELETE FROM tbl_name` (without a `WHERE` clause) in `autocommit` mode, the sequence starts over for all storage engines except `InnoDB` and `MyISAM`. There are some exceptions to this behavior for `InnoDB` tables, as discussed in Section 14.6.1.6, “AUTO_INCREMENT Handling in InnoDB”.

For `MyISAM` tables, you can specify an `AUTO_INCREMENT` secondary column in a multiple-column key. In this case, reuse of values deleted from the top of the sequence occurs even for `MyISAM` tables. See Section 3.6.9, “Using AUTO_INCREMENT”.

Modifiers

The `DELETE` statement supports the following modifiers:

- If you specify the `LOW_PRIORITY` modifier, the server delays execution of the `DELETE` until no other clients are reading from the table. This affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).
- For `MyISAM` tables, if you use the `QUICK` modifier, the storage engine does not merge index leaves during delete, which may speed up some kinds of delete operations.
- The `IGNORE` modifier causes MySQL to ignore errors during the process of deleting rows. (Errors encountered during the parsing stage are processed in the usual manner.) Errors that are ignored due to the use of `IGNORE` are returned as warnings.

Order of Deletion

If the `DELETE` statement includes an `ORDER BY` clause, rows are deleted in the order specified by the clause. This is useful primarily in conjunction with `LIMIT`. For example, the following statement finds rows matching the `WHERE` clause, sorts them by `timestamp_column`, and deletes the first (oldest) one:

```
DELETE FROM somelog WHERE user = 'jcole'
ORDER BY timestamp_column LIMIT 1;
```

`ORDER BY` also helps to delete rows in an order required to avoid referential integrity violations.

InnoDB Tables

If you are deleting many rows from a large table, you may exceed the lock table size for an `InnoDB` table. To avoid this problem, or simply to minimize the time that the table remains locked, the following strategy (which does not use `DELETE` at all) might be helpful:

1. Select the rows *not* to be deleted into an empty table that has the same structure as the original table:

   ```
   INSERT INTO t_copy SELECT * FROM t WHERE ... ;
   ```

2. Use `RENAME TABLE` to atomically move the original table out of the way and rename the copy to the original name:

   ```
   RENAME TABLE t TO t_old, t_copy TO t;
   ```

3. Drop the original table:

   ```
   DROP TABLE t_old;
   ```

No other sessions can access the tables involved while `RENAME TABLE` executes, so the rename operation is not subject to concurrency problems. See Section 13.1.32, “RENAME TABLE Statement”.

MyISAM Tables

In MyISAM tables, deleted rows are maintained in a linked list and subsequent INSERT operations reuse old row positions. To reclaim unused space and reduce file sizes, use the OPTIMIZE TABLE statement or the myisamchk utility to reorganize tables. OPTIMIZE TABLE is easier to use, but myisamchk is faster. See Section 13.7.2.4, "OPTIMIZE TABLE Statement", and Section 4.6.3, "myisamchk — MyISAM Table-Maintenance Utility".

The QUICK modifier affects whether index leaves are merged for delete operations. DELETE QUICK is most useful for applications where index values for deleted rows are replaced by similar index values from rows inserted later. In this case, the holes left by deleted values are reused.

DELETE QUICK is not useful when deleted values lead to underfilled index blocks spanning a range of index values for which new inserts occur again. In this case, use of QUICK can lead to wasted space in the index that remains unreclaimed. Here is an example of such a scenario:

1. Create a table that contains an indexed AUTO_INCREMENT column.
2. Insert many rows into the table. Each insert results in an index value that is added to the high end of the index.
3. Delete a block of rows at the low end of the column range using DELETE QUICK.

In this scenario, the index blocks associated with the deleted index values become underfilled but are not merged with other index blocks due to the use of QUICK. They remain underfilled when new inserts occur, because new rows do not have index values in the deleted range. Furthermore, they remain underfilled even if you later use DELETE without QUICK, unless some of the deleted index values happen to lie in index blocks within or adjacent to the underfilled blocks. To reclaim unused index space under these circumstances, use OPTIMIZE TABLE.

If you are going to delete many rows from a table, it might be faster to use DELETE QUICK followed by OPTIMIZE TABLE. This rebuilds the index rather than performing many index block merge operations.

Multi-Table Deletes

You can specify multiple tables in a DELETE statement to delete rows from one or more tables depending on the condition in the WHERE clause. You cannot use ORDER BY or LIMIT in a multiple-table DELETE. The *table_references* clause lists the tables involved in the join, as described in Section 13.2.9.2, "JOIN Clause".

For the first multiple-table syntax, only matching rows from the tables listed before the FROM clause are deleted. For the second multiple-table syntax, only matching rows from the tables listed in the FROM clause (before the USING clause) are deleted. The effect is that you can delete rows from many tables at the same time and have additional tables that are used only for searching:

```
DELETE t1, t2 FROM t1 INNER JOIN t2 INNER JOIN t3
WHERE t1.id=t2.id AND t2.id=t3.id;
```

Or:

```
DELETE FROM t1, t2 USING t1 INNER JOIN t2 INNER JOIN t3
WHERE t1.id=t2.id AND t2.id=t3.id;
```

These statements use all three tables when searching for rows to delete, but delete matching rows only from tables t1 and t2.

The preceding examples use INNER JOIN, but multiple-table DELETE statements can use other types of join permitted in SELECT statements, such as LEFT JOIN. For example, to delete rows that exist in t1 that have no match in t2, use a LEFT JOIN:

```
DELETE t1 FROM t1 LEFT JOIN t2 ON t1.id=t2.id WHERE t2.id IS NULL;
```

The syntax permits `.*` after each *`tbl_name`* for compatibility with `Access`.

If you use a multiple-table `DELETE` statement involving `InnoDB` tables for which there are foreign key constraints, the MySQL optimizer might process tables in an order that differs from that of their parent/child relationship. In this case, the statement fails and rolls back. Instead, you should delete from a single table and rely on the `ON DELETE` capabilities that `InnoDB` provides to cause the other tables to be modified accordingly.

Note

If you declare an alias for a table, you must use the alias when referring to the table:

```
DELETE t1 FROM test AS t1, test2 WHERE ...
```

Table aliases in a multiple-table `DELETE` should be declared only in the *`table_references`* part of the statement. Elsewhere, alias references are permitted but not alias declarations.

Correct:

```
DELETE a1, a2 FROM t1 AS a1 INNER JOIN t2 AS a2
WHERE a1.id=a2.id;

DELETE FROM a1, a2 USING t1 AS a1 INNER JOIN t2 AS a2
WHERE a1.id=a2.id;
```

Incorrect:

```
DELETE t1 AS a1, t2 AS a2 FROM t1 INNER JOIN t2
WHERE a1.id=a2.id;

DELETE FROM t1 AS a1, t2 AS a2 USING t1 INNER JOIN t2
WHERE a1.id=a2.id;
```

13.2.3 DO Statement

```
DO expr [, expr] ...
```

`DO` executes the expressions but does not return any results. In most respects, `DO` is shorthand for `SELECT expr, ...`, but has the advantage that it is slightly faster when you do not care about the result.

`DO` is useful primarily with functions that have side effects, such as `RELEASE_LOCK()`.

Example: This `SELECT` statement pauses, but also produces a result set:

```
mysql> SELECT SLEEP(5);
+----------+
| SLEEP(5) |
+----------+
|        0 |
+----------+
1 row in set (5.02 sec)
```

`DO`, on the other hand, pauses without producing a result set.:

```
mysql> DO SLEEP(5);
Query OK, 0 rows affected (4.99 sec)
```

This could be useful, for example in a stored function or trigger, which prohibit statements that produce result sets.

`DO` only executes expressions. It cannot be used in all cases where `SELECT` can be used. For example, `DO id FROM t1` is invalid because it references a table.

13.2.4 HANDLER Statement

```
HANDLER tbl_name OPEN [ [AS] alias]

HANDLER tbl_name READ index_name { = | <= | >= | < | > } (value1,value2,...)
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ index_name { FIRST | NEXT | PREV | LAST }
    [ WHERE where_condition ] [LIMIT ... ]
HANDLER tbl_name READ { FIRST | NEXT }
    [ WHERE where_condition ] [LIMIT ... ]

HANDLER tbl_name CLOSE
```

The `HANDLER` statement provides direct access to table storage engine interfaces. It is available for `InnoDB` and `MyISAM` tables.

The `HANDLER ... OPEN` statement opens a table, making it accessible using subsequent `HANDLER ... READ` statements. This table object is not shared by other sessions and is not closed until the session calls `HANDLER ... CLOSE` or the session terminates.

If you open the table using an alias, further references to the open table with other `HANDLER` statements must use the alias rather than the table name. If you do not use an alias, but open the table using a table name qualified by the database name, further references must use the unqualified table name. For example, for a table opened using `mydb.mytable`, further references must use `mytable`.

The first `HANDLER ... READ` syntax fetches a row where the index specified satisfies the given values and the `WHERE` condition is met. If you have a multiple-column index, specify the index column values as a comma-separated list. Either specify values for all the columns in the index, or specify values for a leftmost prefix of the index columns. Suppose that an index `my_idx` includes three columns named `col_a`, `col_b`, and `col_c`, in that order. The `HANDLER` statement can specify values for all three columns in the index, or for the columns in a leftmost prefix. For example:

```
HANDLER ... READ my_idx = (col_a_val,col_b_val,col_c_val) ...
HANDLER ... READ my_idx = (col_a_val,col_b_val) ...
HANDLER ... READ my_idx = (col_a_val) ...
```

To employ the `HANDLER` interface to refer to a table's `PRIMARY KEY`, use the quoted identifier `` `PRIMARY` ``:

```
HANDLER tbl_name READ `PRIMARY` ...
```

The second `HANDLER ... READ` syntax fetches a row from the table in index order that matches the `WHERE` condition.

The third `HANDLER ... READ` syntax fetches a row from the table in natural row order that matches the `WHERE` condition. It is faster than `HANDLER tbl_name READ index_name` when a full table scan is desired. Natural row order is the order in which rows are stored in a `MyISAM` table data file. This statement works for `InnoDB` tables as well, but there is no such concept because there is no separate data file.

Without a `LIMIT` clause, all forms of `HANDLER ... READ` fetch a single row if one is available. To return a specific number of rows, include a `LIMIT` clause. It has the same syntax as for the `SELECT` statement. See Section 13.2.9, “SELECT Statement”.

`HANDLER ... CLOSE` closes a table that was opened with `HANDLER ... OPEN`.

There are several reasons to use the `HANDLER` interface instead of normal `SELECT` statements:

- `HANDLER` is faster than `SELECT`:
  - A designated storage engine handler object is allocated for the `HANDLER ... OPEN`. The object is reused for subsequent `HANDLER` statements for that table; it need not be reinitialized for each one.

  - There is less parsing involved.
  - There is no optimizer or query-checking overhead.
  - The handler interface does not have to provide a consistent look of the data (for example, dirty reads are permitted), so the storage engine can use optimizations that `SELECT` does not normally permit.

- `HANDLER` makes it easier to port to MySQL applications that use a low-level `ISAM`-like interface. (See Section 14.20, “InnoDB memcached Plugin” for an alternative way to adapt applications that use the key-value store paradigm.)
- `HANDLER` enables you to traverse a database in a manner that is difficult (or even impossible) to accomplish with `SELECT`. The `HANDLER` interface is a more natural way to look at data when working with applications that provide an interactive user interface to the database.

`HANDLER` is a somewhat low-level statement. For example, it does not provide consistency. That is, `HANDLER ... OPEN` does *not* take a snapshot of the table, and does *not* lock the table. This means that after a `HANDLER ... OPEN` statement is issued, table data can be modified (by the current session or other sessions) and these modifications might be only partially visible to `HANDLER ... NEXT` or `HANDLER ... PREV` scans.

An open handler can be closed and marked for reopen, in which case the handler loses its position in the table. This occurs when both of the following circumstances are true:

- Any session executes `FLUSH TABLES` or DDL statements on the handler's table.
- The session in which the handler is open executes non-`HANDLER` statements that use tables.

`TRUNCATE TABLE` for a table closes all handlers for the table that were opened with `HANDLER OPEN`.

If a table is flushed with `FLUSH TABLES tbl_name WITH READ LOCK` was opened with `HANDLER`, the handler is implicitly flushed and loses its position.

`HANDLER` is not supported with partitioned tables.

13.2.5 INSERT Statement

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [(col_name [, col_name] ...)]
    {VALUES | VALUE} (value_list) [, (value_list)] ...
    [ON DUPLICATE KEY UPDATE assignment_list]

INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    SET assignment_list
    [ON DUPLICATE KEY UPDATE assignment_list]

INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [(col_name [, col_name] ...)]
    SELECT ...
    [ON DUPLICATE KEY UPDATE assignment_list]

value:
    {expr | DEFAULT}

value_list:
    value [, value] ...
```

```
assignment:
    col_name = value

assignment_list:
    assignment [, assignment] ...
```

`INSERT` inserts new rows into an existing table. The `INSERT ... VALUES` and `INSERT ... SET` forms of the statement insert rows based on explicitly specified values. The `INSERT ... SELECT` form inserts rows selected from another table or tables. `INSERT` with an `ON DUPLICATE KEY UPDATE` clause enables existing rows to be updated if a row to be inserted would cause a duplicate value in a `UNIQUE` index or `PRIMARY KEY`.

For additional information about `INSERT ... SELECT` and `INSERT ... ON DUPLICATE KEY UPDATE`, see Section 13.2.5.1, “INSERT ... SELECT Statement”, and Section 13.2.5.2, “INSERT ... ON DUPLICATE KEY UPDATE Statement”.

Inserting into a table requires the `INSERT` privilege for the table. If the `ON DUPLICATE KEY UPDATE` clause is used and a duplicate key causes an `UPDATE` to be performed instead, the statement requires the `UPDATE` privilege for the columns to be updated. For columns that are read but not modified you need only the `SELECT` privilege (such as for a column referenced only on the right hand side of an *`col_name`*=*`expr`* assignment in an `ON DUPLICATE KEY UPDATE` clause).

When inserting into a partitioned table, you can control which partitions and subpartitions accept new rows. The `PARTITION` option takes a list of the comma-separated names of one or more partitions or subpartitions (or both) of the table. If any of the rows to be inserted by a given `INSERT` statement do not match one of the partitions listed, the `INSERT` statement fails with the error `Found a row not matching the given partition set`. For more information and examples, see Section 19.5, “Partition Selection”.

`tbl_name` is the table into which rows should be inserted. Specify the columns for which the statement provides values as follows:

- Provide a parenthesized list of comma-separated column names following the table name. In this case, a value for each named column must be provided by the `VALUES` list or the `SELECT` statement.
- If you do not specify a list of column names for `INSERT ... VALUES` or `INSERT ... SELECT`, values for every column in the table must be provided by the `VALUES` list or the `SELECT` statement. If you do not know the order of the columns in the table, use `DESCRIBE` *`tbl_name`* to find out.
- A `SET` clause indicates columns explicitly by name, together with the value to assign each one.

Column values can be given in several ways:

- If strict SQL mode is not enabled, any column not explicitly given a value is set to its default (explicit or implicit) value. For example, if you specify a column list that does not name all the columns in the table, unnamed columns are set to their default values. Default value assignment is described in Section 11.5, “Data Type Default Values”. See also Section 1.7.3.3, “Constraints on Invalid Data”.

  If strict SQL mode is enabled, an `INSERT` statement generates an error if it does not specify an explicit value for every column that has no default value. See Section 5.1.10, “Server SQL Modes”.

- If both the column list and the `VALUES` list are empty, `INSERT` creates a row with each column set to its default value:

  ```
  INSERT INTO tbl_name () VALUES();
  ```

  If strict mode is not enabled, MySQL uses the implicit default value for any column that has no explicitly defined default. If strict mode is enabled, an error occurs if any column has no default value.

- Use the keyword `DEFAULT` to set a column explicitly to its default value. This makes it easier to write `INSERT` statements that assign values to all but a few columns, because it enables you to

avoid writing an incomplete VALUES list that does not include a value for each column in the table. Otherwise, you must provide the list of column names corresponding to each value in the VALUES list.

- In expressions, you can use DEFAULT(*col_name*) to produce the default value for column *col_name*.
- Type conversion of an expression *expr* that provides a column value might occur if the expression data type does not match the column data type. Conversion of a given value can result in different inserted values depending on the column type. For example, inserting the string '1999.0e-2' into an INT, FLOAT, DECIMAL(10,6), or YEAR column inserts the value 1999, 19.9921, 19.992100, or 1999, respectively. The value stored in the INT and YEAR columns is 1999 because the string-to-number conversion looks only at as much of the initial part of the string as may be considered a valid integer or year. For the FLOAT and DECIMAL columns, the string-to-number conversion considers the entire string a valid numeric value.
- An expression *expr* can refer to any column that was set earlier in a value list. For example, you can do this because the value for col2 refers to col1, which has previously been assigned:

```
INSERT INTO tbl_name (col1,col2) VALUES(15,col1*2);
```

But the following is not legal, because the value for col1 refers to col2, which is assigned after col1:

```
INSERT INTO tbl_name (col1,col2) VALUES(col2*2,15);
```

An exception occurs for columns that contain AUTO_INCREMENT values. Because AUTO_INCREMENT values are generated after other value assignments, any reference to an AUTO_INCREMENT column in the assignment returns a 0.

INSERT statements that use VALUES syntax can insert multiple rows. To do this, include multiple lists of comma-separated column values, with lists enclosed within parentheses and separated by commas. Example:

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3),(4,5,6),(7,8,9);
```

Each values list must contain exactly as many values as are to be inserted per row. The following statement is invalid because it contains one list of nine values, rather than three lists of three values each:

```
INSERT INTO tbl_name (a,b,c) VALUES(1,2,3,4,5,6,7,8,9);
```

VALUE is a synonym for VALUES in this context. Neither implies anything about the number of values lists, nor about the number of values per list. Either may be used whether there is a single values list or multiple lists, and regardless of the number of values per list.

The affected-rows value for an INSERT can be obtained using the ROW_COUNT() SQL function or the mysql_affected_rows() C API function. See Section 12.16, “Information Functions”, and mysql_affected_rows().

If you use an INSERT ... VALUES statement with multiple value lists or INSERT ... SELECT, the statement returns an information string in this format:

```
Records: N1 Duplicates: N2 Warnings: N3
```

If you are using the C API, the information string can be obtained by invoking the mysql_info() function. See mysql_info().

Records indicates the number of rows processed by the statement. (This is not necessarily the number of rows actually inserted because Duplicates can be nonzero.) Duplicates indicates the

number of rows that could not be inserted because they would duplicate some existing unique index value. `Warnings` indicates the number of attempts to insert column values that were problematic in some way. Warnings can occur under any of the following conditions:

- Inserting `NULL` into a column that has been declared `NOT NULL`. For multiple-row `INSERT` statements or `INSERT INTO ... SELECT` statements, the column is set to the implicit default value for the column data type. This is `0` for numeric types, the empty string (`''`) for string types, and the "zero" value for date and time types. `INSERT INTO ... SELECT` statements are handled the same way as multiple-row inserts because the server does not examine the result set from the `SELECT` to see whether it returns a single row. (For a single-row `INSERT`, no warning occurs when `NULL` is inserted into a `NOT NULL` column. Instead, the statement fails with an error.)

- Setting a numeric column to a value that lies outside the column's range. The value is clipped to the closest endpoint of the range.

- Assigning a value such as `'10.34 a'` to a numeric column. The trailing nonnumeric text is stripped off and the remaining numeric part is inserted. If the string value has no leading numeric part, the column is set to `0`.

- Inserting a string into a string column (`CHAR`, `VARCHAR`, `TEXT`, or `BLOB`) that exceeds the column's maximum length. The value is truncated to the column's maximum length.

- Inserting a value into a date or time column that is illegal for the data type. The column is set to the appropriate zero value for the type.

- For `INSERT` examples involving `AUTO_INCREMENT` column values, see Section 3.6.9, "Using AUTO_INCREMENT".

  If `INSERT` inserts a row into a table that has an `AUTO_INCREMENT` column, you can find the value used for that column by using the `LAST_INSERT_ID()` SQL function or the `mysql_insert_id()` C API function.

Note

These two functions do not always behave identically. The behavior of `INSERT` statements with respect to `AUTO_INCREMENT` columns is discussed further in Section 12.16, "Information Functions", and mysql_insert_id().

The `INSERT` statement supports the following modifiers:

- If you use the `DELAYED` modifier, the server puts the row or rows to be inserted into a buffer, and the client issuing the `INSERT DELAYED` statement can then continue immediately. If the table is in use, the server holds the rows. When the table is free, the server begins inserting rows, checking periodically to see whether there are any new read requests for the table. If there are, the delayed row queue is suspended until the table becomes free again. See Section 13.2.5.3, "INSERT DELAYED Statement".

  `DELAYED` is ignored with `INSERT ... SELECT` or `INSERT ... ON DUPLICATE KEY UPDATE`.

  `DELAYED` is also disregarded for an `INSERT` that uses functions accessing tables or triggers, or that is called from a function or a trigger.

Note

As of MySQL 5.6.6, `INSERT DELAYED` is deprecated; expect it to be removed in a future release. Use `INSERT` (without `DELAYED`) instead.

- If you use the `LOW_PRIORITY` modifier, execution of the `INSERT` is delayed until no other clients are reading from the table. This includes other clients that began reading while existing clients are reading, and while the `INSERT LOW_PRIORITY` statement is waiting. It is possible, therefore, for a client that issues an `INSERT LOW_PRIORITY` statement to wait for a very long time (or even

forever) in a read-heavy environment. (This is in contrast to `INSERT DELAYED`, which lets the client continue at once.)

`LOW_PRIORITY` affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).

Note

`LOW_PRIORITY` should normally not be used with `MyISAM` tables because doing so disables concurrent inserts. See Section 8.11.3, “Concurrent Inserts”.

- If you specify `HIGH_PRIORITY`, it overrides the effect of the `--low-priority-updates` option if the server was started with that option. It also causes concurrent inserts not to be used. See Section 8.11.3, “Concurrent Inserts”.

  `HIGH_PRIORITY` affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).

- If you use the `IGNORE` modifier, errors that occur while executing the `INSERT` statement are ignored. For example, without `IGNORE`, a row that duplicates an existing `UNIQUE` index or `PRIMARY KEY` value in the table causes a duplicate-key error and the statement is aborted. With `IGNORE`, the row is discarded and no error occurs. Ignored errors may generate warnings instead, although duplicate-key errors do not.

  `IGNORE` has a similar effect on inserts into partitioned tables where no partition matching a given value is found. Without `IGNORE`, such `INSERT` statements are aborted with an error. When `INSERT IGNORE` is used, the insert operation fails silently for rows containing the unmatched value, but inserts rows that are matched. For an example, see Section 19.2.2, “LIST Partitioning”.

  Data conversions that would trigger errors abort the statement if `IGNORE` is not specified. With `IGNORE`, invalid values are adjusted to the closest values and inserted; warnings are produced but the statement does not abort. You can determine with the `mysql_info()` C API function how many rows were actually inserted into the table.

  You can use `REPLACE` instead of `INSERT` to overwrite old rows. `REPLACE` is the counterpart to `INSERT IGNORE` in the treatment of new rows that contain unique key values that duplicate old rows: The new rows replace the old rows rather than being discarded. See Section 13.2.8, “REPLACE Statement”.

- If you specify `ON DUPLICATE KEY UPDATE`, and a row is inserted that would cause a duplicate value in a `UNIQUE` index or `PRIMARY KEY`, an `UPDATE` of the old row occurs. The affected-rows value per row is 1 if the row is inserted as a new row, 2 if an existing row is updated, and 0 if an existing row is set to its current values. If you specify the `CLIENT_FOUND_ROWS` flag to the `mysql_real_connect()` C API function when connecting to `mysqld`, the affected-rows value is 1 (not 0) if an existing row is set to its current values. See Section 13.2.5.2, “INSERT ... ON DUPLICATE KEY UPDATE Statement”.

An `INSERT` statement affecting a partitioned table using a storage engine such as `MyISAM` that employs table-level locks locks only those partitions into which rows are actually inserted. (For storage engines such as `InnoDB` that employ row-level locking, no locking of partitions takes place.) For more information, see Section 19.6.4, “Partitioning and Locking”.

13.2.5.1 INSERT ... SELECT Statement

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [(col_name [, col_name] ...)]
    SELECT ...
    [ON DUPLICATE KEY UPDATE assignment_list]
```

```
value:
    {expr | DEFAULT}

assignment:
    col_name = value

assignment_list:
    assignment [, assignment] ...
```

With `INSERT ... SELECT`, you can quickly insert many rows into a table from the result of a `SELECT` statement, which can select from one or many tables. For example:

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

The following conditions hold for `INSERT ... SELECT` statements:

- Specify `IGNORE` to ignore rows that would cause duplicate-key violations.
- `DELAYED` is ignored with `INSERT ... SELECT`.
- The target table of the `INSERT` statement may appear in the `FROM` clause of the `SELECT` part of the query. However, you cannot insert into a table and select from the same table in a subquery.

  When selecting from and inserting into the same table, MySQL creates an internal temporary table to hold the rows from the `SELECT` and then inserts those rows into the target table. However, you cannot use `INSERT INTO t ... SELECT ... FROM t` when `t` is a `TEMPORARY` table, because `TEMPORARY` tables cannot be referred to twice in the same statement. See Section 8.4.4, “Internal Temporary Table Use in MySQL”, and Section B.3.6.2, “TEMPORARY Table Problems”.
- `AUTO_INCREMENT` columns work as usual.
- To ensure that the binary log can be used to re-create the original tables, MySQL does not permit concurrent inserts for `INSERT ... SELECT` statements (see Section 8.11.3, “Concurrent Inserts”).
- To avoid ambiguous column reference problems when the `SELECT` and the `INSERT` refer to the same table, provide a unique alias for each table used in the `SELECT` part, and qualify column names in that part with the appropriate alias.

You can explicitly select which partitions or subpartitions (or both) of the source or target table (or both) are to be used with a `PARTITION` option following the name of the table. When `PARTITION` is used with the name of the source table in the `SELECT` portion of the statement, rows are selected only from the partitions or subpartitions named in its partition list. When `PARTITION` is used with the name of the target table for the `INSERT` portion of the statement, it must be possible to insert all rows selected into the partitions or subpartitions named in the partition list following the option. Otherwise, the `INSERT ... SELECT` statement fails. For more information and examples, see Section 19.5, “Partition Selection”.

For `INSERT ... SELECT` statements, see Section 13.2.5.2, “INSERT ... ON DUPLICATE KEY UPDATE Statement” for conditions under which the `SELECT` columns can be referred to in an `ON DUPLICATE KEY UPDATE` clause.

The order in which a `SELECT` statement with no `ORDER BY` clause returns rows is nondeterministic. This means that, when using replication, there is no guarantee that such a `SELECT` returns rows in the same order on the source and the replica, which can lead to inconsistencies between them. To prevent this from occurring, always write `INSERT ... SELECT` statements that are to be replicated using an `ORDER BY` clause that produces the same row order on the source and the replica. See also Section 17.4.1.17, “Replication and LIMIT”.

Due to this issue, `INSERT ... SELECT ON DUPLICATE KEY UPDATE` and `INSERT IGNORE ... SELECT` statements are flagged as unsafe for statement-based replication. Such statements produce a

warning in the error log when using statement-based mode and are written to the binary log using the row-based format when using `MIXED` mode. (Bug #11758262, Bug #50439)

See also Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”.

An `INSERT ... SELECT` statement affecting partitioned tables using a storage engine such as `MyISAM` that employs table-level locks locks all partitions of the target table; however, only those partitions that are actually read from the source table are locked. (This does not occur with tables using storage engines such as `InnoDB` that employ row-level locking.) For more information, see Section 19.6.4, “Partitioning and Locking”.

13.2.5.2 INSERT ... ON DUPLICATE KEY UPDATE Statement

If you specify an `ON DUPLICATE KEY UPDATE` clause and a row to be inserted would cause a duplicate value in a `UNIQUE` index or `PRIMARY KEY`, an `UPDATE` of the old row occurs. For example, if column `a` is declared as `UNIQUE` and contains the value `1`, the following two statements have similar effect:

```
INSERT INTO t1 (a,b,c) VALUES (1,2,3)
  ON DUPLICATE KEY UPDATE c=c+1;

UPDATE t1 SET c=c+1 WHERE a=1;
```

The effects are not quite identical: For an `InnoDB` table where `a` is an auto-increment column, the `INSERT` statement increases the auto-increment value but the `UPDATE` does not.

If column `b` is also unique, the `INSERT` is equivalent to this `UPDATE` statement instead:

```
UPDATE t1 SET c=c+1 WHERE a=1 OR b=2 LIMIT 1;
```

If `a=1 OR b=2` matches several rows, only *one* row is updated. In general, you should try to avoid using an `ON DUPLICATE KEY UPDATE` clause on tables with multiple unique indexes.

With `ON DUPLICATE KEY UPDATE`, the affected-rows value per row is 1 if the row is inserted as a new row, 2 if an existing row is updated, and 0 if an existing row is set to its current values. If you specify the `CLIENT_FOUND_ROWS` flag to the `mysql_real_connect()` C API function when connecting to `mysqld`, the affected-rows value is 1 (not 0) if an existing row is set to its current values.

If a table contains an `AUTO_INCREMENT` column and `INSERT ... ON DUPLICATE KEY UPDATE` inserts or updates a row, the `LAST_INSERT_ID()` function returns the `AUTO_INCREMENT` value.

The `ON DUPLICATE KEY UPDATE` clause can contain multiple column assignments, separated by commas.

In assignment value expressions in the `ON DUPLICATE KEY UPDATE` clause, you can use the `VALUES(col_name)` function to refer to column values from the `INSERT` portion of the `INSERT ... ON DUPLICATE KEY UPDATE` statement. In other words, `VALUES(col_name)` in the `ON DUPLICATE KEY UPDATE` clause refers to the value of *`col_name`* that would be inserted, had no duplicate-key conflict occurred. This function is especially useful in multiple-row inserts. The `VALUES()` function is meaningful only in the `ON DUPLICATE KEY UPDATE` clause or `INSERT` statements and returns `NULL` otherwise. Example:

```
INSERT INTO t1 (a,b,c) VALUES (1,2,3),(4,5,6)
  ON DUPLICATE KEY UPDATE c=VALUES(a)+VALUES(b);
```

That statement is identical to the following two statements:

```
INSERT INTO t1 (a,b,c) VALUES (1,2,3)
  ON DUPLICATE KEY UPDATE c=3;
INSERT INTO t1 (a,b,c) VALUES (4,5,6)
  ON DUPLICATE KEY UPDATE c=9;
```

The `DELAYED` option is ignored when you use `ON DUPLICATE KEY UPDATE`.

For `INSERT ... SELECT` statements, these rules apply regarding acceptable forms of `SELECT` query expressions that you can refer to in an `ON DUPLICATE KEY UPDATE` clause:

- References to columns from queries on a single table, which may be a derived table.
- References to columns from queries on a join over multiple tables.
- References to columns from `DISTINCT` queries.
- References to columns in other tables, as long as the `SELECT` does not use `GROUP BY`. One side effect is that you must qualify references to nonunique column names.

References to columns from a `UNION` do not work reliably. To work around this restriction, rewrite the `UNION` as a derived table so that its rows can be treated as a single-table result set. For example, this statement can produce incorrect results:

```
INSERT INTO t1 (a, b)
  SELECT c, d FROM t2
  UNION
  SELECT e, f FROM t3
ON DUPLICATE KEY UPDATE b = b + c;
```

Instead, use an equivalent statement that rewrites the `UNION` as a derived table:

```
INSERT INTO t1 (a, b)
SELECT * FROM
  (SELECT c, d FROM t2
   UNION
   SELECT e, f FROM t3) AS dt
ON DUPLICATE KEY UPDATE b = b + c;
```

The technique of rewriting a query as a derived table also enables references to columns from `GROUP BY` queries.

Because the results of `INSERT ... SELECT` statements depend on the ordering of rows from the `SELECT` and this order cannot always be guaranteed, it is possible when logging `INSERT ... SELECT ON DUPLICATE KEY UPDATE` statements for the source and the replica to diverge. Thus, `INSERT ... SELECT ON DUPLICATE KEY UPDATE` statements are flagged as unsafe for statement-based replication. Such statements produce a warning in the error log when using statement-based mode and are written to the binary log using the row-based format when using `MIXED` mode. An `INSERT ... ON DUPLICATE KEY UPDATE` statement against a table having more than one unique or primary key is also marked as unsafe. (Bug #11765650, Bug #58637)

See also Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”.

An `INSERT ... ON DUPLICATE KEY UPDATE` on a partitioned table using a storage engine such as `MyISAM` that employs table-level locks locks any partitions of the table in which a partitioning key column is updated. (This does not occur with tables using storage engines such as `InnoDB` that employ row-level locking.) For more information, see Section 19.6.4, “Partitioning and Locking”.

13.2.5.3 INSERT DELAYED Statement

```
INSERT DELAYED ...
```

The `DELAYED` option for the `INSERT` statement is a MySQL extension to standard SQL that can be used for certain kinds of tables (such as `MyISAM`). When a client uses `INSERT DELAYED`, it gets an okay from the server at once, and the row is queued to be inserted when the table is not in use by any other thread.

Note

`INSERT DELAYED` is slower than a normal `INSERT` if the table is not otherwise in use. There is also the additional overhead for the server to handle a separate

> thread for each table for which there are delayed rows. This means that you should use `INSERT DELAYED` only when you are really sure that you need it.
>
> As of MySQL 5.6.6, `INSERT DELAYED` is deprecated; expect it to be removed in a future release. Use `INSERT` (without `DELAYED`) instead.

The queued rows are held only in memory until they are inserted into the table. This means that if you terminate `mysqld` forcibly (for example, with `kill -9`) or if `mysqld` dies unexpectedly, *any queued rows that have not been written to disk are lost*.

There are some constraints on the use of `DELAYED`:

- `INSERT DELAYED` works only with `MyISAM`, `MEMORY`, `ARCHIVE`, and `BLACKHOLE` tables. For engines that do not support `DELAYED`, an error occurs.
- An error occurs for `INSERT DELAYED` if used with a table that has been locked with `LOCK TABLES` because the insert must be handled by a separate thread, not by the session that holds the lock.
- For `MyISAM` tables, if there are no free blocks in the middle of the data file, concurrent `SELECT` and `INSERT` statements are supported. Under these circumstances, you very seldom need to use `INSERT DELAYED` with `MyISAM`.
- `INSERT DELAYED` should be used only for `INSERT` statements that specify value lists. The server ignores `DELAYED` for `INSERT ... SELECT` or `INSERT ... ON DUPLICATE KEY UPDATE` statements.
- Because the `INSERT DELAYED` statement returns immediately, before the rows are inserted, you cannot use `LAST_INSERT_ID()` to get the `AUTO_INCREMENT` value that the statement might generate.
- `DELAYED` rows are not visible to `SELECT` statements until they actually have been inserted.
- `INSERT DELAYED` is handled as a simple `INSERT` (that is, without the `DELAYED` option) whenever the value of `binlog_format` is `STATEMENT` or `MIXED`. (In the latter case, the statement does not trigger a switch to row-based logging, and so is logged using the statement-based format.)

  This does not apply when using row-based binary logging mode (`binlog_format` set to `ROW`), in which `INSERT DELAYED` statements are always executed using the `DELAYED` option as specified, and logged as row-update events.
- `DELAYED` is ignored on replica servers, so that `INSERT DELAYED` is treated as a normal `INSERT` on replicas. This is because `DELAYED` could cause the slave to have different data than the source.
- Pending `INSERT DELAYED` statements are lost if a table is write locked and `ALTER TABLE` is used to modify the table structure.
- `INSERT DELAYED` is not supported for views.
- `INSERT DELAYED` is not supported for partitioned tables.

The following describes in detail what happens when you use the `DELAYED` option to `INSERT` or `REPLACE`. In this description, the “thread” is the thread that received an `INSERT DELAYED` statement and “handler” is the thread that handles all `INSERT DELAYED` statements for a particular table.

- When a thread executes a `DELAYED` statement for a table, a handler thread is created to process all `DELAYED` statements for the table, if no such handler already exists.
- The thread checks whether the handler has previously acquired a `DELAYED` lock; if not, it tells the handler thread to do so. The `DELAYED` lock can be obtained even if other threads have a `READ` or `WRITE` lock on the table. However, the handler waits for all `ALTER TABLE` locks or `FLUSH TABLES` statements to finish, to ensure that the table structure is up to date.

- The thread executes the `INSERT` statement, but instead of writing the row to the table, it puts a copy of the final row into a queue that is managed by the handler thread. Any syntax errors are noticed by the thread and reported to the client program.
- The client cannot obtain from the server the number of duplicate rows or the `AUTO_INCREMENT` value for the resulting row, because the `INSERT` returns before the insert operation has been completed. (If you use the C API, the `mysql_info()` function does not return anything meaningful, for the same reason.)
- The binary log is updated by the handler thread when the row is inserted into the table. In case of multiple-row inserts, the binary log is updated when the first row is inserted.
- Each time that `delayed_insert_limit` rows are written, the handler checks whether any `SELECT` statements are still pending. If so, it permits these to execute before continuing.
- When the handler has no more rows in its queue, the table is unlocked. If no new `INSERT DELAYED` statements are received within `delayed_insert_timeout` seconds, the handler terminates.
- If more than `delayed_queue_size` rows are pending in a specific handler queue, the thread requesting `INSERT DELAYED` waits until there is room in the queue. This is done to ensure that `mysqld` does not use all memory for the delayed memory queue.
- The handler thread shows up in the MySQL process list with `delayed_insert` in the `Command` column. It is killed if you execute a `FLUSH TABLES` statement or kill it with `KILL *thread_id*`. However, before exiting, it first stores all queued rows into the table. During this time it does not accept any new `INSERT` statements from other threads. If you execute an `INSERT DELAYED` statement after this, a new handler thread is created.

  This means that `INSERT DELAYED` statements have higher priority than normal `INSERT` statements if there is an `INSERT DELAYED` handler running. Other update statements have to wait until the `INSERT DELAYED` queue is empty, someone terminates the handler thread (with `KILL *thread_id*`), or someone executes a `FLUSH TABLES`.
- The following status variables provide information about `INSERT DELAYED` statements.

Status Variable	Meaning
`Delayed_insert_threads`	Number of handler threads
`Delayed_writes`	Number of rows written with `INSERT DELAYED`
`Not_flushed_delayed_rows`	Number of rows waiting to be written

You can view these variables by issuing a `SHOW STATUS` statement or by executing a `mysqladmin extended-status` command.

13.2.6 LOAD DATA Statement

```
LOAD DATA
    [LOW_PRIORITY | CONCURRENT] [LOCAL]
    INFILE 'file_name'
    [REPLACE | IGNORE]
    INTO TABLE tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [CHARACTER SET charset_name]
    [{FIELDS | COLUMNS}
        [TERMINATED BY 'string']
        [[OPTIONALLY] ENCLOSED BY 'char']
        [ESCAPED BY 'char']
    ]
    [LINES
        [STARTING BY 'string']
        [TERMINATED BY 'string']
    ]
```

```
    [IGNORE number {LINES | ROWS}]
    [(col_name_or_user_var
        [, col_name_or_user_var] ...)]
    [SET col_name={expr | DEFAULT}
        [, col_name={expr | DEFAULT}] ...]
```

The `LOAD DATA` statement reads rows from a text file into a table at a very high speed. `LOAD DATA` is the complement of `SELECT ... INTO OUTFILE`. (See Section 13.2.9.1, “SELECT ... INTO Statement”.) To write data from a table to a file, use `SELECT ... INTO OUTFILE`. To read the file back into a table, use `LOAD DATA`. The syntax of the `FIELDS` and `LINES` clauses is the same for both statements.

You can also load data files by using the `mysqlimport` utility; see Section 4.5.5, “`mysqlimport` — A Data Import Program”. `mysqlimport` operates by sending a `LOAD DATA` statement to the server.

For more information about the efficiency of `INSERT` versus `LOAD DATA` and speeding up `LOAD DATA`, see Section 8.2.4.1, “Optimizing INSERT Statements”.

- Partitioned Table Support
- Input File Name, Location, and Content Interpretation
- Concurrency Considerations
- Duplicate-Key Handling
- Index Handling
- Field and Line Handling
- Column List Specification
- Input Preprocessing
- Statement Result Information
- Miscellaneous Topics

Partitioned Table Support

`LOAD DATA` supports explicit partition selection using the `PARTITION` option with a list of one or more comma-separated names of partitions, subpartitions, or both. When this option is used, if any rows from the file cannot be inserted into any of the partitions or subpartitions named in the list, the statement fails with the error `Found a row not matching the given partition set`. For more information and examples, see Section 19.5, “Partition Selection”.

For partitioned tables using storage engines that employ table locks, such as `MyISAM`, `LOAD DATA` cannot prune any partition locks. This does not apply to tables using storage engines that employ row-level locking, such as `InnoDB`. For more information, see Section 19.6.4, “Partitioning and Locking”.

Input File Name, Location, and Content Interpretation

The file name must be given as a literal string. On Windows, specify backslashes in path names as forward slashes or doubled backslashes. The `character_set_filesystem` system variable controls the interpretation of the file name character set.

The server uses the character set indicated by the `character_set_database` system variable to interpret the information in the file. `SET NAMES` and the setting of `character_set_client` do not affect interpretation of input. If the contents of the input file use a character set that differs from the default, it is usually preferable to specify the character set of the file by using the `CHARACTER SET` clause. A character set of `binary` specifies “no conversion.”

`LOAD DATA` interprets all fields in the file as having the same character set, regardless of the data types of the columns into which field values are loaded. For proper interpretation of file contents, you

must ensure that it was written with the correct character set. For example, if you write a data file with `mysqldump -T` or by issuing a `SELECT ... INTO OUTFILE` statement in `mysql`, be sure to use a `--default-character-set` option so that output is written in the character set to be used when the file is loaded with `LOAD DATA`.

Note

It is not possible to load data files that use the `ucs2`, `utf16`, `utf16le`, or `utf32` character set.

The `LOCAL` modifier affects expected location of the file and error handling, as described later. `LOCAL` works only if your server and your client both have been configured to permit it. For example, if `mysqld` was started with the `local_infile` system variable disabled, `LOCAL` does not work. See Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”.

The `LOCAL` modifier affects where the file is expected to be found:

- If `LOCAL` is specified, the file is read by the client program on the client host and sent to the server. The file can be given as a full path name to specify its exact location. If given as a relative path name, the name is interpreted relative to the directory in which the client program was started.

  When using `LOCAL` with `LOAD DATA`, a copy of the file is created in the directory where the MySQL server stores temporary files. See Section B.3.3.5, “Where MySQL Stores Temporary Files”. Lack of sufficient space for the copy in this directory can cause the `LOAD DATA LOCAL` statement to fail.

- If `LOCAL` is not specified, the file must be located on the server host and is read directly by the server. The server uses the following rules to locate the file:

  - If the file name is an absolute path name, the server uses it as given.
  - If the file name is a relative path name with one or more leading components, the server searches for the file relative to the server's data directory.
  - If a file name with no leading components is given, the server looks for the file in the database directory of the default database.

In the non-`LOCAL` case, these rules mean that a file named as `./myfile.txt` is read from the server's data directory, whereas the file named as `myfile.txt` is read from the database directory of the default database. For example, if `db1` is the default database, the following `LOAD DATA` statement reads the file `data.txt` from the database directory for `db1`, even though the statement explicitly loads the file into a table in the `db2` database:

```
LOAD DATA INFILE 'data.txt' INTO TABLE db2.my_table;
```

Non-`LOCAL` load operations read text files located on the server. For security reasons, such operations require that you have the `FILE` privilege. See Section 6.2.2, “Privileges Provided by MySQL”. Also, non-`LOCAL` load operations are subject to the `secure_file_priv` system variable setting. If the variable value is a nonempty directory name, the file to be loaded must be located in that directory. If the variable value is empty (which is insecure), the file need only be readable by the server.

Using `LOCAL` is a bit slower than letting the server access the files directly, because the file contents must be sent over the connection by the client to the server. On the other hand, you do not need the `FILE` privilege to load local files.

`LOCAL` also affects error handling:

- With `LOAD DATA`, data-interpretation and duplicate-key errors terminate the operation.
- With `LOAD DATA LOCAL`, data-interpretation and duplicate-key errors become warnings and the operation continues because the server has no way to stop transmission of the file in the middle of the operation. For duplicate-key errors, this is the same as if `IGNORE` is specified. `IGNORE` is explained further later in this section.

Concurrency Considerations

If you use the `LOW_PRIORITY` modifier, execution of the `LOAD DATA` statement is delayed until no other clients are reading from the table. This affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).

If you specify the `CONCURRENT` modifier with a `MyISAM` table that satisfies the condition for concurrent inserts (that is, it contains no free blocks in the middle), other threads can retrieve data from the table while `LOAD DATA` is executing. This modifier affects the performance of `LOAD DATA` a bit, even if no other thread is using the table at the same time.

For information about `LOAD DATA` in relation to replication, see Section 17.4.1.18, “Replication and LOAD DATA”.

Duplicate-Key Handling

The `REPLACE` and `IGNORE` modifiers control handling of new (input) rows that duplicate existing rows on unique key values:

- If you specify `REPLACE`, new rows replace existing rows. In other words, rows that have the same value for a primary key or unique index as an existing row replace the existing row. See Section 13.2.8, “REPLACE Statement”.
- If you specify `IGNORE`, new rows that duplicate an existing row on a unique key value are discarded.
- If you do not specify either modifier, the behavior depends on whether the `LOCAL` modifier is specified. Without `LOCAL`, an error occurs when a duplicate key value is found, and the rest of the text file is ignored. With `LOCAL`, the default behavior is the same as if `IGNORE` is specified; this is because the server has no way to stop transmission of the file in the middle of the operation.

Index Handling

To ignore foreign key constraints during the load operation, execute a `SET foreign_key_checks = 0` statement before executing `LOAD DATA`.

If you use `LOAD DATA` on an empty `MyISAM` table, all nonunique indexes are created in a separate batch (as for `REPAIR TABLE`). Normally, this makes `LOAD DATA` much faster when you have many indexes. In some extreme cases, you can create the indexes even faster by turning them off with `ALTER TABLE ... DISABLE KEYS` before loading the file into the table and using `ALTER TABLE ... ENABLE KEYS` to re-create the indexes after loading the file. See Section 8.2.4.1, “Optimizing INSERT Statements”.

Field and Line Handling

For both the `LOAD DATA` and `SELECT ... INTO OUTFILE` statements, the syntax of the `FIELDS` and `LINES` clauses is the same. Both clauses are optional, but `FIELDS` must precede `LINES` if both are specified.

If you specify a `FIELDS` clause, each of its subclauses (`TERMINATED BY`, `[OPTIONALLY] ENCLOSED BY`, and `ESCAPED BY`) is also optional, except that you must specify at least one of them. Arguments to these clauses are permitted to contain only ASCII characters.

If you specify no `FIELDS` or `LINES` clause, the defaults are the same as if you had written this:

```
FIELDS TERMINATED BY '\t' ENCLOSED BY '' ESCAPED BY '\\'
LINES TERMINATED BY '\n' STARTING BY ''
```

Backslash is the MySQL escape character within strings in SQL statements. Thus, to specify a literal backslash, you must specify two backslashes for the value to be interpreted as a single backslash. The escape sequences `'\t'` and `'\n'` specify tab and newline characters, respectively.

In other words, the defaults cause `LOAD DATA` to act as follows when reading input:

- Look for line boundaries at newlines.
- Do not skip any line prefix.
- Break lines into fields at tabs.
- Do not expect fields to be enclosed within any quoting characters.
- Interpret characters preceded by the escape character `\` as escape sequences. For example, `\t`, `\n`, and `\\` signify tab, newline, and backslash, respectively. See the discussion of `FIELDS ESCAPED BY` later for the full list of escape sequences.

Conversely, the defaults cause `SELECT ... INTO OUTFILE` to act as follows when writing output:

- Write tabs between fields.
- Do not enclose fields within any quoting characters.
- Use `\` to escape instances of tab, newline, or `\` that occur within field values.
- Write newlines at the ends of lines.

Note

For a text file generated on a Windows system, proper file reading might require `LINES TERMINATED BY '\r\n'` because Windows programs typically use two characters as a line terminator. Some programs, such as `WordPad`, might use `\r` as a line terminator when writing files. To read such files, use `LINES TERMINATED BY '\r'`.

If all the input lines have a common prefix that you want to ignore, you can use `LINES STARTING BY 'prefix_string'` to skip the prefix *and anything before it*. If a line does not include the prefix, the entire line is skipped. Suppose that you issue the following statement:

```
LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test
  FIELDS TERMINATED BY ','  LINES STARTING BY 'xxx';
```

If the data file looks like this:

```
xxx"abc",1
something xxx"def",2
"ghi",3
```

The resulting rows are `("abc",1)` and `("def",2)`. The third row in the file is skipped because it does not contain the prefix.

The `IGNORE number LINES` option can be used to ignore lines at the start of the file. For example, you can use `IGNORE 1 LINES` to skip an initial header line containing column names:

```
LOAD DATA INFILE '/tmp/test.txt' INTO TABLE test IGNORE 1 LINES;
```

When you use `SELECT ... INTO OUTFILE` in tandem with `LOAD DATA` to write data from a database into a file and then read the file back into the database later, the field- and line-handling options for both statements must match. Otherwise, `LOAD DATA` does not interpret the contents of the file properly. Suppose that you use `SELECT ... INTO OUTFILE` to write a file with fields delimited by commas:

```
SELECT * INTO OUTFILE 'data.txt'
  FIELDS TERMINATED BY ','
  FROM table2;
```

To read the comma-delimited file, the correct statement would be:

```
LOAD DATA INFILE 'data.txt' INTO TABLE table2
```

```
  FIELDS TERMINATED BY ',';
```

If instead you tried to read the file with the statement shown following, it wouldn't work because it instructs `LOAD DATA` to look for tabs between fields:

```
LOAD DATA INFILE 'data.txt' INTO TABLE table2
  FIELDS TERMINATED BY '\t';
```

The likely result is that each input line would be interpreted as a single field.

`LOAD DATA` can be used to read files obtained from external sources. For example, many programs can export data in comma-separated values (CSV) format, such that lines have fields separated by commas and enclosed within double quotation marks, with an initial line of column names. If the lines in such a file are terminated by carriage return/newline pairs, the statement shown here illustrates the field- and line-handling options you would use to load the file:

```
LOAD DATA INFILE 'data.txt' INTO TABLE tbl_name
  FIELDS TERMINATED BY ',' ENCLOSED BY '"'
  LINES TERMINATED BY '\r\n'
  IGNORE 1 LINES;
```

If the input values are not necessarily enclosed within quotation marks, use `OPTIONALLY` before the `ENCLOSED BY` option.

Any of the field- or line-handling options can specify an empty string (`''`). If not empty, the `FIELDS [OPTIONALLY] ENCLOSED BY` and `FIELDS ESCAPED BY` values must be a single character. The `FIELDS TERMINATED BY`, `LINES STARTING BY`, and `LINES TERMINATED BY` values can be more than one character. For example, to write lines that are terminated by carriage return/linefeed pairs, or to read a file containing such lines, specify a `LINES TERMINATED BY '\r\n'` clause.

To read a file containing jokes that are separated by lines consisting of `%%`, you can do this

```
CREATE TABLE jokes
  (a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  joke TEXT NOT NULL);
LOAD DATA INFILE '/tmp/jokes.txt' INTO TABLE jokes
  FIELDS TERMINATED BY ''
  LINES TERMINATED BY '\n%%\n' (joke);
```

`FIELDS [OPTIONALLY] ENCLOSED BY` controls quoting of fields. For output (`SELECT ... INTO OUTFILE`), if you omit the word `OPTIONALLY`, all fields are enclosed by the `ENCLOSED BY` character. An example of such output (using a comma as the field delimiter) is shown here:

```
"1","a string","100.20"
"2","a string containing a , comma","102.20"
"3","a string containing a \" quote","102.20"
"4","a string containing a \", quote and comma","102.20"
```

If you specify `OPTIONALLY`, the `ENCLOSED BY` character is used only to enclose values from columns that have a string data type (such as `CHAR`, `BINARY`, `TEXT`, or `ENUM`):

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a \" quote",102.20
4,"a string containing a \", quote and comma",102.20
```

Occurrences of the `ENCLOSED BY` character within a field value are escaped by prefixing them with the `ESCAPED BY` character. Also, if you specify an empty `ESCAPED BY` value, it is possible to inadvertently generate output that cannot be read properly by `LOAD DATA`. For example, the preceding output just shown would appear as follows if the escape character is empty. Observe that the second field in the fourth line contains a comma following the quote, which (erroneously) appears to terminate the field:

```
1,"a string",100.20
2,"a string containing a , comma",102.20
3,"a string containing a " quote",102.20
4,"a string containing a ", quote and comma",102.20
```

For input, the ENCLOSED BY character, if present, is stripped from the ends of field values. (This is true regardless of whether OPTIONALLY is specified; OPTIONALLY has no effect on input interpretation.) Occurrences of the ENCLOSED BY character preceded by the ESCAPED BY character are interpreted as part of the current field value.

If the field begins with the ENCLOSED BY character, instances of that character are recognized as terminating a field value only if followed by the field or line TERMINATED BY sequence. To avoid ambiguity, occurrences of the ENCLOSED BY character within a field value can be doubled and are interpreted as a single instance of the character. For example, if ENCLOSED BY '"' is specified, quotation marks are handled as shown here:

```
"The ""BIG"" boss"  -> The "BIG" boss
The "BIG" boss      -> The "BIG" boss
The ""BIG"" boss    -> The ""BIG"" boss
```

FIELDS ESCAPED BY controls how to read or write special characters:

- For input, if the FIELDS ESCAPED BY character is not empty, occurrences of that character are stripped and the following character is taken literally as part of a field value. Some two-character sequences that are exceptions, where the first character is the escape character. These sequences are shown in the following table (using \ for the escape character). The rules for NULL handling are described later in this section.

Character	Escape Sequence
\0	An ASCII NUL (X'00') character
\b	A backspace character
\n	A newline (linefeed) character
\r	A carriage return character
\t	A tab character.
\Z	ASCII 26 (Control+Z)
\N	NULL

  For more information about \-escape syntax, see Section 9.1.1, “String Literals”.

  If the FIELDS ESCAPED BY character is empty, escape-sequence interpretation does not occur.

- For output, if the FIELDS ESCAPED BY character is not empty, it is used to prefix the following characters on output:

  - The FIELDS ESCAPED BY character.
  - The FIELDS [OPTIONALLY] ENCLOSED BY character.
  - The first character of the FIELDS TERMINATED BY and LINES TERMINATED BY values, if the ENCLOSED BY character is empty or unspecified.
  - ASCII 0 (what is actually written following the escape character is ASCII 0, not a zero-valued byte).

  If the FIELDS ESCAPED BY character is empty, no characters are escaped and NULL is output as NULL, not \N. It is probably not a good idea to specify an empty escape character, particularly if field values in your data contain any of the characters in the list just given.

In certain cases, field- and line-handling options interact:

- If LINES TERMINATED BY is an empty string and FIELDS TERMINATED BY is nonempty, lines are also terminated with FIELDS TERMINATED BY.
- If the FIELDS TERMINATED BY and FIELDS ENCLOSED BY values are both empty (''), a fixed-row (nondelimited) format is used. With fixed-row format, no delimiters are used between fields

(but you can still have a line terminator). Instead, column values are read and written using a field width wide enough to hold all values in the field. For `TINYINT`, `SMALLINT`, `MEDIUMINT`, `INT`, and `BIGINT`, the field widths are 4, 6, 8, 11, and 20, respectively, no matter what the declared display width is.

`LINES TERMINATED BY` is still used to separate lines. If a line does not contain all fields, the rest of the columns are set to their default values. If you do not have a line terminator, you should set this to `''`. In this case, the text file must contain all fields for each row.

Fixed-row format also affects handling of `NULL` values, as described later.

Note

Fixed-size format does not work if you are using a multibyte character set.

Handling of `NULL` values varies according to the `FIELDS` and `LINES` options in use:

- For the default `FIELDS` and `LINES` values, `NULL` is written as a field value of `\N` for output, and a field value of `\N` is read as `NULL` for input (assuming that the `ESCAPED BY` character is `\`).
- If `FIELDS ENCLOSED BY` is not empty, a field containing the literal word `NULL` as its value is read as a `NULL` value. This differs from the word `NULL` enclosed within `FIELDS ENCLOSED BY` characters, which is read as the string `'NULL'`.
- If `FIELDS ESCAPED BY` is empty, `NULL` is written as the word `NULL`.
- With fixed-row format (which is used when `FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` are both empty), `NULL` is written as an empty string. This causes both `NULL` values and empty strings in the table to be indistinguishable when written to the file because both are written as empty strings. If you need to be able to tell the two apart when reading the file back in, you should not use fixed-row format.

An attempt to load `NULL` into a `NOT NULL` column causes assignment of the implicit default value for the column's data type and a warning, or an error in strict SQL mode. Implicit default values are discussed in Section 11.5, “Data Type Default Values”.

Some cases are not supported by `LOAD DATA`:

- Fixed-size rows (`FIELDS TERMINATED BY` and `FIELDS ENCLOSED BY` both empty) and `BLOB` or `TEXT` columns.
- If you specify one separator that is the same as or a prefix of another, `LOAD DATA` cannot interpret the input properly. For example, the following `FIELDS` clause would cause problems:

```
FIELDS TERMINATED BY '"' ENCLOSED BY '"'
```

- If `FIELDS ESCAPED BY` is empty, a field value that contains an occurrence of `FIELDS ENCLOSED BY` or `LINES TERMINATED BY` followed by the `FIELDS TERMINATED BY` value causes `LOAD DATA` to stop reading a field or line too early. This happens because `LOAD DATA` cannot properly determine where the field or line value ends.

Column List Specification

The following example loads all columns of the `persondata` table:

```
LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata;
```

By default, when no column list is provided at the end of the `LOAD DATA` statement, input lines are expected to contain a field for each table column. If you want to load only some of a table's columns, specify a column list:

```
LOAD DATA INFILE 'persondata.txt' INTO TABLE persondata
(col_name_or_user_var [, col_name_or_user_var] ...);
```

You must also specify a column list if the order of the fields in the input file differs from the order of the columns in the table. Otherwise, MySQL cannot tell how to match input fields with table columns.

Input Preprocessing

Each *`col_name_or_user_var`* value is either a column name or a user variable. With user variables, the `SET` clause enables you to perform preprocessing transformations on their values before assigning the result to columns.

User variables in the `SET` clause can be used in several ways. The following example uses the first input column directly for the value of `t1.column1`, and assigns the second input column to a user variable that is subjected to a division operation before being used for the value of `t1.column2`:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, @var1)
  SET column2 = @var1/100;
```

The `SET` clause can be used to supply values not derived from the input file. The following statement sets `column3` to the current date and time:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, column2)
  SET column3 = CURRENT_TIMESTAMP;
```

You can also discard an input value by assigning it to a user variable and not assigning the variable to a table column:

```
LOAD DATA INFILE 'file.txt'
  INTO TABLE t1
  (column1, @dummy, column2, @dummy, column3);
```

Use of the column/variable list and `SET` clause is subject to the following restrictions:

- Assignments in the `SET` clause should have only column names on the left hand side of assignment operators.
- You can use subqueries in the right hand side of `SET` assignments. A subquery that returns a value to be assigned to a column may be a scalar subquery only. Also, you cannot use a subquery to select from the table that is being loaded.
- Lines ignored by an `IGNORE` clause are not processed for the column/variable list or `SET` clause.
- User variables cannot be used when loading data with fixed-row format because user variables do not have a display width.

When processing an input line, `LOAD DATA` splits it into fields and uses the values according to the column/variable list and the `SET` clause, if they are present. Then the resulting row is inserted into the table. If there are `BEFORE INSERT` or `AFTER INSERT` triggers for the table, they are activated before or after inserting the row, respectively.

If an input line has too many fields, the extra fields are ignored and the number of warnings is incremented.

If an input line has too few fields, the table columns for which input fields are missing are set to their default values. Default value assignment is described in Section 11.5, “Data Type Default Values”.

An empty field value is interpreted different from a missing field:

- For string types, the column is set to the empty string.
- For numeric types, the column is set to `0`.
- For date and time types, the column is set to the appropriate “zero” value for the type. See Section 11.2, “Date and Time Data Types”.

These are the same values that result if you assign an empty string explicitly to a string, numeric, or date or time type explicitly in an `INSERT` or `UPDATE` statement.

Treatment of empty or incorrect field values differs from that just described if the SQL mode is set to a restrictive value. For example, if `sql_mode` is set to `TRADITIONAL`, conversion of an empty value or a value such as `'x'` for a numeric column results in an error, not conversion to 0. (With `LOCAL` or `IGNORE`, warnings occur rather than errors, even with a restrictive `sql_mode` value, and the row is inserted using the same closest-value behavior used for nonrestrictive SQL modes. This occurs because the server has no way to stop transmission of the file in the middle of the operation.)

`TIMESTAMP` columns are set to the current date and time only if there is a `NULL` value for the column (that is, `\N`) and the column is not declared to permit `NULL` values, or if the `TIMESTAMP` column's default value is the current timestamp and it is omitted from the field list when a field list is specified.

`LOAD DATA` regards all input as strings, so you cannot use numeric values for `ENUM` or `SET` columns the way you can with `INSERT` statements. All `ENUM` and `SET` values must be specified as strings.

`BIT` values cannot be loaded directly using binary notation (for example, `b'011010'`). To work around this, use the `SET` clause to strip off the leading `b'` and trailing `'` and perform a base-2 to base-10 conversion so that MySQL loads the values into the `BIT` column properly:

```
shell> cat /tmp/bit_test.txt
b'10'
b'1111111'
shell> mysql test
mysql> LOAD DATA INFILE '/tmp/bit_test.txt'
       INTO TABLE bit_test (@var1)
       SET b = CAST(CONV(MID(@var1, 3, LENGTH(@var1)-3), 2, 10) AS UNSIGNED);
Query OK, 2 rows affected (0.00 sec)
Records: 2  Deleted: 0  Skipped: 0  Warnings: 0

mysql> SELECT BIN(b+0) FROM bit_test;
+----------+
| BIN(b+0) |
+----------+
| 10       |
| 1111111  |
+----------+
2 rows in set (0.00 sec)
```

For `BIT` values in `0b` binary notation (for example, `0b011010`), use this `SET` clause instead to strip off the leading `0b`:

```
SET b = CAST(CONV(MID(@var1, 3, LENGTH(@var1)-2), 2, 10) AS UNSIGNED)
```

Statement Result Information

When the `LOAD DATA` statement finishes, it returns an information string in the following format:

```
Records: 1  Deleted: 0  Skipped: 0  Warnings: 0
```

Warnings occur under the same circumstances as when values are inserted using the `INSERT` statement (see Section 13.2.5, “INSERT Statement”), except that `LOAD DATA` also generates warnings when there are too few or too many fields in the input row.

You can use `SHOW WARNINGS` to get a list of the first `max_error_count` warnings as information about what went wrong. See Section 13.7.5.41, “SHOW WARNINGS Statement”.

If you are using the C API, you can get information about the statement by calling the `mysql_info()` function. See mysql_info().

Miscellaneous Topics

On Unix, if you need `LOAD DATA` to read from a pipe, you can use the following technique (the example loads a listing of the `/` directory into the table `db1.t1`):

```
mkfifo /mysql/data/db1/ls.dat
chmod 666 /mysql/data/db1/ls.dat
find / -ls > /mysql/data/db1/ls.dat &
mysql -e "LOAD DATA INFILE 'ls.dat' INTO TABLE t1" db1
```

Here you must run the command that generates the data to be loaded and the `mysql` commands either on separate terminals, or run the data generation process in the background (as shown in the preceding example). If you do not do this, the pipe blocks until data is read by the `mysql` process.

13.2.7 LOAD XML Statement

```
LOAD XML
    [LOW_PRIORITY | CONCURRENT] [LOCAL]
    INFILE 'file_name'
    [REPLACE | IGNORE]
    INTO TABLE [db_name.]tbl_name
    [CHARACTER SET charset_name]
    [ROWS IDENTIFIED BY '<tagname>']
    [IGNORE number {LINES | ROWS}]
    [(field_name_or_user_var
        [, field_name_or_user_var] ...)]
    [SET col_name={expr | DEFAULT},
        [, col_name={expr | DEFAULT}] ...]
```

The `LOAD XML` statement reads data from an XML file into a table. The *`file_name`* must be given as a literal string. The *`tagname`* in the optional `ROWS IDENTIFIED BY` clause must also be given as a literal string, and must be surrounded by angle brackets (`<` and `>`).

`LOAD XML` acts as the complement of running the `mysql` client in XML output mode (that is, starting the client with the `--xml` option). To write data from a table to an XML file, you can invoke the `mysql` client with the `--xml` and `-e` options from the system shell, as shown here:

```
shell> mysql --xml -e 'SELECT * FROM mydb.mytable' > file.xml
```

To read the file back into a table, use `LOAD XML`. By default, the `<row>` element is considered to be the equivalent of a database table row; this can be changed using the `ROWS IDENTIFIED BY` clause.

This statement supports three different XML formats:

- Column names as attributes and column values as attribute values:

  ```
  <row column1="value1" column2="value2" .../>
  ```

- Column names as tags and column values as the content of these tags:

  ```
  <row>
    <column1>value1</column1>
    <column2>value2</column2>
  </row>
  ```

- Column names are the `name` attributes of `<field>` tags, and values are the contents of these tags:

  ```
  <row>
    <field name='column1'>value1</field>
    <field name='column2'>value2</field>
  </row>
  ```

  This is the format used by other MySQL tools, such as `mysqldump`.

All three formats can be used in the same XML file; the import routine automatically detects the format for each row and interprets it correctly. Tags are matched based on the tag or attribute name and the column name.

Prior to MySQL 5.6.27, `LOAD XML` did not handle empty XML elements in the form `<element/>` correctly. (Bug #67542, Bug #16171518)

In MySQL 5.6, `LOAD XML` does not support `CDATA` sections in the source XML. This limitation is removed in MySQL 8.0. (Bug #30753708, Bug #98199)

The following clauses work essentially the same way for `LOAD XML` as they do for `LOAD DATA`:

- `LOW_PRIORITY` or `CONCURRENT`
- `LOCAL`
- `REPLACE` or `IGNORE`
- `CHARACTER SET`
- `SET`

See Section 13.2.6, "LOAD DATA Statement", for more information about these clauses.

`(field_name_or_user_var, ...)` is a list of one or more comma-separated XML fields or user variables. The name of a user variable used for this purpose must match the name of a field from the XML file, prefixed with `@`. You can use field names to select only desired fields. User variables can be employed to store the corresponding field values for subsequent re-use.

The `IGNORE number LINES` or `IGNORE number ROWS` clause causes the first `number` rows in the XML file to be skipped. It is analogous to the `LOAD DATA` statement's `IGNORE ... LINES` clause.

Suppose that we have a table named `person`, created as shown here:

```
USE test;

CREATE TABLE person (
    person_id INT NOT NULL PRIMARY KEY,
    fname VARCHAR(40) NULL,
    lname VARCHAR(40) NULL,
    created TIMESTAMP
);
```

Suppose further that this table is initially empty.

Now suppose that we have a simple XML file `person.xml`, whose contents are as shown here:

```
<list>
  <person person_id="1" fname="Kapek" lname="Sainnouine"/>
  <person person_id="2" fname="Sajon" lname="Rondela"/>
  <person person_id="3"><fname>Likame</fname><lname>Örrtmons</lname></person>
  <person person_id="4"><fname>Slar</fname><lname>Manlanth</lname></person>
  <person><field name="person_id">5</field><field name="fname">Stoma</field>
    <field name="lname">Milu</field></person>
  <person><field name="person_id">6</field><field name="fname">Nirtam</field>
    <field name="lname">Sklöd</field></person>
  <person person_id="7"><fname>Sungam</fname><lname>Dulbåd</lname></person>
  <person person_id="8" fname="Sraref" lname="Encmelt"/>
</list>
```

Each of the permissible XML formats discussed previously is represented in this example file.

To import the data in `person.xml` into the `person` table, you can use this statement:

```
mysql> LOAD XML LOCAL INFILE 'person.xml'
    ->   INTO TABLE person
    ->   ROWS IDENTIFIED BY '<person>';

Query OK, 8 rows affected (0.00 sec)
Records: 8  Deleted: 0  Skipped: 0  Warnings: 0
```

Here, we assume that `person.xml` is located in the MySQL data directory. If the file cannot be found, the following error results:

```
ERROR 2 (HY000): File '/person.xml' not found (Errcode: 2)
```

The `ROWS IDENTIFIED BY '<person>'` clause means that each `<person>` element in the XML file is considered equivalent to a row in the table into which the data is to be imported. In this case, this is the `person` table in the `test` database.

As can be seen by the response from the server, 8 rows were imported into the `test.person` table. This can be verified by a simple `SELECT` statement:

```
mysql> SELECT * FROM person;
+-----------+--------+------------+---------------------+
| person_id | fname  | lname      | created             |
+-----------+--------+------------+---------------------+
|         1 | Kapek  | Sainnouine | 2007-07-13 16:18:47 |
|         2 | Sajon  | Rondela    | 2007-07-13 16:18:47 |
|         3 | Likame | Örrtmons   | 2007-07-13 16:18:47 |
|         4 | Slar   | Manlanth   | 2007-07-13 16:18:47 |
|         5 | Stoma  | Nilu       | 2007-07-13 16:18:47 |
|         6 | Nirtam | Sklöd      | 2007-07-13 16:18:47 |
|         7 | Sungam | Dulbåd     | 2007-07-13 16:18:47 |
|         8 | Sreraf | Encmelt    | 2007-07-13 16:18:47 |
+-----------+--------+------------+---------------------+
8 rows in set (0.00 sec)
```

This shows, as stated earlier in this section, that any or all of the 3 permitted XML formats may appear in a single file and be read using `LOAD XML`.

The inverse of the import operation just shown—that is, dumping MySQL table data into an XML file—can be accomplished using the `mysql` client from the system shell, as shown here:

```
shell> mysql --xml -e "SELECT * FROM test.person" > person-dump.xml
shell> cat person-dump.xml
<?xml version="1.0"?>

<resultset statement="SELECT * FROM test.person" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <row>
 <field name="person_id">1</field>
 <field name="fname">Kapek</field>
 <field name="lname">Sainnouine</field>
  </row>

  <row>
 <field name="person_id">2</field>
 <field name="fname">Sajon</field>
 <field name="lname">Rondela</field>
  </row>

  <row>
 <field name="person_id">3</field>
 <field name="fname">Likema</field>
 <field name="lname">Örrtmons</field>
  </row>

  <row>
 <field name="person_id">4</field>
 <field name="fname">Slar</field>
 <field name="lname">Manlanth</field>
  </row>

  <row>
 <field name="person_id">5</field>
 <field name="fname">Stoma</field>
 <field name="lname">Nilu</field>
  </row>

  <row>
 <field name="person_id">6</field>
 <field name="fname">Nirtam</field>
 <field name="lname">Sklöd</field>
  </row>

  <row>
```

```
 <field name="person_id">7</field>
 <field name="fname">Sungam</field>
 <field name="lname">Dulbåd</field>
  </row>

  <row>
 <field name="person_id">8</field>
 <field name="fname">Sreraf</field>
 <field name="lname">Encmelt</field>
  </row>
</resultset>
```

Note

The `--xml` option causes the `mysql` client to use XML formatting for its output; the `-e` option causes the client to execute the SQL statement immediately following the option. See Section 4.5.1, “`mysql` — The MySQL Command-Line Client”.

You can verify that the dump is valid by creating a copy of the `person` table and importing the dump file into the new table, like this:

```
mysql> USE test;
mysql> CREATE TABLE person2 LIKE person;
Query OK, 0 rows affected (0.00 sec)

mysql> LOAD XML LOCAL INFILE 'person-dump.xml'
    ->   INTO TABLE person2;
Query OK, 8 rows affected (0.01 sec)
Records: 8  Deleted: 0  Skipped: 0  Warnings: 0

mysql> SELECT * FROM person2;
+-----------+--------+------------+---------------------+
| person_id | fname  | lname      | created             |
+-----------+--------+------------+---------------------+
|         1 | Kapek  | Sainnouine | 2007-07-13 16:18:47 |
|         2 | Sajon  | Rondela    | 2007-07-13 16:18:47 |
|         3 | Likema | Örrtmons   | 2007-07-13 16:18:47 |
|         4 | Slar   | Manlanth   | 2007-07-13 16:18:47 |
|         5 | Stoma  | Nilu       | 2007-07-13 16:18:47 |
|         6 | Nirtam | Sklöd      | 2007-07-13 16:18:47 |
|         7 | Sungam | Dulbåd     | 2007-07-13 16:18:47 |
|         8 | Sreraf | Encmelt    | 2007-07-13 16:18:47 |
+-----------+--------+------------+---------------------+
8 rows in set (0.00 sec)
```

There is no requirement that every field in the XML file be matched with a column in the corresponding table. Fields which have no corresponding columns are skipped. You can see this by first emptying the `person2` table and dropping the `created` column, then using the same `LOAD XML` statement we just employed previously, like this:

```
mysql> TRUNCATE person2;
Query OK, 8 rows affected (0.26 sec)

mysql> ALTER TABLE person2 DROP COLUMN created;
Query OK, 0 rows affected (0.52 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW CREATE TABLE person2\G
*************************** 1. row ***************************
       Table: person2
Create Table: CREATE TABLE `person2` (
  `person_id` int(11) NOT NULL,
  `fname` varchar(40) DEFAULT NULL,
  `lname` varchar(40) DEFAULT NULL,
  PRIMARY KEY (`person_id`)
) ENGINE=InnoDB DEFAULT CHARSET=utf8
1 row in set (0.00 sec)

mysql> LOAD XML LOCAL INFILE 'person-dump.xml'
```

```
    ->   INTO TABLE person2;
Query OK, 8 rows affected (0.01 sec)
Records: 8  Deleted: 0  Skipped: 0  Warnings: 0

mysql> SELECT * FROM person2;
+-----------+--------+------------+
| person_id | fname  | lname      |
+-----------+--------+------------+
|         1 | Kapek  | Sainnouine |
|         2 | Sajon  | Rondela    |
|         3 | Likema | Örrtmons   |
|         4 | Slar   | Manlanth   |
|         5 | Stoma  | Nilu       |
|         6 | Nirtam | Sklöd      |
|         7 | Sungam | Dulbåd     |
|         8 | Sreraf | Encmelt    |
+-----------+--------+------------+
8 rows in set (0.00 sec)
```

The order in which the fields are given within each row of the XML file does not affect the operation of `LOAD XML`; the field order can vary from row to row, and is not required to be in the same order as the corresponding columns in the table.

As mentioned previously, you can use a *`(field_name_or_user_var, ...)`* list of one or more XML fields (to select desired fields only) or user variables (to store the corresponding field values for later use). User variables can be especially useful when you want to insert data from an XML file into table columns whose names do not match those of the XML fields. To see how this works, we first create a table named `individual` whose structure matches that of the `person` table, but whose columns are named differently:

```
mysql> CREATE TABLE individual (
    ->     individual_id INT NOT NULL PRIMARY KEY,
    ->     name1 VARCHAR(40) NULL,
    ->     name2 VARCHAR(40) NULL,
    ->     made TIMESTAMP
    -> );
Query OK, 0 rows affected (0.42 sec)
```

In this case, you cannot simply load the XML file directly into the table, because the field and column names do not match:

```
mysql> LOAD XML INFILE '../bin/person-dump.xml' INTO TABLE test.individual;
ERROR 1263 (22004): Column set to default value; NULL supplied to NOT NULL column 'individual_id' at ro
```

This happens because the MySQL server looks for field names matching the column names of the target table. You can work around this problem by selecting the field values into user variables, then setting the target table's columns equal to the values of those variables using `SET`. You can perform both of these operations in a single statement, as shown here:

```
mysql> LOAD XML INFILE '../bin/person-dump.xml'
    ->     INTO TABLE test.individual (@person_id, @fname, @lname, @created)
    ->     SET individual_id=@person_id, name1=@fname, name2=@lname, made=@created;
Query OK, 8 rows affected (0.05 sec)
Records: 8  Deleted: 0  Skipped: 0  Warnings: 0

mysql> SELECT * FROM individual;
+---------------+--------+------------+---------------------+
| individual_id | name1  | name2      | made                |
+---------------+--------+------------+---------------------+
|             1 | Kapek  | Sainnouine | 2007-07-13 16:18:47 |
|             2 | Sajon  | Rondela    | 2007-07-13 16:18:47 |
|             3 | Likema | Örrtmons   | 2007-07-13 16:18:47 |
|             4 | Slar   | Manlanth   | 2007-07-13 16:18:47 |
|             5 | Stoma  | Nilu       | 2007-07-13 16:18:47 |
|             6 | Nirtam | Sklöd      | 2007-07-13 16:18:47 |
|             7 | Sungam | Dulbåd     | 2007-07-13 16:18:47 |
|             8 | Srraf  | Encmelt    | 2007-07-13 16:18:47 |
+---------------+--------+------------+---------------------+
8 rows in set (0.00 sec)
```

The names of the user variables *must* match those of the corresponding fields from the XML file, with the addition of the required `@` prefix to indicate that they are variables. The user variables need not be listed or assigned in the same order as the corresponding fields.

Using a `ROWS IDENTIFIED BY '<tagname>'` clause, it is possible to import data from the same XML file into database tables with different definitions. For this example, suppose that you have a file named `address.xml` which contains the following XML:

```
<?xml version="1.0"?>

<list>
  <person person_id="1">
    <fname>Robert</fname>
    <lname>Jones</lname>
    <address address_id="1" street="Mill Creek Road" zip="45365" city="Sidney"/>
    <address address_id="2" street="Main Street" zip="28681" city="Taylorsville"/>
  </person>

  <person person_id="2">
    <fname>Mary</fname>
    <lname>Smith</lname>
    <address address_id="3" street="River Road" zip="80239" city="Denver"/>
    <!-- <address address_id="4" street="North Street" zip="37920" city="Knoxville"/> -->
  </person>

</list>
```

You can again use the `test.person` table as defined previously in this section, after clearing all the existing records from the table and then showing its structure as shown here:

```
mysql< TRUNCATE person;
Query OK, 0 rows affected (0.04 sec)

mysql< SHOW CREATE TABLE person\G
*************************** 1. row ***************************
       Table: person
Create Table: CREATE TABLE `person` (
  `person_id` int(11) NOT NULL,
  `fname` varchar(40) DEFAULT NULL,
  `lname` varchar(40) DEFAULT NULL,
  `created` timestamp NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (`person_id`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

Now create an `address` table in the `test` database using the following `CREATE TABLE` statement:

```
CREATE TABLE address (
    address_id INT NOT NULL PRIMARY KEY,
    person_id INT NULL,
    street VARCHAR(40) NULL,
    zip INT NULL,
    city VARCHAR(40) NULL,
    created TIMESTAMP
);
```

To import the data from the XML file into the `person` table, execute the following `LOAD XML` statement, which specifies that rows are to be specified by the `<person>` element, as shown here;

```
mysql> LOAD XML LOCAL INFILE 'address.xml'
    ->   INTO TABLE person
    ->   ROWS IDENTIFIED BY '<person>';
Query OK, 2 rows affected (0.00 sec)
Records: 2  Deleted: 0  Skipped: 0  Warnings: 0
```

You can verify that the records were imported using a `SELECT` statement:

```
mysql> SELECT * FROM person;
+-----------+--------+-------+---------------------+
| person_id | fname  | lname | created             |
```

```
+----------+--------+-------+---------------------+
|         1 | Robert | Jones | 2007-07-24 17:37:06 |
|         2 | Mary   | Smith | 2007-07-24 17:37:06 |
+----------+--------+-------+---------------------+
2 rows in set (0.00 sec)
```

Since the `<address>` elements in the XML file have no corresponding columns in the `person` table, they are skipped.

To import the data from the `<address>` elements into the `address` table, use the `LOAD XML` statement shown here:

```
mysql> LOAD XML LOCAL INFILE 'address.xml'
    ->   INTO TABLE address
    ->   ROWS IDENTIFIED BY '<address>';
Query OK, 3 rows affected (0.00 sec)
Records: 3  Deleted: 0  Skipped: 0  Warnings: 0
```

You can see that the data was imported using a `SELECT` statement such as this one:

```
mysql> SELECT * FROM address;
+------------+-----------+-----------------+-------+-------------+---------------------+
| address_id | person_id | street          | zip   | city        | created             |
+------------+-----------+-----------------+-------+-------------+---------------------+
|          1 |         1 | Mill Creek Road | 45365 | Sidney      | 2007-07-24 17:37:37 |
|          2 |         1 | Main Street     | 28681 | Taylorsville | 2007-07-24 17:37:37 |
|          3 |         2 | River Road      | 80239 | Denver      | 2007-07-24 17:37:37 |
+------------+-----------+-----------------+-------+-------------+---------------------+
3 rows in set (0.00 sec)
```

The data from the `<address>` element that is enclosed in XML comments is not imported. However, since there is a `person_id` column in the `address` table, the value of the `person_id` attribute from the parent `<person>` element for each `<address>` *is* imported into the `address` table.

Security Considerations. As with the `LOAD DATA` statement, the transfer of the XML file from the client host to the server host is initiated by the MySQL server. In theory, a patched server could be built that would tell the client program to transfer a file of the server's choosing rather than the file named by the client in the `LOAD XML` statement. Such a server could access any file on the client host to which the client user has read access.

In a Web environment, clients usually connect to MySQL from a Web server. A user that can run any command against the MySQL server can use `LOAD XML LOCAL` to read any files to which the Web server process has read access. In this environment, the client with respect to the MySQL server is actually the Web server, not the remote program being run by the user who connects to the Web server.

You can disable loading of XML files from clients by starting the server with `--local-infile=0` or `--local-infile=OFF`. This option can also be used when starting the `mysql` client to disable `LOAD XML` for the duration of the client session.

To prevent a client from loading XML files from the server, do not grant the `FILE` privilege to the corresponding MySQL user account, or revoke this privilege if the client user account already has it.

Important

Revoking the `FILE` privilege (or not granting it in the first place) keeps the user only from executing the `LOAD XML` statement (as well as the `LOAD_FILE()` function; it does *not* prevent the user from executing `LOAD XML LOCAL`. To disallow this statement, you must start the server or the client with `--local-infile=OFF`.

In other words, the `FILE` privilege affects only whether the client can read files on the server; it has no bearing on whether the client can read files on the local file system.

For partitioned tables using storage engines that employ table locks, such as MyISAM, any locks caused by LOAD XML perform locks on all partitions of the table. This does not apply to tables using storage engines which employ row-level locking, such as InnoDB. For more information, see Section 19.6.4, “Partitioning and Locking”.

13.2.8 REPLACE Statement

```
REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [(col_name [, col_name] ...)]
    {VALUES | VALUE} (value_list) [, (value_list)] ...

REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    SET assignment_list

REPLACE [LOW_PRIORITY | DELAYED]
    [INTO] tbl_name
    [PARTITION (partition_name [, partition_name] ...)]
    [(col_name [, col_name] ...)]
    SELECT ...

value:
    {expr | DEFAULT}

value_list:
    value [, value] ...

assignment:
    col_name = value

assignment_list:
    assignment [, assignment] ...
```

REPLACE works exactly like INSERT, except that if an old row in the table has the same value as a new row for a PRIMARY KEY or a UNIQUE index, the old row is deleted before the new row is inserted. See Section 13.2.5, “INSERT Statement”.

REPLACE is a MySQL extension to the SQL standard. It either inserts, or *deletes* and inserts. For another MySQL extension to standard SQL—that either inserts or *updates*—see Section 13.2.5.2, “INSERT ... ON DUPLICATE KEY UPDATE Statement”.

Note

REPLACE makes sense only if a table has a PRIMARY KEY or UNIQUE index. Otherwise, it becomes equivalent to INSERT, because there is no index to be used to determine whether a new row duplicates another.

Values for all columns are taken from the values specified in the REPLACE statement. Any missing columns are set to their default values, just as happens for INSERT. You cannot refer to values from the current row and use them in the new row. If you use an assignment such as SET *col_name* = *col_name* + 1, the reference to the column name on the right hand side is treated as DEFAULT(*col_name*), so the assignment is equivalent to SET *col_name* = DEFAULT(*col_name*) + 1.

To use REPLACE, you must have both the INSERT and DELETE privileges for the table.

REPLACE supports explicit partition selection using the PARTITION keyword with a list of comma-separated names of partitions, subpartitions, or both. As with INSERT, if it is not possible to insert the new row into any of these partitions or subpartitions, the REPLACE statement fails with the error Found a row not matching the given partition set. For more information and examples, see Section 19.5, “Partition Selection”.

The REPLACE statement returns a count to indicate the number of rows affected. This is the sum of the rows deleted and inserted. If the count is 1 for a single-row REPLACE, a row was inserted and no rows were deleted. If the count is greater than 1, one or more old rows were deleted before the new row was inserted. It is possible for a single row to replace more than one old row if the table contains multiple unique indexes and the new row duplicates values for different old rows in different unique indexes.

The affected-rows count makes it easy to determine whether REPLACE only added a row or whether it also replaced any rows: Check whether the count is 1 (added) or greater (replaced).

If you are using the C API, the affected-rows count can be obtained using the mysql_affected_rows() function.

You cannot replace into a table and select from the same table in a subquery.

MySQL uses the following algorithm for REPLACE (and LOAD DATA ... REPLACE):

1. Try to insert the new row into the table
2. While the insertion fails because a duplicate-key error occurs for a primary key or unique index:
   a. Delete from the table the conflicting row that has the duplicate key value
   b. Try again to insert the new row into the table

It is possible that in the case of a duplicate-key error, a storage engine may perform the REPLACE as an update rather than a delete plus insert, but the semantics are the same. There are no user-visible effects other than a possible difference in how the storage engine increments Handler_*xxx* status variables.

Because the results of REPLACE ... SELECT statements depend on the ordering of rows from the SELECT and this order cannot always be guaranteed, it is possible when logging these statements for the source and the slave to diverge. For this reason, REPLACE ... SELECT statements are flagged as unsafe for statement-based replication. such statements produce a warning in the error log when using statement-based mode and are written to the binary log using the row-based format when using MIXED mode. See also Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”.

When modifying an existing table that is not partitioned to accommodate partitioning, or, when modifying the partitioning of an already partitioned table, you may consider altering the table's primary key (see Section 19.6.1, “Partitioning Keys, Primary Keys, and Unique Keys”). You should be aware that, if you do this, the results of REPLACE statements may be affected, just as they would be if you modified the primary key of a nonpartitioned table. Consider the table created by the following CREATE TABLE statement:

```
CREATE TABLE test (
  id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  data VARCHAR(64) DEFAULT NULL,
  ts TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (id)
);
```

When we create this table and run the statements shown in the mysql client, the result is as follows:

```
mysql> REPLACE INTO test VALUES (1, 'Old', '2014-08-20 18:47:00');
Query OK, 1 row affected (0.04 sec)

mysql> REPLACE INTO test VALUES (1, 'New', '2014-08-20 18:47:42');
Query OK, 2 rows affected (0.04 sec)

mysql> SELECT * FROM test;
+----+------+---------------------+
| id | data | ts                  |
+----+------+---------------------+
|  1 | New  | 2014-08-20 18:47:42 |
+----+------+---------------------+
```

```
1 row in set (0.00 sec)
```

Now we create a second table almost identical to the first, except that the primary key now covers 2 columns, as shown here (emphasized text):

```
CREATE TABLE test2 (
  id INT UNSIGNED NOT NULL AUTO_INCREMENT,
  data VARCHAR(64) DEFAULT NULL,
  ts TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
  PRIMARY KEY (id, ts)
);
```

When we run on `test2` the same two `REPLACE` statements as we did on the original `test` table, we obtain a different result:

```
mysql> REPLACE INTO test2 VALUES (1, 'Old', '2014-08-20 18:47:00');
Query OK, 1 row affected (0.05 sec)

mysql> REPLACE INTO test2 VALUES (1, 'New', '2014-08-20 18:47:42');
Query OK, 1 row affected (0.06 sec)

mysql> SELECT * FROM test2;
+----+------+---------------------+
| id | data | ts                  |
+----+------+---------------------+
|  1 | Old  | 2014-08-20 18:47:00 |
|  1 | New  | 2014-08-20 18:47:42 |
+----+------+---------------------+
2 rows in set (0.00 sec)
```

This is due to the fact that, when run on `test2`, both the `id` and `ts` column values must match those of an existing row for the row to be replaced; otherwise, a row is inserted.

A `REPLACE` statement affecting a partitioned table using a storage engine such as `MyISAM` that employs table-level locks locks only those partitions containing rows that match the `REPLACE` statement `WHERE` clause, as long as none of the table partitioning columns are updated; otherwise the entire table is locked. (For storage engines such as `InnoDB` that employ row-level locking, no locking of partitions takes place.) For more information, see Section 19.6.4, “Partitioning and Locking”.

13.2.9 SELECT Statement

```
SELECT
    [ALL | DISTINCT | DISTINCTROW ]
    [HIGH_PRIORITY]
    [STRAIGHT_JOIN]
    [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
    [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
    select_expr [, select_expr] ...
    [into_option]
    [FROM table_references
      [PARTITION partition_list]]
    [WHERE where_condition]
    [GROUP BY {col_name | expr | position}
      [ASC | DESC], ... [WITH ROLLUP]]
    [HAVING where_condition]
    [ORDER BY {col_name | expr | position}
      [ASC | DESC], ...]
    [LIMIT {[offset,] row_count | row_count OFFSET offset}]
    [PROCEDURE procedure_name(argument_list)]
    [into_option]
    [FOR UPDATE | LOCK IN SHARE MODE]

into_option: {
    INTO OUTFILE 'file_name'
        [CHARACTER SET charset_name]
        export_options
  | INTO DUMPFILE 'file_name'
  | INTO var_name [, var_name] ...
}
```

`SELECT` is used to retrieve rows selected from one or more tables, and can include `UNION` statements and subqueries. See Section 13.2.9.3, “UNION Clause”, and Section 13.2.10, “Subqueries”.

The most commonly used clauses of `SELECT` statements are these:

- Each *`select_expr`* indicates a column that you want to retrieve. There must be at least one *`select_expr`*.
- *`table_references`* indicates the table or tables from which to retrieve rows. Its syntax is described in Section 13.2.9.2, “JOIN Clause”.
- `SELECT` supports explicit partition selection using the `PARTITION` keyword with a list of partitions or subpartitions (or both) following the name of the table in a *`table_reference`* (see Section 13.2.9.2, “JOIN Clause”). In this case, rows are selected only from the partitions listed, and any other partitions of the table are ignored. For more information and examples, see Section 19.5, “Partition Selection”.

  `SELECT ... PARTITION` from tables using storage engines such as `MyISAM` that perform table-level locks (and thus partition locks) lock only the partitions or subpartitions named by the `PARTITION` option.

  For more information, see Section 19.6.4, “Partitioning and Locking”.
- The `WHERE` clause, if given, indicates the condition or conditions that rows must satisfy to be selected. *`where_condition`* is an expression that evaluates to true for each row to be selected. The statement selects all rows if there is no `WHERE` clause.

  In the `WHERE` expression, you can use any of the functions and operators that MySQL supports, except for aggregate (group) functions. See Section 9.5, “Expressions”, and Chapter 12, *Functions and Operators*.

`SELECT` can also be used to retrieve rows computed without reference to any table.

For example:

```
mysql> SELECT 1 + 1;
        -> 2
```

You are permitted to specify `DUAL` as a dummy table name in situations where no tables are referenced:

```
mysql> SELECT 1 + 1 FROM DUAL;
        -> 2
```

`DUAL` is purely for the convenience of people who require that all `SELECT` statements should have `FROM` and possibly other clauses. MySQL may ignore the clauses. MySQL does not require `FROM DUAL` if no tables are referenced.

In general, clauses used must be given in exactly the order shown in the syntax description. For example, a `HAVING` clause must come after any `GROUP BY` clause and before any `ORDER BY` clause. The `INTO` clause, if present, can appear in any position indicated by the syntax description, but within a given statement can appear only once, not in multiple positions. For more information about `INTO`, see Section 13.2.9.1, “SELECT ... INTO Statement”.

The list of *`select_expr`* terms comprises the select list that indicates which columns to retrieve. Terms specify a column or expression or can use `*`-shorthand:

- A select list consisting only of a single unqualified `*` can be used as shorthand to select all columns from all tables:

  ```
  SELECT * FROM t1 INNER JOIN t2 ...
  ```
- *`tbl_name`*`.*` can be used as a qualified shorthand to select all columns from the named table:

  ```
  SELECT t1.*, t2.* FROM t1 INNER JOIN t2 ...
  ```

- Use of an unqualified * with other items in the select list may produce a parse error. To avoid this problem, use a qualified *tbl_name*.* reference:

```
SELECT AVG(score), t1.* FROM t1 ...
```

The following list provides additional information about other SELECT clauses:

- A *select_expr* can be given an alias using AS *alias_name*. The alias is used as the expression's column name and can be used in GROUP BY, ORDER BY, or HAVING clauses. For example:

```
SELECT CONCAT(last_name,', ',first_name) AS full_name
  FROM mytable ORDER BY full_name;
```

The AS keyword is optional when aliasing a *select_expr* with an identifier. The preceding example could have been written like this:

```
SELECT CONCAT(last_name,', ',first_name) full_name
  FROM mytable ORDER BY full_name;
```

However, because the AS is optional, a subtle problem can occur if you forget the comma between two *select_expr* expressions: MySQL interprets the second as an alias name. For example, in the following statement, columnb is treated as an alias name:

```
SELECT columna columnb FROM mytable;
```

For this reason, it is good practice to be in the habit of using AS explicitly when specifying column aliases.

It is not permissible to refer to a column alias in a WHERE clause, because the column value might not yet be determined when the WHERE clause is executed. See Section B.3.4.4, “Problems with Column Aliases”.

- The FROM *table_references* clause indicates the table or tables from which to retrieve rows. If you name more than one table, you are performing a join. For information on join syntax, see Section 13.2.9.2, “JOIN Clause”. For each table specified, you can optionally specify an alias.

```
tbl_name [[AS] alias] [index_hint]
```

The use of index hints provides the optimizer with information about how to choose indexes during query processing. For a description of the syntax for specifying these hints, see Section 8.9.3, “Index Hints”.

You can use SET max_seeks_for_key=*value* as an alternative way to force MySQL to prefer key scans instead of table scans. See Section 5.1.7, “Server System Variables”.

- You can refer to a table within the default database as *tbl_name*, or as *db_name*.*tbl_name* to specify a database explicitly. You can refer to a column as *col_name*, *tbl_name*.*col_name*, or *db_name*.*tbl_name*.*col_name*. You need not specify a *tbl_name* or *db_name*.*tbl_name* prefix for a column reference unless the reference would be ambiguous. See Section 9.2.2, “Identifier Qualifiers”, for examples of ambiguity that require the more explicit column reference forms.

- A table reference can be aliased using *tbl_name* AS *alias_name* or *tbl_name* *alias_name*. These statements are equivalent:

```
SELECT t1.name, t2.salary FROM employee AS t1, info AS t2
  WHERE t1.name = t2.name;

SELECT t1.name, t2.salary FROM employee t1, info t2
  WHERE t1.name = t2.name;
```

- Columns selected for output can be referred to in ORDER BY and GROUP BY clauses using column names, column aliases, or column positions. Column positions are integers and begin with 1:

```
SELECT college, region, seed FROM tournament
  ORDER BY region, seed;

SELECT college, region AS r, seed AS s FROM tournament
  ORDER BY r, s;

SELECT college, region, seed FROM tournament
  ORDER BY 2, 3;
```

To sort in reverse order, add the `DESC` (descending) keyword to the name of the column in the `ORDER BY` clause that you are sorting by. The default is ascending order; this can be specified explicitly using the `ASC` keyword.

If `ORDER BY` occurs within a parenthesized query expression and also is applied in the outer query, the results are undefined and may change in a future MySQL version.

Use of column positions is deprecated because the syntax has been removed from the SQL standard.

- MySQL extends the `GROUP BY` clause so that you can also specify `ASC` and `DESC` after columns named in the clause:

```
SELECT a, COUNT(b) FROM test_table GROUP BY a DESC;
```

- If you use `GROUP BY`, output rows are sorted according to the `GROUP BY` columns as if you had an `ORDER BY` for the same columns. To avoid the overhead of sorting that `GROUP BY` produces, add `ORDER BY NULL`:

```
SELECT a, COUNT(b) FROM test_table GROUP BY a ORDER BY NULL;
```

Relying on implicit `GROUP BY` sorting (that is, sorting in the absence of `ASC` or `DESC` designators) is deprecated. To produce a given sort order, use explicit `ASC` or `DESC` designators for `GROUP BY` columns or provide an `ORDER BY` clause.

- When you use `ORDER BY` or `GROUP BY` to sort a column in a `SELECT`, the server sorts values using only the initial number of bytes indicated by the `max_sort_length` system variable.

- MySQL extends the use of `GROUP BY` to permit selecting fields that are not mentioned in the `GROUP BY` clause. If you are not getting the results that you expect from your query, please read the description of `GROUP BY` found in Section 12.19, “Aggregate Functions”.

- `GROUP BY` permits a `WITH ROLLUP` modifier. See Section 12.19.2, “GROUP BY Modifiers”.

- The `HAVING` clause is applied nearly last, just before items are sent to the client, with no optimization. (`LIMIT` is applied after `HAVING`.)

  The SQL standard requires that `HAVING` must reference only columns in the `GROUP BY` clause or columns used in aggregate functions. However, MySQL supports an extension to this behavior, and permits `HAVING` to refer to columns in the `SELECT` list and columns in outer subqueries as well.

  If the `HAVING` clause refers to a column that is ambiguous, a warning occurs. In the following statement, `col2` is ambiguous because it is used as both an alias and a column name:

```
SELECT COUNT(col1) AS col2 FROM t GROUP BY col2 HAVING col2 = 2;
```

  Preference is given to standard SQL behavior, so if a `HAVING` column name is used both in `GROUP BY` and as an aliased column in the output column list, preference is given to the column in the `GROUP BY` column.

- Do not use `HAVING` for items that should be in the `WHERE` clause. For example, do not write the following:

```
SELECT col_name FROM tbl_name HAVING col_name > 0;
```

Write this instead:

```
SELECT col_name FROM tbl_name WHERE col_name > 0;
```

- The HAVING clause can refer to aggregate functions, which the WHERE clause cannot:

```
SELECT user, MAX(salary) FROM users
  GROUP BY user HAVING MAX(salary) > 10;
```

(This did not work in some older versions of MySQL.)

- MySQL permits duplicate column names. That is, there can be more than one *select_expr* with the same name. This is an extension to standard SQL. Because MySQL also permits GROUP BY and HAVING to refer to *select_expr* values, this can result in an ambiguity:

```
SELECT 12 AS a, a FROM t GROUP BY a;
```

In that statement, both columns have the name a. To ensure that the correct column is used for grouping, use different names for each *select_expr*.

- MySQL resolves unqualified column or alias references in ORDER BY clauses by searching in the *select_expr* values, then in the columns of the tables in the FROM clause. For GROUP BY or HAVING clauses, it searches the FROM clause before searching in the *select_expr* values. (For GROUP BY and HAVING, this differs from the pre-MySQL 5.0 behavior that used the same rules as for ORDER BY.)

- The LIMIT clause can be used to constrain the number of rows returned by the SELECT statement. LIMIT takes one or two numeric arguments, which must both be nonnegative integer constants, with these exceptions:

  - Within prepared statements, LIMIT parameters can be specified using ? placeholder markers.
  - Within stored programs, LIMIT parameters can be specified using integer-valued routine parameters or local variables.

  With two arguments, the first argument specifies the offset of the first row to return, and the second specifies the maximum number of rows to return. The offset of the initial row is 0 (not 1):

```
SELECT * FROM tbl LIMIT 5,10;  # Retrieve rows 6-15
```

To retrieve all rows from a certain offset up to the end of the result set, you can use some large number for the second parameter. This statement retrieves all rows from the 96th row to the last:

```
SELECT * FROM tbl LIMIT 95,18446744073709551615;
```

With one argument, the value specifies the number of rows to return from the beginning of the result set:

```
SELECT * FROM tbl LIMIT 5;     # Retrieve first 5 rows
```

In other words, LIMIT *row_count* is equivalent to LIMIT 0, *row_count*.

For prepared statements, you can use placeholders. The following statements return one row from the tbl table:

```
SET @a=1;
PREPARE STMT FROM 'SELECT * FROM tbl LIMIT ?';
EXECUTE STMT USING @a;
```

The following statements return the second to sixth row from the tbl table:

```
SET @skip=1; SET @numrows=5;
PREPARE STMT FROM 'SELECT * FROM tbl LIMIT ?, ?';
EXECUTE STMT USING @skip, @numrows;
```

For compatibility with PostgreSQL, MySQL also supports the `LIMIT row_count OFFSET offset` syntax.

If `LIMIT` occurs within a parenthesized query expression and also is applied in the outer query, the results are undefined and may change in a future MySQL version.

- A `PROCEDURE` clause names a procedure that should process the data in the result set. For an example, see Section 8.4.2.4, “Using PROCEDURE ANALYSE”, which describes `ANALYSE`, a procedure that can be used to obtain suggestions for optimal column data types that may help reduce table sizes.

  A `PROCEDURE` clause is not permitted in a `UNION` statement.

- The `SELECT ... INTO` form of `SELECT` enables the query result to be written to a file or stored in variables. For more information, see Section 13.2.9.1, “SELECT ... INTO Statement”.

- If you use `FOR UPDATE` with a storage engine that uses page or row locks, rows examined by the query are write-locked until the end of the current transaction. Using `LOCK IN SHARE MODE` sets a shared lock that permits other transactions to read the examined rows but not to update or delete them. See Section 14.7.2.4, “Locking Reads”.

  In addition, you cannot use `FOR UPDATE` as part of the `SELECT` in a statement such as `CREATE TABLE new_table SELECT ... FROM old_table ...`. (If you attempt to do so, the statement is rejected with the error `Can't update table 'old_table' while 'new_table' is being created`.) This is a change in behavior from MySQL 5.5 and earlier, which permitted `CREATE TABLE ... SELECT` statements to make changes in tables other than the table being created.

Following the `SELECT` keyword, you can use a number of modifiers that affect the operation of the statement. `HIGH_PRIORITY`, `STRAIGHT_JOIN`, and modifiers beginning with `SQL_` are MySQL extensions to standard SQL.

- The `ALL` and `DISTINCT` modifiers specify whether duplicate rows should be returned. `ALL` (the default) specifies that all matching rows should be returned, including duplicates. `DISTINCT` specifies removal of duplicate rows from the result set. It is an error to specify both modifiers. `DISTINCTROW` is a synonym for `DISTINCT`.

- `HIGH_PRIORITY` gives the `SELECT` higher priority than a statement that updates a table. You should use this only for queries that are very fast and must be done at once. A `SELECT HIGH_PRIORITY` query that is issued while the table is locked for reading runs even if there is an update statement waiting for the table to be free. This affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).

  `HIGH_PRIORITY` cannot be used with `SELECT` statements that are part of a `UNION`.

- `STRAIGHT_JOIN` forces the optimizer to join the tables in the order in which they are listed in the `FROM` clause. You can use this to speed up a query if the optimizer joins the tables in nonoptimal order. `STRAIGHT_JOIN` also can be used in the *`table_references`* list. See Section 13.2.9.2, “JOIN Clause”.

  `STRAIGHT_JOIN` does not apply to any table that the optimizer treats as a `const` or `system` table. Such a table produces a single row, is read during the optimization phase of query execution, and references to its columns are replaced with the appropriate column values before query execution proceeds. These tables appear first in the query plan displayed by `EXPLAIN`. See Section 8.8.1, “Optimizing Queries with EXPLAIN”. This exception may not apply to `const` or `system` tables that are used on the `NULL`-complemented side of an outer join (that is, the right-side table of a `LEFT JOIN` or the left-side table of a `RIGHT JOIN`.

- `SQL_BIG_RESULT` or `SQL_SMALL_RESULT` can be used with `GROUP BY` or `DISTINCT` to tell the optimizer that the result set has many rows or is small, respectively. For `SQL_BIG_RESULT`,

MySQL directly uses disk-based temporary tables if they are created, and prefers sorting to using a temporary table with a key on the `GROUP BY` elements. For `SQL_SMALL_RESULT`, MySQL uses in-memory temporary tables to store the resulting table instead of using sorting. This should not normally be needed.

- `SQL_BUFFER_RESULT` forces the result to be put into a temporary table. This helps MySQL free the table locks early and helps in cases where it takes a long time to send the result set to the client. This modifier can be used only for top-level `SELECT` statements, not for subqueries or following `UNION`.

- `SQL_CALC_FOUND_ROWS` tells MySQL to calculate how many rows there would be in the result set, disregarding any `LIMIT` clause. The number of rows can then be retrieved with `SELECT FOUND_ROWS()`. See Section 12.16, “Information Functions”.

- The `SQL_CACHE` and `SQL_NO_CACHE` modifiers affect caching of query results in the query cache (see Section 8.10.3, “The MySQL Query Cache”). `SQL_CACHE` tells MySQL to store the result in the query cache if it is cacheable and the value of the `query_cache_type` system variable is `2` or `DEMAND`. With `SQL_NO_CACHE`, the server does not use the query cache. It neither checks the query cache to see whether the result is already cached, nor does it cache the query result. (Due to a limitation in the parser, a space character must precede and follow the `SQL_NO_CACHE` keyword; a nonspace such as a newline causes the server to check the query cache to see whether the result is already cached.)

  These two modifiers are mutually exclusive and an error occurs if they are both specified. Also, these modifiers are not permitted in subqueries (including subqueries in the `FROM` clause), and `SELECT` statements in unions other than the first `SELECT`.

  For views, `SQL_NO_CACHE` applies if it appears in any `SELECT` in the query. For a cacheable query, `SQL_CACHE` applies if it appears in the first `SELECT` of a view referred to by the query.

A `SELECT` from a partitioned table using a storage engine such as `MyISAM` that employs table-level locks locks only those partitions containing rows that match the `SELECT` statement `WHERE` clause. (This does not occur with storage engines such as `InnoDB` that employ row-level locking.) For more information, see Section 19.6.4, “Partitioning and Locking”.

13.2.9.1 SELECT ... INTO Statement

The `SELECT ... INTO` form of `SELECT` enables a query result to be stored in variables or written to a file:

- `SELECT ... INTO` *`var_list`* selects column values and stores them into variables.

- `SELECT ... INTO OUTFILE` writes the selected rows to a file. Column and line terminators can be specified to produce a specific output format.

- `SELECT ... INTO DUMPFILE` writes a single row to a file without any formatting.

A given `SELECT` statement can contain at most one `INTO` clause, although as shown by the `SELECT` syntax description (see Section 13.2.9, “SELECT Statement”), the `INTO` can appear in different positions:

- Before `FROM`. Example:

```
SELECT * INTO @myvar FROM t1;
```

- Before a trailing locking clause. Example:

```
SELECT * FROM t1 INTO @myvar FOR UPDATE;
```

An `INTO` clause should not be used in a nested `SELECT` because such a `SELECT` must return its result to the outer context. There are also constraints on the use of `INTO` within `UNION` statements; see Section 13.2.9.3, “UNION Clause”.

For the `INTO var_list` variant:

- *`var_list`* names a list of one or more variables, each of which can be a user-defined variable, stored procedure or function parameter, or stored program local variable. (Within a prepared `SELECT ... INTO var_list` statement, only user-defined variables are permitted; see Section 13.6.4.2, “Local Variable Scope and Resolution”.)
- The selected values are assigned to the variables. The number of variables must match the number of columns. The query should return a single row. If the query returns no rows, a warning with error code 1329 occurs (`No data`), and the variable values remain unchanged. If the query returns multiple rows, error 1172 occurs (`Result consisted of more than one row`). If it is possible that the statement may retrieve multiple rows, you can use `LIMIT 1` to limit the result set to a single row.

```
SELECT id, data INTO @x, @y FROM test.t1 LIMIT 1;
```

User variable names are not case-sensitive. See Section 9.4, “User-Defined Variables”.

The `SELECT ... INTO OUTFILE 'file_name'` form of `SELECT` writes the selected rows to a file. The file is created on the server host, so you must have the `FILE` privilege to use this syntax. *`file_name`* cannot be an existing file, which among other things prevents files such as `/etc/passwd` and database tables from being modified. The `character_set_filesystem` system variable controls the interpretation of the file name.

The `SELECT ... INTO OUTFILE` statement is intended to enable dumping a table to a text file on the server host. To create the resulting file on some other host, `SELECT ... INTO OUTFILE` normally is unsuitable because there is no way to write a path to the file relative to the server host file system, unless the location of the file on the remote host can be accessed using a network-mapped path on the server host file system.

Alternatively, if the MySQL client software is installed on the remote host, you can use a client command such as `mysql -e "SELECT ..." > file_name` to generate the file on that host.

`SELECT ... INTO OUTFILE` is the complement of `LOAD DATA`. Column values are written converted to the character set specified in the `CHARACTER SET` clause. If no such clause is present, values are dumped using the `binary` character set. In effect, there is no character set conversion. If a result set contains columns in several character sets, so does the output data file, and it may not be possible to reload the file correctly.

The syntax for the *`export_options`* part of the statement consists of the same `FIELDS` and `LINES` clauses that are used with the `LOAD DATA` statement. For information about the `FIELDS` and `LINES` clauses, including their default values and permissible values, see Section 13.2.6, “LOAD DATA Statement”.

`FIELDS ESCAPED BY` controls how to write special characters. If the `FIELDS ESCAPED BY` character is not empty, it is used when necessary to avoid ambiguity as a prefix that precedes following characters on output:

- The `FIELDS ESCAPED BY` character
- The `FIELDS [OPTIONALLY] ENCLOSED BY` character
- The first character of the `FIELDS TERMINATED BY` and `LINES TERMINATED BY` values
- ASCII `NUL` (the zero-valued byte; what is actually written following the escape character is ASCII `0`, not a zero-valued byte)

The `FIELDS TERMINATED BY`, `ENCLOSED BY`, `ESCAPED BY`, or `LINES TERMINATED BY` characters *must* be escaped so that you can read the file back in reliably. ASCII `NUL` is escaped to make it easier to view with some pagers.

The resulting file need not conform to SQL syntax, so nothing else need be escaped.

If the FIELDS ESCAPED BY character is empty, no characters are escaped and NULL is output as NULL, not \N. It is probably not a good idea to specify an empty escape character, particularly if field values in your data contain any of the characters in the list just given.

Here is an example that produces a file in the comma-separated values (CSV) format used by many programs:

```
SELECT a,b,a+b INTO OUTFILE '/tmp/result.txt'
  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"'
  LINES TERMINATED BY '\n'
  FROM test_table;
```

If you use INTO DUMPFILE instead of INTO OUTFILE, MySQL writes only one row into the file, without any column or line termination and without performing any escape processing. This is useful for selecting a BLOB value and storing it in a file.

Note

Any file created by INTO OUTFILE or INTO DUMPFILE is writable by all users on the server host. The reason for this is that the MySQL server cannot create a file that is owned by anyone other than the user under whose account it is running. (You should *never* run mysqld as root for this and other reasons.) The file thus must be world-writable so that you can manipulate its contents.

If the secure_file_priv system variable is set to a nonempty directory name, the file to be written must be located in that directory.

In the context of SELECT ... INTO statements that occur as part of events executed by the Event Scheduler, diagnostics messages (not only errors, but also warnings) are written to the error log, and, on Windows, to the application event log. For additional information, see Section 20.4.5, “Event Scheduler Status”.

13.2.9.2 JOIN Clause

MySQL supports the following JOIN syntax for the *table_references* part of SELECT statements and multiple-table DELETE and UPDATE statements:

```
table_references:
    escaped_table_reference [, escaped_table_reference] ...

escaped_table_reference: {
    table_reference
  | { OJ table_reference }
}

table_reference: {
    table_factor
  | joined_table
}

table_factor: {
    tbl_name [PARTITION (partition_names)]
        [[AS] alias] [index_hint_list]
  | table_subquery [AS] alias
  | ( table_references )
}

joined_table: {
    table_reference [INNER | CROSS] JOIN table_factor [join_specification]
  | table_reference STRAIGHT_JOIN table_factor
  | table_reference STRAIGHT_JOIN table_factor ON search_condition
  | table_reference {LEFT|RIGHT} [OUTER] JOIN table_reference join_specification
  | table_reference NATURAL [{LEFT|RIGHT} [OUTER]] JOIN table_factor
}

join_specification: {
```

```
    ON search_condition
  | USING (join_column_list)
}

join_column_list:
    column_name [, column_name] ...

index_hint_list:
    index_hint [, index_hint] ...

index_hint: {
    USE {INDEX|KEY}
      [FOR {JOIN|ORDER BY|GROUP BY}] ([index_list])
  | {IGNORE|FORCE} {INDEX|KEY}
      [FOR {JOIN|ORDER BY|GROUP BY}] (index_list)
}

index_list:
    index_name [, index_name] ...
```

A table reference is also known as a join expression.

A table reference (when it refers to a partitioned table) may contain a `PARTITION` option, including a list of comma-separated partitions, subpartitions, or both. This option follows the name of the table and precedes any alias declaration. The effect of this option is that rows are selected only from the listed partitions or subpartitions. Any partitions or subpartitions not named in the list are ignored. For more information and examples, see Section 19.5, “Partition Selection”.

The syntax of *`table_factor`* is extended in MySQL in comparison with standard SQL. The standard accepts only *`table_reference`*, not a list of them inside a pair of parentheses.

This is a conservative extension if each comma in a list of *`table_reference`* items is considered as equivalent to an inner join. For example:

```
SELECT * FROM t1 LEFT JOIN (t2, t3, t4)
                 ON (t2.a = t1.a AND t3.b = t1.b AND t4.c = t1.c)
```

is equivalent to:

```
SELECT * FROM t1 LEFT JOIN (t2 CROSS JOIN t3 CROSS JOIN t4)
                 ON (t2.a = t1.a AND t3.b = t1.b AND t4.c = t1.c)
```

In MySQL, `JOIN`, `CROSS JOIN`, and `INNER JOIN` are syntactic equivalents (they can replace each other). In standard SQL, they are not equivalent. `INNER JOIN` is used with an `ON` clause, `CROSS JOIN` is used otherwise.

In general, parentheses can be ignored in join expressions containing only inner join operations. MySQL also supports nested joins. See Section 8.2.1.7, “Nested Join Optimization”.

Index hints can be specified to affect how the MySQL optimizer makes use of indexes. For more information, see Section 8.9.3, “Index Hints”. The `optimizer_switch` system variable is another way to influence optimizer use of indexes. See Section 8.9.2, “Switchable Optimizations”.

The following list describes general factors to take into account when writing joins:

- A table reference can be aliased using *`tbl_name AS alias_name`* or *`tbl_name alias_name`*:

  ```
  SELECT t1.name, t2.salary
    FROM employee AS t1 INNER JOIN info AS t2 ON t1.name = t2.name;

  SELECT t1.name, t2.salary
    FROM employee t1 INNER JOIN info t2 ON t1.name = t2.name;
  ```

- A *`table_subquery`* is also known as a derived table or subquery in the `FROM` clause. See Section 13.2.10.8, “Derived Tables”. Such subqueries *must* include an alias to give the subquery result a table name. A trivial example follows:

```
SELECT * FROM (SELECT 1, 2, 3) AS t1;
```

- The maximum number of tables that can be referenced in a single join is 61.
- `INNER JOIN` and `,` (comma) are semantically equivalent in the absence of a join condition: both produce a Cartesian product between the specified tables (that is, each and every row in the first table is joined to each and every row in the second table).

  However, the precedence of the comma operator is less than that of `INNER JOIN`, `CROSS JOIN`, `LEFT JOIN`, and so on. If you mix comma joins with the other join types when there is a join condition, an error of the form `Unknown column 'col_name' in 'on clause'` may occur. Information about dealing with this problem is given later in this section.
- The *`search_condition`* used with `ON` is any conditional expression of the form that can be used in a `WHERE` clause. Generally, the `ON` clause serves for conditions that specify how to join tables, and the `WHERE` clause restricts which rows to include in the result set.
- If there is no matching row for the right table in the `ON` or `USING` part in a `LEFT JOIN`, a row with all columns set to `NULL` is used for the right table. You can use this fact to find rows in a table that have no counterpart in another table:

  ```
  SELECT left_tbl.*
    FROM left_tbl LEFT JOIN right_tbl ON left_tbl.id = right_tbl.id
    WHERE right_tbl.id IS NULL;
  ```

  This example finds all rows in `left_tbl` with an `id` value that is not present in `right_tbl` (that is, all rows in `left_tbl` with no corresponding row in `right_tbl`). See Section 8.2.1.8, “Outer Join Optimization”.
- The `USING(`*`join_column_list`*`)` clause names a list of columns that must exist in both tables. If tables `a` and `b` both contain columns `c1`, `c2`, and `c3`, the following join compares corresponding columns from the two tables:

  ```
  a LEFT JOIN b USING (c1, c2, c3)
  ```
- The `NATURAL [LEFT] JOIN` of two tables is defined to be semantically equivalent to an `INNER JOIN` or a `LEFT JOIN` with a `USING` clause that names all columns that exist in both tables.
- `RIGHT JOIN` works analogously to `LEFT JOIN`. To keep code portable across databases, it is recommended that you use `LEFT JOIN` instead of `RIGHT JOIN`.
- The `{ OJ ... }` syntax shown in the join syntax description exists only for compatibility with ODBC. The curly braces in the syntax should be written literally; they are not metasyntax as used elsewhere in syntax descriptions.

  ```
  SELECT left_tbl.*
      FROM { OJ left_tbl LEFT OUTER JOIN right_tbl
             ON left_tbl.id = right_tbl.id }
      WHERE right_tbl.id IS NULL;
  ```

  You can use other types of joins within `{ OJ ... }`, such as `INNER JOIN` or `RIGHT OUTER JOIN`. This helps with compatibility with some third-party applications, but is not official ODBC syntax.
- `STRAIGHT_JOIN` is similar to `JOIN`, except that the left table is always read before the right table. This can be used for those (few) cases for which the join optimizer processes the tables in a suboptimal order.

Some join examples:

```
SELECT * FROM table1, table2;

SELECT * FROM table1 INNER JOIN table2 ON table1.id = table2.id;
```

```
SELECT * FROM table1 LEFT JOIN table2 ON table1.id = table2.id;

SELECT * FROM table1 LEFT JOIN table2 USING (id);

SELECT * FROM table1 LEFT JOIN table2 ON table1.id = table2.id
  LEFT JOIN table3 ON table2.id = table3.id;
```

Natural joins and joins with `USING`, including outer join variants, are processed according to the SQL:2003 standard:

- Redundant columns of a `NATURAL` join do not appear. Consider this set of statements:

```
CREATE TABLE t1 (i INT, j INT);
CREATE TABLE t2 (k INT, j INT);
INSERT INTO t1 VALUES(1, 1);
INSERT INTO t2 VALUES(1, 1);
SELECT * FROM t1 NATURAL JOIN t2;
SELECT * FROM t1 JOIN t2 USING (j);
```

  In the first `SELECT` statement, column `j` appears in both tables and thus becomes a join column, so, according to standard SQL, it should appear only once in the output, not twice. Similarly, in the second SELECT statement, column `j` is named in the `USING` clause and should appear only once in the output, not twice.

  Thus, the statements produce this output:

```
+------+------+------+
| j    | i    | k    |
+------+------+------+
|    1 |    1 |    1 |
+------+------+------+
+------+------+------+
| j    | i    | k    |
+------+------+------+
|    1 |    1 |    1 |
+------+------+------+
```

  Redundant column elimination and column ordering occurs according to standard SQL, producing this display order:

  - First, coalesced common columns of the two joined tables, in the order in which they occur in the first table
  - Second, columns unique to the first table, in order in which they occur in that table
  - Third, columns unique to the second table, in order in which they occur in that table

  The single result column that replaces two common columns is defined using the coalesce operation. That is, for two `t1.a` and `t2.a` the resulting single join column `a` is defined as `a = COALESCE(t1.a, t2.a)`, where:

```
COALESCE(x, y) = (CASE WHEN x IS NOT NULL THEN x ELSE y END)
```

  If the join operation is any other join, the result columns of the join consist of the concatenation of all columns of the joined tables.

  A consequence of the definition of coalesced columns is that, for outer joins, the coalesced column contains the value of the non-`NULL` column if one of the two columns is always `NULL`. If neither or both columns are `NULL`, both common columns have the same value, so it doesn't matter which one is chosen as the value of the coalesced column. A simple way to interpret this is to consider that a coalesced column of an outer join is represented by the common column of the inner table of a `JOIN`. Suppose that the tables `t1(a, b)` and `t2(a, c)` have the following contents:

```
t1    t2
----  ----
```

```
1 x   2 z
2 y   3 w
```

Then, for this join, column `a` contains the values of `t1.a`:

```
mysql> SELECT * FROM t1 NATURAL LEFT JOIN t2;
+------+------+------+
| a    | b    | c    |
+------+------+------+
|    1 | x    | NULL |
|    2 | y    | z    |
+------+------+------+
```

By contrast, for this join, column `a` contains the values of `t2.a`.

```
mysql> SELECT * FROM t1 NATURAL RIGHT JOIN t2;
+------+------+------+
| a    | c    | b    |
+------+------+------+
|    2 | z    | y    |
|    3 | w    | NULL |
+------+------+------+
```

Compare those results to the otherwise equivalent queries with `JOIN ... ON`:

```
mysql> SELECT * FROM t1 LEFT JOIN t2 ON (t1.a = t2.a);
+------+------+------+------+
| a    | b    | a    | c    |
+------+------+------+------+
|    1 | x    | NULL | NULL |
|    2 | y    |    2 | z    |
+------+------+------+------+
```

```
mysql> SELECT * FROM t1 RIGHT JOIN t2 ON (t1.a = t2.a);
+------+------+------+------+
| a    | b    | a    | c    |
+------+------+------+------+
|    2 | y    |    2 | z    |
| NULL | NULL |    3 | w    |
+------+------+------+------+
```

- A `USING` clause can be rewritten as an `ON` clause that compares corresponding columns. However, although `USING` and `ON` are similar, they are not quite the same. Consider the following two queries:

  ```
  a LEFT JOIN b USING (c1, c2, c3)
  a LEFT JOIN b ON a.c1 = b.c1 AND a.c2 = b.c2 AND a.c3 = b.c3
  ```

  With respect to determining which rows satisfy the join condition, both joins are semantically identical.

  With respect to determining which columns to display for `SELECT *` expansion, the two joins are not semantically identical. The `USING` join selects the coalesced value of corresponding columns, whereas the `ON` join selects all columns from all tables. For the `USING` join, `SELECT *` selects these values:

  ```
  COALESCE(a.c1, b.c1), COALESCE(a.c2, b.c2), COALESCE(a.c3, b.c3)
  ```

  For the `ON` join, `SELECT *` selects these values:

  ```
  a.c1, a.c2, a.c3, b.c1, b.c2, b.c3
  ```

  With an inner join, `COALESCE(a.c1, b.c1)` is the same as either `a.c1` or `b.c1` because both columns have the same value. With an outer join (such as `LEFT JOIN`), one of the two columns can be `NULL`. That column is omitted from the result.

- An `ON` clause can refer only to its operands.

  Example:

```
CREATE TABLE t1 (i1 INT);
CREATE TABLE t2 (i2 INT);
CREATE TABLE t3 (i3 INT);
SELECT * FROM t1 JOIN t2 ON (i1 = i3) JOIN t3;
```

The statement fails with an `Unknown column 'i3' in 'on clause'` error because `i3` is a column in `t3`, which is not an operand of the `ON` clause. To enable the join to be processed, rewrite the statement as follows:

```
SELECT * FROM t1 JOIN t2 JOIN t3 ON (i1 = i3);
```

- `JOIN` has higher precedence than the comma operator (`,`), so the join expression `t1, t2 JOIN t3` is interpreted as `(t1, (t2 JOIN t3))`, not as `((t1, t2) JOIN t3)`. This affects statements that use an `ON` clause because that clause can refer only to columns in the operands of the join, and the precedence affects interpretation of what those operands are.

  Example:

  ```
  CREATE TABLE t1 (i1 INT, j1 INT);
  CREATE TABLE t2 (i2 INT, j2 INT);
  CREATE TABLE t3 (i3 INT, j3 INT);
  INSERT INTO t1 VALUES(1, 1);
  INSERT INTO t2 VALUES(1, 1);
  INSERT INTO t3 VALUES(1, 1);
  SELECT * FROM t1, t2 JOIN t3 ON (t1.i1 = t3.i3);
  ```

  The `JOIN` takes precedence over the comma operator, so the operands for the `ON` clause are `t2` and `t3`. Because `t1.i1` is not a column in either of the operands, the result is an `Unknown column 't1.i1' in 'on clause'` error.

  To enable the join to be processed, use either of these strategies:

  - Group the first two tables explicitly with parentheses so that the operands for the `ON` clause are `(t1, t2)` and `t3`:

    ```
    SELECT * FROM (t1, t2) JOIN t3 ON (t1.i1 = t3.i3);
    ```

  - Avoid the use of the comma operator and use `JOIN` instead:

    ```
    SELECT * FROM t1 JOIN t2 JOIN t3 ON (t1.i1 = t3.i3);
    ```

  The same precedence interpretation also applies to statements that mix the comma operator with `INNER JOIN`, `CROSS JOIN`, `LEFT JOIN`, and `RIGHT JOIN`, all of which have higher precedence than the comma operator.

- A MySQL extension compared to the SQL:2003 standard is that MySQL permits you to qualify the common (coalesced) columns of `NATURAL` or `USING` joins, whereas the standard disallows that.

13.2.9.3 UNION Clause

```
SELECT ...
UNION [ALL | DISTINCT] SELECT ...
[UNION [ALL | DISTINCT] SELECT ...]
```

`UNION` combines the result from multiple `SELECT` statements into a single result set.

- Result Set Column Names and Data Types
- UNION DISTINCT and UNION ALL
- ORDER BY and LIMIT in Unions
- UNION Restrictions

Result Set Column Names and Data Types

The result set column names are taken from the column names of the first `SELECT` statement.

Selected columns listed in corresponding positions of each `SELECT` statement should have the same data type. For example, the first column selected by the first statement should have the same type as the first column selected by the other statements. If the data types of corresponding `SELECT` columns do not match, the types and lengths of the columns in the `UNION` result take into account the values retrieved by all the `SELECT` statements. For example, consider the following, where the column length is not constrained to the length of the value from the first `SELECT`:

```
mysql> SELECT REPEAT('a',1) UNION SELECT REPEAT('b',20);
+----------------------+
| REPEAT('a',1)        |
+----------------------+
| a                    |
| bbbbbbbbbbbbbbbbbbbb |
+----------------------+
```

UNION DISTINCT and UNION ALL

By default, duplicate rows are removed from `UNION` results. The optional `DISTINCT` keyword has the same effect but makes it explicit. With the optional `ALL` keyword, duplicate-row removal does not occur and the result includes all matching rows from all the `SELECT` statements.

You can mix `UNION ALL` and `UNION DISTINCT` in the same query. Mixed `UNION` types are treated such that a `DISTINCT` union overrides any `ALL` union to its left. A `DISTINCT` union can be produced explicitly by using `UNION DISTINCT` or implicitly by using `UNION` with no following `DISTINCT` or `ALL` keyword.

ORDER BY and LIMIT in Unions

To apply an `ORDER BY` or `LIMIT` clause to an individual `SELECT`, parenthesize the `SELECT` and place the clause inside the parentheses:

```
(SELECT a FROM t1 WHERE a=10 AND B=1 ORDER BY a LIMIT 10)
UNION
(SELECT a FROM t2 WHERE a=11 AND B=2 ORDER BY a LIMIT 10);
```

Use of `ORDER BY` for individual `SELECT` statements implies nothing about the order in which the rows appear in the final result because `UNION` by default produces an unordered set of rows. Therefore, `ORDER BY` in this context typically is used in conjunction with `LIMIT`, to determine the subset of the selected rows to retrieve for the `SELECT`, even though it does not necessarily affect the order of those rows in the final `UNION` result. If `ORDER BY` appears without `LIMIT` in a `SELECT`, it is optimized away because it has no effect.

To use an `ORDER BY` or `LIMIT` clause to sort or limit the entire `UNION` result, parenthesize the individual `SELECT` statements and place the `ORDER BY` or `LIMIT` after the last one:

```
(SELECT a FROM t1 WHERE a=10 AND B=1)
UNION
(SELECT a FROM t2 WHERE a=11 AND B=2)
ORDER BY a LIMIT 10;
```

A statement without parentheses is equivalent to one parenthesized as just shown.

This kind of `ORDER BY` cannot use column references that include a table name (that is, names in *`tbl_name.col_name`* format). Instead, provide a column alias in the first `SELECT` statement and refer to the alias in the `ORDER BY`. (Alternatively, refer to the column in the `ORDER BY` using its column position. However, use of column positions is deprecated.)

Also, if a column to be sorted is aliased, the `ORDER BY` clause *must* refer to the alias, not the column name. The first of the following statements is permitted, but the second fails with an `Unknown column 'a' in 'order clause'` error:

```
(SELECT a AS b FROM t) UNION (SELECT ...) ORDER BY b;
(SELECT a AS b FROM t) UNION (SELECT ...) ORDER BY a;
```

To cause rows in a `UNION` result to consist of the sets of rows retrieved by each `SELECT` one after the other, select an additional column in each `SELECT` to use as a sort column and add an `ORDER BY` that sorts on that column following the last `SELECT`:

```
(SELECT 1 AS sort_col, col1a, col1b, ... FROM t1)
UNION
(SELECT 2, col2a, col2b, ... FROM t2) ORDER BY sort_col;
```

To additionally maintain sort order within individual `SELECT` results, add a secondary column to the `ORDER BY` clause:

```
(SELECT 1 AS sort_col, col1a, col1b, ... FROM t1)
UNION
(SELECT 2, col2a, col2b, ... FROM t2) ORDER BY sort_col, col1a;
```

Use of an additional column also enables you to determine which `SELECT` each row comes from. Extra columns can provide other identifying information as well, such as a string that indicates a table name.

UNION Restrictions

In a `UNION`, the `SELECT` statements are normal select statements, but with the following restrictions:

- `HIGH_PRIORITY` in the first `SELECT` has no effect. `HIGH_PRIORITY` in any subsequent `SELECT` produces a syntax error.
- Only the last `SELECT` statement can use an `INTO` clause. However, the entire `UNION` result is written to the `INTO` output destination.

13.2.10 Subqueries

A subquery is a `SELECT` statement within another statement.

All subquery forms and operations that the SQL standard requires are supported, as well as a few features that are MySQL-specific.

Here is an example of a subquery:

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

In this example, `SELECT * FROM t1 ...` is the *outer query* (or *outer statement*), and `(SELECT column1 FROM t2)` is the *subquery*. We say that the subquery is *nested* within the outer query, and in fact it is possible to nest subqueries within other subqueries, to a considerable depth. A subquery must always appear within parentheses.

The main advantages of subqueries are:

- They allow queries that are *structured* so that it is possible to isolate each part of a statement.
- They provide alternative ways to perform operations that would otherwise require complex joins and unions.
- Many people find subqueries more readable than complex joins or unions. Indeed, it was the innovation of subqueries that gave people the original idea of calling the early SQL “Structured Query Language.”

Here is an example statement that shows the major points about subquery syntax as specified by the SQL standard and supported in MySQL:

```
DELETE FROM t1
```

```
WHERE s11 > ANY
 (SELECT COUNT(*) /* no hint */ FROM t2
  WHERE NOT EXISTS
   (SELECT * FROM t3
    WHERE ROW(5*t2.s1,77)=
     (SELECT 50,11*s1 FROM t4 UNION SELECT 50,77 FROM
      (SELECT * FROM t5) AS t5)));
```

A subquery can return a scalar (a single value), a single row, a single column, or a table (one or more rows of one or more columns). These are called scalar, column, row, and table subqueries. Subqueries that return a particular kind of result often can be used only in certain contexts, as described in the following sections.

There are few restrictions on the type of statements in which subqueries can be used. A subquery can contain many of the keywords or clauses that an ordinary `SELECT` can contain: `DISTINCT`, `GROUP BY`, `ORDER BY`, `LIMIT`, joins, index hints, `UNION` constructs, comments, functions, and so on.

A subquery's outer statement can be any one of: `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `SET`, or `DO`.

In MySQL, you cannot modify a table and select from the same table in a subquery. This applies to statements such as `DELETE`, `INSERT`, `REPLACE`, `UPDATE`, and (because subqueries can be used in the `SET` clause) `LOAD DATA`.

For information about how the optimizer handles subqueries, see Section 8.2.2, “Optimizing Subqueries and Derived Tables”. For a discussion of restrictions on subquery use, including performance issues for certain forms of subquery syntax, see Section 13.2.10.12, “Restrictions on Subqueries”.

13.2.10.1 The Subquery as Scalar Operand

In its simplest form, a subquery is a scalar subquery that returns a single value. A scalar subquery is a simple operand, and you can use it almost anywhere a single column value or literal is legal, and you can expect it to have those characteristics that all operands have: a data type, a length, an indication that it can be `NULL`, and so on. For example:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5) NOT NULL);
INSERT INTO t1 VALUES(100, 'abcde');
SELECT (SELECT s2 FROM t1);
```

The subquery in this `SELECT` returns a single value (`'abcde'`) that has a data type of `CHAR`, a length of 5, a character set and collation equal to the defaults in effect at `CREATE TABLE` time, and an indication that the value in the column can be `NULL`. Nullability of the value selected by a scalar subquery is not copied because if the subquery result is empty, the result is `NULL`. For the subquery just shown, if `t1` were empty, the result would be `NULL` even though `s2` is `NOT NULL`.

There are a few contexts in which a scalar subquery cannot be used. If a statement permits only a literal value, you cannot use a subquery. For example, `LIMIT` requires literal integer arguments, and `LOAD DATA` requires a literal string file name. You cannot use subqueries to supply these values.

When you see examples in the following sections that contain the rather spartan construct `(SELECT column1 FROM t1)`, imagine that your own code contains much more diverse and complex constructions.

Suppose that we make two tables:

```
CREATE TABLE t1 (s1 INT);
INSERT INTO t1 VALUES (1);
CREATE TABLE t2 (s1 INT);
INSERT INTO t2 VALUES (2);
```

Then perform a `SELECT`:

```
SELECT (SELECT s1 FROM t2) FROM t1;
```

The result is `2` because there is a row in `t2` containing a column `s1` that has a value of `2`.

A scalar subquery can be part of an expression, but remember the parentheses, even if the subquery is an operand that provides an argument for a function. For example:

```
SELECT UPPER((SELECT s1 FROM t1)) FROM t2;
```

13.2.10.2 Comparisons Using Subqueries

The most common use of a subquery is in the form:

```
non_subquery_operand comparison_operator (subquery)
```

Where *`comparison_operator`* is one of these operators:

```
=  >  <  >=  <=  <>  !=  <=>
```

For example:

```
... WHERE 'a' = (SELECT column1 FROM t1)
```

MySQL also permits this construct:

```
non_subquery_operand LIKE (subquery)
```

At one time the only legal place for a subquery was on the right side of a comparison, and you might still find some old DBMSs that insist on this.

Here is an example of a common-form subquery comparison that you cannot do with a join. It finds all the rows in table `t1` for which the `column1` value is equal to a maximum value in table `t2`:

```
SELECT * FROM t1
  WHERE column1 = (SELECT MAX(column2) FROM t2);
```

Here is another example, which again is impossible with a join because it involves aggregating for one of the tables. It finds all rows in table `t1` containing a value that occurs twice in a given column:

```
SELECT * FROM t1 AS t
  WHERE 2 = (SELECT COUNT(*) FROM t1 WHERE t1.id = t.id);
```

For a comparison of the subquery to a scalar, the subquery must return a scalar. For a comparison of the subquery to a row constructor, the subquery must be a row subquery that returns a row with the same number of values as the row constructor. See Section 13.2.10.5, “Row Subqueries”.

13.2.10.3 Subqueries with ANY, IN, or SOME

Syntax:

```
operand comparison_operator ANY (subquery)
operand IN (subquery)
operand comparison_operator SOME (subquery)
```

Where *`comparison_operator`* is one of these operators:

```
=  >  <  >=  <=  <>  !=
```

The `ANY` keyword, which must follow a comparison operator, means “return `TRUE` if the comparison is `TRUE` for `ANY` of the values in the column that the subquery returns.” For example:

```
SELECT s1 FROM t1 WHERE s1 > ANY (SELECT s1 FROM t2);
```

Suppose that there is a row in table `t1` containing `(10)`. The expression is `TRUE` if table `t2` contains `(21,14,7)` because there is a value `7` in `t2` that is less than `10`. The expression is `FALSE` if table

`t2` contains `(20,10)`, or if table `t2` is empty. The expression is *unknown* (that is, `NULL`) if table `t2` contains `(NULL,NULL,NULL)`.

When used with a subquery, the word `IN` is an alias for `= ANY`. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 = ANY (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 IN    (SELECT s1 FROM t2);
```

`IN` and `= ANY` are not synonyms when used with an expression list. `IN` can take an expression list, but `= ANY` cannot. See Section 12.4.2, “Comparison Functions and Operators”.

`NOT IN` is not an alias for `<> ANY`, but for `<> ALL`. See Section 13.2.10.4, “Subqueries with ALL”.

The word `SOME` is an alias for `ANY`. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 <> ANY  (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 <> SOME (SELECT s1 FROM t2);
```

Use of the word `SOME` is rare, but this example shows why it might be useful. To most people, the English phrase “a is not equal to any b” means “there is no b which is equal to a,” but that is not what is meant by the SQL syntax. The syntax means “there is some b to which a is not equal.” Using `<> SOME` instead helps ensure that everyone understands the true meaning of the query.

13.2.10.4 Subqueries with ALL

Syntax:

```
operand comparison_operator ALL (subquery)
```

The word `ALL`, which must follow a comparison operator, means “return `TRUE` if the comparison is `TRUE` for `ALL` of the values in the column that the subquery returns.” For example:

```
SELECT s1 FROM t1 WHERE s1 > ALL (SELECT s1 FROM t2);
```

Suppose that there is a row in table `t1` containing `(10)`. The expression is `TRUE` if table `t2` contains `(-5,0,+5)` because `10` is greater than all three values in `t2`. The expression is `FALSE` if table `t2` contains `(12,6,NULL,-100)` because there is a single value `12` in table `t2` that is greater than `10`. The expression is *unknown* (that is, `NULL`) if table `t2` contains `(0,NULL,1)`.

Finally, the expression is `TRUE` if table `t2` is empty. So, the following expression is `TRUE` when table `t2` is empty:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT s1 FROM t2);
```

But this expression is `NULL` when table `t2` is empty:

```
SELECT * FROM t1 WHERE 1 > (SELECT s1 FROM t2);
```

In addition, the following expression is `NULL` when table `t2` is empty:

```
SELECT * FROM t1 WHERE 1 > ALL (SELECT MAX(s1) FROM t2);
```

In general, *tables containing `NULL` values* and *empty tables* are “edge cases.” When writing subqueries, always consider whether you have taken those two possibilities into account.

`NOT IN` is an alias for `<> ALL`. Thus, these two statements are the same:

```
SELECT s1 FROM t1 WHERE s1 <> ALL (SELECT s1 FROM t2);
SELECT s1 FROM t1 WHERE s1 NOT IN (SELECT s1 FROM t2);
```

13.2.10.5 Row Subqueries

Scalar or column subqueries return a single value or a column of values. A *row subquery* is a subquery variant that returns a single row and can thus return more than one column value. Legal operators for row subquery comparisons are:

```
=  >  <  >=  <=  <>  !=  <=>
```

Here are two examples:

```
SELECT * FROM t1
  WHERE (col1,col2) = (SELECT col3, col4 FROM t2 WHERE id = 10);
SELECT * FROM t1
  WHERE ROW(col1,col2) = (SELECT col3, col4 FROM t2 WHERE id = 10);
```

For both queries, if the table `t2` contains a single row with `id = 10`, the subquery returns a single row. If this row has `col3` and `col4` values equal to the `col1` and `col2` values of any rows in `t1`, the `WHERE` expression is `TRUE` and each query returns those `t1` rows. If the `t2` row `col3` and `col4` values are not equal the `col1` and `col2` values of any `t1` row, the expression is `FALSE` and the query returns an empty result set. The expression is *unknown* (that is, `NULL`) if the subquery produces no rows. An error occurs if the subquery produces multiple rows because a row subquery can return at most one row.

For information about how each operator works for row comparisons, see Section 12.4.2, “Comparison Functions and Operators”.

The expressions `(1,2)` and `ROW(1,2)` are sometimes called *row constructors*. The two are equivalent. The row constructor and the row returned by the subquery must contain the same number of values.

A row constructor is used for comparisons with subqueries that return two or more columns. When a subquery returns a single column, this is regarded as a scalar value and not as a row, so a row constructor cannot be used with a subquery that does not return at least two columns. Thus, the following query fails with a syntax error:

```
SELECT * FROM t1 WHERE ROW(1) = (SELECT column1 FROM t2)
```

Row constructors are legal in other contexts. For example, the following two statements are semantically equivalent (and are handled in the same way by the optimizer):

```
SELECT * FROM t1 WHERE (column1,column2) = (1,1);
SELECT * FROM t1 WHERE column1 = 1 AND column2 = 1;
```

The following query answers the request, “find all rows in table `t1` that also exist in table `t2`”:

```
SELECT column1,column2,column3
  FROM t1
  WHERE (column1,column2,column3) IN
         (SELECT column1,column2,column3 FROM t2);
```

For more information about the optimizer and row constructors, see Section 8.2.1.18, “Row Constructor Expression Optimization”

13.2.10.6 Subqueries with EXISTS or NOT EXISTS

If a subquery returns any rows at all, `EXISTS subquery` is `TRUE`, and `NOT EXISTS subquery` is `FALSE`. For example:

```
SELECT column1 FROM t1 WHERE EXISTS (SELECT * FROM t2);
```

Traditionally, an `EXISTS` subquery starts with `SELECT *`, but it could begin with `SELECT 5` or `SELECT column1` or anything at all. MySQL ignores the `SELECT` list in such a subquery, so it makes no difference.

For the preceding example, if `t2` contains any rows, even rows with nothing but `NULL` values, the `EXISTS` condition is `TRUE`. This is actually an unlikely example because a `[NOT] EXISTS` subquery almost always contains correlations. Here are some more realistic examples:

- What kind of store is present in one or more cities?

```
SELECT DISTINCT store_type FROM stores
  WHERE EXISTS (SELECT * FROM cities_stores
                WHERE cities_stores.store_type = stores.store_type);
```

- What kind of store is present in no cities?

```
SELECT DISTINCT store_type FROM stores
  WHERE NOT EXISTS (SELECT * FROM cities_stores
                    WHERE cities_stores.store_type = stores.store_type);
```

- What kind of store is present in all cities?

```
SELECT DISTINCT store_type FROM stores s1
  WHERE NOT EXISTS (
    SELECT * FROM cities WHERE NOT EXISTS (
      SELECT * FROM cities_stores
       WHERE cities_stores.city = cities.city
       AND cities_stores.store_type = stores.store_type));
```

The last example is a double-nested `NOT EXISTS` query. That is, it has a `NOT EXISTS` clause within a `NOT EXISTS` clause. Formally, it answers the question “does a city exist with a store that is not in `Stores`”? But it is easier to say that a nested `NOT EXISTS` answers the question “is *x* `TRUE` for all *y*?”

13.2.10.7 Correlated Subqueries

A *correlated subquery* is a subquery that contains a reference to a table that also appears in the outer query. For example:

```
SELECT * FROM t1
  WHERE column1 = ANY (SELECT column1 FROM t2
                       WHERE t2.column2 = t1.column2);
```

Notice that the subquery contains a reference to a column of `t1`, even though the subquery's `FROM` clause does not mention a table `t1`. So, MySQL looks outside the subquery, and finds `t1` in the outer query.

Suppose that table `t1` contains a row where `column1 = 5` and `column2 = 6`; meanwhile, table `t2` contains a row where `column1 = 5` and `column2 = 7`. The simple expression `... WHERE column1 = ANY (SELECT column1 FROM t2)` would be `TRUE`, but in this example, the `WHERE` clause within the subquery is `FALSE` (because `(5,6)` is not equal to `(5,7)`), so the expression as a whole is `FALSE`.

Scoping rule: MySQL evaluates from inside to outside. For example:

```
SELECT column1 FROM t1 AS x
  WHERE x.column1 = (SELECT column1 FROM t2 AS x
    WHERE x.column1 = (SELECT column1 FROM t3
      WHERE x.column2 = t3.column1));
```

In this statement, `x.column2` must be a column in table `t2` because `SELECT column1 FROM t2 AS x ...` renames `t2`. It is not a column in table `t1` because `SELECT column1 FROM t1 ...` is an outer query that is *farther out*.

For subqueries in `HAVING` or `ORDER BY` clauses, MySQL also looks for column names in the outer select list.

For certain cases, a correlated subquery is optimized. For example:

```
val IN (SELECT key_val FROM tbl_name WHERE correlated_condition)
```

Otherwise, they are inefficient and likely to be slow. Rewriting the query as a join might improve performance.

Aggregate functions in correlated subqueries may contain outer references, provided the function contains nothing but outer references, and provided the function is not contained in another function or expression.

13.2.10.8 Derived Tables

A derived table is an expression that generates a table within the scope of a query `FROM` clause. For example, a subquery in a `SELECT` statement `FROM` clause is a derived table:

```
SELECT ... FROM (subquery) [AS] tbl_name ...
```

The `[AS] tbl_name` clause is mandatory because every table in a `FROM` clause must have a name. Any columns in the derived table must have unique names.

For the sake of illustration, assume that you have this table:

```
CREATE TABLE t1 (s1 INT, s2 CHAR(5), s3 FLOAT);
```

Here is how to use a subquery in the `FROM` clause, using the example table:

```
INSERT INTO t1 VALUES (1,'1',1.0);
INSERT INTO t1 VALUES (2,'2',2.0);
SELECT sb1,sb2,sb3
  FROM (SELECT s1 AS sb1, s2 AS sb2, s3*2 AS sb3 FROM t1) AS sb
  WHERE sb1 > 1;
```

Result:

```
+------+------+------+
| sb1  | sb2  | sb3  |
+------+------+------+
|    2 | 2    |    4 |
+------+------+------+
```

Here is another example: Suppose that you want to know the average of a set of sums for a grouped table. This does not work:

```
SELECT AVG(SUM(column1)) FROM t1 GROUP BY column1;
```

However, this query provides the desired information:

```
SELECT AVG(sum_column1)
  FROM (SELECT SUM(column1) AS sum_column1
        FROM t1 GROUP BY column1) AS t1;
```

Notice that the column name used within the subquery (`sum_column1`) is recognized in the outer query.

A derived table can return a scalar, column, row, or table.

Derived tables are subject to these restrictions:

- A derived table cannot be a correlated subquery.
- A derived table cannot contain references to other tables of the same `SELECT`.
- A derived table cannot contain outer references. This is a MySQL restriction, not a restriction of the SQL standard.

The optimizer determines information about derived tables in such a way that `EXPLAIN` does not need to materialize them. See Section 8.2.2.4, “Optimizing Derived Tables”.

It is possible under certain circumstances that using `EXPLAIN SELECT` modifies table data. This can occur if the outer query accesses any tables and an inner query invokes a stored function that changes one or more rows of a table. Suppose that there are two tables `t1` and `t2` in database `d1`, and a stored function `f1` that modifies `t2`, created as shown here:

```
CREATE DATABASE d1;
USE d1;
CREATE TABLE t1 (c1 INT);
CREATE TABLE t2 (c1 INT);
CREATE FUNCTION f1(p1 INT) RETURNS INT
  BEGIN
    INSERT INTO t2 VALUES (p1);
    RETURN p1;
  END;
```

Referencing the function directly in an `EXPLAIN SELECT` has no effect on `t2`, as shown here:

```
mysql> SELECT * FROM t2;
Empty set (0.02 sec)

mysql> EXPLAIN SELECT f1(5)\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: NULL
         type: NULL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
        Extra: No tables used
1 row in set (0.01 sec)

mysql> SELECT * FROM t2;
Empty set (0.01 sec)
```

This is because the `SELECT` statement did not reference any tables, as can be seen in the `table` and `Extra` columns of the output. This is also true of the following nested `SELECT`:

```
mysql> EXPLAIN SELECT NOW() AS a1, (SELECT f1(5)) AS a2\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: NULL
         type: NULL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
     filtered: NULL
        Extra: No tables used
1 row in set, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-------+------+-----------------------------------------+
| Level | Code | Message                                 |
+-------+------+-----------------------------------------+
| Note  | 1249 | Select 2 was reduced during optimization |
+-------+------+-----------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM t2;
Empty set (0.00 sec)
```

However, if the outer `SELECT` references any tables, the optimizer executes the statement in the subquery as well, with the result that `t2` is modified:

```
mysql> EXPLAIN SELECT * FROM t1 AS a1, (SELECT f1(5)) AS a2\G
*************************** 1. row ***************************
           id: 1
  select_type: PRIMARY
        table: <derived2>
         type: system
possible_keys: NULL
```

```
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1
        Extra: NULL
*************************** 2. row ***************************
           id: 1
  select_type: PRIMARY
        table: a1
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 1
        Extra: NULL
*************************** 3. row ***************************
           id: 2
  select_type: DERIVED
        table: NULL
         type: NULL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: NULL
        Extra: No tables used
3 rows in set (0.00 sec)

mysql> SELECT * FROM t2;
+------+
| c1   |
+------+
|    5 |
+------+
1 row in set (0.00 sec)
```

This also means that an `EXPLAIN SELECT` statement such as the one shown here may take a long time to execute because the `BENCHMARK()` function is executed once for each row in `t1`:

```
EXPLAIN SELECT * FROM t1 AS a1, (SELECT BENCHMARK(1000000, MD5(NOW())));
```

13.2.10.9 Subquery Errors

There are some errors that apply only to subqueries. This section describes them.

- Unsupported subquery syntax:

  ```
  ERROR 1235 (ER_NOT_SUPPORTED_YET)
  SQLSTATE = 42000
  Message = "This version of MySQL doesn't yet support
  'LIMIT & IN/ALL/ANY/SOME subquery'"
  ```

  This means that MySQL does not support statements of the following form:

  ```
  SELECT * FROM t1 WHERE s1 IN (SELECT s2 FROM t2 ORDER BY s1 LIMIT 1)
  ```

- Incorrect number of columns from subquery:

  ```
  ERROR 1241 (ER_OPERAND_COL)
  SQLSTATE = 21000
  Message = "Operand should contain 1 column(s)"
  ```

  This error occurs in cases like this:

  ```
  SELECT (SELECT column1, column2 FROM t2) FROM t1;
  ```

  You may use a subquery that returns multiple columns, if the purpose is row comparison. In other contexts, the subquery must be a scalar operand. See Section 13.2.10.5, “Row Subqueries”.

- Incorrect number of rows from subquery:

```
ERROR 1242 (ER_SUBSELECT_NO_1_ROW)
SQLSTATE = 21000
Message = "Subquery returns more than 1 row"
```

  This error occurs for statements where the subquery must return at most one row but returns multiple rows. Consider the following example:

```
SELECT * FROM t1 WHERE column1 = (SELECT column1 FROM t2);
```

  If `SELECT column1 FROM t2` returns just one row, the previous query works. If the subquery returns more than one row, error 1242 occurs. In that case, the query should be rewritten as:

```
SELECT * FROM t1 WHERE column1 = ANY (SELECT column1 FROM t2);
```

- Incorrectly used table in subquery:

```
Error 1093 (ER_UPDATE_TABLE_USED)
SQLSTATE = HY000
Message = "You can't specify target table 'x'
for update in FROM clause"
```

  This error occurs in cases such as the following, which attempts to modify a table and select from the same table in the subquery:

```
UPDATE t1 SET column2 = (SELECT MAX(column1) FROM t1);
```

  You can use a subquery for assignment within an `UPDATE` statement because subqueries are legal in `UPDATE` and `DELETE` statements as well as in `SELECT` statements. However, you cannot use the same table (in this case, table `t1`) for both the subquery `FROM` clause and the update target.

For transactional storage engines, the failure of a subquery causes the entire statement to fail. For nontransactional storage engines, data modifications made before the error was encountered are preserved.

13.2.10.10 Optimizing Subqueries

Development is ongoing, so no optimization tip is reliable for the long term. The following list provides some interesting tricks that you might want to play with. See also Section 8.2.2, “Optimizing Subqueries and Derived Tables”.

- Use subquery clauses that affect the number or order of the rows in the subquery. For example:

```
SELECT * FROM t1 WHERE t1.column1 IN
  (SELECT column1 FROM t2 ORDER BY column1);
SELECT * FROM t1 WHERE t1.column1 IN
  (SELECT DISTINCT column1 FROM t2);
SELECT * FROM t1 WHERE EXISTS
  (SELECT * FROM t2 LIMIT 1);
```

- Replace a join with a subquery. For example, try this:

```
SELECT DISTINCT column1 FROM t1 WHERE t1.column1 IN (
  SELECT column1 FROM t2);
```

  Instead of this:

```
SELECT DISTINCT t1.column1 FROM t1, t2
  WHERE t1.column1 = t2.column1;
```

- Some subqueries can be transformed to joins for compatibility with older versions of MySQL that do not support subqueries. However, in some cases, converting a subquery to a join may improve performance. See Section 13.2.10.11, “Rewriting Subqueries as Joins”.

- Move clauses from outside to inside the subquery. For example, use this query:

```
SELECT * FROM t1
  WHERE s1 IN (SELECT s1 FROM t1 UNION ALL SELECT s1 FROM t2);
```

Instead of this query:

```
SELECT * FROM t1
  WHERE s1 IN (SELECT s1 FROM t1) OR s1 IN (SELECT s1 FROM t2);
```

For another example, use this query:

```
SELECT (SELECT column1 + 5 FROM t1) FROM t2;
```

Instead of this query:

```
SELECT (SELECT column1 FROM t1) + 5 FROM t2;
```

- Use a row subquery instead of a correlated subquery. For example, use this query:

```
SELECT * FROM t1
  WHERE (column1,column2) IN (SELECT column1,column2 FROM t2);
```

Instead of this query:

```
SELECT * FROM t1
  WHERE EXISTS (SELECT * FROM t2 WHERE t2.column1=t1.column1
                AND t2.column2=t1.column2);
```

- Use `NOT (a = ANY (...))` rather than `a <> ALL (...)`.
- Use `x = ANY (table containing (1,2))` rather than `x=1 OR x=2`.
- Use `= ANY` rather than `EXISTS`.
- For uncorrelated subqueries that always return one row, `IN` is always slower than `=`. For example, use this query:

```
SELECT * FROM t1
  WHERE t1.col_name = (SELECT a FROM t2 WHERE b = some_const);
```

Instead of this query:

```
SELECT * FROM t1
  WHERE t1.col_name IN (SELECT a FROM t2 WHERE b = some_const);
```

These tricks might cause programs to go faster or slower. Using MySQL facilities like the `BENCHMARK()` function, you can get an idea about what helps in your own situation. See Section 12.16, “Information Functions”.

Some optimizations that MySQL itself makes are:

- MySQL executes uncorrelated subqueries only once. Use `EXPLAIN` to make sure that a given subquery really is uncorrelated.
- MySQL rewrites `IN`, `ALL`, `ANY`, and `SOME` subqueries in an attempt to take advantage of the possibility that the select-list columns in the subquery are indexed.
- MySQL replaces subqueries of the following form with an index-lookup function, which `EXPLAIN` describes as a special join type (`unique_subquery` or `index_subquery`):

```
... IN (SELECT indexed_column FROM single_table ...)
```

- MySQL enhances expressions of the following form with an expression involving `MIN()` or `MAX()`, unless `NULL` values or empty sets are involved:

```
value {ALL|ANY|SOME} {> | < | >= | <=} (uncorrelated subquery)
```

For example, this `WHERE` clause:

```
WHERE 5 > ALL (SELECT x FROM t)
```

might be treated by the optimizer like this:

```
WHERE 5 > (SELECT MAX(x) FROM t)
```

See also MySQL Internals: How MySQL Transforms Subqueries.

13.2.10.11 Rewriting Subqueries as Joins

Sometimes there are other ways to test membership in a set of values than by using a subquery. Also, on some occasions, it is not only possible to rewrite a query without a subquery, but it can be more efficient to make use of some of these techniques rather than to use subqueries. One of these is the `IN()` construct:

For example, this query:

```
SELECT * FROM t1 WHERE id IN (SELECT id FROM t2);
```

Can be rewritten as:

```
SELECT DISTINCT t1.* FROM t1, t2 WHERE t1.id=t2.id;
```

The queries:

```
SELECT * FROM t1 WHERE id NOT IN (SELECT id FROM t2);
SELECT * FROM t1 WHERE NOT EXISTS (SELECT id FROM t2 WHERE t1.id=t2.id);
```

Can be rewritten as:

```
SELECT table1.*
  FROM table1 LEFT JOIN table2 ON table1.id=table2.id
  WHERE table2.id IS NULL;
```

A `LEFT [OUTER] JOIN` can be faster than an equivalent subquery because the server might be able to optimize it better—a fact that is not specific to MySQL Server alone. Prior to SQL-92, outer joins did not exist, so subqueries were the only way to do certain things. Today, MySQL Server and many other modern database systems offer a wide range of outer join types.

MySQL Server supports multiple-table `DELETE` statements that can be used to efficiently delete rows based on information from one table or even from many tables at the same time. Multiple-table `UPDATE` statements are also supported. See Section 13.2.2, “DELETE Statement”, and Section 13.2.11, “UPDATE Statement”.

13.2.10.12 Restrictions on Subqueries

- In general, you cannot modify a table and select from the same table in a subquery. For example, this limitation applies to statements of the following forms:

  ```
  DELETE FROM t WHERE ... (SELECT ... FROM t ...);
  UPDATE t ... WHERE col = (SELECT ... FROM t ...);
  {INSERT|REPLACE} INTO t (SELECT ... FROM t ...);
  ```

  Exception: The preceding prohibition does not apply if for the modified table you are using a derived table and that derived table is materialized rather than merged into the outer query. Example:

  ```
  UPDATE t ... WHERE col = (SELECT * FROM (SELECT ... FROM t...) AS _t ...);
  ```

  Here the result from the derived table is materialized as a temporary table, so the relevant rows in `t` have already been selected by the time the update to `t` takes place.

- Row comparison operations are only partially supported:

- For *`expr [NOT] IN subquery`*, *`expr`* can be an *`n`*-tuple (specified using row constructor syntax) and the subquery can return rows of *`n`*-tuples. The permitted syntax is therefore more specifically expressed as *`row_constructor [NOT] IN table_subquery`*

- For *`expr op {ALL|ANY|SOME} subquery`*, *`expr`* must be a scalar value and the subquery must be a column subquery; it cannot return multiple-column rows.

In other words, for a subquery that returns rows of *`n`*-tuples, this is supported:

```
(expr_1, ..., expr_n) [NOT] IN table_subquery
```

But this is not supported:

```
(expr_1, ..., expr_n) op {ALL|ANY|SOME} subquery
```

The reason for supporting row comparisons for `IN` but not for the others is that `IN` is implemented by rewriting it as a sequence of `=` comparisons and `AND` operations. This approach cannot be used for `ALL`, `ANY`, or `SOME`.

- Subqueries in the `FROM` clause cannot be correlated subqueries. They are materialized in whole (evaluated to produce a result set) during query execution, so they cannot be evaluated per row of the outer query. Before MySQL 5.6.3, materialization takes place before evaluation of the outer query. As of 5.6.3, the optimizer delays materialization until the result is needed, which may permit materialization to be avoided. See Section 8.2.2.4, “Optimizing Derived Tables”.

- MySQL does not support `LIMIT` in subqueries for certain subquery operators:

```
mysql> SELECT * FROM t1
       WHERE s1 IN (SELECT s2 FROM t2 ORDER BY s1 LIMIT 1);
ERROR 1235 (42000): This version of MySQL doesn't yet support
 'LIMIT & IN/ALL/ANY/SOME subquery'
```

- MySQL permits a subquery to refer to a stored function that has data-modifying side effects such as inserting rows into a table. For example, if `f()` inserts rows, the following query can modify data:

```
SELECT ... WHERE x IN (SELECT f() ...);
```

This behavior is an extension to the SQL standard. In MySQL, it can produce nondeterministic results because `f()` might be executed a different number of times for different executions of a given query depending on how the optimizer chooses to handle it.

For statement-based or mixed-format replication, one implication of this indeterminism is that such a query can produce different results on the source and its slaves.

- Before MySQL 5.6.3, a subquery in the `FROM` clause is evaluated by materializing the result into a temporary table, and this table does not use indexes. As of 5.6.3, the optimizer creates an index on the materialized table if this results in faster query execution. See Section 8.2.2.4, “Optimizing Derived Tables”.

13.2.11 UPDATE Statement

`UPDATE` is a DML statement that modifies rows in a table.

Single-table syntax:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_reference
    SET assignment_list
    [WHERE where_condition]
    [ORDER BY ...]
    [LIMIT row_count]

value:
    {expr | DEFAULT}
```

```
assignment:
    col_name = value

assignment_list:
    assignment [, assignment] ...
```

Multiple-table syntax:

```
UPDATE [LOW_PRIORITY] [IGNORE] table_references
    SET assignment_list
    [WHERE where_condition]
```

For the single-table syntax, the `UPDATE` statement updates columns of existing rows in the named table with new values. The `SET` clause indicates which columns to modify and the values they should be given. Each value can be given as an expression, or the keyword `DEFAULT` to set a column explicitly to its default value. The `WHERE` clause, if given, specifies the conditions that identify which rows to update. With no `WHERE` clause, all rows are updated. If the `ORDER BY` clause is specified, the rows are updated in the order that is specified. The `LIMIT` clause places a limit on the number of rows that can be updated.

For the multiple-table syntax, `UPDATE` updates rows in each table named in *`table_references`* that satisfy the conditions. Each matching row is updated once, even if it matches the conditions multiple times. For multiple-table syntax, `ORDER BY` and `LIMIT` cannot be used.

For partitioned tables, both the single-single and multiple-table forms of this statement support the use of a `PARTITION` option as part of a table reference. This option takes a list of one or more partitions or subpartitions (or both). Only the partitions (or subpartitions) listed are checked for matches, and a row that is not in any of these partitions or subpartitions is not updated, whether it satisfies the *`where_condition`* or not.

Note

Unlike the case when using `PARTITION` with an `INSERT` or `REPLACE` statement, an otherwise valid `UPDATE ... PARTITION` statement is considered successful even if no rows in the listed partitions (or subpartitions) match the *`where_condition`*.

For more information and examples, see Section 19.5, “Partition Selection”.

`where_condition` is an expression that evaluates to true for each row to be updated. For expression syntax, see Section 9.5, “Expressions”.

`table_references` and *`where_condition`* are specified as described in Section 13.2.9, “SELECT Statement”.

You need the `UPDATE` privilege only for columns referenced in an `UPDATE` that are actually updated. You need only the `SELECT` privilege for any columns that are read but not modified.

The `UPDATE` statement supports the following modifiers:

- With the `LOW_PRIORITY` modifier, execution of the `UPDATE` is delayed until no other clients are reading from the table. This affects only storage engines that use only table-level locking (such as `MyISAM`, `MEMORY`, and `MERGE`).
- With the `IGNORE` modifier, the update statement does not abort even if errors occur during the update. Rows for which duplicate-key conflicts occur on a unique key value are not updated. Rows updated to values that would cause data conversion errors are updated to the closest valid values instead.

`UPDATE IGNORE` statements, including those having an `ORDER BY` clause, are flagged as unsafe for statement-based replication. (This is because the order in which the rows are updated determines which rows are ignored.) Such statements produce a warning in the error log when using statement-

based mode and are written to the binary log using the row-based format when using `MIXED` mode. (Bug #11758262, Bug #50439) See Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”, for more information.

If you access a column from the table to be updated in an expression, `UPDATE` uses the current value of the column. For example, the following statement sets `col1` to one more than its current value:

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

The second assignment in the following statement sets `col2` to the current (updated) `col1` value, not the original `col1` value. The result is that `col1` and `col2` have the same value. This behavior differs from standard SQL.

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

Single-table `UPDATE` assignments are generally evaluated from left to right. For multiple-table updates, there is no guarantee that assignments are carried out in any particular order.

If you set a column to the value it currently has, MySQL notices this and does not update it.

If you update a column that has been declared `NOT NULL` by setting to `NULL`, an error occurs if strict SQL mode is enabled; otherwise, the column is set to the implicit default value for the column data type and the warning count is incremented. The implicit default value is `0` for numeric types, the empty string (`''`) for string types, and the “zero” value for date and time types. See Section 11.5, “Data Type Default Values”.

`UPDATE` returns the number of rows that were actually changed. The `mysql_info()` C API function returns the number of rows that were matched and updated and the number of warnings that occurred during the `UPDATE`.

You can use `LIMIT` *`row_count`* to restrict the scope of the `UPDATE`. A `LIMIT` clause is a rows-matched restriction. The statement stops as soon as it has found *`row_count`* rows that satisfy the `WHERE` clause, whether or not they actually were changed.

If an `UPDATE` statement includes an `ORDER BY` clause, the rows are updated in the order specified by the clause. This can be useful in certain situations that might otherwise result in an error. Suppose that a table `t` contains a column `id` that has a unique index. The following statement could fail with a duplicate-key error, depending on the order in which rows are updated:

```
UPDATE t SET id = id + 1;
```

For example, if the table contains 1 and 2 in the `id` column and 1 is updated to 2 before 2 is updated to 3, an error occurs. To avoid this problem, add an `ORDER BY` clause to cause the rows with larger `id` values to be updated before those with smaller values:

```
UPDATE t SET id = id + 1 ORDER BY id DESC;
```

You can also perform `UPDATE` operations covering multiple tables. However, you cannot use `ORDER BY` or `LIMIT` with a multiple-table `UPDATE`. The *`table_references`* clause lists the tables involved in the join. Its syntax is described in Section 13.2.9.2, “JOIN Clause”. Here is an example:

```
UPDATE items,month SET items.price=month.price
WHERE items.id=month.id;
```

The preceding example shows an inner join that uses the comma operator, but multiple-table `UPDATE` statements can use any type of join permitted in `SELECT` statements, such as `LEFT JOIN`.

If you use a multiple-table `UPDATE` statement involving `InnoDB` tables for which there are foreign key constraints, the MySQL optimizer might process tables in an order that differs from that of their parent/child relationship. In this case, the statement fails and rolls back. Instead, update a single table and rely on the `ON UPDATE` capabilities that `InnoDB` provides to cause the other tables to be modified accordingly. See Section 13.1.17.5, “FOREIGN KEY Constraints”.

You cannot update a table and select directly from the same table in a subquery. You can work around this by using a multi-table update in which one of the tables is derived from the table that you actually wish to update, and referring to the derived table using an alias. Suppose you wish to update a table named `items` which is defined using the statement shown here:

```
CREATE TABLE items (
    id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    wholesale DECIMAL(6,2) NOT NULL DEFAULT 0.00,
    retail DECIMAL(6,2) NOT NULL DEFAULT 0.00,
    quantity BIGINT NOT NULL DEFAULT 0
);
```

To reduce the retail price of any items for which the markup is 30% or greater and of which you have fewer than one hundred in stock, you might try to use an `UPDATE` statement such as the one following, which uses a subquery in the `WHERE` clause. As shown here, this statement does not work:

```
mysql> UPDATE items
     > SET retail = retail * 0.9
     > WHERE id IN
     >     (SELECT id FROM items
     >         WHERE retail / wholesale >= 1.3 AND quantity > 100);
ERROR 1093 (HY000): You can't specify target table 'items' for update in FROM clause
```

Instead, you can employ a multi-table update in which the subquery is moved into the list of tables to be updated, using an alias to reference it in the outermost `WHERE` clause, like this:

```
UPDATE items,
       (SELECT id FROM items
        WHERE id IN
            (SELECT id FROM items
             WHERE retail / wholesale >= 1.3 AND quantity < 100))
        AS discounted
SET items.retail = items.retail * 0.9
WHERE items.id = discounted.id;
```

Because the optimizer tries by default to merge the derived table `discounted` into the outermost query block, this works only if you force materialization of the derived table. You can do this by setting the `derived_merge` flag of the `optimizer_switch` system variable to `off` before running the update, or by using the `NO_MERGE` optimizer hint, as shown here:

```
UPDATE /*+ NO_MERGE(discounted) */ items,
       (SELECT id FROM items
        WHERE retail / wholesale >= 1.3 AND quantity < 100)
        AS discounted
    SET items.retail = items.retail * 0.9
    WHERE items.id = discounted.id;
```

The advantage of using the optimizer hint in such a case is that it applies only within the query block where it is used, so that it is not necessary to change the value of `optimizer_switch` again after executing the `UPDATE`.

Another possibility is to rewrite the subquery so that it does not use `IN` or `EXISTS`, like this:

```
UPDATE items,
       (SELECT id, retail / wholesale AS markup, quantity FROM items)
       AS discounted
    SET items.retail = items.retail * 0.9
    WHERE discounted.markup >= 1.3
    AND discounted.quantity < 100
    AND items.id = discounted.id;
```

In this case, the subquery is materialized by default rather than merged, so it is not necessary to disable merging of the derived table.

An `UPDATE` on a partitioned table using a storage engine such as `MyISAM` that employs table-level locks locks only those partitions containing rows that match the `UPDATE` statement `WHERE` clause, as long as none of the table partitioning columns are updated. (For storage engines such as

InnoDB that employ row-level locking, no locking of partitions takes place.) For more information, see Section 19.6.4, “Partitioning and Locking”.

13.3 Transactional and Locking Statements

MySQL supports local transactions (within a given client session) through statements such as SET autocommit, START TRANSACTION, COMMIT, and ROLLBACK. See Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”. XA transaction support enables MySQL to participate in distributed transactions as well. See Section 13.3.7, “XA Transactions”.

13.3.1 START TRANSACTION, COMMIT, and ROLLBACK Statements

```
START TRANSACTION
    [transaction_characteristic [, transaction_characteristic] ...]

transaction_characteristic: {
    WITH CONSISTENT SNAPSHOT
  | READ WRITE
  | READ ONLY
}

BEGIN [WORK]
COMMIT [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
ROLLBACK [WORK] [AND [NO] CHAIN] [[NO] RELEASE]
SET autocommit = {0 | 1}
```

These statements provide control over use of transactions:

- START TRANSACTION or BEGIN start a new transaction.
- COMMIT commits the current transaction, making its changes permanent.
- ROLLBACK rolls back the current transaction, canceling its changes.
- SET autocommit disables or enables the default autocommit mode for the current session.

By default, MySQL runs with autocommit mode enabled. This means that, when not otherwise inside a transaction, each statement is atomic, as if it were surrounded by START TRANSACTION and COMMIT. You cannot use ROLLBACK to undo the effect; however, if an error occurs during statement execution, the statement is rolled back.

To disable autocommit mode implicitly for a single series of statements, use the START TRANSACTION statement:

```
START TRANSACTION;
SELECT @A:=SUM(salary) FROM table1 WHERE type=1;
UPDATE table2 SET summary=@A WHERE type=1;
COMMIT;
```

With START TRANSACTION, autocommit remains disabled until you end the transaction with COMMIT or ROLLBACK. The autocommit mode then reverts to its previous state.

START TRANSACTION permits several modifiers that control transaction characteristics. To specify multiple modifiers, separate them by commas.

- The WITH CONSISTENT SNAPSHOT modifier starts a consistent read for storage engines that are capable of it. This applies only to InnoDB. The effect is the same as issuing a START TRANSACTION followed by a SELECT from any InnoDB table. See Section 14.7.2.3, “Consistent Nonlocking Reads”. The WITH CONSISTENT SNAPSHOT modifier does not change the current transaction isolation level, so it provides a consistent snapshot only if the current isolation level is one that permits a consistent read. The only isolation level that permits a consistent read is REPEATABLE READ. For all other isolation levels, the WITH CONSISTENT SNAPSHOT clause is ignored.

- The `READ WRITE` and `READ ONLY` modifiers set the transaction access mode. They permit or prohibit changes to tables used in the transaction. The `READ ONLY` restriction prevents the transaction from modifying or locking both transactional and nontransactional tables that are visible to other transactions; the transaction can still modify or lock temporary tables. These modifiers are available as of MySQL 5.6.5.

  MySQL enables extra optimizations for queries on `InnoDB` tables when the transaction is known to be read-only. Specifying `READ ONLY` ensures these optimizations are applied in cases where the read-only status cannot be determined automatically. See Section 8.5.3, "Optimizing InnoDB Read-Only Transactions" for more information.

  If no access mode is specified, the default mode applies. Unless the default has been changed, it is read/write. It is not permitted to specify both `READ WRITE` and `READ ONLY` in the same statement.

  In read-only mode, it remains possible to change tables created with the `TEMPORARY` keyword using DML statements. Changes made with DDL statements are not permitted, just as with permanent tables.

  For additional information about transaction access mode, including ways to change the default mode, see Section 13.3.6, "SET TRANSACTION Statement".

  If the `read_only` system variable is enabled, explicitly starting a transaction with `START TRANSACTION READ WRITE` requires the `SUPER` privilege.

Important

Many APIs used for writing MySQL client applications (such as JDBC) provide their own methods for starting transactions that can (and sometimes should) be used instead of sending a `START TRANSACTION` statement from the client. See Chapter 23, *Connectors and APIs*, or the documentation for your API, for more information.

To disable autocommit mode explicitly, use the following statement:

```
SET autocommit=0;
```

After disabling autocommit mode by setting the `autocommit` variable to zero, changes to transaction-safe tables (such as those for `InnoDB` or `NDB`) are not made permanent immediately. You must use `COMMIT` to store your changes to disk or `ROLLBACK` to ignore the changes.

`autocommit` is a session variable and must be set for each session. To disable autocommit mode for each new connection, see the description of the `autocommit` system variable at Section 5.1.7, "Server System Variables".

`BEGIN` and `BEGIN WORK` are supported as aliases of `START TRANSACTION` for initiating a transaction. `START TRANSACTION` is standard SQL syntax, is the recommended way to start an ad-hoc transaction, and permits modifiers that `BEGIN` does not.

The `BEGIN` statement differs from the use of the `BEGIN` keyword that starts a `BEGIN ... END` compound statement. The latter does not begin a transaction. See Section 13.6.1, "BEGIN ... END Compound Statement".

Note

Within all stored programs (stored procedures and functions, triggers, and events), the parser treats `BEGIN [WORK]` as the beginning of a `BEGIN ... END` block. Begin a transaction in this context with `START TRANSACTION` instead.

The optional `WORK` keyword is supported for `COMMIT` and `ROLLBACK`, as are the `CHAIN` and `RELEASE` clauses. `CHAIN` and `RELEASE` can be used for additional control over transaction completion. The

value of the `completion_type` system variable determines the default completion behavior. See Section 5.1.7, “Server System Variables”.

The `AND CHAIN` clause causes a new transaction to begin as soon as the current one ends, and the new transaction has the same isolation level as the just-terminated transaction. The new transaction also uses the same access mode (`READ WRITE` or `READ ONLY`) as the just-terminated transaction. The `RELEASE` clause causes the server to disconnect the current client session after terminating the current transaction. Including the `NO` keyword suppresses `CHAIN` or `RELEASE` completion, which can be useful if the `completion_type` system variable is set to cause chaining or release completion by default.

Beginning a transaction causes any pending transaction to be committed. See Section 13.3.3, “Statements That Cause an Implicit Commit”, for more information.

Beginning a transaction also causes table locks acquired with `LOCK TABLES` to be released, as though you had executed `UNLOCK TABLES`. Beginning a transaction does not release a global read lock acquired with `FLUSH TABLES WITH READ LOCK`.

For best results, transactions should be performed using only tables managed by a single transaction-safe storage engine. Otherwise, the following problems can occur:

- If you use tables from more than one transaction-safe storage engine (such as `InnoDB`), and the transaction isolation level is not `SERIALIZABLE`, it is possible that when one transaction commits, another ongoing transaction that uses the same tables sees only some of the changes made by the first transaction. That is, the atomicity of transactions is not guaranteed with mixed engines and inconsistencies can result. (If mixed-engine transactions are infrequent, you can use `SET TRANSACTION ISOLATION LEVEL` to set the isolation level to `SERIALIZABLE` on a per-transaction basis as necessary.)

- If you use tables that are not transaction-safe within a transaction, changes to those tables are stored at once, regardless of the status of autocommit mode.

- If you issue a `ROLLBACK` statement after updating a nontransactional table within a transaction, an `ER_WARNING_NOT_COMPLETE_ROLLBACK` warning occurs. Changes to transaction-safe tables are rolled back, but not changes to nontransaction-safe tables.

Each transaction is stored in the binary log in one chunk, upon `COMMIT`. Transactions that are rolled back are not logged. (**Exception**: Modifications to nontransactional tables cannot be rolled back. If a transaction that is rolled back includes modifications to nontransactional tables, the entire transaction is logged with a `ROLLBACK` statement at the end to ensure that modifications to the nontransactional tables are replicated.) See Section 5.4.4, “The Binary Log”.

You can change the isolation level or access mode for transactions with the `SET TRANSACTION` statement. See Section 13.3.6, “SET TRANSACTION Statement”.

Rolling back can be a slow operation that may occur implicitly without the user having explicitly asked for it (for example, when an error occurs). Because of this, `SHOW PROCESSLIST` displays `Rolling back` in the `State` column for the session, not only for explicit rollbacks performed with the `ROLLBACK` statement but also for implicit rollbacks.

Note

In MySQL 5.6, `BEGIN`, `COMMIT`, and `ROLLBACK` are not affected by `--replicate-do-db` or `--replicate-ignore-db` rules.

When `InnoDB` performs a complete rollback of a transaction, all locks set by the transaction are released. If a single SQL statement within a transaction rolls back as a result of an error, such as a duplicate key error, locks set by the statement are preserved while the transaction remains active. This happens because `InnoDB` stores row locks in a format such that it cannot know afterward which lock was set by which statement.

If a SELECT statement within a transaction calls a stored function, and a statement within the stored function fails, that statement rolls back. If ROLLBACK is executed for the transaction subsequently, the entire transaction rolls back.

13.3.2 Statements That Cannot Be Rolled Back

Some statements cannot be rolled back. In general, these include data definition language (DDL) statements, such as those that create or drop databases, those that create, drop, or alter tables or stored routines.

You should design your transactions not to include such statements. If you issue a statement early in a transaction that cannot be rolled back, and then another statement later fails, the full effect of the transaction cannot be rolled back in such cases by issuing a ROLLBACK statement.

13.3.3 Statements That Cause an Implicit Commit

The statements listed in this section (and any synonyms for them) implicitly end any transaction active in the current session, as if you had done a COMMIT before executing the statement.

Most of these statements also cause an implicit commit after executing. The intent is to handle each such statement in its own special transaction because it cannot be rolled back anyway. Transaction-control and locking statements are exceptions: If an implicit commit occurs before execution, another does not occur after.

- **Data definition language (DDL) statements that define or modify database objects.** ALTER DATABASE ... UPGRADE DATA DIRECTORY NAME, ALTER EVENT, ALTER PROCEDURE, ALTER SERVER, ALTER TABLE, ALTER VIEW, CREATE DATABASE, CREATE EVENT, CREATE INDEX, CREATE PROCEDURE, CREATE SERVER, CREATE TABLE, CREATE TRIGGER, CREATE VIEW, DROP DATABASE, DROP EVENT, DROP INDEX, DROP PROCEDURE, DROP SERVER, DROP TABLE, DROP TRIGGER, DROP VIEW, RENAME TABLE, TRUNCATE TABLE.

  ALTER FUNCTION, CREATE FUNCTION and DROP FUNCTION also cause an implicit commit when used with stored functions, but not with user-defined functions. (ALTER FUNCTION can only be used with stored functions.)

  CREATE TABLE and DROP TABLE statements do not commit a transaction if the TEMPORARY keyword is used. (This does not apply to other operations on temporary tables such as ALTER TABLE and CREATE INDEX, which do cause a commit.) However, although no implicit commit occurs, neither can the statement be rolled back, which means that the use of such statements causes transactional atomicity to be violated. For example, if you use CREATE TEMPORARY TABLE and then roll back the transaction, the table remains in existence.

  The CREATE TABLE statement in InnoDB is processed as a single transaction. This means that a ROLLBACK from the user does not undo CREATE TABLE statements the user made during that transaction.

  CREATE TABLE ... SELECT causes an implicit commit before and after the statement is executed when you are creating nontemporary tables. (No commit occurs for CREATE TEMPORARY TABLE ... SELECT.)

- **Statements that implicitly use or modify tables in the mysql database.** ALTER USER, CREATE USER, DROP USER, GRANT, RENAME USER, REVOKE, SET PASSWORD.

- **Transaction-control and locking statements.** BEGIN, LOCK TABLES, SET autocommit = 1 (if the value is not already 1), START TRANSACTION, UNLOCK TABLES.

  UNLOCK TABLES commits a transaction only if any tables currently have been locked with LOCK TABLES to acquire nontransactional table locks. A commit does not occur for UNLOCK TABLES following FLUSH TABLES WITH READ LOCK because the latter statement does not acquire table-level locks.

Transactions cannot be nested. This is a consequence of the implicit commit performed for any current transaction when you issue a `START TRANSACTION` statement or one of its synonyms.

Statements that cause an implicit commit cannot be used in an XA transaction while the transaction is in an `ACTIVE` state.

The `BEGIN` statement differs from the use of the `BEGIN` keyword that starts a `BEGIN ... END` compound statement. The latter does not cause an implicit commit. See Section 13.6.1, "BEGIN ... END Compound Statement".

- **Data loading statements.** `LOAD DATA`. `LOAD DATA` causes an implicit commit only for tables using the `NDB` storage engine.
- **Administrative statements.** `ANALYZE TABLE`, `CACHE INDEX`, `CHECK TABLE`, `FLUSH`, `LOAD INDEX INTO CACHE`, `OPTIMIZE TABLE`, `REPAIR TABLE`, `RESET`.
- **Replication control statements**. `START SLAVE`, `STOP SLAVE`, `RESET SLAVE`, `CHANGE MASTER TO`.

13.3.4 SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Statements

```
SAVEPOINT identifier
ROLLBACK [WORK] TO [SAVEPOINT] identifier
RELEASE SAVEPOINT identifier
```

`InnoDB` supports the SQL statements `SAVEPOINT`, `ROLLBACK TO SAVEPOINT`, `RELEASE SAVEPOINT` and the optional `WORK` keyword for `ROLLBACK`.

The `SAVEPOINT` statement sets a named transaction savepoint with a name of *`identifier`*. If the current transaction has a savepoint with the same name, the old savepoint is deleted and a new one is set.

The `ROLLBACK TO SAVEPOINT` statement rolls back a transaction to the named savepoint without terminating the transaction. Modifications that the current transaction made to rows after the savepoint was set are undone in the rollback, but `InnoDB` does *not* release the row locks that were stored in memory after the savepoint. (For a new inserted row, the lock information is carried by the transaction ID stored in the row; the lock is not separately stored in memory. In this case, the row lock is released in the undo.) Savepoints that were set at a later time than the named savepoint are deleted.

If the `ROLLBACK TO SAVEPOINT` statement returns the following error, it means that no savepoint with the specified name exists:

```
ERROR 1305 (42000): SAVEPOINT identifier does not exist
```

The `RELEASE SAVEPOINT` statement removes the named savepoint from the set of savepoints of the current transaction. No commit or rollback occurs. It is an error if the savepoint does not exist.

All savepoints of the current transaction are deleted if you execute a `COMMIT`, or a `ROLLBACK` that does not name a savepoint.

A new savepoint level is created when a stored function is invoked or a trigger is activated. The savepoints on previous levels become unavailable and thus do not conflict with savepoints on the new level. When the function or trigger terminates, any savepoints it created are released and the previous savepoint level is restored.

13.3.5 LOCK TABLES and UNLOCK TABLES Statements

```
LOCK TABLES
```

```
    tbl_name [[AS] alias] lock_type
    [, tbl_name [[AS] alias] lock_type] ...

lock_type: {
    READ [LOCAL]
  | [LOW_PRIORITY] WRITE
}

UNLOCK TABLES
```

MySQL enables client sessions to acquire table locks explicitly for the purpose of cooperating with other sessions for access to tables, or to prevent other sessions from modifying tables during periods when a session requires exclusive access to them. A session can acquire or release locks only for itself. One session cannot acquire locks for another session or release locks held by another session.

Locks may be used to emulate transactions or to get more speed when updating tables. This is explained in more detail in Table-Locking Restrictions and Conditions.

`LOCK TABLES` explicitly acquires table locks for the current client session. Table locks can be acquired for base tables or views. You must have the `LOCK TABLES` privilege, and the `SELECT` privilege for each object to be locked.

For view locking, `LOCK TABLES` adds all base tables used in the view to the set of tables to be locked and locks them automatically. As of MySQL 5.6.50, `LOCK TABLES` checks that the view definer has the proper privileges on the tables underlying the view.

If you lock a table explicitly with `LOCK TABLES`, any tables used in triggers are also locked implicitly, as described in LOCK TABLES and Triggers.

`UNLOCK TABLES` explicitly releases any table locks held by the current session. `LOCK TABLES` implicitly releases any table locks held by the current session before acquiring new locks.

Another use for `UNLOCK TABLES` is to release the global read lock acquired with the `FLUSH TABLES WITH READ LOCK` statement, which enables you to lock all tables in all databases. See Section 13.7.6.3, “FLUSH Statement”. (This is a very convenient way to get backups if you have a file system such as Veritas that can take snapshots in time.)

A table lock protects only against inappropriate reads or writes by other sessions. A session holding a `WRITE` lock can perform table-level operations such as `DROP TABLE` or `TRUNCATE TABLE`. For sessions holding a `READ` lock, `DROP TABLE` and `TRUNCATE TABLE` operations are not permitted.

The following discussion applies only to non-`TEMPORARY` tables. `LOCK TABLES` is permitted (but ignored) for a `TEMPORARY` table. The table can be accessed freely by the session within which it was created, regardless of what other locking may be in effect. No lock is necessary because no other session can see the table.

- Table Lock Acquisition
- Table Lock Release
- Interaction of Table Locking and Transactions
- LOCK TABLES and Triggers
- Table-Locking Restrictions and Conditions

Table Lock Acquisition

To acquire table locks within the current session, use the `LOCK TABLES` statement, which acquires metadata locks (see Section 8.11.4, “Metadata Locking”).

The following lock types are available:

`READ [LOCAL]` lock:

- The session that holds the lock can read the table (but not write it).
- Multiple sessions can acquire a `READ` lock for the table at the same time.
- Other sessions can read the table without explicitly acquiring a `READ` lock.
- The `LOCAL` modifier enables nonconflicting `INSERT` statements (concurrent inserts) by other sessions to execute while the lock is held. (See Section 8.11.3, “Concurrent Inserts”.) However, `READ LOCAL` cannot be used if you are going to manipulate the database using processes external to the server while you hold the lock. For `InnoDB` tables, `READ LOCAL` is the same as `READ`.

`[LOW_PRIORITY] WRITE` lock:

- The session that holds the lock can read and write the table.
- Only the session that holds the lock can access the table. No other session can access it until the lock is released.
- Lock requests for the table by other sessions block while the `WRITE` lock is held.
- The `LOW_PRIORITY` modifier has no effect. In previous versions of MySQL, it affected locking behavior, but this is no longer true. It is now deprecated and its use produces a warning. Use `WRITE` without `LOW_PRIORITY` instead.

`WRITE` locks normally have higher priority than `READ` locks to ensure that updates are processed as soon as possible. This means that if one session obtains a `READ` lock and then another session requests a `WRITE` lock, subsequent `READ` lock requests wait until the session that requested the `WRITE` lock has obtained the lock and released it. (An exception to this policy can occur for small values of the `max_write_lock_count` system variable; see Section 8.11.4, “Metadata Locking”.)

If the `LOCK TABLES` statement must wait due to locks held by other sessions on any of the tables, it blocks until all locks can be acquired.

A session that requires locks must acquire all the locks that it needs in a single `LOCK TABLES` statement. While the locks thus obtained are held, the session can access only the locked tables. For example, in the following sequence of statements, an error occurs for the attempt to access `t2` because it was not locked in the `LOCK TABLES` statement:

```
mysql> LOCK TABLES t1 READ;
mysql> SELECT COUNT(*) FROM t1;
+----------+
| COUNT(*) |
+----------+
|        3 |
+----------+
mysql> SELECT COUNT(*) FROM t2;
ERROR 1100 (HY000): Table 't2' was not locked with LOCK TABLES
```

Tables in the `INFORMATION_SCHEMA` database are an exception. They can be accessed without being locked explicitly even while a session holds table locks obtained with `LOCK TABLES`.

You cannot refer to a locked table multiple times in a single query using the same name. Use aliases instead, and obtain a separate lock for the table and each alias:

```
mysql> LOCK TABLE t WRITE, t AS t1 READ;
mysql> INSERT INTO t SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> INSERT INTO t SELECT * FROM t AS t1;
```

The error occurs for the first `INSERT` because there are two references to the same name for a locked table. The second `INSERT` succeeds because the references to the table use different names.

If your statements refer to a table by means of an alias, you must lock the table using that same alias. It does not work to lock the table without specifying the alias:

```
mysql> LOCK TABLE t READ;
mysql> SELECT * FROM t AS myalias;
ERROR 1100: Table 'myalias' was not locked with LOCK TABLES
```

Conversely, if you lock a table using an alias, you must refer to it in your statements using that alias:

```
mysql> LOCK TABLE t AS myalias READ;
mysql> SELECT * FROM t;
ERROR 1100: Table 't' was not locked with LOCK TABLES
mysql> SELECT * FROM t AS myalias;
```

Note

`LOCK TABLES` or `UNLOCK TABLES`, when applied to a partitioned table, always locks or unlocks the entire table; these statements do not support partition lock pruning. See Section 19.6.4, “Partitioning and Locking”.

Table Lock Release

When the table locks held by a session are released, they are all released at the same time. A session can release its locks explicitly, or locks may be released implicitly under certain conditions.

- A session can release its locks explicitly with `UNLOCK TABLES`.
- If a session issues a `LOCK TABLES` statement to acquire a lock while already holding locks, its existing locks are released implicitly before the new locks are granted.
- If a session begins a transaction (for example, with `START TRANSACTION`), an implicit `UNLOCK TABLES` is performed, which causes existing locks to be released. (For additional information about the interaction between table locking and transactions, see Interaction of Table Locking and Transactions.)

If the connection for a client session terminates, whether normally or abnormally, the server implicitly releases all table locks held by the session (transactional and nontransactional). If the client reconnects, the locks are no longer be in effect. In addition, if the client had an active transaction, the server rolls back the transaction upon disconnect, and if reconnect occurs, the new session begins with autocommit enabled. For this reason, clients may wish to disable auto-reconnect. With auto-reconnect in effect, the client is not notified if reconnect occurs but any table locks or current transaction are lost. With auto-reconnect disabled, if the connection drops, an error occurs for the next statement issued. The client can detect the error and take appropriate action such as reacquiring the locks or redoing the transaction. See C API Automatic Reconnection Control.

Note

If you use `ALTER TABLE` on a locked table, it may become unlocked. For example, if you attempt a second `ALTER TABLE` operation, the result may be an error `Table 'tbl_name' was not locked with LOCK TABLES`. To handle this, lock the table again prior to the second alteration. See also Section B.3.6.1, “Problems with ALTER TABLE”.

Interaction of Table Locking and Transactions

`LOCK TABLES` and `UNLOCK TABLES` interact with the use of transactions as follows:

- `LOCK TABLES` is not transaction-safe and implicitly commits any active transaction before attempting to lock the tables.
- `UNLOCK TABLES` implicitly commits any active transaction, but only if `LOCK TABLES` has been used to acquire table locks. For example, in the following set of statements, `UNLOCK TABLES` releases the global read lock but does not commit the transaction because no table locks are in effect:

```
FLUSH TABLES WITH READ LOCK;
START TRANSACTION;
SELECT ... ;
UNLOCK TABLES;
```

- Beginning a transaction (for example, with `START TRANSACTION`) implicitly commits any current transaction and releases existing table locks.
- `FLUSH TABLES WITH READ LOCK` acquires a global read lock and not table locks, so it is not subject to the same behavior as `LOCK TABLES` and `UNLOCK TABLES` with respect to table locking and implicit commits. For example, `START TRANSACTION` does not release the global read lock. See Section 13.7.6.3, “FLUSH Statement”.
- Other statements that implicitly cause transactions to be committed do not release existing table locks. For a list of such statements, see Section 13.3.3, “Statements That Cause an Implicit Commit”.
- The correct way to use `LOCK TABLES` and `UNLOCK TABLES` with transactional tables, such as `InnoDB` tables, is to begin a transaction with `SET autocommit = 0` (not `START TRANSACTION`) followed by `LOCK TABLES`, and to not call `UNLOCK TABLES` until you commit the transaction explicitly. For example, if you need to write to table `t1` and read from table `t2`, you can do this:

  ```
  SET autocommit=0;
  LOCK TABLES t1 WRITE, t2 READ, ...;
  ... do something with tables t1 and t2 here ...
  COMMIT;
  UNLOCK TABLES;
  ```

  When you call `LOCK TABLES`, `InnoDB` internally takes its own table lock, and MySQL takes its own table lock. `InnoDB` releases its internal table lock at the next commit, but for MySQL to release its table lock, you have to call `UNLOCK TABLES`. You should not have `autocommit = 1`, because then `InnoDB` releases its internal table lock immediately after the call of `LOCK TABLES`, and deadlocks can very easily happen. `InnoDB` does not acquire the internal table lock at all if `autocommit = 1`, to help old applications avoid unnecessary deadlocks.
- `ROLLBACK` does not release table locks.

LOCK TABLES and Triggers

If you lock a table explicitly with `LOCK TABLES`, any tables used in triggers are also locked implicitly:

- The locks are taken as the same time as those acquired explicitly with the `LOCK TABLES` statement.
- The lock on a table used in a trigger depends on whether the table is used only for reading. If so, a read lock suffices. Otherwise, a write lock is used.
- If a table is locked explicitly for reading with `LOCK TABLES`, but needs to be locked for writing because it might be modified within a trigger, a write lock is taken rather than a read lock. (That is, an implicit write lock needed due to the table's appearance within a trigger causes an explicit read lock request for the table to be converted to a write lock request.)

Suppose that you lock two tables, `t1` and `t2`, using this statement:

```
LOCK TABLES t1 WRITE, t2 READ;
```

If `t1` or `t2` have any triggers, tables used within the triggers are also locked. Suppose that `t1` has a trigger defined like this:

```
CREATE TRIGGER t1_a_ins AFTER INSERT ON t1 FOR EACH ROW
BEGIN
  UPDATE t4 SET count = count+1
      WHERE id = NEW.id AND EXISTS (SELECT a FROM t3);
  INSERT INTO t2 VALUES(1, 2);
```

```
END;
```

The result of the `LOCK TABLES` statement is that `t1` and `t2` are locked because they appear in the statement, and `t3` and `t4` are locked because they are used within the trigger:

- `t1` is locked for writing per the `WRITE` lock request.
- `t2` is locked for writing, even though the request is for a `READ` lock. This occurs because `t2` is inserted into within the trigger, so the `READ` request is converted to a `WRITE` request.
- `t3` is locked for reading because it is only read from within the trigger.
- `t4` is locked for writing because it might be updated within the trigger.

Table-Locking Restrictions and Conditions

You can safely use `KILL` to terminate a session that is waiting for a table lock. See Section 13.7.6.4, "KILL Statement".

Do *not* lock any tables that you are using with `INSERT DELAYED`. An `INSERT DELAYED` in this case results in an error because the insert must be handled by a separate thread, not by the session which holds the lock.

`LOCK TABLES` and `UNLOCK TABLES` cannot be used within stored programs.

Tables in the `performance_schema` database cannot be locked with `LOCK TABLES`, except the `setup_xxx` tables.

The following statements are prohibited while a `LOCK TABLES` statement is in effect: `CREATE TABLE`, `CREATE TABLE ... LIKE`, `CREATE VIEW`, `DROP VIEW`, and DDL statements on stored functions and procedures and events.

For some operations, system tables in the `mysql` database must be accessed. For example, the `HELP` statement requires the contents of the server-side help tables, and `CONVERT_TZ()` might need to read the time zone tables. The server implicitly locks the system tables for reading as necessary so that you need not lock them explicitly. These tables are treated as just described:

```
mysql.help_category
mysql.help_keyword
mysql.help_relation
mysql.help_topic
mysql.proc
mysql.time_zone
mysql.time_zone_leap_second
mysql.time_zone_name
mysql.time_zone_transition
mysql.time_zone_transition_type
```

If you want to explicitly place a `WRITE` lock on any of those tables with a `LOCK TABLES` statement, the table must be the only one locked; no other table can be locked with the same statement.

Normally, you do not need to lock tables, because all single `UPDATE` statements are atomic; no other session can interfere with any other currently executing SQL statement. However, there are a few cases when locking tables may provide an advantage:

- If you are going to run many operations on a set of `MyISAM` tables, it is much faster to lock the tables you are going to use. Locking `MyISAM` tables speeds up inserting, updating, or deleting on them because MySQL does not flush the key cache for the locked tables until `UNLOCK TABLES` is called. Normally, the key cache is flushed after each SQL statement.

  The downside to locking the tables is that no session can update a `READ`-locked table (including the one holding the lock) and no session can access a `WRITE`-locked table other than the one holding the lock.

- If you are using tables for a nontransactional storage engine, you must use `LOCK TABLES` if you want to ensure that no other session modifies the tables between a `SELECT` and an `UPDATE`. The example shown here requires `LOCK TABLES` to execute safely:

```
LOCK TABLES trans READ, customer WRITE;
SELECT SUM(value) FROM trans WHERE customer_id=some_id;
UPDATE customer
  SET total_value=sum_from_previous_statement
  WHERE customer_id=some_id;
UNLOCK TABLES;
```

  Without `LOCK TABLES`, it is possible that another session might insert a new row in the `trans` table between execution of the `SELECT` and `UPDATE` statements.

You can avoid using `LOCK TABLES` in many cases by using relative updates (`UPDATE customer SET value=value+new_value`) or the `LAST_INSERT_ID()` function.

You can also avoid locking tables in some cases by using the user-level advisory lock functions `GET_LOCK()` and `RELEASE_LOCK()`. These locks are saved in a hash table in the server and implemented with `pthread_mutex_lock()` and `pthread_mutex_unlock()` for high speed. See Section 12.15, “Locking Functions”.

See Section 8.11.1, “Internal Locking Methods”, for more information on locking policy.

13.3.6 SET TRANSACTION Statement

```
SET [GLOBAL | SESSION] TRANSACTION
    transaction_characteristic [, transaction_characteristic] ...

transaction_characteristic: {
    ISOLATION LEVEL level
  | access_mode
}

level: {
     REPEATABLE READ
   | READ COMMITTED
   | READ UNCOMMITTED
   | SERIALIZABLE
}

access_mode: {
     READ WRITE
   | READ ONLY
}
```

This statement specifies transaction characteristics. It takes a list of one or more characteristic values separated by commas. Each characteristic value sets the transaction isolation level or access mode. The isolation level is used for operations on `InnoDB` tables. The access mode specifies whether transactions operate in read/write or read-only mode.

In addition, `SET TRANSACTION` can include an optional `GLOBAL` or `SESSION` keyword to indicate the scope of the statement.

- Transaction Isolation Levels
- Transaction Access Mode
- Transaction Characteristic Scope

Transaction Isolation Levels

To set the transaction isolation level, use an `ISOLATION LEVEL level` clause. It is not permitted to specify multiple `ISOLATION LEVEL` clauses in the same `SET TRANSACTION` statement.

The default isolation level is `REPEATABLE READ`. Other permitted values are `READ COMMITTED`, `READ UNCOMMITTED`, and `SERIALIZABLE`. For information about these isolation levels, see Section 14.7.2.1, “Transaction Isolation Levels”.

Transaction Access Mode

To set the transaction access mode, use a `READ WRITE` or `READ ONLY` clause. It is not permitted to specify multiple access-mode clauses in the same `SET TRANSACTION` statement.

By default, a transaction takes place in read/write mode, with both reads and writes permitted to tables used in the transaction. This mode may be specified explicitly using `SET TRANSACTION` with an access mode of `READ WRITE`.

If the transaction access mode is set to `READ ONLY`, changes to tables are prohibited. This may enable storage engines to make performance improvements that are possible when writes are not permitted.

In read-only mode, it remains possible to change tables created with the `TEMPORARY` keyword using DML statements. Changes made with DDL statements are not permitted, just as with permanent tables.

The `READ WRITE` and `READ ONLY` access modes also may be specified for an individual transaction using the `START TRANSACTION` statement.

Transaction Characteristic Scope

You can set transaction characteristics globally, for the current session, or for the next transaction only:

- With the `GLOBAL` keyword:
  - The statement applies globally for all subsequent sessions.
  - Existing sessions are unaffected.
- With the `SESSION` keyword:
  - The statement applies to all subsequent transactions performed within the current session.
  - The statement is permitted within transactions, but does not affect the current ongoing transaction.
  - If executed between transactions, the statement overrides any preceding statement that sets the next-transaction value of the named characteristics.
- Without any `SESSION` or `GLOBAL` keyword:
  - The statement applies only to the next single transaction performed within the session.
  - Subsequent transactions revert to using the session value of the named characteristics.
  - The statement is not permitted within transactions:

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.02 sec)

mysql> SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
ERROR 1568 (25001): Transaction characteristics can't be changed
while a transaction is in progress
```

A change to global transaction characteristics requires the `SUPER` privilege. Any session is free to change its session characteristics (even in the middle of a transaction), or the characteristics for its next transaction (prior to the start of that transaction).

To set the global isolation level at server startup, use the `--transaction-isolation=level` option on the command line or in an option file. Values of *`level`* for this option use dashes rather than

spaces, so the permissible values are `READ-UNCOMMITTED`, `READ-COMMITTED`, `REPEATABLE-READ`, or `SERIALIZABLE`.

Similarly, to set the global transaction access mode at server startup, use the `--transaction-read-only` option. The default is `OFF` (read/write mode) but the value can be set to `ON` for a mode of read only.

For example, to set the isolation level to `REPEATABLE READ` and the access mode to `READ WRITE`, use these lines in the `[mysqld]` section of an option file:

```
[mysqld]
transaction-isolation = REPEATABLE-READ
transaction-read-only = OFF
```

At runtime, characteristics at the global, session, and next-transaction scope levels can be set indirectly using the `SET TRANSACTION` statement, as described previously. They can also be set directly using the `SET` statement to assign values to the `tx_isolation` and `tx_read_only` system variables:

- `SET TRANSACTION` permits optional `GLOBAL` and `SESSION` keywords for setting transaction characteristics at different scope levels.
- The `SET` statement for assigning values to the `tx_isolation` and `tx_read_only` system variables has syntaxes for setting these variables at different scope levels.

The following tables show the characteristic scope level set by each `SET TRANSACTION` and variable-assignment syntax.

Table 13.6 SET TRANSACTION Syntax for Transaction Characteristics

Syntax	Affected Characteristic Scope
`SET GLOBAL TRANSACTION` *`transaction_characteristic`*	Global
`SET SESSION TRANSACTION` *`transaction_characteristic`*	Session
`SET TRANSACTION` *`transaction_characteristic`*	Next transaction only

Table 13.7 SET Syntax for Transaction Characteristics

Syntax	Affected Characteristic Scope
`SET GLOBAL` *`var_name = value`*	Global
`SET @@GLOBAL.`*`var_name = value`*	Global
`SET PERSIST` *`var_name = value`*	Global
`SET @@PERSIST.`*`var_name = value`*	Global
`SET PERSIST_ONLY` *`var_name = value`*	No runtime effect
`SET @@PERSIST_ONLY.`*`var_name = value`*	No runtime effect
`SET SESSION` *`var_name = value`*	Session
`SET @@SESSION.`*`var_name = value`*	Session
`SET` *`var_name = value`*	Session
`SET @@`*`var_name = value`*	Next transaction only

It is possible to check the global and session values of transaction characteristics at runtime:

```
SELECT @@GLOBAL.tx_isolation, @@GLOBAL.tx_read_only;
SELECT @@SESSION.tx_isolation, @@SESSION.tx_read_only;
```

13.3.7 XA Transactions

Support for XA transactions is available for the `InnoDB` storage engine. The MySQL XA implementation is based on the X/Open CAE document *Distributed Transaction Processing: The XA Specification*. This document is published by The Open Group and available at http://www.opengroup.org/public/pubs/catalog/c193.htm. Limitations of the current XA implementation are described in Section 13.3.7.3, “Restrictions on XA Transactions”.

On the client side, there are no special requirements. The XA interface to a MySQL server consists of SQL statements that begin with the `XA` keyword. MySQL client programs must be able to send SQL statements and to understand the semantics of the XA statement interface. They do not need be linked against a recent client library. Older client libraries also work.

Among the MySQL Connectors, MySQL Connector/J 5.0.0 and higher supports XA directly, by means of a class interface that handles the XA SQL statement interface for you.

XA supports distributed transactions, that is, the ability to permit multiple separate transactional resources to participate in a global transaction. Transactional resources often are RDBMSs but may be other kinds of resources.

A global transaction involves several actions that are transactional in themselves, but that all must either complete successfully as a group, or all be rolled back as a group. In essence, this extends ACID properties “up a level” so that multiple ACID transactions can be executed in concert as components of a global operation that also has ACID properties. (As with nondistributed transactions, `SERIALIZABLE` may be preferred if your applications are sensitive to read phenomena. `REPEATABLE READ` may not be sufficient for distributed transactions.)

Some examples of distributed transactions:

- An application may act as an integration tool that combines a messaging service with an RDBMS. The application makes sure that transactions dealing with message sending, retrieval, and processing that also involve a transactional database all happen in a global transaction. You can think of this as “transactional email.”
- An application performs actions that involve different database servers, such as a MySQL server and an Oracle server (or multiple MySQL servers), where actions that involve multiple servers must happen as part of a global transaction, rather than as separate transactions local to each server.
- A bank keeps account information in an RDBMS and distributes and receives money through automated teller machines (ATMs). It is necessary to ensure that ATM actions are correctly reflected in the accounts, but this cannot be done with the RDBMS alone. A global transaction manager integrates the ATM and database resources to ensure overall consistency of financial transactions.

Applications that use global transactions involve one or more Resource Managers and a Transaction Manager:

- A Resource Manager (RM) provides access to transactional resources. A database server is one kind of resource manager. It must be possible to either commit or roll back transactions managed by the RM.
- A Transaction Manager (TM) coordinates the transactions that are part of a global transaction. It communicates with the RMs that handle each of these transactions. The individual transactions within a global transaction are “branches” of the global transaction. Global transactions and their branches are identified by a naming scheme described later.

The MySQL implementation of XA enables a MySQL server to act as a Resource Manager that handles XA transactions within a global transaction. A client program that connects to the MySQL server acts as the Transaction Manager.

To carry out a global transaction, it is necessary to know which components are involved, and bring each component to a point when it can be committed or rolled back. Depending on what each

component reports about its ability to succeed, they must all commit or roll back as an atomic group. That is, either all components must commit, or all components must roll back. To manage a global transaction, it is necessary to take into account that any component or the connecting network might fail.

The process for executing a global transaction uses two-phase commit (2PC). This takes place after the actions performed by the branches of the global transaction have been executed.

1. In the first phase, all branches are prepared. That is, they are told by the TM to get ready to commit. Typically, this means each RM that manages a branch records the actions for the branch in stable storage. The branches indicate whether they are able to do this, and these results are used for the second phase.

2. In the second phase, the TM tells the RMs whether to commit or roll back. If all branches indicated when they were prepared that they are able to commit, all branches are told to commit. If any branch indicated when it was prepared that it was not able to commit, all branches are told to roll back.

In some cases, a global transaction might use one-phase commit (1PC). For example, when a Transaction Manager finds that a global transaction consists of only one transactional resource (that is, a single branch), that resource can be told to prepare and commit at the same time.

13.3.7.1 XA Transaction SQL Statements

To perform XA transactions in MySQL, use the following statements:

```
XA {START|BEGIN} xid [JOIN|RESUME]

XA END xid [SUSPEND [FOR MIGRATE]]

XA PREPARE xid

XA COMMIT xid [ONE PHASE]

XA ROLLBACK xid

XA RECOVER
```

For `XA START`, the `JOIN` and `RESUME` clauses are recognized but have no effect.

For `XA END` the `SUSPEND [FOR MIGRATE]` clause is recognized but has no effect.

Each XA statement begins with the `XA` keyword, and most of them require an *`xid`* value. An *`xid`* is an XA transaction identifier. It indicates which transaction the statement applies to. *`xid`* values are supplied by the client, or generated by the MySQL server. An *`xid`* value has from one to three parts:

```
xid: gtrid [, bqual [, formatID ]]
```

`gtrid` is a global transaction identifier, *`bqual`* is a branch qualifier, and *`formatID`* is a number that identifies the format used by the *`gtrid`* and *`bqual`* values. As indicated by the syntax, *`bqual`* and *`formatID`* are optional. The default *`bqual`* value is `''` if not given. The default *`formatID`* value is 1 if not given.

`gtrid` and *`bqual`* must be string literals, each up to 64 bytes (not characters) long. *`gtrid`* and *`bqual`* can be specified in several ways. You can use a quoted string (`'ab'`), hex string (`X'6162'`, `0x6162`), or bit value (`b'nnnn'`).

`formatID` is an unsigned integer.

The *`gtrid`* and *`bqual`* values are interpreted in bytes by the MySQL server's underlying XA support routines. However, while an SQL statement containing an XA statement is being parsed, the server works with some specific character set. To be safe, write *`gtrid`* and *`bqual`* as hex strings.

xid values typically are generated by the Transaction Manager. Values generated by one TM must be different from values generated by other TMs. A given TM must be able to recognize its own *xid* values in a list of values returned by the XA RECOVER statement.

XA START *xid* starts an XA transaction with the given *xid* value. Each XA transaction must have a unique *xid* value, so the value must not currently be used by another XA transaction. Uniqueness is assessed using the *gtrid* and *bqual* values. All following XA statements for the XA transaction must be specified using the same *xid* value as that given in the XA START statement. If you use any of those statements but specify an *xid* value that does not correspond to some existing XA transaction, an error occurs.

One or more XA transactions can be part of the same global transaction. All XA transactions within a given global transaction must use the same *gtrid* value in the *xid* value. For this reason, *gtrid* values must be globally unique so that there is no ambiguity about which global transaction a given XA transaction is part of. The *bqual* part of the *xid* value must be different for each XA transaction within a global transaction. (The requirement that *bqual* values be different is a limitation of the current MySQL XA implementation. It is not part of the XA specification.)

The XA RECOVER statement returns information for those XA transactions on the MySQL server that are in the PREPARED state. (See Section 13.3.7.2, “XA Transaction States”.) The output includes a row for each such XA transaction on the server, regardless of which client started it.

XA RECOVER output rows look like this (for an example *xid* value consisting of the parts 'abc', 'def', and 7):

```
mysql> XA RECOVER;
+----------+--------------+--------------+--------+
| formatID | gtrid_length | bqual_length | data   |
+----------+--------------+--------------+--------+
|        7 |            3 |            3 | abcdef |
+----------+--------------+--------------+--------+
```

The output columns have the following meanings:

- formatID is the *formatID* part of the transaction *xid*
- gtrid_length is the length in bytes of the *gtrid* part of the *xid*
- bqual_length is the length in bytes of the *bqual* part of the *xid*
- data is the concatenation of the *gtrid* and *bqual* parts of the *xid*

13.3.7.2 XA Transaction States

An XA transaction progresses through the following states:

1. Use XA START to start an XA transaction and put it in the ACTIVE state.
2. For an ACTIVE XA transaction, issue the SQL statements that make up the transaction, and then issue an XA END statement. XA END puts the transaction in the IDLE state.
3. For an IDLE XA transaction, you can issue either an XA PREPARE statement or an XA COMMIT ... ONE PHASE statement:
   - XA PREPARE puts the transaction in the PREPARED state. An XA RECOVER statement at this point includes the transaction's *xid* value in its output, because XA RECOVER lists all XA transactions that are in the PREPARED state.
   - XA COMMIT ... ONE PHASE prepares and commits the transaction. The *xid* value is not listed by XA RECOVER because the transaction terminates.
4. For a PREPARED XA transaction, you can issue an XA COMMIT statement to commit and terminate the transaction, or XA ROLLBACK to roll back and terminate the transaction.

Here is a simple XA transaction that inserts a row into a table as part of a global transaction:

```
mysql> XA START 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO mytable (i) VALUES(10);
Query OK, 1 row affected (0.04 sec)

mysql> XA END 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> XA PREPARE 'xatest';
Query OK, 0 rows affected (0.00 sec)

mysql> XA COMMIT 'xatest';
Query OK, 0 rows affected (0.00 sec)
```

Within the context of a given client connection, XA transactions and local (non-XA) transactions are mutually exclusive. For example, if `XA START` has been issued to begin an XA transaction, a local transaction cannot be started until the XA transaction has been committed or rolled back. Conversely, if a local transaction has been started with `START TRANSACTION`, no XA statements can be used until the transaction has been committed or rolled back.

If an XA transaction is in the `ACTIVE` state, you cannot issue any statements that cause an implicit commit. That would violate the XA contract because you could not roll back the XA transaction. You receive the following error if you try to execute such a statement:

```
ERROR 1399 (XAE07): XAER_RMFAIL: The command cannot be executed
when global transaction is in the ACTIVE state
```

Statements to which the preceding remark applies are listed at Section 13.3.3, “Statements That Cause an Implicit Commit”.

13.3.7.3 Restrictions on XA Transactions

XA transaction support is limited to the `InnoDB` storage engine.

For “external XA,” a MySQL server acts as a Resource Manager and client programs act as Transaction Managers. For “Internal XA”, storage engines within a MySQL server act as RMs, and the server itself acts as a TM. Internal XA support is limited by the capabilities of individual storage engines. Internal XA is required for handling XA transactions that involve more than one storage engine. The implementation of internal XA requires that a storage engine support two-phase commit at the table handler level, and currently this is true only for `InnoDB`.

For `XA START`, the `JOIN` and `RESUME` clauses are recognized but have no effect.

For `XA END` the `SUSPEND [FOR MIGRATE]` clause is recognized but has no effect.

The requirement that the *`bqual`* part of the *`xid`* value be different for each XA transaction within a global transaction is a limitation of the current MySQL XA implementation. It is not part of the XA specification.

If an XA transaction has reached the `PREPARED` state and the MySQL server is killed (for example, with `kill -9` on Unix) or shuts down abnormally, the transaction can be continued after the server restarts. However, if the client reconnects and commits the transaction, the transaction is absent from the binary log even though it has been committed. This means the data and the binary log have gone out of synchrony. An implication is that XA cannot be used safely together with replication.

It is possible that the server rolls back a pending XA transaction, even one that has reached the `PREPARED` state. This happens if a client connection terminates and the server continues to run, or if clients are connected and the server shuts down gracefully. (In the latter case, the server marks each connection to be terminated, and then rolls back the `PREPARED` XA transaction associated with it.) It should be possible to commit or roll back a `PREPARED` XA transaction, but this cannot be done without changes to the binary logging mechanism.

`FLUSH TABLES WITH READ LOCK` is not compatible with XA transactions.

13.4 Replication Statements

Replication can be controlled through the SQL interface using the statements described in this section. One group of statements controls replication source servers, the other controls replicas.

13.4.1 SQL Statements for Controlling Replication Source Servers

This section discusses statements for managing replication source servers. Section 13.4.2, “SQL Statements for Controlling Replica Servers”, discusses statements for managing replica servers.

In addition to the statements described here, the following `SHOW` statements are used with replication source servers. For information about these statements, see Section 13.7.5, “SHOW Statements”.

- `SHOW BINARY LOGS`
- `SHOW BINLOG EVENTS`
- `SHOW MASTER STATUS`
- `SHOW SLAVE HOSTS`

13.4.1.1 PURGE BINARY LOGS Statement

```
PURGE { BINARY | MASTER } LOGS {
    TO 'log_name'
  | BEFORE datetime_expr
}
```

The binary log is a set of files that contain information about data modifications made by the MySQL server. The log consists of a set of binary log files, plus an index file (see Section 5.4.4, “The Binary Log”).

The `PURGE BINARY LOGS` statement deletes all the binary log files listed in the log index file prior to the specified log file name or date. `BINARY` and `MASTER` are synonyms. Deleted log files also are removed from the list recorded in the index file, so that the given log file becomes the first in the list.

This statement has no effect if the server was not started with the `--log-bin` option to enable binary logging.

Examples:

```
PURGE BINARY LOGS TO 'mysql-bin.010';
PURGE BINARY LOGS BEFORE '2019-04-02 22:46:26';
```

The `BEFORE` variant's *`datetime_expr`* argument should evaluate to a `DATETIME` value (a value in *`'YYYY-MM-DD hh:mm:ss'`* format).

This statement is safe to run while replicas are replicating. You need not stop them. If you have an active replica that currently is reading one of the log files you are trying to delete, this statement does not delete the log file that is in use or any log files later than that one, but it deletes any earlier log files. In MySQL 5.6.12 and later, a warning message is issued in this situation. (Bug #13727933) However, if a replica is not connected and you happen to purge one of the log files it has yet to read, the replica cannot replicate after it reconnects.

To safely purge binary log files, follow this procedure:

1. On each replica server, use `SHOW SLAVE STATUS` to check which log file it is reading.
2. Obtain a listing of the binary log files on the source with `SHOW BINARY LOGS`.

3. Determine the earliest log file among all the replicas. This is the target file. If all the replicas are up to date, this is the last log file on the list.

4. Make a backup of all the log files you are about to delete. (This step is optional, but always advisable.)

5. Purge all log files up to but not including the target file.

You can also set the `expire_logs_days` system variable to expire binary log files automatically after a given number of days (see Section 5.1.7, “Server System Variables”). If you are using replication, you should set the variable no lower than the maximum number of days your replicas might lag behind the source.

`PURGE BINARY LOGS TO` and `PURGE BINARY LOGS BEFORE` both fail with an error when binary log files listed in the `.index` file had been removed from the system by some other means (such as using `rm` on Linux). (Bug #18199, Bug #18453) To handle such errors, edit the `.index` file (which is a simple text file) manually to ensure that it lists only the binary log files that are actually present, then run again the `PURGE BINARY LOGS` statement that failed.

13.4.1.2 RESET MASTER Statement

```
RESET MASTER
```

Deletes all binary log files listed in the index file, resets the binary log index file to be empty, and creates a new binary log file.

`RESET MASTER` also clears the values of the `gtid_purged` system variable as well as the global value of the `gtid_executed` system variable (but not its session value); that is, executing this statement sets each of these values to an empty string (`''`).

This statement is intended to be used only when the source is started for the first time.

Important

The effects of `RESET MASTER` differ from those of `PURGE BINARY LOGS` in 2 key ways:

1. `RESET MASTER` removes *all* binary log files that are listed in the index file, leaving only a single, empty binary log file with a numeric suffix of `.000001`, whereas the numbering is not reset by `PURGE BINARY LOGS`.

2. `RESET MASTER` is *not* intended to be used while any replicas are running. The behavior of `RESET MASTER` when used while replicas are running is undefined (and thus unsupported), whereas `PURGE BINARY LOGS` may be safely used while replicas are running.

See also Section 13.4.1.1, “PURGE BINARY LOGS Statement”.

`RESET MASTER` can prove useful when you first set up the source and the replica, so that you can verify the setup as follows:

1. Start the source and replica, and start replication (see Section 17.1.1, “How to Set Up Replication”).

2. Execute a few test queries on the source.

3. Check that the queries were replicated to the replica.

4. When replication is running correctly, issue `STOP SLAVE` followed by `RESET SLAVE` on the replica, then verify that any unwanted data no longer exists on the replica.

5. Issue `RESET MASTER` on the source to clean up the test queries.

After verifying the setup and getting rid of any unwanted and log files generated by testing, you can start the replica and begin replicating.

13.4.1.3 SET sql_log_bin Statement

```
SET sql_log_bin = {OFF|ON}
```

The `sql_log_bin` variable controls whether logging to the binary log is enabled for the current session (assuming that the binary log itself is enabled). The default value is `ON`. To disable or enable binary logging for the current session, set the session `sql_log_bin` variable to `OFF` or `ON`.

Set this variable to `OFF` for a session to temporarily disable binary logging while making changes to the source you do not want replicated to the replica.

Setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.8.1, “System Variable Privileges”.

It is not possible to set the session value of `sql_log_bin` within a transaction or subquery.

Setting this variable to `OFF` prevents GTIDs from being assigned to transactions in the binary log. If you are using GTIDs for replication, this means that, even when binary logging is later enabled once again, the GTIDs written into the log from this point do not account for any transactions that occurred in the meantime—in effect, those transactions are lost.

As of MySQL 5.6.22, the global `sql_log_bin` variable is read only and cannot be modified. The global scope is deprecated; expect it to be removed in a future MySQL release. Prior to 5.6.22, `sql_log_bin` can be set as a global or session variable. Setting `sql_log_bin` globally is only detected when a new session is started. Any sessions previously running are not impacted when setting `sql_log_bin` globally.

Warning

Incorrect use of `sql_log_bin` with a global scope means any changes made in an already running session are *still* being recorded to the binary log and therefore replicated. Exercise extreme caution using `sql_log_bin` with a global scope as the above situation could cause unexpected results including replication failure.

13.4.2 SQL Statements for Controlling Replica Servers

This section discusses statements for managing replica servers. Section 13.4.1, “SQL Statements for Controlling Replication Source Servers”, discusses statements for managing source servers.

In addition to the statements described here, `SHOW SLAVE STATUS` and `SHOW RELAYLOG EVENTS` are also used with replicas. For information about these statements, see Section 13.7.5.35, “SHOW SLAVE STATUS Statement”, and Section 13.7.5.33, “SHOW RELAYLOG EVENTS Statement”.

13.4.2.1 CHANGE MASTER TO Statement

```
CHANGE MASTER TO option [, option] ...

option: {
    MASTER_BIND = 'interface_name'
  | MASTER_HOST = 'host_name'
  | MASTER_USER = 'user_name'
  | MASTER_PASSWORD = 'password'
  | MASTER_PORT = port_num
  | MASTER_CONNECT_RETRY = interval
  | MASTER_RETRY_COUNT = count
  | MASTER_DELAY = interval
  | MASTER_HEARTBEAT_PERIOD = interval
```

```
  | MASTER_LOG_FILE = 'source_log_name'
  | MASTER_LOG_POS = source_log_pos
  | MASTER_AUTO_POSITION = {0|1}
  | RELAY_LOG_FILE = 'relay_log_name'
  | RELAY_LOG_POS = relay_log_pos
  | MASTER_SSL = {0|1}
  | MASTER_SSL_CA = 'ca_file_name'
  | MASTER_SSL_CAPATH = 'ca_directory_name'
  | MASTER_SSL_CERT = 'cert_file_name'
  | MASTER_SSL_CRL = 'crl_file_name'
  | MASTER_SSL_CRLPATH = 'crl_directory_name'
  | MASTER_SSL_KEY = 'key_file_name'
  | MASTER_SSL_CIPHER = 'cipher_list'
  | MASTER_SSL_VERIFY_SERVER_CERT = {0|1}
  | IGNORE_SERVER_IDS = (server_id_list)
}

server_id_list:
    [server_id [, server_id] ... ]
```

`CHANGE MASTER TO` changes the parameters that the replica uses for connecting to the replication source server, for reading the source's binary log, and reading the replica's relay log. It also updates the contents of the replication metadata repositories (see Section 17.2.2, “Relay Log and Replication Metadata Repositories”). `CHANGE MASTER TO` requires the `SUPER` privilege.

To use `CHANGE MASTER TO`, the replication threads must be stopped (use `STOP SLAVE` if necessary). In MySQL 5.6.11 and later, `gtid_next` must also be set to `AUTOMATIC` (Bug #16062608).

Options not specified retain their value, except as indicated in the following discussion. Thus, in most cases, there is no need to specify options that do not change. For example, if the password to connect to your MySQL source has changed, issue these statements to tell the replica about the new password:

```
STOP SLAVE; -- if replication was running
CHANGE MASTER TO MASTER_PASSWORD='new3cret';
START SLAVE; -- if you want to restart replication
```

`MASTER_HOST`, `MASTER_USER`, `MASTER_PASSWORD`, and `MASTER_PORT` provide information to the replica about how to connect to its source:

- `MASTER_HOST` and `MASTER_PORT` are the host name (or IP address) of the source host and its TCP/IP port.

  Note

  Replication cannot use Unix socket files. You must be able to connect to the source MySQL server using TCP/IP.

  If you specify the `MASTER_HOST` or `MASTER_PORT` option, the replica assumes that the source server is different from before (even if the option value is the same as its current value). In this case, the old values for the source's binary log file name and position are considered no longer applicable, so if you do not specify `MASTER_LOG_FILE` and `MASTER_LOG_POS` in the statement, `MASTER_LOG_FILE=''` and `MASTER_LOG_POS=4` are silently appended to it.

  Setting `MASTER_HOST=''` (that is, setting its value explicitly to an empty string) is *not* the same as not setting `MASTER_HOST` at all. Beginning with MySQL 5.5, trying to set `MASTER_HOST` to an empty string fails with an error. Previously, setting `MASTER_HOST` to an empty string caused `START SLAVE` subsequently to fail. (Bug #28796)

  In MySQL 5.6.5 and later, values used for `MASTER_HOST` and other `CHANGE MASTER TO` options are checked for linefeed (`\n` or `0x0A`) characters; the presence of such characters in these values causes the statement to fail with `ER_MASTER_INFO`. (Bug #11758581, Bug #50801)

- `MASTER_USER` and `MASTER_PASSWORD` are the user name and password of the account to use for connecting to the source. If you specify `MASTER_PASSWORD`, `MASTER_USER` is also required. The

password used for a replication user account in a `CHANGE MASTER TO` statement is limited to 32 characters in length; if the password is longer, the statement succeeds, but any excess characters are silently truncated. This is an issue specific to MySQL Replication, which is fixed in MySQL 5.7. (Bug #11752299, Bug #43439)

It is possible to set an empty user name by specifying `MASTER_USER=''`, but the replication channel cannot be started with an empty user name. Only set an empty `MASTER_USER` user name if you need to clear previously used credentials from the replica's repositories for security purposes, and do not attempt to use the channel afterwards. (Bug #13427949)

The text of a running `CHANGE MASTER TO` statement, including values for `MASTER_USER` and `MASTER_PASSWORD`, can be seen in the output of a concurrent `SHOW PROCESSLIST` statement. (The complete text of a `START SLAVE` statement is also visible to `SHOW PROCESSLIST`.)

The `MASTER_SSL_xxx` options provide information about using SSL for the connection. They correspond to the `--ssl-xxx` options described in Command Options for Encrypted Connections, and Section 17.3.8, “Setting Up Replication to Use Encrypted Connections”. These options can be changed even on replicas that are compiled without SSL support. They are saved to the connection metadata repository, but are ignored if the replica does not have SSL support enabled. `MASTER_SSL_CRL` and `MASTER_SSL_CRLPATH` were added in MySQL 5.6.3.

`MASTER_CONNECT_RETRY` specifies how many seconds to wait between connect retries. The default is 60.

`MASTER_RETRY_COUNT`, added in MySQL 5.6.1, limits the *number* of reconnection attempts and updates the value of the `Master_Retry_Count` column in the output of `SHOW SLAVE STATUS` (also added in MySQL 5.6.1). The default value is 24 * 3600 = 86400. `MASTER_RETRY_COUNT` is intended to replace the older `--master-retry-count` server option, and is now the preferred method for setting this limit. You are encouraged not to rely on `--master-retry-count` in new applications and, when upgrading to MySQL 5.6.1 or later from earlier versions of MySQL, to update any existing applications that rely on it, so that they use `CHANGE MASTER TO ... MASTER_RETRY_COUNT` instead.

`MASTER_DELAY` specifies how many seconds behind the source the replica must lag. An event received from the source is not executed until at least *`interval`* seconds later than its execution on the source. The default is 0. An error occurs if *`interval`* is not a nonnegative integer in the range from 0 to $2^{31}-1$. For more information, see Section 17.3.10, “Delayed Replication”. This option was added in MySQL 5.6.0.

`MASTER_BIND` is for use on replicas having multiple network interfaces, and determines which of the replica's network interfaces is chosen for connecting to the source.

The address configured with this option, if any, can be seen in the `Master_Bind` column of the output from `SHOW SLAVE STATUS`. If you are using tables for the replication metadata repositories (the server was started with `master_info_repository=TABLE`), the value can also be seen as the `Master_bind` column of the `mysql.slave_master_info` table.

The ability to bind a replica to a specific network interface was added in MySQL 5.6.2. This is also supported by MySQL NDB Cluster 7.3.1 and later.

`MASTER_HEARTBEAT_PERIOD` sets the interval in seconds between replication heartbeats. Whenever the source's binary log is updated with an event, the waiting period for the next heartbeat is reset. *`interval`* is a decimal value having the range 0 to 4294967 seconds and a resolution in milliseconds; the smallest nonzero value is 0.001. Heartbeats are sent by the source only if there are no unsent events in the binary log file for a period longer than *`interval`*.

If you are logging source connection information to tables, `MASTER_HEARTBEAT_PERIOD` can be seen as the value of the `Heartbeat` column of the `mysql.slave_master_info` table.

Setting *`interval`* to 0 disables heartbeats altogether. The default value for *`interval`* is equal to the value of `slave_net_timeout` divided by 2.

Setting `@@GLOBAL.slave_net_timeout` to a value less than that of the current heartbeat interval results in a warning being issued. The effect of issuing `RESET SLAVE` on the heartbeat interval is to reset it to the default value.

`MASTER_LOG_FILE` and `MASTER_LOG_POS` are the coordinates at which the replication I/O thread should begin reading from the source the next time the thread starts. `RELAY_LOG_FILE` and `RELAY_LOG_POS` are the coordinates at which the replication SQL thread should begin reading from the relay log the next time the thread starts. If you specify either of `MASTER_LOG_FILE` or `MASTER_LOG_POS`, you cannot specify `RELAY_LOG_FILE` or `RELAY_LOG_POS`. In MySQL 5.6.5 and later, if you specify either of `MASTER_LOG_FILE` or `MASTER_LOG_POS`, you also cannot specify `MASTER_AUTO_POSITION = 1` (described later in this section). If neither of `MASTER_LOG_FILE` or `MASTER_LOG_POS` is specified, the replica uses the last coordinates of the *replication SQL thread* before `CHANGE MASTER TO` was issued. This ensures that there is no discontinuity in replication, even if the replication SQL thread was late compared to the replication I/O thread, when you merely want to change, say, the password to use.

`MASTER_AUTO_POSITION` was added in MySQL 5.6.5. If `MASTER_AUTO_POSITION = 1` is used with `CHANGE MASTER TO`, the replica attempts to connect to the source using the GTID-based replication protocol.

When using GTIDs, the replica tells the source which transactions it has already received, executed, or both. To compute this set, it reads the global value of `gtid_executed` and the value of the `Retrieved_gtid_set` column from `SHOW SLAVE STATUS`. Since the GTID of the last transmitted transaction is included in `Retrieved_gtid_set` even if the transaction was only partially transmitted, the last received GTID is subtracted from this set. Thus, the replica computes the following set:

```
UNION(@@GLOBAL.gtid_executed, Retrieved_gtid_set - last_received_GTID)
```

This set is sent to the source as part of the initial handshake, and the source sends back all transactions that it has executed which are not part of the set. If any of these transactions have been already purged from the source's binary log, the source sends the error `ER_MASTER_HAS_PURGED_REQUIRED_GTIDS` to the replica, and replication does not start.

When GTID-based replication is employed, the coordinates represented by `MASTER_LOG_FILE` and `MASTER_LOG_POS` are not used, and global transaction identifiers are used instead. Thus the use of either or both of these options together with `MASTER_AUTO_POSITION` causes an error.

Beginning with MySQL 5.6.10, you can see whether replication is running with autopositioning enabled by checking the output of `SHOW SLAVE STATUS`. (Bug #15992220)

`gtid_mode` must also be enabled before issuing `CHANGE MASTER TO ... MASTER_AUTO_POSITION = 1`. Otherwise, the statement fails with an error.

To revert to the older file-based replication protocol after using GTIDs, you can issue a new `CHANGE MASTER TO` statement that specifies `MASTER_AUTO_POSITION = 0`, as well as at least one of `MASTER_LOG_FILE` or `MASTER_LOG_POS`.

`CHANGE MASTER TO` *deletes all relay log files* and starts a new one, unless you specify `RELAY_LOG_FILE` or `RELAY_LOG_POS`. In that case, relay log files are kept; the `relay_log_purge` global variable is set silently to 0.

Prior to MySQL 5.6.2, `RELAY_LOG_FILE` required an absolute path. Beginning with MySQL 5.6.2, the path can be relative, in which case it is assumed to be relative to the replica's data directory. (Bug #12190)

`IGNORE_SERVER_IDS` takes a comma-separated list of 0 or more server IDs. Events originating from the corresponding servers are ignored, with the exception of log rotation and deletion events, which are still recorded in the relay log.

In circular replication, the originating server normally acts as the terminator of its own events, so that they are not applied more than once. Thus, this option is useful in circular replication when one of the

servers in the circle is removed. Suppose that you have a circular replication setup with 4 servers, having server IDs 1, 2, 3, and 4, and server 3 fails. When bridging the gap by starting replication from server 2 to server 4, you can include `IGNORE_SERVER_IDS = (3)` in the `CHANGE MASTER TO` statement that you issue on server 4 to tell it to use server 2 as its source instead of server 3. Doing so causes it to ignore and not to propagate any statements that originated with the server that is no longer in use.

When a `CHANGE MASTER TO` statement is issued without any `IGNORE_SERVER_IDS` option, any existing list is preserved. To clear the list of ignored servers, it is necessary to use the option with an empty list:

```
CHANGE MASTER TO IGNORE_SERVER_IDS = ();
```

`RESET SLAVE ALL` has no effect on the server ID list. This issue is fixed in MySQL 5.7. (Bug #18816897)

If `IGNORE_SERVER_IDS` contains the server's own ID and the server was started with the `--replicate-same-server-id` option enabled, an error results.

In MySQL 5.6, the connection metadata repository and the output of `SHOW SLAVE STATUS` provide the list of servers that are currently ignored. For more information, see Section 17.2.2.2, “Replication Metadata Repositories”, and Section 13.7.5.35, “SHOW SLAVE STATUS Statement”.

In MySQL 5.6, invoking `CHANGE MASTER TO` causes the previous values for `MASTER_HOST`, `MASTER_PORT`, `MASTER_LOG_FILE`, and `MASTER_LOG_POS` to be written to the error log, along with other information about the replica's state prior to execution.

In MySQL 5.6.7 and later, `CHANGE MASTER TO` causes an implicit commit of an ongoing transaction. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

`CHANGE MASTER TO` is useful for setting up a replica when you have the snapshot of the source and have recorded the source binary log coordinates corresponding to the time of the snapshot. After loading the snapshot into the replica to synchronize it with the source, you can run `CHANGE MASTER TO MASTER_LOG_FILE='log_name', MASTER_LOG_POS=log_pos` on the replica to specify the coordinates at which the replica should begin reading the source's binary log.

The following example changes the source server the replica uses and establishes the source's binary log coordinates from which the replica begins reading. This is used when you want to set up the replica to replicate the source:

```
CHANGE MASTER TO
  MASTER_HOST='source2.example.com',
  MASTER_USER='replication',
  MASTER_PASSWORD='bigs3cret',
  MASTER_PORT=3306,
  MASTER_LOG_FILE='source2-bin.001',
  MASTER_LOG_POS=4,
  MASTER_CONNECT_RETRY=10;
```

The next example shows an operation that is less frequently employed. It is used when the replica has relay log files that you want it to execute again for some reason. To do this, the source need not be reachable. You need only use `CHANGE MASTER TO` and start the SQL thread (`START SLAVE SQL_THREAD`):

```
CHANGE MASTER TO
  RELAY_LOG_FILE='replica-relay-bin.006',
  RELAY_LOG_POS=4025;
```

The following table shows the maximum permissible length for the string-valued options.

Option	Maximum Length
`MASTER_HOST`	60

Option	Maximum Length
MASTER_USER	16
MASTER_PASSWORD	32
MASTER_LOG_FILE	255
RELAY_LOG_FILE	255
MASTER_SSL_CA	255
MASTER_SSL_CAPATH	255
MASTER_SSL_CERT	255
MASTER_SSL_CRL	255
MASTER_SSL_CRLPATH	255
MASTER_SSL_KEY	255
MASTER_SSL_CIPHER	511

13.4.2.2 MASTER_POS_WAIT() Statement

```
SELECT MASTER_POS_WAIT('source_log_file', source_log_pos [, timeout])
```

This is actually a function, not a statement. It is used to ensure that the replica has read and executed events up to a given position in the source's binary log. See Section 12.20, “Miscellaneous Functions”, for a full description.

13.4.2.3 RESET SLAVE Statement

```
RESET SLAVE [ALL]
```

`RESET SLAVE` makes the replica forget its replication position in the source's binary log. This statement is meant to be used for a clean start: It clears the replication metadata repositories, deletes all the relay log files, and starts a new relay log file. It also resets to 0 the replication delay specified with the `MASTER_DELAY` option to `CHANGE MASTER TO`. `RESET SLAVE` does not change the values of `gtid_executed` or `gtid_purged`. To use `RESET SLAVE`, the replication threads must be stopped, so on a running replica use `STOP SLAVE` before issuing `RESET SLAVE`.

Note

All relay log files are deleted, even if they have not been completely executed by the replication SQL thread. (This is a condition likely to exist on a replica if you have issued a `STOP SLAVE` statement or if the replica is highly loaded.)

In MySQL 5.6 (unlike the case in MySQL 5.1 and earlier), `RESET SLAVE` does not change any replication connection parameters such as the source's host name and port or the replication user account name and password, which are retained in memory. This means that `START SLAVE` can be issued without requiring a `CHANGE MASTER TO` statement following `RESET SLAVE`.

Connection parameters are reset if the replica `mysqld` is shut down following `RESET SLAVE`. In MySQL 5.6.3 and later, you can instead use `RESET SLAVE ALL` to reset these connection parameters (Bug #11809016).

`RESET SLAVE ALL` does not clear the `IGNORE_SERVER_IDS` list set by `CHANGE MASTER TO`. This issue is fixed in MySQL 5.7. (Bug #18816897)

In MySQL 5.6.7 and later, `RESET SLAVE` causes an implicit commit of an ongoing transaction. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

If the replication SQL thread was in the middle of replicating temporary tables when it was stopped, and `RESET SLAVE` is issued, these replicated temporary tables are deleted on the replica.

Note

When used on an NDB Cluster replica SQL node, `RESET SLAVE` clears the `mysql.ndb_apply_status` table. You should keep in mind when using this statement that `ndb_apply_status` uses the `NDB` storage engine and so is shared by all SQL nodes attached to the replica cluster.

Beginning with MySQL NDB Cluster 7.4.9, you can override this behavior by issuing `SET GLOBAL @@ndb_clear_apply_status=OFF` prior to executing `RESET SLAVE`, which keeps the replica from purging the `ndb_apply_status` table in such cases.

13.4.2.4 SET GLOBAL sql_slave_skip_counter Statement

```
SET GLOBAL sql_slave_skip_counter = N
```

This statement skips the next *N* events from the source. This is useful for recovering from replication stops caused by a statement.

This statement is valid only when the replication threads are not running. Otherwise, it produces an error.

When using this statement, it is important to understand that the binary log is actually organized as a sequence of groups known as *event groups*. Each event group consists of a sequence of events.

- For transactional tables, an event group corresponds to a transaction.
- For nontransactional tables, an event group corresponds to a single SQL statement.

Note

A single transaction can contain changes to both transactional and nontransactional tables.

When you use `SET GLOBAL sql_slave_skip_counter` to skip events and the result is in the middle of a group, the replica continues to skip events until it reaches the end of the group. Execution then starts with the next event group.

13.4.2.5 START SLAVE Statement

```
START SLAVE [thread_types] [until_option] [connection_options]

thread_types:
    [thread_type [, thread_type] ... ]

thread_type:
    IO_THREAD | SQL_THREAD

until_option:
    UNTIL {   {SQL_BEFORE_GTIDS | SQL_AFTER_GTIDS} = gtid_set
          |   MASTER_LOG_FILE = 'log_name', MASTER_LOG_POS = log_pos
          |   RELAY_LOG_FILE = 'log_name', RELAY_LOG_POS = log_pos
          |   SQL_AFTER_MTS_GAPS  }

connection_options:
    [USER='user_name'] [PASSWORD='user_pass'] [DEFAULT_AUTH='plugin_name'] [PLUGIN_DIR='plugin_dir']


gtid_set:
    uuid_set [, uuid_set] ...
    | ''

uuid_set:
    uuid:interval[:interval]...
```

```
uuid:
    hhhhhhhh-hhhh-hhhh-hhhh-hhhhhhhhhhhh

h:
    [0-9,A-F]

interval:
    n[-n]

    (n >= 1)
```

`START SLAVE` with no *`thread_type`* options starts both of the replication threads. The I/O thread reads events from the replication source server and stores them in the relay log. The SQL thread reads events from the relay log and executes them. `START SLAVE` requires the `SUPER` privilege.

If `START SLAVE` succeeds in starting the replication threads, it returns without any error. However, even in that case, it might be that the replication threads start and then later stop (for example, because they do not manage to connect to the source or read its binary log, or some other problem). `START SLAVE` does not warn you about this. You must check the replica's error log for error messages generated by the replication threads, or check that they are running satisfactorily with `SHOW SLAVE STATUS`.

`START SLAVE` causes an implicit commit of an ongoing transaction. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

Beginning with MySQL 5.6.11, `gtid_next` must be set to `AUTOMATIC` before issuing this statement (Bug #16062608).

`START SLAVE` supports pluggable user-password authentication with the `USER`, `PASSWORD`, `DEFAULT_AUTH` and `PLUGIN_DIR` options, as described in the following list:

- `USER`: User name. Cannot be set to an empty or null string, or left unset if `PASSWORD` is used.
- `PASSWORD`: Password.
- `DEFAULT_AUTH`: Name of plugin; default is MySQL native authentication.
- `PLUGIN_DIR`: Location of plugin.

You cannot use the `SQL_THREAD` option when specifying any of `USER`, `PASSWORD`, `DEFAULT_AUTH`, or `PLUGIN_DIR`, unless the `IO_THREAD` option is also provided.

For more information, see Section 6.2.11, “Pluggable Authentication”.

If an insecure connection is used with any these options, the server issues the warning `Sending passwords in plain text without SSL/TLS is extremely insecure`.

`START SLAVE ... UNTIL` supports two additional options for use with global transaction identifiers (GTIDs) (see Section 17.1.3, “Replication with Global Transaction Identifiers”). Each of these takes a set of one or more global transaction identifiers *`gtid_set`* as an argument (see GTID Sets, for more information).

When no *`thread_type`* is specified, `START SLAVE UNTIL SQL_BEFORE_GTIDS` causes the replication SQL thread to process transactions until it has reached the *first* transaction whose GTID is listed in the *`gtid_set`*. `START SLAVE UNTIL SQL_AFTER_GTIDS` causes the replication threads to process all transactions until the *`last`* transaction in the *`gtid_set`* has been processed by both threads. In other words, `START SLAVE UNTIL SQL_BEFORE_GTIDS` causes the replication SQL thread to process all transactions occurring before the first GTID in the *`gtid_set`* is reached, and `START SLAVE UNTIL SQL_AFTER_GTIDS` causes the replication threads to handle all transactions, including those whose GTIDs are found in *`gtid_set`*, until each has encountered a transaction

whose GTID is not part of the set. `SQL_BEFORE_GTIDS` and `SQL_AFTER_GTIDS` each support the `SQL_THREAD` and `IO_THREAD` options, although using `IO_THREAD` with them currently has no effect.

For example, `START SLAVE SQL_THREAD UNTIL SQL_BEFORE_GTIDS = 3E11FA47-71CA-11E1-9E33-C80AA9429562:11-56` causes the replication SQL thread to process all transactions originating from the source whose `server_uuid` is `3E11FA47-71CA-11E1-9E33-C80AA9429562` until it encounters the transaction having sequence number 11; it then stops without processing this transaction. In other words, all transactions up to and including the transaction with sequence number 10 are processed. Executing `START SLAVE SQL_THREAD UNTIL SQL_AFTER_GTIDS = 3E11FA47-71CA-11E1-9E33-C80AA9429562:11-56`, on the other hand, would cause the replication SQL thread to obtain all transactions just mentioned from the source, including all of the transactions having the sequence numbers 11 through 56, and then to stop without processing any additional transactions; that is, the transaction having sequence number 56 would be the last transaction fetched by the replication SQL thread.

Prior to MySQL 5.6.14, `SQL_AFTER_GTIDS` did not stop the replica once the indicated transaction was completed, but waited until another GTID event was received (Bug #14767986).

Use the `START SLAVE UNTIL SQL_AFTER_MTS_GAPS` statement to cause a multithreaded replica's SQL threads to run until no more gaps are found in the relay log, and then to stop. This statement can take an `SQL_THREAD` option, but the effects of the statement remain unchanged. It has no effect on the replication I/O thread (and cannot be used with the `IO_THREAD` option). `START SLAVE UNTIL SQL_AFTER_MTS_GAPS` should be used before switching the replica from multithreaded mode to single-threaded mode (that is, when resetting `slave_parallel_workers` back to 0 from a positive, nonzero value) after the replica has failed with errors in multithreaded mode.

To change a failed multithreaded replica to single-threaded mode, you can issue the following series of statements, in the order shown:

```
START SLAVE UNTIL SQL_AFTER_MTS_GAPS;

SET @@GLOBAL.slave_parallel_workers = 0;

START SLAVE SQL_THREAD;
```

If you were running the failed multithreaded replica with `relay_log_recovery` enabled, then you must issue `START SLAVE UNTIL SQL_AFTER_MTS_GAPS` prior to executing `CHANGE MASTER TO`. Otherwise the latter statement fails.

Note

It is possible to view the entire text of a running `START SLAVE ...` statement, including any `USER` or `PASSWORD` values used, in the output of `SHOW PROCESSLIST`. This is also true for the text of a running `CHANGE MASTER TO` statement, including any values it employs for `MASTER_USER` or `MASTER_PASSWORD`.

`START SLAVE` sends an acknowledgment to the user after both the I/O thread and the SQL thread have started. However, the I/O thread may not yet have connected. For this reason, a successful `START SLAVE` causes `SHOW SLAVE STATUS` to show `Slave_SQL_Running=Yes`, but this does not guarantee that `Slave_IO_Running=Yes` (because `Slave_IO_Running=Yes` only if the I/O thread is running *and connected*). For more information, see Section 13.7.5.35, “SHOW SLAVE STATUS Statement”, and Section 17.1.5.1, “Checking Replication Status”.

You can add `IO_THREAD` and `SQL_THREAD` options to the statement to name which of the threads to start. The `SQL_THREAD` option is disallowed when specifying any of `USER`, `PASSWORD`, `DEFAULT_AUTH`, or `PLUGIN_DIR`, unless the `IO_THREAD` option is also provided.

An `UNTIL` clause (*`until_option`*, in the preceding grammar) may be added to specify that the replica should start and run until the SQL thread reaches a given point in the source's binary log, specified by the `MASTER_LOG_POS` and `MASTER_LOG_FILE` options, or a given point in the replica's relay log,

indicated with the RELAY_LOG_POS and RELAY_LOG_FILE options. When the SQL thread reaches the point specified, it stops. If the SQL_THREAD option is specified in the statement, it starts only the SQL thread. Otherwise, it starts both replication threads. If the SQL thread is running, the UNTIL clause is ignored and a warning is issued. You cannot use an UNTIL clause with the IO_THREAD option.

It is also possible with START SLAVE UNTIL to specify a stop point relative to a given GTID or set of GTIDs using one of the options SQL_BEFORE_GTIDS or SQL_AFTER_GTIDS, as explained previously in this section. When using one of these options, you can specify SQL_THREAD, IO_THREAD, both of these, or neither of them. If you specify only SQL_THREAD, then only the replication SQL thread is affected by the statement; if only IO_THREAD is used, then only the replication I/O thread is affected. If both SQL_THREAD and IO_THREAD are used, or if neither of them is used, then both the replication threads are affected by the statement.

The UNTIL clause is not supported for multithreaded replicas except when also using SQL_AFTER_MTS_GAPS.

For an UNTIL clause, you must specify any one of the following:

- *Both* a log file name and a position in that file
- *Either* of SQL_BEFORE_GTIDS or SQL_AFTER_GTIDS
- SQL_AFTER_MTS_GAPS

Do not mix binary log and relay log options. Do not mix log file options with GTID options.

Any UNTIL condition is reset by a subsequent STOP SLAVE statement, a START SLAVE statement that includes no UNTIL clause, or a server restart.

When specifying a log file and position, you can use the IO_THREAD option with START SLAVE ... UNTIL even though only the SQL thread is affected by this statement. The IO_THREAD option is ignored in such cases. The preceding restriction does not apply when using one of the GTID options (SQL_BEFORE_GTIDS and SQL_AFTER_GTIDS); the GTID options support both SQL_THREAD and IO_THREAD, as explained previously in this section.

The UNTIL clause can be useful for debugging replication, or to cause replication to proceed until just before the point where you want to avoid having the replica replicate an event. For example, if an unwise DROP TABLE statement was executed on the source, you can use UNTIL to tell the replica to execute up to that point but no farther. To find what the event is, use mysqlbinlog with the source's binary log or the replica's relay log, or by using a SHOW BINLOG EVENTS statement.

If you are using UNTIL to have the replica process replicated queries in sections, it is recommended that you start the replica with the --skip-slave-start option to prevent the SQL thread from running when the replica server starts. It is probably best to use this option in an option file rather than on the command line, so that an unexpected server restart does not cause it to be forgotten.

The SHOW SLAVE STATUS statement includes output fields that display the current values of the UNTIL condition.

13.4.2.6 STOP SLAVE Statement

```
STOP SLAVE [thread_types]

thread_types:
    [thread_type [, thread_type] ... ]

thread_type: IO_THREAD | SQL_THREAD
```

Stops the replication threads. STOP SLAVE requires the SUPER privilege. Recommended best practice is to execute STOP SLAVE on the replica before stopping the replica server (see Section 5.1.15, “The Server Shutdown Process”, for more information).

When using the row-based logging format: You should execute `STOP SLAVE` or `STOP SLAVE SQL_THREAD` on the replica prior to shutting down the replica server if you are replicating any tables that use a nontransactional storage engine (see the *Note* later in this section).

Like `START SLAVE`, this statement may be used with the `IO_THREAD` and `SQL_THREAD` options to name the thread or threads to be stopped.

`STOP SLAVE` causes an implicit commit of an ongoing transaction. See Section 13.3.3, "Statements That Cause an Implicit Commit".

Beginning with MySQL 5.6.11, `gtid_next` must be set to `AUTOMATIC` before issuing this statement (Bug #16062608).

In MySQL 5.6.13 and later, you can control how long `STOP SLAVE` waits before timing out by setting the `rpl_stop_slave_timeout` system variable. This can be used to avoid deadlocks between `STOP SLAVE` and other SQL statements using different client connections to the replica. When the timeout value is reached, the issuing client returns an error message and stops waiting, but the `STOP SLAVE` instruction remains in effect. Once the replication threads are no longer busy, the `STOP SLAVE` statement is executed and the replica stops. (Bug #16856735)

If the current replication event group has modified one or more nontransactional tables, `STOP SLAVE` waits for up to 60 seconds for the event group to complete, unless you issue a `KILL QUERY` or `KILL CONNECTION` statement for the replication SQL thread. If the event group remains incomplete after the timeout, an error message is logged. (Bug #319, Bug #38205)

In old versions of MySQL (before 4.0.5), this statement was called `SLAVE STOP`. That syntax is no longer accepted as of MySQL 5.6.1.

13.5 Prepared Statements

MySQL 5.6 provides support for server-side prepared statements. This support takes advantage of the efficient client/server binary protocol. Using prepared statements with placeholders for parameter values has the following benefits:

- Less overhead for parsing the statement each time it is executed. Typically, database applications process large volumes of almost-identical statements, with only changes to literal or variable values in clauses such as `WHERE` for queries and deletes, `SET` for updates, and `VALUES` for inserts.
- Protection against SQL injection attacks. The parameter values can contain unescaped SQL quote and delimiter characters.

The following sections provide an overview of the characteristics of prepared statements:

- Prepared Statements in Application Programs
- Prepared Statements in SQL Scripts
- PREPARE, EXECUTE, and DEALLOCATE PREPARE Statements
- SQL Syntax Permitted in Prepared Statements

Prepared Statements in Application Programs

You can use server-side prepared statements through client programming interfaces, including the MySQL C API client library for C programs, MySQL Connector/J for Java programs, and MySQL Connector/NET for programs using .NET technologies. For example, the C API provides a set of function calls that make up its prepared statement API. See C API Prepared Statements. Other language interfaces can provide support for prepared statements that use the binary protocol by linking in the C client library, one example being the `mysqli` extension, available in PHP 5.0 and higher.

Prepared Statements in SQL Scripts

An alternative SQL interface to prepared statements is available. This interface is not as efficient as using the binary protocol through a prepared statement API, but requires no programming because it is available directly at the SQL level:

- You can use it when no programming interface is available to you.
- You can use it from any program that can send SQL statements to the server to be executed, such as the `mysql` client program.
- You can use it even if the client is using an old version of the client library.

SQL syntax for prepared statements is intended to be used for situations such as these:

- To test how prepared statements work in your application before coding it.
- To use prepared statements when you do not have access to a programming API that supports them.
- To interactively troubleshoot application issues with prepared statements.
- To create a test case that reproduces a problem with prepared statements, so that you can file a bug report.

PREPARE, EXECUTE, and DEALLOCATE PREPARE Statements

SQL syntax for prepared statements is based on three SQL statements:

- `PREPARE` prepares a statement for execution (see Section 13.5.1, “PREPARE Statement”).
- `EXECUTE` executes a prepared statement (see Section 13.5.2, “EXECUTE Statement”).
- `DEALLOCATE PREPARE` releases a prepared statement (see Section 13.5.3, “DEALLOCATE PREPARE Statement”).

The following examples show two equivalent ways of preparing a statement that computes the hypotenuse of a triangle given the lengths of the two sides.

The first example shows how to create a prepared statement by using a string literal to supply the text of the statement:

```
mysql> PREPARE stmt1 FROM 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> SET @a = 3;
mysql> SET @b = 4;
mysql> EXECUTE stmt1 USING @a, @b;
+------------+
| hypotenuse |
+------------+
|          5 |
+------------+
mysql> DEALLOCATE PREPARE stmt1;
```

The second example is similar, but supplies the text of the statement as a user variable:

```
mysql> SET @s = 'SELECT SQRT(POW(?,2) + POW(?,2)) AS hypotenuse';
mysql> PREPARE stmt2 FROM @s;
mysql> SET @a = 6;
mysql> SET @b = 8;
mysql> EXECUTE stmt2 USING @a, @b;
+------------+
| hypotenuse |
+------------+
```

```
|         10 |
+------------+
mysql> DEALLOCATE PREPARE stmt2;
```

Here is an additional example that demonstrates how to choose the table on which to perform a query at runtime, by storing the name of the table as a user variable:

```
mysql> USE test;
mysql> CREATE TABLE t1 (a INT NOT NULL);
mysql> INSERT INTO t1 VALUES (4), (8), (11), (32), (80);

mysql> SET @table = 't1';
mysql> SET @s = CONCAT('SELECT * FROM ', @table);

mysql> PREPARE stmt3 FROM @s;
mysql> EXECUTE stmt3;
+----+
| a  |
+----+
|  4 |
|  8 |
| 11 |
| 32 |
| 80 |
+----+

mysql> DEALLOCATE PREPARE stmt3;
```

A prepared statement is specific to the session in which it was created. If you terminate a session without deallocating a previously prepared statement, the server deallocates it automatically.

A prepared statement is also global to the session. If you create a prepared statement within a stored routine, it is not deallocated when the stored routine ends.

To guard against too many prepared statements being created simultaneously, set the `max_prepared_stmt_count` system variable. To prevent the use of prepared statements, set the value to 0.

SQL Syntax Permitted in Prepared Statements

The following SQL statements can be used as prepared statements:

```
ALTER TABLE
ALTER USER
ANALYZE TABLE
CACHE INDEX
CALL
CHANGE MASTER
CHECKSUM {TABLE | TABLES}
COMMIT
{CREATE | RENAME | DROP} DATABASE
{CREATE | DROP} INDEX
{CREATE | RENAME | DROP} TABLE
{CREATE | RENAME | DROP} USER
{CREATE | DROP} VIEW
DELETE
DO
FLUSH {TABLE | TABLES | TABLES WITH READ LOCK | HOSTS | PRIVILEGES
  | LOGS | STATUS | MASTER | SLAVE | DES_KEY_FILE | USER_RESOURCES}
GRANT
INSERT
INSTALL PLUGIN
KILL
LOAD INDEX INTO CACHE
OPTIMIZE TABLE
REPAIR TABLE
REPLACE
RESET {MASTER | SLAVE | QUERY CACHE}
```

```
REVOKE
SELECT
SET
SHOW {AUTHORS | CONTRIBUTORS | WARNINGS | ERRORS}
SHOW BINLOG EVENTS
SHOW CREATE {PROCEDURE | FUNCTION | EVENT | TABLE | VIEW}
SHOW {MASTER | BINARY} LOGS
SHOW {MASTER | SLAVE} STATUS
SLAVE {START | STOP}
TRUNCATE TABLE
UNINSTALL PLUGIN
UPDATE
```

Other statements are not supported.

Generally, statements not permitted in SQL prepared statements are also not permitted in stored programs. Exceptions are noted in Section 20.8, “Restrictions on Stored Programs”.

Metadata changes to tables or views referred to by prepared statements are detected and cause automatic repreparation of the statement when it is next executed. For more information, see Section 8.10.4, “Caching of Prepared Statements and Stored Programs”.

Placeholders can be used for the arguments of the `LIMIT` clause when using prepared statements. See Section 13.2.9, “SELECT Statement”.

In prepared `CALL` statements used with `PREPARE` and `EXECUTE`, placeholder support for `OUT` and `INOUT` parameters is available beginning with MySQL 5.6. See Section 13.2.1, “CALL Statement”, for an example and a workaround for earlier versions. Placeholders can be used for `IN` parameters regardless of version.

SQL syntax for prepared statements cannot be used in nested fashion. That is, a statement passed to `PREPARE` cannot itself be a `PREPARE`, `EXECUTE`, or `DEALLOCATE PREPARE` statement.

SQL syntax for prepared statements is distinct from using prepared statement API calls. For example, you cannot use the `mysql_stmt_prepare()` C API function to prepare a `PREPARE`, `EXECUTE`, or `DEALLOCATE PREPARE` statement.

SQL syntax for prepared statements can be used within stored procedures, but not in stored functions or triggers. However, a cursor cannot be used for a dynamic statement that is prepared and executed with `PREPARE` and `EXECUTE`. The statement for a cursor is checked at cursor creation time, so the statement cannot be dynamic.

SQL syntax for prepared statements does not support multi-statements (that is, multiple statements within a single string separated by `;` characters).

Prepared statements use the query cache under the conditions described in Section 8.10.3.1, “How the Query Cache Operates”.

To write C programs that use the `CALL` SQL statement to execute stored procedures that contain prepared statements, the `CLIENT_MULTI_RESULTS` flag must be enabled. This is because each `CALL` returns a result to indicate the call status, in addition to any result sets that might be returned by statements executed within the procedure.

`CLIENT_MULTI_RESULTS` can be enabled when you call `mysql_real_connect()`, either explicitly by passing the `CLIENT_MULTI_RESULTS` flag itself, or implicitly by passing `CLIENT_MULTI_STATEMENTS` (which also enables `CLIENT_MULTI_RESULTS`). For additional information, see Section 13.2.1, “CALL Statement”.

13.5.1 PREPARE Statement

```
PREPARE stmt_name FROM preparable_stmt
```

The PREPARE statement prepares a SQL statement and assigns it a name, *stmt_name*, by which to refer to the statement later. The prepared statement is executed with EXECUTE and released with DEALLOCATE PREPARE. For examples, see Section 13.5, “Prepared Statements”.

Statement names are not case-sensitive. *preparable_stmt* is either a string literal or a user variable that contains the text of the SQL statement. The text must represent a single statement, not multiple statements. Within the statement, ? characters can be used as parameter markers to indicate where data values are to be bound to the query later when you execute it. The ? characters should not be enclosed within quotation marks, even if you intend to bind them to string values. Parameter markers can be used only where data values should appear, not for SQL keywords, identifiers, and so forth.

If a prepared statement with the given name already exists, it is deallocated implicitly before the new statement is prepared. This means that if the new statement contains an error and cannot be prepared, an error is returned and no statement with the given name exists.

The scope of a prepared statement is the session within which it is created, which as several implications:

- A prepared statement created in one session is not available to other sessions.
- When a session ends, whether normally or abnormally, its prepared statements no longer exist. If auto-reconnect is enabled, the client is not notified that the connection was lost. For this reason, clients may wish to disable auto-reconnect. See C API Automatic Reconnection Control.
- A prepared statement created within a stored program continues to exist after the program finishes executing and can be executed outside the program later.
- A statement prepared in stored program context cannot refer to stored procedure or function parameters or local variables because they go out of scope when the program ends and would be unavailable were the statement to be executed later outside the program. As a workaround, refer instead to user-defined variables, which also have session scope; see Section 9.4, “User-Defined Variables”.

13.5.2 EXECUTE Statement

```
EXECUTE stmt_name
    [USING @var_name [, @var_name] ...]
```

After preparing a statement with PREPARE, you execute it with an EXECUTE statement that refers to the prepared statement name. If the prepared statement contains any parameter markers, you must supply a USING clause that lists user variables containing the values to be bound to the parameters. Parameter values can be supplied only by user variables, and the USING clause must name exactly as many variables as the number of parameter markers in the statement.

You can execute a given prepared statement multiple times, passing different variables to it or setting the variables to different values before each execution.

For examples, see Section 13.5, “Prepared Statements”.

13.5.3 DEALLOCATE PREPARE Statement

```
{DEALLOCATE | DROP} PREPARE stmt_name
```

To deallocate a prepared statement produced with PREPARE, use a DEALLOCATE PREPARE statement that refers to the prepared statement name. Attempting to execute a prepared statement after deallocating it results in an error. If too many prepared statements are created and not deallocated by either the DEALLOCATE PREPARE statement or the end of the session, you might encounter the upper limit enforced by the max_prepared_stmt_count system variable.

For examples, see Section 13.5, “Prepared Statements”.

13.6 Compound Statements

This section describes the syntax for the `BEGIN ... END` compound statement and other statements that can be used in the body of stored programs: Stored procedures and functions, triggers, and events. These objects are defined in terms of SQL code that is stored on the server for later invocation (see Chapter 20, *Stored Objects*).

A compound statement is a block that can contain other blocks; declarations for variables, condition handlers, and cursors; and flow control constructs such as loops and conditional tests.

13.6.1 BEGIN ... END Compound Statement

```
[begin_label:] BEGIN
    [statement_list]
END [end_label]
```

`BEGIN ... END` syntax is used for writing compound statements, which can appear within stored programs (stored procedures and functions, triggers, and events). A compound statement can contain multiple statements, enclosed by the `BEGIN` and `END` keywords. *`statement_list`* represents a list of one or more statements, each terminated by a semicolon (`;`) statement delimiter. The *`statement_list`* itself is optional, so the empty compound statement (`BEGIN END`) is legal.

`BEGIN ... END` blocks can be nested.

Use of multiple statements requires that a client is able to send statement strings containing the `;` statement delimiter. In the `mysql` command-line client, this is handled with the `delimiter` command. Changing the `;` end-of-statement delimiter (for example, to `//`) permit `;` to be used in a program body. For an example, see Section 20.1, “Defining Stored Programs”.

A `BEGIN ... END` block can be labeled. See Section 13.6.2, “Statement Labels”.

The optional `[NOT] ATOMIC` clause is not supported. This means that no transactional savepoint is set at the start of the instruction block and the `BEGIN` clause used in this context has no effect on the current transaction.

Note

Within all stored programs, the parser treats `BEGIN [WORK]` as the beginning of a `BEGIN ... END` block. To begin a transaction in this context, use `START TRANSACTION` instead.

13.6.2 Statement Labels

```
[begin_label:] BEGIN
    [statement_list]
END [end_label]

[begin_label:] LOOP
    statement_list
END LOOP [end_label]

[begin_label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [end_label]

[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

Labels are permitted for `BEGIN ... END` blocks and for the `LOOP`, `REPEAT`, and `WHILE` statements. Label use for those statements follows these rules:

- *`begin_label`* must be followed by a colon.

- *`begin_label`* can be given without *`end_label`*. If *`end_label`* is present, it must be the same as *`begin_label`*.
- *`end_label`* cannot be given without *`begin_label`*.
- Labels at the same nesting level must be distinct.
- Labels can be up to 16 characters long.

To refer to a label within the labeled construct, use an `ITERATE` or `LEAVE` statement. The following example uses those statements to continue iterating or terminate the loop:

```
CREATE PROCEDURE doiterate(p1 INT)
BEGIN
  label1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN ITERATE label1; END IF;
    LEAVE label1;
  END LOOP label1;
END;
```

The scope of a block label does not include the code for handlers declared within the block. For details, see Section 13.6.7.2, “DECLARE ... HANDLER Statement”.

13.6.3 DECLARE Statement

The `DECLARE` statement is used to define various items local to a program:

- Local variables. See Section 13.6.4, “Variables in Stored Programs”.
- Conditions and handlers. See Section 13.6.7, “Condition Handling”.
- Cursors. See Section 13.6.6, “Cursors”.

`DECLARE` is permitted only inside a `BEGIN ... END` compound statement and must be at its start, before any other statements.

Declarations must follow a certain order. Cursor declarations must appear before handler declarations. Variable and condition declarations must appear before cursor or handler declarations.

13.6.4 Variables in Stored Programs

System variables and user-defined variables can be used in stored programs, just as they can be used outside stored-program context. In addition, stored programs can use `DECLARE` to define local variables, and stored routines (procedures and functions) can be declared to take parameters that communicate values between the routine and its caller.

- To declare local variables, use the `DECLARE` statement, as described in Section 13.6.4.1, “Local Variable DECLARE Statement”.
- Variables can be set directly with the `SET` statement. See Section 13.7.4.1, “SET Syntax for Variable Assignment”.
- Results from queries can be retrieved into local variables using `SELECT ... INTO` *`var_list`* or by opening a cursor and using `FETCH ... INTO` *`var_list`*. See Section 13.2.9.1, “SELECT ... INTO Statement”, and Section 13.6.6, “Cursors”.

For information about the scope of local variables and how MySQL resolves ambiguous names, see Section 13.6.4.2, “Local Variable Scope and Resolution”.

It is not permitted to assign the value `DEFAULT` to stored procedure or function parameters or stored program local variables (for example with a `SET` *`var_name`* `= DEFAULT` statement). As of MySQL 5.6.6, this results in a syntax error.

13.6.4.1 Local Variable DECLARE Statement

```
DECLARE var_name [, var_name] ... type [DEFAULT value]
```

This statement declares local variables within stored programs. To provide a default value for a variable, include a `DEFAULT` clause. The value can be specified as an expression; it need not be a constant. If the `DEFAULT` clause is missing, the initial value is `NULL`.

Local variables are treated like stored routine parameters with respect to data type and overflow checking. See Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”.

Variable declarations must appear before cursor or handler declarations.

Local variable names are not case-sensitive. Permissible characters and quoting rules are the same as for other identifiers, as described in Section 9.2, “Schema Object Names”.

The scope of a local variable is the `BEGIN ... END` block within which it is declared. The variable can be referred to in blocks nested within the declaring block, except those blocks that declare a variable with the same name.

For examples of variable declarations, see Section 13.6.4.2, “Local Variable Scope and Resolution”.

13.6.4.2 Local Variable Scope and Resolution

The scope of a local variable is the `BEGIN ... END` block within which it is declared. The variable can be referred to in blocks nested within the declaring block, except those blocks that declare a variable with the same name.

Because local variables are in scope only during stored program execution, references to them are not permitted in prepared statements created within a stored program. Prepared statement scope is the current session, not the stored program, so the statement could be executed after the program ends, at which point the variables would no longer be in scope. For example, `SELECT ... INTO local_var` cannot be used as a prepared statement. This restriction also applies to stored procedure and function parameters. See Section 13.5.1, “PREPARE Statement”.

A local variable should not have the same name as a table column. If an SQL statement, such as a `SELECT ... INTO` statement, contains a reference to a column and a declared local variable with the same name, MySQL currently interprets the reference as the name of a variable. Consider the following procedure definition:

```
CREATE PROCEDURE sp1 (x VARCHAR(5))
BEGIN
  DECLARE xname VARCHAR(5) DEFAULT 'bob';
  DECLARE newname VARCHAR(5);
  DECLARE xid INT;

  SELECT xname, id INTO newname, xid
    FROM table1 WHERE xname = xname;
  SELECT newname;
END;
```

MySQL interprets `xname` in the `SELECT` statement as a reference to the `xname` *variable* rather than the `xname` *column*. Consequently, when the procedure `sp1()`is called, the `newname` variable returns the value `'bob'` regardless of the value of the `table1.xname` column.

Similarly, the cursor definition in the following procedure contains a `SELECT` statement that refers to `xname`. MySQL interprets this as a reference to the variable of that name rather than a column reference.

```
CREATE PROCEDURE sp2 (x VARCHAR(5))
BEGIN
  DECLARE xname VARCHAR(5) DEFAULT 'bob';
```

```
  DECLARE newname VARCHAR(5);
  DECLARE xid INT;
  DECLARE done TINYINT DEFAULT 0;
  DECLARE cur1 CURSOR FOR SELECT xname, id FROM table1;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = 1;

  OPEN cur1;
  read_loop: LOOP
    FETCH FROM cur1 INTO newname, xid;
    IF done THEN LEAVE read_loop; END IF;
    SELECT newname;
  END LOOP;
  CLOSE cur1;
END;
```

See also Section 20.8, “Restrictions on Stored Programs”.

13.6.5 Flow Control Statements

MySQL supports the `IF`, `CASE`, `ITERATE`, `LEAVE LOOP`, `WHILE`, and `REPEAT` constructs for flow control within stored programs. It also supports `RETURN` within stored functions.

Many of these constructs contain other statements, as indicated by the grammar specifications in the following sections. Such constructs may be nested. For example, an `IF` statement might contain a `WHILE` loop, which itself contains a `CASE` statement.

MySQL does not support `FOR` loops.

13.6.5.1 CASE Statement

```
CASE case_value
    WHEN when_value THEN statement_list
    [WHEN when_value THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

Or:

```
CASE
    WHEN search_condition THEN statement_list
    [WHEN search_condition THEN statement_list] ...
    [ELSE statement_list]
END CASE
```

The `CASE` statement for stored programs implements a complex conditional construct.

Note

There is also a `CASE` *operator*, which differs from the `CASE` *statement* described here. See Section 12.5, “Flow Control Functions”. The `CASE` statement cannot have an `ELSE NULL` clause, and it is terminated with `END CASE` instead of `END`.

For the first syntax, *`case_value`* is an expression. This value is compared to the *`when_value`* expression in each `WHEN` clause until one of them is equal. When an equal *`when_value`* is found, the corresponding `THEN` clause *`statement_list`* executes. If no *`when_value`* is equal, the `ELSE` clause *`statement_list`* executes, if there is one.

This syntax cannot be used to test for equality with `NULL` because `NULL = NULL` is false. See Section 3.3.4.6, “Working with NULL Values”.

For the second syntax, each `WHEN` clause *`search_condition`* expression is evaluated until one is true, at which point its corresponding `THEN` clause *`statement_list`* executes. If no *`search_condition`* is equal, the `ELSE` clause *`statement_list`* executes, if there is one.

If no *`when_value`* or *`search_condition`* matches the value tested and the `CASE` statement contains no `ELSE` clause, a `Case not found for CASE statement` error results.

Each *`statement_list`* consists of one or more SQL statements; an empty *`statement_list`* is not permitted.

To handle situations where no value is matched by any `WHEN` clause, use an `ELSE` containing an empty `BEGIN ... END` block, as shown in this example. (The indentation used here in the `ELSE` clause is for purposes of clarity only, and is not otherwise significant.)

```
DELIMITER |

CREATE PROCEDURE p()
  BEGIN
    DECLARE v INT DEFAULT 1;

    CASE v
      WHEN 2 THEN SELECT v;
      WHEN 3 THEN SELECT 0;
      ELSE
        BEGIN
        END;
    END CASE;
  END;
  |
```

13.6.5.2 IF Statement

```
IF search_condition THEN statement_list
    [ELSEIF search_condition THEN statement_list] ...
    [ELSE statement_list]
END IF
```

The `IF` statement for stored programs implements a basic conditional construct.

Note

There is also an `IF()` *function*, which differs from the `IF` *statement* described here. See Section 12.5, “Flow Control Functions”. The `IF` statement can have `THEN`, `ELSE`, and `ELSEIF` clauses, and it is terminated with `END IF`.

If a given *`search_condition`* evaluates to true, the corresponding `THEN` or `ELSEIF` clause *`statement_list`* executes. If no *`search_condition`* matches, the `ELSE` clause *`statement_list`* executes.

Each *`statement_list`* consists of one or more SQL statements; an empty *`statement_list`* is not permitted.

An `IF ... END IF` block, like all other flow-control blocks used within stored programs, must be terminated with a semicolon, as shown in this example:

```
DELIMITER //

CREATE FUNCTION SimpleCompare(n INT, m INT)
  RETURNS VARCHAR(20)

  BEGIN
    DECLARE s VARCHAR(20);

    IF n > m THEN SET s = '>';
    ELSEIF n = m THEN SET s = '=';
    ELSE SET s = '<';
    END IF;

    SET s = CONCAT(n, ' ', s, ' ', m);
```

```
    RETURN s;
  END //

DELIMITER ;
```

As with other flow-control constructs, `IF ... END IF` blocks may be nested within other flow-control constructs, including other `IF` statements. Each `IF` must be terminated by its own `END IF` followed by a semicolon. You can use indentation to make nested flow-control blocks more easily readable by humans (although this is not required by MySQL), as shown here:

```
DELIMITER //

CREATE FUNCTION VerboseCompare (n INT, m INT)
  RETURNS VARCHAR(50)

  BEGIN
    DECLARE s VARCHAR(50);

    IF n = m THEN SET s = 'equals';
    ELSE
      IF n > m THEN SET s = 'greater';
      ELSE SET s = 'less';
      END IF;

      SET s = CONCAT('is ', s, ' than');
    END IF;

    SET s = CONCAT(n, ' ', s, ' ', m, '.');

    RETURN s;
  END //

DELIMITER ;
```

In this example, the inner `IF` is evaluated only if `n` is not equal to `m`.

13.6.5.3 ITERATE Statement

```
ITERATE label
```

`ITERATE` can appear only within `LOOP`, `REPEAT`, and `WHILE` statements. `ITERATE` means “start the loop again.”

For an example, see Section 13.6.5.5, “LOOP Statement”.

13.6.5.4 LEAVE Statement

```
LEAVE label
```

This statement is used to exit the flow control construct that has the given label. If the label is for the outermost stored program block, `LEAVE` exits the program.

`LEAVE` can be used within `BEGIN ... END` or loop constructs (`LOOP`, `REPEAT`, `WHILE`).

For an example, see Section 13.6.5.5, “LOOP Statement”.

13.6.5.5 LOOP Statement

```
[begin_label:] LOOP
    statement_list
END LOOP [end_label]
```

`LOOP` implements a simple loop construct, enabling repeated execution of the statement list, which consists of one or more statements, each terminated by a semicolon (`;`) statement delimiter. The

statements within the loop are repeated until the loop is terminated. Usually, this is accomplished with a LEAVE statement. Within a stored function, RETURN can also be used, which exits the function entirely.

Neglecting to include a loop-termination statement results in an infinite loop.

A LOOP statement can be labeled. For the rules regarding label use, see Section 13.6.2, “Statement Labels”.

Example:

```
CREATE PROCEDURE doiterate(p1 INT)
BEGIN
  label1: LOOP
    SET p1 = p1 + 1;
    IF p1 < 10 THEN
      ITERATE label1;
    END IF;
    LEAVE label1;
  END LOOP label1;
  SET @x = p1;
END;
```

13.6.5.6 REPEAT Statement

```
[begin_label:] REPEAT
    statement_list
UNTIL search_condition
END REPEAT [end_label]
```

The statement list within a REPEAT statement is repeated until the *search_condition* expression is true. Thus, a REPEAT always enters the loop at least once. *statement_list* consists of one or more statements, each terminated by a semicolon (;) statement delimiter.

A REPEAT statement can be labeled. For the rules regarding label use, see Section 13.6.2, “Statement Labels”.

Example:

```
mysql> delimiter //

mysql> CREATE PROCEDURE dorepeat(p1 INT)
       BEGIN
         SET @x = 0;
         REPEAT
           SET @x = @x + 1;
         UNTIL @x > p1 END REPEAT;
       END
       //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL dorepeat(1000)//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
+------+
| @x   |
+------+
| 1001 |
+------+
1 row in set (0.00 sec)
```

13.6.5.7 RETURN Statement

```
RETURN expr
```

The RETURN statement terminates execution of a stored function and returns the value *expr* to the function caller. There must be at least one RETURN statement in a stored function. There may be more than one if the function has multiple exit points.

This statement is not used in stored procedures, triggers, or events. The `LEAVE` statement can be used to exit a stored program of those types.

13.6.5.8 WHILE Statement

```
[begin_label:] WHILE search_condition DO
    statement_list
END WHILE [end_label]
```

The statement list within a `WHILE` statement is repeated as long as the *`search_condition`* expression is true. *`statement_list`* consists of one or more SQL statements, each terminated by a semicolon (`;`) statement delimiter.

A `WHILE` statement can be labeled. For the rules regarding label use, see Section 13.6.2, “Statement Labels”.

Example:

```
CREATE PROCEDURE dowhile()
BEGIN
  DECLARE v1 INT DEFAULT 5;

  WHILE v1 > 0 DO
    ...
    SET v1 = v1 - 1;
  END WHILE;
END;
```

13.6.6 Cursors

MySQL supports cursors inside stored programs. The syntax is as in embedded SQL. Cursors have these properties:

- Asensitive: The server may or may not make a copy of its result table
- Read only: Not updatable
- Nonscrollable: Can be traversed only in one direction and cannot skip rows

Cursor declarations must appear before handler declarations and after variable and condition declarations.

Example:

```
CREATE PROCEDURE curdemo()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE a CHAR(16);
  DECLARE b, c INT;
  DECLARE cur1 CURSOR FOR SELECT id,data FROM test.t1;
  DECLARE cur2 CURSOR FOR SELECT i FROM test.t2;
  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

  OPEN cur1;
  OPEN cur2;

  read_loop: LOOP
    FETCH cur1 INTO a, b;
    FETCH cur2 INTO c;
    IF done THEN
      LEAVE read_loop;
    END IF;
    IF b < c THEN
      INSERT INTO test.t3 VALUES (a,b);
    ELSE
      INSERT INTO test.t3 VALUES (a,c);
```

```
    END IF;
  END LOOP;

  CLOSE cur1;
  CLOSE cur2;
END;
```

13.6.6.1 Cursor CLOSE Statement

```
CLOSE cursor_name
```

This statement closes a previously opened cursor. For an example, see Section 13.6.6, “Cursors”.

An error occurs if the cursor is not open.

If not closed explicitly, a cursor is closed at the end of the `BEGIN ... END` block in which it was declared.

13.6.6.2 Cursor DECLARE Statement

```
DECLARE cursor_name CURSOR FOR select_statement
```

This statement declares a cursor and associates it with a `SELECT` statement that retrieves the rows to be traversed by the cursor. To fetch the rows later, use a `FETCH` statement. The number of columns retrieved by the `SELECT` statement must match the number of output variables specified in the `FETCH` statement.

The `SELECT` statement cannot have an `INTO` clause.

Cursor declarations must appear before handler declarations and after variable and condition declarations.

A stored program may contain multiple cursor declarations, but each cursor declared in a given block must have a unique name. For an example, see Section 13.6.6, “Cursors”.

For information available through `SHOW` statements, it is possible in many cases to obtain equivalent information by using a cursor with an `INFORMATION_SCHEMA` table.

13.6.6.3 Cursor FETCH Statement

```
FETCH [[NEXT] FROM] cursor_name INTO var_name [, var_name] ...
```

This statement fetches the next row for the `SELECT` statement associated with the specified cursor (which must be open), and advances the cursor pointer. If a row exists, the fetched columns are stored in the named variables. The number of columns retrieved by the `SELECT` statement must match the number of output variables specified in the `FETCH` statement.

If no more rows are available, a No Data condition occurs with SQLSTATE value `'02000'`. To detect this condition, you can set up a handler for it (or for a `NOT FOUND` condition). For an example, see Section 13.6.6, “Cursors”.

Be aware that another operation, such as a `SELECT` or another `FETCH`, may also cause the handler to execute by raising the same condition. If it is necessary to distinguish which operation raised the condition, place the operation within its own `BEGIN ... END` block so that it can be associated with its own handler.

13.6.6.4 Cursor OPEN Statement

```
OPEN cursor_name
```

This statement opens a previously declared cursor. For an example, see Section 13.6.6, “Cursors”.

13.6.6.5 Restrictions on Server-Side Cursors

Server-side cursors are implemented in the C API using the `mysql_stmt_attr_set()` function. The same implementation is used for cursors in stored routines. A server-side cursor enables a result set to be generated on the server side, but not transferred to the client except for those rows that the client requests. For example, if a client executes a query but is only interested in the first row, the remaining rows are not transferred.

In MySQL, a server-side cursor is materialized into an internal temporary table. Initially, this is a `MEMORY` table, but is converted to a `MyISAM` table when its size exceeds the minimum value of the `max_heap_table_size` and `tmp_table_size` system variables. The same restrictions apply to internal temporary tables created to hold the result set for a cursor as for other uses of internal temporary tables. See Section 8.4.4, “Internal Temporary Table Use in MySQL”. One limitation of the implementation is that for a large result set, retrieving its rows through a cursor might be slow.

Cursors are read only; you cannot use a cursor to update rows.

`UPDATE WHERE CURRENT OF` and `DELETE WHERE CURRENT OF` are not implemented, because updatable cursors are not supported.

Cursors are nonholdable (not held open after a commit).

Cursors are asensitive.

Cursors are nonscrollable.

Cursors are not named. The statement handler acts as the cursor ID.

You can have open only a single cursor per prepared statement. If you need several cursors, you must prepare several statements.

You cannot use a cursor for a statement that generates a result set if the statement is not supported in prepared mode. This includes statements such as `CHECK TABLE`, `HANDLER READ`, and `SHOW BINLOG EVENTS`.

13.6.7 Condition Handling

Conditions may arise during stored program execution that require special handling, such as exiting the current program block or continuing execution. Handlers can be defined for general conditions such as warnings or exceptions, or for specific conditions such as a particular error code. Specific conditions can be assigned names and referred to that way in handlers.

To name a condition, use the `DECLARE ... CONDITION` statement. To declare a handler, use the `DECLARE ... HANDLER` statement. See Section 13.6.7.1, “DECLARE ... CONDITION Statement”, and Section 13.6.7.2, “DECLARE ... HANDLER Statement”. For information about how the server chooses handlers when a condition occurs, see Section 13.6.7.6, “Scope Rules for Handlers”.

To raise a condition, use the `SIGNAL` statement. To modify condition information within a condition handler, use `RESIGNAL`. See Section 13.6.7.1, “DECLARE ... CONDITION Statement”, and Section 13.6.7.2, “DECLARE ... HANDLER Statement”.

To retrieve information from the diagnostics area, use the `GET DIAGNOSTICS` statement (see Section 13.6.7.3, “GET DIAGNOSTICS Statement”). For information about the diagnostics area, see Section 13.6.7.7, “The MySQL Diagnostics Area”.

13.6.7.1 DECLARE ... CONDITION Statement

```
DECLARE condition_name CONDITION FOR condition_value
```

```
condition_value: {
    mysql_error_code
  | SQLSTATE [VALUE] sqlstate_value
}
```

The `DECLARE ... CONDITION` statement declares a named error condition, associating a name with a condition that needs specific handling. The name can be referred to in a subsequent `DECLARE ... HANDLER` statement (see Section 13.6.7.2, “DECLARE ... HANDLER Statement”).

Condition declarations must appear before cursor or handler declarations.

The *`condition_value`* for `DECLARE ... CONDITION` indicates the specific condition or class of conditions to associate with the condition name. It can take the following forms:

- *`mysql_error_code`*: An integer literal indicating a MySQL error code.

  Do not use MySQL error code 0 because that indicates success rather than an error condition. For a list of MySQL error codes, see Server Error Message Reference.

- SQLSTATE [VALUE] *`sqlstate_value`*: A 5-character string literal indicating an SQLSTATE value.

  Do not use SQLSTATE values that begin with `'00'` because those indicate success rather than an error condition. For a list of SQLSTATE values, see Server Error Message Reference.

Condition names referred to in `SIGNAL` or use `RESIGNAL` statements must be associated with SQLSTATE values, not MySQL error codes.

Using names for conditions can help make stored program code clearer. For example, this handler applies to attempts to drop a nonexistent table, but that is apparent only if you know that 1051 is the MySQL error code for “unknown table”:

```
DECLARE CONTINUE HANDLER FOR 1051
  BEGIN
    -- body of handler
  END;
```

By declaring a name for the condition, the purpose of the handler is more readily seen:

```
DECLARE no_such_table CONDITION FOR 1051;
DECLARE CONTINUE HANDLER FOR no_such_table
  BEGIN
    -- body of handler
  END;
```

Here is a named condition for the same condition, but based on the corresponding SQLSTATE value rather than the MySQL error code:

```
DECLARE no_such_table CONDITION FOR SQLSTATE '42S02';
DECLARE CONTINUE HANDLER FOR no_such_table
  BEGIN
    -- body of handler
  END;
```

13.6.7.2 DECLARE ... HANDLER Statement

```
DECLARE handler_action HANDLER
    FOR condition_value [, condition_value] ...
    statement

handler_action: {
    CONTINUE
  | EXIT
  | UNDO
}
```

```
condition_value: {
    mysql_error_code
  | SQLSTATE [VALUE] sqlstate_value
  | condition_name
  | SQLWARNING
  | NOT FOUND
  | SQLEXCEPTION
}
```

The `DECLARE ... HANDLER` statement specifies a handler that deals with one or more conditions. If one of these conditions occurs, the specified *`statement`* executes. *`statement`* can be a simple statement such as `SET var_name = value`, or a compound statement written using `BEGIN` and `END` (see Section 13.6.1, “BEGIN ... END Compound Statement”).

Handler declarations must appear after variable or condition declarations.

The *`handler_action`* value indicates what action the handler takes after execution of the handler statement:

- `CONTINUE`: Execution of the current program continues.
- `EXIT`: Execution terminates for the `BEGIN ... END` compound statement in which the handler is declared. This is true even if the condition occurs in an inner block.
- `UNDO`: Not supported.

The *`condition_value`* for `DECLARE ... HANDLER` indicates the specific condition or class of conditions that activates the handler. It can take the following forms:

- *`mysql_error_code`*: An integer literal indicating a MySQL error code, such as 1051 to specify “unknown table”:

  ```
  DECLARE CONTINUE HANDLER FOR 1051
    BEGIN
      -- body of handler
    END;
  ```

  Do not use MySQL error code 0 because that indicates success rather than an error condition. For a list of MySQL error codes, see Server Error Message Reference.

- SQLSTATE [VALUE] *`sqlstate_value`*: A 5-character string literal indicating an SQLSTATE value, such as `'42S01'` to specify “unknown table”:

  ```
  DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
    BEGIN
      -- body of handler
    END;
  ```

  Do not use SQLSTATE values that begin with `'00'` because those indicate success rather than an error condition. For a list of SQLSTATE values, see Server Error Message Reference.

- *`condition_name`*: A condition name previously specified with `DECLARE ... CONDITION`. A condition name can be associated with a MySQL error code or SQLSTATE value. See Section 13.6.7.1, “DECLARE ... CONDITION Statement”.

- `SQLWARNING`: Shorthand for the class of SQLSTATE values that begin with `'01'`.

  ```
  DECLARE CONTINUE HANDLER FOR SQLWARNING
    BEGIN
      -- body of handler
    END;
  ```

- `NOT FOUND`: Shorthand for the class of SQLSTATE values that begin with `'02'`. This is relevant within the context of cursors and is used to control what happens when a cursor reaches the end of a

data set. If no more rows are available, a No Data condition occurs with SQLSTATE value `'02000'`. To detect this condition, you can set up a handler for it or for a `NOT FOUND` condition.

```
DECLARE CONTINUE HANDLER FOR NOT FOUND
  BEGIN
    -- body of handler
  END;
```

For another example, see Section 13.6.6, “Cursors”. The `NOT FOUND` condition also occurs for `SELECT ... INTO var_list` statements that retrieve no rows.

- `SQLEXCEPTION`: Shorthand for the class of SQLSTATE values that do not begin with `'00'`, `'01'`, or `'02'`.

```
DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
  BEGIN
    -- body of handler
  END;
```

For information about how the server chooses handlers when a condition occurs, see Section 13.6.7.6, “Scope Rules for Handlers”.

If a condition occurs for which no handler has been declared, the action taken depends on the condition class:

- For `SQLEXCEPTION` conditions, the stored program terminates at the statement that raised the condition, as if there were an `EXIT` handler. If the program was called by another stored program, the calling program handles the condition using the handler selection rules applied to its own handlers.
- For `SQLWARNING` conditions, the program continues executing, as if there were a `CONTINUE` handler.
- For `NOT FOUND` conditions, if the condition was raised normally, the action is `CONTINUE`. If it was raised by `SIGNAL` or `RESIGNAL`, the action is `EXIT`.

The following example uses a handler for `SQLSTATE '23000'`, which occurs for a duplicate-key error:

```
mysql> CREATE TABLE test.t (s1 INT, PRIMARY KEY (s1));
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter //

mysql> CREATE PROCEDURE handlerdemo ()
       BEGIN
         DECLARE CONTINUE HANDLER FOR SQLSTATE '23000' SET @x2 = 1;
         SET @x = 1;
         INSERT INTO test.t VALUES (1);
         SET @x = 2;
         INSERT INTO test.t VALUES (1);
         SET @x = 3;
       END;
       //
Query OK, 0 rows affected (0.00 sec)

mysql> CALL handlerdemo()//
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x//
    +------+
    | @x   |
    +------+
    | 3    |
    +------+
    1 row in set (0.00 sec)
```

Notice that `@x` is `3` after the procedure executes, which shows that execution continued to the end of the procedure after the error occurred. If the `DECLARE ... HANDLER` statement had not been

present, MySQL would have taken the default action (`EXIT`) after the second `INSERT` failed due to the `PRIMARY KEY` constraint, and `SELECT @x` would have returned `2`.

To ignore a condition, declare a `CONTINUE` handler for it and associate it with an empty block. For example:

```
DECLARE CONTINUE HANDLER FOR SQLWARNING BEGIN END;
```

The scope of a block label does not include the code for handlers declared within the block. Therefore, the statement associated with a handler cannot use `ITERATE` or `LEAVE` to refer to labels for blocks that enclose the handler declaration. Consider the following example, where the `REPEAT` block has a label of `retry`:

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE i INT DEFAULT 3;
  retry:
    REPEAT
      BEGIN
        DECLARE CONTINUE HANDLER FOR SQLWARNING
          BEGIN
            ITERATE retry;    # illegal
          END;
        IF i < 0 THEN
          LEAVE retry;        # legal
        END IF;
        SET i = i - 1;
      END;
    UNTIL FALSE END REPEAT;
END;
```

The `retry` label is in scope for the `IF` statement within the block. It is not in scope for the `CONTINUE` handler, so the reference there is invalid and results in an error:

```
ERROR 1308 (42000): LEAVE with no matching label: retry
```

To avoid references to outer labels in handlers, use one of these strategies:

- To leave the block, use an `EXIT` handler. If no block cleanup is required, the `BEGIN ... END` handler body can be empty:

  ```
  DECLARE EXIT HANDLER FOR SQLWARNING BEGIN END;
  ```

  Otherwise, put the cleanup statements in the handler body:

  ```
  DECLARE EXIT HANDLER FOR SQLWARNING
    BEGIN
      block cleanup statements
    END;
  ```

- To continue execution, set a status variable in a `CONTINUE` handler that can be checked in the enclosing block to determine whether the handler was invoked. The following example uses the variable `done` for this purpose:

  ```
  CREATE PROCEDURE p ()
  BEGIN
    DECLARE i INT DEFAULT 3;
    DECLARE done INT DEFAULT FALSE;
    retry:
      REPEAT
        BEGIN
          DECLARE CONTINUE HANDLER FOR SQLWARNING
            BEGIN
              SET done = TRUE;
            END;
          IF done OR i < 0 THEN
            LEAVE retry;
  ```

```
        END IF;
        SET i = i - 1;
      END;
    UNTIL FALSE END REPEAT;
END;
```

13.6.7.3 GET DIAGNOSTICS Statement

```
GET [CURRENT] DIAGNOSTICS {
    statement_information_item
    [, statement_information_item] ...
  | CONDITION condition_number
    condition_information_item
    [, condition_information_item] ...
}

statement_information_item:
    target = statement_information_item_name

condition_information_item:
    target = condition_information_item_name

statement_information_item_name: {
    NUMBER
  | ROW_COUNT
}

condition_information_item_name: {
    CLASS_ORIGIN
  | SUBCLASS_ORIGIN
  | RETURNED_SQLSTATE
  | MESSAGE_TEXT
  | MYSQL_ERRNO
  | CONSTRAINT_CATALOG
  | CONSTRAINT_SCHEMA
  | CONSTRAINT_NAME
  | CATALOG_NAME
  | SCHEMA_NAME
  | TABLE_NAME
  | COLUMN_NAME
  | CURSOR_NAME
}

condition_number, target:
    (see following discussion)
```

SQL statements produce diagnostic information that populates the diagnostics area. The `GET DIAGNOSTICS` statement enables applications to inspect this information. It is available as of MySQL 5.6.4. (You can also use `SHOW WARNINGS` or `SHOW ERRORS` to see conditions or errors.)

No special privileges are required to execute `GET DIAGNOSTICS`.

The keyword `CURRENT` means to retrieve information from the current diagnostics area. In MySQL, it has no effect because that is the default behavior.

`GET DIAGNOSTICS` is typically used in a handler within a stored program, but it is a MySQL extension that it is permitted outside handler context to check the execution of any SQL statement. For example, if you invoke the `mysql` client program, you can enter these statements at the prompt:

```
mysql> DROP TABLE test.no_such_table;
ERROR 1051 (42S02): Unknown table 'test.no_such_table'
mysql> GET DIAGNOSTICS CONDITION 1
         @p1 = RETURNED_SQLSTATE, @p2 = MESSAGE_TEXT;
mysql> SELECT @p1, @p2;
+-------+-----------------------------------+
| @p1   | @p2                               |
+-------+-----------------------------------+
| 42S02 | Unknown table 'test.no_such_table' |
+-------+-----------------------------------+
```

For a description of the diagnostics area, see Section 13.6.7.7, “The MySQL Diagnostics Area”. Briefly, it contains two kinds of information:

- Statement information, such as the number of conditions that occurred or the affected-rows count.
- Condition information, such as the error code and message. If a statement raises multiple conditions, this part of the diagnostics area has a condition area for each one. If a statement raises no conditions, this part of the diagnostics area is empty.

For a statement that produces three conditions, the diagnostics area contains statement and condition information like this:

```
Statement information:
  row count
  ... other statement information items ...
Condition area list:
  Condition area 1:
    error code for condition 1
    error message for condition 1
    ... other condition information items ...
  Condition area 2:
    error code for condition 2:
    error message for condition 2
    ... other condition information items ...
  Condition area 3:
    error code for condition 3
    error message for condition 3
    ... other condition information items ...
```

`GET DIAGNOSTICS` can obtain either statement or condition information, but not both in the same statement:

- To obtain statement information, retrieve the desired statement items into target variables. This instance of `GET DIAGNOSTICS` assigns the number of available conditions and the rows-affected count to the user variables `@p1` and `@p2`:

  ```
  GET DIAGNOSTICS @p1 = NUMBER, @p2 = ROW_COUNT;
  ```

- To obtain condition information, specify the condition number and retrieve the desired condition items into target variables. This instance of `GET DIAGNOSTICS` assigns the SQLSTATE value and error message to the user variables `@p3` and `@p4`:

  ```
  GET DIAGNOSTICS CONDITION 1
    @p3 = RETURNED_SQLSTATE, @p4 = MESSAGE_TEXT;
  ```

The retrieval list specifies one or more *`target = item_name`* assignments, separated by commas. Each assignment names a target variable and either a *`statement_information_item_name`* or *`condition_information_item_name`* designator, depending on whether the statement retrieves statement or condition information.

Valid *`target`* designators for storing item information can be stored procedure or function parameters, stored program local variables declared with `DECLARE`, or user-defined variables.

Valid *`condition_number`* designators can be stored procedure or function parameters, stored program local variables declared with `DECLARE`, user-defined variables, system variables, or literals. A character literal may include a *`_charset`* introducer. A warning occurs if the condition number is not in the range from 1 to the number of condition areas that have information. In this case, the warning is added to the diagnostics area without clearing it.

When a condition occurs, MySQL does not populate all condition items recognized by `GET DIAGNOSTICS`. For example:

```
mysql> GET DIAGNOSTICS CONDITION 1
         @p5 = SCHEMA_NAME, @p6 = TABLE_NAME;
```

```
mysql> SELECT @p5, @p6;
+------+------+
| @p5  | @p6  |
+------+------+
|      |      |
+------+------+
```

In standard SQL, if there are multiple conditions, the first condition relates to the `SQLSTATE` value returned for the previous SQL statement. In MySQL, this is not guaranteed. To get the main error, you cannot do this:

```
GET DIAGNOSTICS CONDITION 1 @errno = MYSQL_ERRNO;
```

Instead, retrieve the condition count first, then use it to specify which condition number to inspect:

```
GET DIAGNOSTICS @cno = NUMBER;
GET DIAGNOSTICS CONDITION @cno @errno = MYSQL_ERRNO;
```

For information about permissible statement and condition information items, and which ones are populated when a condition occurs, see Diagnostics Area Information Items.

Here is an example that uses `GET DIAGNOSTICS` and an exception handler in stored procedure context to assess the outcome of an insert operation. If the insert was successful, the procedure uses `GET DIAGNOSTICS` to get the rows-affected count. This shows that you can use `GET DIAGNOSTICS` multiple times to retrieve information about a statement as long as the diagnostics area has not been cleared.

```
CREATE PROCEDURE do_insert(value INT)
BEGIN
  -- Declare variables to hold diagnostics area information
  DECLARE code CHAR(5) DEFAULT '00000';
  DECLARE msg TEXT;
  DECLARE nrows INT;
  DECLARE result TEXT;
  -- Declare exception handler for failed insert
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    BEGIN
      GET DIAGNOSTICS CONDITION 1
        code = RETURNED_SQLSTATE, msg = MESSAGE_TEXT;
    END;

  -- Perform the insert
  INSERT INTO t1 (int_col) VALUES(value);
  -- Check whether the insert was successful
  IF code = '00000' THEN
    GET DIAGNOSTICS nrows = ROW_COUNT;
    SET result = CONCAT('insert succeeded, row count = ',nrows);
  ELSE
    SET result = CONCAT('insert failed, error = ',code,', message = ',msg);
  END IF;
  -- Say what happened
  SELECT result;
END;
```

Suppose that `t1.int_col` is an integer column that is declared as `NOT NULL`. The procedure produces these results when invoked to insert non-`NULL` and `NULL` values, respectively:

```
mysql> CALL do_insert(1);
+---------------------------------+
| result                          |
+---------------------------------+
| insert succeeded, row count = 1 |
+---------------------------------+

mysql> CALL do_insert(NULL);
+-------------------------------------------------------------------------+
| result                                                                  |
+-------------------------------------------------------------------------+
```

```
| insert failed, error = 23000, message = Column 'int_col' cannot be null |
+--------------------------------------------------------------------------+
```

Within a condition handler, `GET DIAGNOSTICS` should be used before other statements that might clear the diagnostics area and cause information to be lost about the condition that activated the handler. For information about when the diagnostics area is set and cleared, see Section 13.6.7.7, “The MySQL Diagnostics Area”.

13.6.7.4 RESIGNAL Statement

```
RESIGNAL [condition_value]
    [SET signal_information_item
    [, signal_information_item] ...]

condition_value: {
    SQLSTATE [VALUE] sqlstate_value
  | condition_name
}

signal_information_item:
    condition_information_item_name = simple_value_specification

condition_information_item_name: {
    CLASS_ORIGIN
  | SUBCLASS_ORIGIN
  | MESSAGE_TEXT
  | MYSQL_ERRNO
  | CONSTRAINT_CATALOG
  | CONSTRAINT_SCHEMA
  | CONSTRAINT_NAME
  | CATALOG_NAME
  | SCHEMA_NAME
  | TABLE_NAME
  | COLUMN_NAME
  | CURSOR_NAME
}

condition_name, simple_value_specification:
    (see following discussion)
```

`RESIGNAL` passes on the error condition information that is available during execution of a condition handler within a compound statement inside a stored procedure or function, trigger, or event. `RESIGNAL` may change some or all information before passing it on. `RESIGNAL` is related to `SIGNAL`, but instead of originating a condition as `SIGNAL` does, `RESIGNAL` relays existing condition information, possibly after modifying it.

`RESIGNAL` makes it possible to both handle an error and return the error information. Otherwise, by executing an SQL statement within the handler, information that caused the handler's activation is destroyed. `RESIGNAL` also can make some procedures shorter if a given handler can handle part of a situation, then pass the condition “up the line” to another handler.

No privileges are required to execute the `RESIGNAL` statement.

All forms of `RESIGNAL` require that the current context be a condition handler. Otherwise, `RESIGNAL` is illegal and a `RESIGNAL when handler not active` error occurs.

To retrieve information from the diagnostics area, use the `GET DIAGNOSTICS` statement (see Section 13.6.7.3, “GET DIAGNOSTICS Statement”). For information about the diagnostics area, see Section 13.6.7.7, “The MySQL Diagnostics Area”.

- RESIGNAL Overview
- RESIGNAL Alone
- RESIGNAL with New Signal Information

- RESIGNAL with a Condition Value and Optional New Signal Information
- RESIGNAL Requires Condition Handler Context

RESIGNAL Overview

For *condition_value* and *signal_information_item*, the definitions and rules are the same for RESIGNAL as for SIGNAL. For example, the *condition_value* can be an SQLSTATE value, and the value can indicate errors, warnings, or "not found." For additional information, see Section 13.6.7.5, "SIGNAL Statement".

The RESIGNAL statement takes *condition_value* and SET clauses, both of which are optional. This leads to several possible uses:

- RESIGNAL alone:

```
RESIGNAL;
```

- RESIGNAL with new signal information:

```
RESIGNAL SET signal_information_item [, signal_information_item] ...;
```

- RESIGNAL with a condition value and possibly new signal information:

```
RESIGNAL condition_value
    [SET signal_information_item [, signal_information_item] ...];
```

These use cases all cause changes to the diagnostics and condition areas:

- A diagnostics area contains one or more condition areas.
- A condition area contains condition information items, such as the SQLSTATE value, MYSQL_ERRNO, or MESSAGE_TEXT.

The maximum number of condition areas in a diagnostics area is determined by the value of the max_error_count system variable. See Diagnostics Area-Related System Variables.

RESIGNAL Alone

A simple RESIGNAL alone means "pass on the error with no change." It restores the last diagnostics area and makes it the current diagnostics area. That is, it "pops" the diagnostics area stack.

Within a condition handler that catches a condition, one use for RESIGNAL alone is to perform some other actions, and then pass on without change the original condition information (the information that existed before entry into the handler).

Example:

```
DROP TABLE IF EXISTS xx;
delimiter //
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    SET @error_count = @error_count + 1;
    IF @a = 0 THEN RESIGNAL; END IF;
  END;
  DROP TABLE xx;
END//
delimiter ;
SET @error_count = 0;
SET @a = 0;
CALL p();
```

Suppose that the DROP TABLE xx statement fails. The diagnostics area stack looks like this:

```
DA 1. ERROR 1051 (42S02): Unknown table 'xx'
```

Then execution enters the `EXIT` handler. It starts by pushing a diagnostics area to the top of the stack, which now looks like this:

```
DA 1. ERROR 1051 (42S02): Unknown table 'xx'
DA 2. ERROR 1051 (42S02): Unknown table 'xx'
```

At this point, the contents of the first (current) and second (stacked) diagnostics areas are the same. The first diagnostics area may be modified by statements executing subsequently within the handler.

Usually a procedure statement clears the first diagnostics area. `BEGIN` is an exception, it does not clear, it does nothing. `SET` is not an exception, it clears, performs the operation, and produces a result of “success.” The diagnostics area stack now looks like this:

```
DA 1. ERROR 0000 (00000): Successful operation
DA 2. ERROR 1051 (42S02): Unknown table 'xx'
```

At this point, if `@a = 0`, `RESIGNAL` pops the diagnostics area stack, which now looks like this:

```
DA 1. ERROR 1051 (42S02): Unknown table 'xx'
```

And that is what the caller sees.

If `@a` is not 0, the handler simply ends, which means that there is no more use for the current diagnostics area (it has been “handled”), so it can be thrown away, causing the stacked diagnostics area to become the current diagnostics area again. The diagnostics area stack looks like this:

```
DA 1. ERROR 0000 (00000): Successful operation
```

The details make it look complex, but the end result is quite useful: Handlers can execute without destroying information about the condition that caused activation of the handler.

RESIGNAL with New Signal Information

`RESIGNAL` with a `SET` clause provides new signal information, so the statement means “pass on the error with changes”:

```
RESIGNAL SET signal_information_item [, signal_information_item] ...;
```

As with `RESIGNAL` alone, the idea is to pop the diagnostics area stack so that the original information goes out. Unlike `RESIGNAL` alone, anything specified in the `SET` clause changes.

Example:

```
DROP TABLE IF EXISTS xx;
delimiter //
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    SET @error_count = @error_count + 1;
    IF @a = 0 THEN RESIGNAL SET MYSQL_ERRNO = 5; END IF;
  END;
  DROP TABLE xx;
END//
delimiter ;
SET @error_count = 0;
SET @a = 0;
CALL p();
```

Remember from the previous discussion that `RESIGNAL` alone results in a diagnostics area stack like this:

```
DA 1. ERROR 1051 (42S02): Unknown table 'xx'
```

The `RESIGNAL SET MYSQL_ERRNO = 5` statement results in this stack instead, which is what the caller sees:

```
DA 1. ERROR 5 (42S02): Unknown table 'xx'
```

In other words, it changes the error number, and nothing else.

The `RESIGNAL` statement can change any or all of the signal information items, making the first condition area of the diagnostics area look quite different.

RESIGNAL with a Condition Value and Optional New Signal Information

`RESIGNAL` with a condition value means “push a condition into the current diagnostics area.” If the `SET` clause is present, it also changes the error information.

```
RESIGNAL condition_value
    [SET signal_information_item [, signal_information_item] ...];
```

This form of `RESIGNAL` restores the last diagnostics area and makes it the current diagnostics area. That is, it “pops” the diagnostics area stack, which is the same as what a simple `RESIGNAL` alone would do. However, it also changes the diagnostics area depending on the condition value or signal information.

Example:

```
DROP TABLE IF EXISTS xx;
delimiter //
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    SET @error_count = @error_count + 1;
    IF @a = 0 THEN RESIGNAL SQLSTATE '45000' SET MYSQL_ERRNO=5; END IF;
  END;
  DROP TABLE xx;
END//
delimiter ;
SET @error_count = 0;
SET @a = 0;
SET @@max_error_count = 2;
CALL p();
SHOW ERRORS;
```

This is similar to the previous example, and the effects are the same, except that if `RESIGNAL` happens, the current condition area looks different at the end. (The reason the condition adds to rather than replaces the existing condition is the use of a condition value.)

The `RESIGNAL` statement includes a condition value (`SQLSTATE '45000'`), so it adds a new condition area, resulting in a diagnostics area stack that looks like this:

```
DA 1. (condition 2) ERROR 1051 (42S02): Unknown table 'xx'
      (condition 1) ERROR 5 (45000) Unknown table 'xx'
```

The result of `CALL p()` and `SHOW ERRORS` for this example is:

```
mysql> CALL p();
ERROR 5 (45000): Unknown table 'xx'
mysql> SHOW ERRORS;
+-------+------+---------------------------------+
| Level | Code | Message                         |
+-------+------+---------------------------------+
| Error | 1051 | Unknown table 'xx'              |
| Error |    5 | Unknown table 'xx'              |
+-------+------+---------------------------------+
```

RESIGNAL Requires Condition Handler Context

All forms of `RESIGNAL` require that the current context be a condition handler. Otherwise, `RESIGNAL` is illegal and a `RESIGNAL when handler not active` error occurs. For example:

```
mysql> CREATE PROCEDURE p () RESIGNAL;
Query OK, 0 rows affected (0.00 sec)

mysql> CALL p();
ERROR 1645 (0K000): RESIGNAL when handler not active
```

Here is a more difficult example:

```
delimiter //
CREATE FUNCTION f () RETURNS INT
BEGIN
  RESIGNAL;
  RETURN 5;
END//
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION SET @a=f();
  SIGNAL SQLSTATE '55555';
END//
delimiter ;
CALL p();
```

`RESIGNAL` occurs within the stored function `f()`. Although `f()` itself is invoked within the context of the `EXIT` handler, execution within `f()` has its own context, which is not handler context. Thus, `RESIGNAL` within `f()` results in a “handler not active” error.

In MySQL 5.5, handler scope rules are less developed. `f()` is considered to execute within handler context and `RESIGNAL` within `f()` is legal.

13.6.7.5 SIGNAL Statement

```
SIGNAL condition_value
    [SET signal_information_item
    [, signal_information_item] ...]

condition_value: {
    SQLSTATE [VALUE] sqlstate_value
  | condition_name
}

signal_information_item:
    condition_information_item_name = simple_value_specification

condition_information_item_name: {
    CLASS_ORIGIN
  | SUBCLASS_ORIGIN
  | MESSAGE_TEXT
  | MYSQL_ERRNO
  | CONSTRAINT_CATALOG
  | CONSTRAINT_SCHEMA
  | CONSTRAINT_NAME
  | CATALOG_NAME
  | SCHEMA_NAME
  | TABLE_NAME
  | COLUMN_NAME
  | CURSOR_NAME
}

condition_name, simple_value_specification:
    (see following discussion)
```

`SIGNAL` is the way to “return” an error. `SIGNAL` provides error information to a handler, to an outer portion of the application, or to the client. Also, it provides control over the error's characteristics (error

number, SQLSTATE value, message). Without SIGNAL, it is necessary to resort to workarounds such as deliberately referring to a nonexistent table to cause a routine to return an error.

No privileges are required to execute the SIGNAL statement.

To retrieve information from the diagnostics area, use the GET DIAGNOSTICS statement (see Section 13.6.7.3, “GET DIAGNOSTICS Statement”). For information about the diagnostics area, see Section 13.6.7.7, “The MySQL Diagnostics Area”.

- SIGNAL Overview
- Signal Condition Information Items
- Effect of Signals on Handlers, Cursors, and Statements

SIGNAL Overview

The *condition_value* in a SIGNAL statement indicates the error value to be returned. It can be an SQLSTATE value (a 5-character string literal) or a *condition_name* that refers to a named condition previously defined with DECLARE ... CONDITION (see Section 13.6.7.1, “DECLARE ... CONDITION Statement”).

An SQLSTATE value can indicate errors, warnings, or “not found.” The first two characters of the value indicate its error class, as discussed in Signal Condition Information Items. Some signal values cause statement termination; see Effect of Signals on Handlers, Cursors, and Statements.

The SQLSTATE value for a SIGNAL statement should not start with '00' because such values indicate success and are not valid for signaling an error. This is true whether the SQLSTATE value is specified directly in the SIGNAL statement or in a named condition referred to in the statement. If the value is invalid, a Bad SQLSTATE error occurs.

To signal a generic SQLSTATE value, use '45000', which means “unhandled user-defined exception.”

The SIGNAL statement optionally includes a SET clause that contains multiple signal items, in a list of *condition_information_item_name* = *simple_value_specification* assignments, separated by commas.

Each *condition_information_item_name* may be specified only once in the SET clause. Otherwise, a Duplicate condition information item error occurs.

Valid *simple_value_specification* designators can be specified using stored procedure or function parameters, stored program local variables declared with DECLARE, user-defined variables, system variables, or literals. A character literal may include a *_charset* introducer.

For information about permissible *condition_information_item_name* values, see Signal Condition Information Items.

The following procedure signals an error or warning depending on the value of pval, its input parameter:

```
CREATE PROCEDURE p (pval INT)
BEGIN
  DECLARE specialty CONDITION FOR SQLSTATE '45000';
  IF pval = 0 THEN
    SIGNAL SQLSTATE '01000';
  ELSEIF pval = 1 THEN
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'An error occurred';
  ELSEIF pval = 2 THEN
    SIGNAL specialty
      SET MESSAGE_TEXT = 'An error occurred';
  ELSE
```

```
    SIGNAL SQLSTATE '01000'
      SET MESSAGE_TEXT = 'A warning occurred', MYSQL_ERRNO = 1000;
    SIGNAL SQLSTATE '45000'
      SET MESSAGE_TEXT = 'An error occurred', MYSQL_ERRNO = 1001;
  END IF;
END;
```

If `pval` is 0, `p()` signals a warning because `SQLSTATE` values that begin with `'01'` are signals in the warning class. The warning does not terminate the procedure, and can be seen with `SHOW WARNINGS` after the procedure returns.

If `pval` is 1, `p()` signals an error and sets the `MESSAGE_TEXT` condition information item. The error terminates the procedure, and the text is returned with the error information.

If `pval` is 2, the same error is signaled, although the `SQLSTATE` value is specified using a named condition in this case.

If `pval` is anything else, `p()` first signals a warning and sets the message text and error number condition information items. This warning does not terminate the procedure, so execution continues and `p()` then signals an error. The error does terminate the procedure. The message text and error number set by the warning are replaced by the values set by the error, which are returned with the error information.

`SIGNAL` is typically used within stored programs, but it is a MySQL extension that it is permitted outside handler context. For example, if you invoke the `mysql` client program, you can enter any of these statements at the prompt:

```
SIGNAL SQLSTATE '77777';

CREATE TRIGGER t_bi BEFORE INSERT ON t
  FOR EACH ROW SIGNAL SQLSTATE '77777';

CREATE EVENT e ON SCHEDULE EVERY 1 SECOND
  DO SIGNAL SQLSTATE '77777';
```

`SIGNAL` executes according to the following rules:

If the `SIGNAL` statement indicates a particular `SQLSTATE` value, that value is used to signal the condition specified. Example:

```
CREATE PROCEDURE p (divisor INT)
BEGIN
  IF divisor = 0 THEN
    SIGNAL SQLSTATE '22012';
  END IF;
END;
```

If the `SIGNAL` statement uses a named condition, the condition must be declared in some scope that applies to the `SIGNAL` statement, and must be defined using an `SQLSTATE` value, not a MySQL error number. Example:

```
CREATE PROCEDURE p (divisor INT)
BEGIN
  DECLARE divide_by_zero CONDITION FOR SQLSTATE '22012';
  IF divisor = 0 THEN
    SIGNAL divide_by_zero;
  END IF;
END;
```

If the named condition does not exist in the scope of the `SIGNAL` statement, an `Undefined CONDITION` error occurs.

If `SIGNAL` refers to a named condition that is defined with a MySQL error number rather than an `SQLSTATE` value, a `SIGNAL/RESIGNAL can only use a CONDITION defined with`

`SQLSTATE` error occurs. The following statements cause that error because the named condition is associated with a MySQL error number:

```
DECLARE no_such_table CONDITION FOR 1051;
SIGNAL no_such_table;
```

If a condition with a given name is declared multiple times in different scopes, the declaration with the most local scope applies. Consider the following procedure:

```
CREATE PROCEDURE p (divisor INT)
BEGIN
  DECLARE my_error CONDITION FOR SQLSTATE '45000';
  IF divisor = 0 THEN
    BEGIN
      DECLARE my_error CONDITION FOR SQLSTATE '22012';
      SIGNAL my_error;
    END;
  END IF;
  SIGNAL my_error;
END;
```

If `divisor` is 0, the first `SIGNAL` statement executes. The innermost `my_error` condition declaration applies, raising `SQLSTATE '22012'`.

If `divisor` is not 0, the second `SIGNAL` statement executes. The outermost `my_error` condition declaration applies, raising `SQLSTATE '45000'`.

For information about how the server chooses handlers when a condition occurs, see Section 13.6.7.6, “Scope Rules for Handlers”.

Signals can be raised within exception handlers:

```
CREATE PROCEDURE p ()
BEGIN
  DECLARE EXIT HANDLER FOR SQLEXCEPTION
  BEGIN
    SIGNAL SQLSTATE VALUE '99999'
      SET MESSAGE_TEXT = 'An error occurred';
  END;
  DROP TABLE no_such_table;
END;
```

`CALL p()` reaches the `DROP TABLE` statement. There is no table named `no_such_table`, so the error handler is activated. The error handler destroys the original error (“no such table”) and makes a new error with `SQLSTATE '99999'` and message `An error occurred`.

Signal Condition Information Items

The following table lists the names of diagnostics area condition information items that can be set in a `SIGNAL` (or `RESIGNAL`) statement. All items are standard SQL except `MYSQL_ERRNO`, which is a MySQL extension. For more information about these items see Section 13.6.7.7, “The MySQL Diagnostics Area”.

```
Item Name             Definition
---------             ----------
CLASS_ORIGIN          VARCHAR(64)
SUBCLASS_ORIGIN       VARCHAR(64)
CONSTRAINT_CATALOG    VARCHAR(64)
CONSTRAINT_SCHEMA     VARCHAR(64)
CONSTRAINT_NAME       VARCHAR(64)
CATALOG_NAME          VARCHAR(64)
SCHEMA_NAME           VARCHAR(64)
TABLE_NAME            VARCHAR(64)
COLUMN_NAME           VARCHAR(64)
CURSOR_NAME           VARCHAR(64)
MESSAGE_TEXT          VARCHAR(128)
```

```
MYSQL_ERRNO           SMALLINT UNSIGNED
```

The character set for character items is UTF-8.

It is illegal to assign `NULL` to a condition information item in a `SIGNAL` statement.

A `SIGNAL` statement always specifies an `SQLSTATE` value, either directly, or indirectly by referring to a named condition defined with an `SQLSTATE` value. The first two characters of an `SQLSTATE` value are its class, and the class determines the default value for the condition information items:

- Class = `'00'` (success)

  Illegal. `SQLSTATE` values that begin with `'00'` indicate success and are not valid for `SIGNAL`.

- Class = `'01'` (warning)

  ```
  MESSAGE_TEXT = 'Unhandled user-defined warning condition';
  MYSQL_ERRNO = ER_SIGNAL_WARN
  ```

- Class = `'02'` (not found)

  ```
  MESSAGE_TEXT = 'Unhandled user-defined not found condition';
  MYSQL_ERRNO = ER_SIGNAL_NOT_FOUND
  ```

- Class > `'02'` (exception)

  ```
  MESSAGE_TEXT = 'Unhandled user-defined exception condition';
  MYSQL_ERRNO = ER_SIGNAL_EXCEPTION
  ```

For legal classes, the other condition information items are set as follows:

```
CLASS_ORIGIN = SUBCLASS_ORIGIN = '';
CONSTRAINT_CATALOG = CONSTRAINT_SCHEMA = CONSTRAINT_NAME = '';
CATALOG_NAME = SCHEMA_NAME = TABLE_NAME = COLUMN_NAME = '';
CURSOR_NAME = '';
```

The error values that are accessible after `SIGNAL` executes are the `SQLSTATE` value raised by the `SIGNAL` statement and the `MESSAGE_TEXT` and `MYSQL_ERRNO` items. These values are available from the C API:

- `mysql_sqlstate()` returns the `SQLSTATE` value.
- `mysql_errno()` returns the `MYSQL_ERRNO` value.
- `mysql_error()` returns the `MESSAGE_TEXT` value.

At the SQL level, the output from `SHOW WARNINGS` and `SHOW ERRORS` indicates the `MYSQL_ERRNO` and `MESSAGE_TEXT` values in the `Code` and `Message` columns.

To retrieve information from the diagnostics area, use the `GET DIAGNOSTICS` statement (see Section 13.6.7.3, “GET DIAGNOSTICS Statement”). For information about the diagnostics area, see Section 13.6.7.7, “The MySQL Diagnostics Area”.

Effect of Signals on Handlers, Cursors, and Statements

Signals have different effects on statement execution depending on the signal class. The class determines how severe an error is. MySQL ignores the value of the `sql_mode` system variable; in particular, strict SQL mode does not matter. MySQL also ignores `IGNORE`: The intent of `SIGNAL` is to raise a user-generated error explicitly, so a signal is never ignored.

In the following descriptions, “unhandled” means that no handler for the signaled `SQLSTATE` value has been defined with `DECLARE ... HANDLER`.

- Class = `'00'` (success)

Illegal. SQLSTATE values that begin with '00' indicate success and are not valid for SIGNAL.

- Class = '01' (warning)

  The value of the warning_count system variable goes up. SHOW WARNINGS shows the signal. SQLWARNING handlers catch the signal.

  Warnings cannot be returned from stored functions because the RETURN statement that causes the function to return clears the diagnostic area. The statement thus clears any warnings that may have been present there (and resets warning_count to 0).

- Class = '02' (not found)

  NOT FOUND handlers catch the signal. There is no effect on cursors. If the signal is unhandled in a stored function, statements end.

- Class > '02' (exception)

  SQLEXCEPTION handlers catch the signal. If the signal is unhandled in a stored function, statements end.

- Class = '40'

  Treated as an ordinary exception.

13.6.7.6 Scope Rules for Handlers

A stored program may include handlers to be invoked when certain conditions occur within the program. The applicability of each handler depends on its location within the program definition and on the condition or conditions that it handles:

- A handler declared in a BEGIN ... END block is in scope only for the SQL statements following the handler declarations in the block. If the handler itself raises a condition, it cannot handle that condition, nor can any other handlers declared in the block. In the following example, handlers H1 and H2 are in scope for conditions raised by statements *stmt1* and *stmt2*. But neither H1 nor H2 are in scope for conditions raised in the body of H1 or H2.

```
BEGIN -- outer block
  DECLARE EXIT HANDLER FOR ...;  -- handler H1
  DECLARE EXIT HANDLER FOR ...;  -- handler H2
  stmt1;
  stmt2;
END;
```

- A handler is in scope only for the block in which it is declared, and cannot be activated for conditions occurring outside that block. In the following example, handler H1 is in scope for *stmt1* in the inner block, but not for *stmt2* in the outer block:

```
BEGIN -- outer block
  BEGIN -- inner block
    DECLARE EXIT HANDLER FOR ...;  -- handler H1
    stmt1;
  END;
  stmt2;
END;
```

- A handler can be specific or general. A specific handler is for a MySQL error code, SQLSTATE value, or condition name. A general handler is for a condition in the SQLWARNING, SQLEXCEPTION, or NOT FOUND class. Condition specificity is related to condition precedence, as described later.

Multiple handlers can be declared in different scopes and with different specificities. For example, there might be a specific MySQL error code handler in an outer block, and a general SQLWARNING handler in an inner block. Or there might be handlers for a specific MySQL error code and the general SQLWARNING class in the same block.

Whether a handler is activated depends not only on its own scope and condition value, but on what other handlers are present. When a condition occurs in a stored program, the server searches for applicable handlers in the current scope (current `BEGIN ... END` block). If there are no applicable handlers, the search continues outward with the handlers in each successive containing scope (block). When the server finds one or more applicable handlers at a given scope, it chooses among them based on condition precedence:

- A MySQL error code handler takes precedence over an `SQLSTATE` value handler.
- An `SQLSTATE` value handler takes precedence over general `SQLWARNING`, `SQLEXCEPTION`, or `NOT FOUND` handlers.
- An `SQLEXCEPTION` handler takes precedence over an `SQLWARNING` handler.
- It is possible to have several applicable handlers with the same precedence. For example, a statement could generate multiple warnings with different error codes, for each of which an error-specific handler exists. In this case, the choice of which handler the server activates is nondeterministic, and may change depending on the circumstances under which the condition occurs.

One implication of the handler selection rules is that if multiple applicable handlers occur in different scopes, handlers with the most local scope take precedence over handlers in outer scopes, even over those for more specific conditions.

If there is no appropriate handler when a condition occurs, the action taken depends on the class of the condition:

- For `SQLEXCEPTION` conditions, the stored program terminates at the statement that raised the condition, as if there were an `EXIT` handler. If the program was called by another stored program, the calling program handles the condition using the handler selection rules applied to its own handlers.
- For `SQLWARNING` conditions, the program continues executing, as if there were a `CONTINUE` handler.
- For `NOT FOUND` conditions, if the condition was raised normally, the action is `CONTINUE`. If it was raised by `SIGNAL` or `RESIGNAL`, the action is `EXIT`.

The following examples demonstrate how MySQL applies the handler selection rules.

This procedure contains two handlers, one for the specific `SQLSTATE` value (`'42S02'`) that occurs for attempts to drop a nonexistent table, and one for the general `SQLEXCEPTION` class:

```
CREATE PROCEDURE p1()
BEGIN
  DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
    SELECT 'SQLSTATE handler was activated' AS msg;
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    SELECT 'SQLEXCEPTION handler was activated' AS msg;

  DROP TABLE test.t;
END;
```

Both handlers are declared in the same block and have the same scope. However, `SQLSTATE` handlers take precedence over `SQLEXCEPTION` handlers, so if the table `t` is nonexistent, the `DROP TABLE` statement raises a condition that activates the `SQLSTATE` handler:

```
mysql> CALL p1();
+-------------------------------+
| msg                           |
+-------------------------------+
| SQLSTATE handler was activated |
+-------------------------------+
```

This procedure contains the same two handlers. But this time, the `DROP TABLE` statement and `SQLEXCEPTION` handler are in an inner block relative to the `SQLSTATE` handler:

```
CREATE PROCEDURE p2()
BEGIN -- outer block
    DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
      SELECT 'SQLSTATE handler was activated' AS msg;
  BEGIN -- inner block
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
      SELECT 'SQLEXCEPTION handler was activated' AS msg;

    DROP TABLE test.t; -- occurs within inner block
  END;
END;
```

In this case, the handler that is more local to where the condition occurs takes precedence. The `SQLEXCEPTION` handler activates, even though it is more general than the `SQLSTATE` handler:

```
mysql> CALL p2();
+-----------------------------------+
| msg                               |
+-----------------------------------+
| SQLEXCEPTION handler was activated |
+-----------------------------------+
```

In this procedure, one of the handlers is declared in a block inner to the scope of the `DROP TABLE` statement:

```
CREATE PROCEDURE p3()
BEGIN -- outer block
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
    SELECT 'SQLEXCEPTION handler was activated' AS msg;
  BEGIN -- inner block
    DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
      SELECT 'SQLSTATE handler was activated' AS msg;
  END;

  DROP TABLE test.t; -- occurs within outer block
END;
```

Only the `SQLEXCEPTION` handler applies because the other one is not in scope for the condition raised by the `DROP TABLE`:

```
mysql> CALL p3();
+-----------------------------------+
| msg                               |
+-----------------------------------+
| SQLEXCEPTION handler was activated |
+-----------------------------------+
```

In this procedure, both handlers are declared in a block inner to the scope of the `DROP TABLE` statement:

```
CREATE PROCEDURE p4()
BEGIN -- outer block
  BEGIN -- inner block
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION
      SELECT 'SQLEXCEPTION handler was activated' AS msg;
    DECLARE CONTINUE HANDLER FOR SQLSTATE '42S02'
      SELECT 'SQLSTATE handler was activated' AS msg;
  END;

  DROP TABLE test.t; -- occurs within outer block
END;
```

Neither handler applies because they are not in scope for the `DROP TABLE`. The condition raised by the statement goes unhandled and terminates the procedure with an error:

```
mysql> CALL p4();
```

```
ERROR 1051 (42S02): Unknown table 'test.t'
```

13.6.7.7 The MySQL Diagnostics Area

SQL statements produce diagnostic information that populates the diagnostics area. Standard SQL has a diagnostics area stack, containing a diagnostics area for each nested execution context. Standard SQL also supports `GET STACKED DIAGNOSTICS` syntax for referring to the second diagnostics area during condition handler execution. MySQL does not support the `STACKED` keyword until MySQL 5.7. In MySQL 5.6, there is a single diagnostics area containing information from the most recent statement that wrote to it.

The following discussion describes the structure of the diagnostics area in MySQL, the information items recognized by MySQL and how statements clear and set the diagnostics area.

- Diagnostics Area Structure
- Diagnostics Area Information Items
- How the Diagnostics Area is Cleared and Populated
- Diagnostics Area-Related System Variables

Diagnostics Area Structure

The diagnostics area contains two kinds of information:

- Statement information, such as the number of conditions that occurred or the affected-rows count.
- Condition information, such as the error code and message. If a statement raises multiple conditions, this part of the diagnostics area has a condition area for each one. If a statement raises no conditions, this part of the diagnostics area is empty.

For a statement that produces three conditions, the diagnostics area contains statement and condition information like this:

```
Statement information:
  row count
  ... other statement information items ...
Condition area list:
  Condition area 1:
    error code for condition 1
    error message for condition 1
    ... other condition information items ...
  Condition area 2:
    error code for condition 2:
    error message for condition 2
    ... other condition information items ...
  Condition area 3:
    error code for condition 3
    error message for condition 3
    ... other condition information items ...
```

Diagnostics Area Information Items

The diagnostics area contains statement and condition information items. Numeric items are integers. The character set for character items is UTF-8. No item can be `NULL`. If a statement or condition item is not set by a statement that populates the diagnostics area, its value is 0 or the empty string, depending on the item data type.

The statement information part of the diagnostics area contains these items:

- `NUMBER`: An integer indicating the number of condition areas that have information.
- `ROW_COUNT`: An integer indicating the number of rows affected by the statement. `ROW_COUNT` has the same value as the `ROW_COUNT()` function (see Section 12.16, “Information Functions”).

The condition information part of the diagnostics area contains a condition area for each condition. Condition areas are numbered from 1 to the value of the NUMBER statement condition item. If NUMBER is 0, there are no condition areas.

Each condition area contains the items in the following list. All items are standard SQL except MYSQL_ERRNO, which is a MySQL extension. The definitions apply for conditions generated other than by a signal (that is, by a SIGNAL or RESIGNAL statement). For nonsignal conditions, MySQL populates only those condition items not described as always empty. The effects of signals on the condition area are described later.

- CLASS_ORIGIN: A string containing the class of the RETURNED_SQLSTATE value. If the RETURNED_SQLSTATE value begins with a class value defined in SQL standards document ISO 9075-2 (section 24.1, SQLSTATE), CLASS_ORIGIN is 'ISO 9075'. Otherwise, CLASS_ORIGIN is 'MySQL'.
- SUBCLASS_ORIGIN: A string containing the subclass of the RETURNED_SQLSTATE value. If CLASS_ORIGIN is 'ISO 9075' or RETURNED_SQLSTATE ends with '000', SUBCLASS_ORIGIN is 'ISO 9075'. Otherwise, SUBCLASS_ORIGIN is 'MySQL'.
- RETURNED_SQLSTATE: A string that indicates the SQLSTATE value for the condition.
- MESSAGE_TEXT: A string that indicates the error message for the condition.
- MYSQL_ERRNO: An integer that indicates the MySQL error code for the condition.
- CONSTRAINT_CATALOG, CONSTRAINT_SCHEMA, CONSTRAINT_NAME: Strings that indicate the catalog, schema, and name for a violated constraint. They are always empty.
- CATALOG_NAME, SCHEMA_NAME, TABLE_NAME, COLUMN_NAME: Strings that indicate the catalog, schema, table, and column related to the condition. They are always empty.
- CURSOR_NAME: A string that indicates the cursor name. This is always empty.

For the RETURNED_SQLSTATE, MESSAGE_TEXT, and MYSQL_ERRNO values for particular errors, see Server Error Message Reference.

If a SIGNAL (or RESIGNAL) statement populates the diagnostics area, its SET clause can assign to any condition information item except RETURNED_SQLSTATE any value that is legal for the item data type. SIGNAL also sets the RETURNED_SQLSTATE value, but not directly in its SET clause. That value comes from the SIGNAL statement SQLSTATE argument.

SIGNAL also sets statement information items. It sets NUMBER to 1. It sets ROW_COUNT to −1 for errors and 0 otherwise.

How the Diagnostics Area is Cleared and Populated

Most nondiagnostic SQL statements populate the diagnostics area automatically, and its contents can be set explicitly with the SIGNAL and RESIGNAL statements. The diagnostics area can be examined with GET DIAGNOSTICS to extract specific items, or with SHOW WARNINGS or SHOW ERRORS to see conditions or errors.

SQL statements clear and set the diagnostics area as follows:

- When the server starts executing a statement after parsing it, it clears the diagnostics area for nondiagnostic statements that use tables. Diagnostic statements do not clear the diagnostics area. These statements are diagnostic:
  - GET DIAGNOSTICS
  - SHOW ERRORS
  - SHOW WARNINGS

- If a statement raises a condition, the diagnostics area is cleared of conditions that belong to earlier statements. The exception is that conditions raised by `GET DIAGNOSTICS` and `RESIGNAL` are added to the diagnostics area without clearing it.

Thus, even a statement that does not normally clear the diagnostics area when it begins executing clears it if the statement raises a condition.

The following example shows the effect of various statements on the diagnostics area, using `SHOW WARNINGS` to display information about conditions stored there.

This `DROP TABLE` statement uses a table, so it clears the diagnostics area and populates it when the condition occurs:

```
mysql> DROP TABLE IF EXISTS test.no_such_table;
Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql> SHOW WARNINGS;
+-------+------+----------------------------------------+
| Level | Code | Message                                |
+-------+------+----------------------------------------+
| Note  | 1051 | Unknown table 'test.no_such_table' |
+-------+------+----------------------------------------+
1 row in set (0.00 sec)
```

This `SET` statement does not use tables and does not generate warnings, so it leaves the diagnostics area unchanged:

```
mysql> SET @x = 1;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW WARNINGS;
+-------+------+----------------------------------------+
| Level | Code | Message                                |
+-------+------+----------------------------------------+
| Note  | 1051 | Unknown table 'test.no_such_table' |
+-------+------+----------------------------------------+
1 row in set (0.00 sec)
```

This `SET` statement generates an error, so it clears and populates the diagnostics area:

```
mysql> SET @x = @@x;
ERROR 1193 (HY000): Unknown system variable 'x'

mysql> SHOW WARNINGS;
+-------+------+-----------------------------+
| Level | Code | Message                     |
+-------+------+-----------------------------+
| Error | 1193 | Unknown system variable 'x' |
+-------+------+-----------------------------+
1 row in set (0.00 sec)
```

The previous `SET` statement produced a single condition, so 1 is the only valid condition number for `GET DIAGNOSTICS` at this point. The following statement uses a condition number of 2, which produces a warning that is added to the diagnostics area without clearing it:

```
mysql> GET DIAGNOSTICS CONDITION 2 @p = MESSAGE_TEXT;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> SHOW WARNINGS;
+-------+------+-----------------------------+
| Level | Code | Message                     |
+-------+------+-----------------------------+
| Error | 1193 | Unknown system variable 'xx' |
| Error | 1753 | Invalid condition number     |
+-------+------+-----------------------------+
2 rows in set (0.00 sec)
```

Now there are two conditions in the diagnostics area, so the same `GET DIAGNOSTICS` statement succeeds:

```
mysql> GET DIAGNOSTICS CONDITION 2 @p = MESSAGE_TEXT;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @p;
+--------------------------+
| @p                       |
+--------------------------+
| Invalid condition number |
+--------------------------+
1 row in set (0.01 sec)
```

Diagnostics Area-Related System Variables

Certain system variables control or are related to some aspects of the diagnostics area:

- `max_error_count` controls the number of condition areas in the diagnostics area. If more conditions than this occur, MySQL silently discards information for the excess conditions. (Conditions added by `RESIGNAL` are always added, with older conditions being discarded as necessary to make room.)
- `warning_count` indicates the number of conditions that occurred. This includes errors, warnings, and notes. Normally, `NUMBER` and `warning_count` are the same. However, as the number of conditions generated exceeds `max_error_count`, the value of `warning_count` continues to rise whereas `NUMBER` remains capped at `max_error_count` because no additional conditions are stored in the diagnostics area.
- `error_count` indicates the number of errors that occurred. This value includes “not found” and exception conditions, but excludes warnings and notes. Like `warning_count`, its value can exceed `max_error_count`.
- If the `sql_notes` system variable is set to 0, notes are not stored and do not increment `warning_count`.

Example: If `max_error_count` is 10, the diagnostics area can contain a maximum of 10 condition areas. Suppose that a statement raises 20 conditions, 12 of which are errors. In that case, the diagnostics area contains the first 10 conditions, `NUMBER` is 10, `warning_count` is 20, and `error_count` is 12.

Changes to the value of `max_error_count` have no effect until the next attempt to modify the diagnostics area. If the diagnostics area contains 10 condition areas and `max_error_count` is set to 5, that has no immediate effect on the size or content of the diagnostics area.

Before MySQL 5.6, statement information items are not available directly. `ROW_COUNT` can be obtained by calling the `ROW_COUNT()` function. `NUMBER` is approximated by the value of the `warning_count` system variable. However, whereas `NUMBER` is capped to the value of `max_error_count`, `warning_count` is not.

13.6.7.8 Condition Handling and OUT or INOUT Parameters

If a stored procedure exits with an unhandled exception, modified values of `OUT` and `INOUT` parameters are not propogated back to the caller.

If an exception is handled by a `CONTINUE` or `EXIT` handler that contains a `RESIGNAL` statement, execution of `RESIGNAL` pops the Diagnostics Area stack, thus signalling the exception (that is, the information that existed before entry into the handler). If the exception is an error, the values of `OUT` and `INOUT` parameters are not propogated back to the caller.

13.6.7.9 Restrictions on Condition Handling

`SIGNAL`, `RESIGNAL`, and `GET DIAGNOSTICS` are not permissible as prepared statements. For example, this statement is invalid:

```
PREPARE stmt1 FROM 'SIGNAL SQLSTATE "02000"';
```

`SQLSTATE` values in class `'04'` are not treated specially. They are handled the same as other exceptions.

Standard SQL has a diagnostics area stack, containing a diagnostics area for each nested execution context. Standard SQL syntax includes `GET STACKED DIAGNOSTICS` for referring to stacked areas. MySQL does not support the `STACKED` keyword because there is a single diagnostics area containing information from the most recent statement that wrote to it. See also Section 13.6.7.7, "The MySQL Diagnostics Area".

In standard SQL, the first condition relates to the `SQLSTATE` value returned for the previous SQL statement. In MySQL, this is not guaranteed, so to get the main error, you cannot do this:

```
GET DIAGNOSTICS CONDITION 1 @errno = MYSQL_ERRNO;
```

Instead, do this:

```
GET DIAGNOSTICS @cno = NUMBER;
GET DIAGNOSTICS CONDITION @cno @errno = MYSQL_ERRNO;
```

13.7 Database Administration Statements

13.7.1 Account Management Statements

MySQL account information is stored in the tables of the `mysql` system database. This database and the access control system are discussed extensively in Chapter 5, *MySQL Server Administration*, which you should consult for additional details.

Important

Some MySQL releases introduce changes to the grant tables to add new privileges or features. To make sure that you can take advantage of any new capabilities, update your grant tables to the current structure whenever you upgrade MySQL. See Section 2.11, "Upgrading MySQL".

When the `read_only` system variable is enabled, account-management statements require the `SUPER` privilege, in addition to any other required privileges. This is because they modify tables in the `mysql` system database.

13.7.1.1 ALTER USER Statement

```
ALTER USER user_specification [, user_specification] ...

user_specification:
    user PASSWORD EXPIRE
```

The `ALTER USER` statement modifies MySQL accounts. An error occurs if you try to modify a nonexistent account.

To use `ALTER USER`, you must have the global `CREATE USER` privilege or the `UPDATE` privilege for the `mysql` system database. When the `read_only` system variable is enabled, `ALTER USER` additionally requires the `SUPER` privilege.

Each account name uses the format described in Section 6.2.4, "Specifying Account Names". The host name part of the account name, if omitted, defaults to `'%'`. It is also possible to specify `CURRENT_USER` or `CURRENT_USER()` to refer to the account associated with the current session.

For each account, `ALTER USER` expires its password. For example:

```
ALTER USER 'jeffrey'@'localhost' PASSWORD EXPIRE;
```

Password expiration for an account affects the corresponding row of the `mysql.user` system table: The server sets the `password_expired` column to `'Y'`.

A client session operates in restricted mode if the account password has been expired. In restricted mode, operations performed within the session result in an error until the user establishes a new account password:

```
mysql> SELECT 1;
ERROR 1820 (HY000): You must SET PASSWORD before executing this statement

mysql> SET PASSWORD = PASSWORD('new_password');
Query OK, 0 rows affected (0.01 sec)

mysql> SELECT 1;
+---+
| 1 |
+---+
| 1 |
+---+
1 row in set (0.00 sec)
```

This restricted mode of operation permits `SET` statements, which is useful if the account password has a hashing format that requires `old_passwords` to be set to a value different from its default before using `SET PASSWORD`.

It is possible for an administrative user to reset the account password, but any existing sessions for the account remain restricted. A client using the account must disconnect and reconnect before statements can be executed successfully.

Note

Although it is possible to “reset” an expired password by setting it to its current value, it is preferable, as a matter of good policy, to choose a different password.

13.7.1.2 CREATE USER Statement

```
CREATE USER
    user [auth_option] [, user [auth_option]] ...

user:
    (see Section 6.2.4, “Specifying Account Names”)

auth_option: {
    IDENTIFIED BY [PASSWORD] 'auth_string'
  | IDENTIFIED WITH auth_plugin [AS 'auth_string']
}
```

The `CREATE USER` statement creates new MySQL accounts. An error occurs if you try to create an account that already exists.

To use `CREATE USER`, you must have the global `CREATE USER` privilege, or the `INSERT` privilege for the `mysql` system database. When the `read_only` system variable is enabled, `CREATE USER` additionally requires the `SUPER` privilege.

Important

Under some circumstances, `CREATE USER` may be recorded in server logs or on the client side in a history file such as `~/.mysql_history`, which means that cleartext passwords may be read by anyone having read access to that information. For information about the conditions under which this occurs for the server logs and how to control it, see Section 6.1.2.3, “Passwords and Logging”.

> For similar information about client-side logging, see Section 4.5.1.3, “mysql Client Logging”.

For each account, `CREATE USER` creates a new row in the `mysql.user` system table with no privileges and assigns the account an authentication plugin. Depending on the syntax used, `CREATE USER` may also assign the account a password.

An account when first created has no privileges. To assign privileges, use the `GRANT` statement.

Each *`user`* value naming an account may be followed by an optional *`auth_option`* value that specifies how authentication occurs for clients that use the account. This part of `CREATE USER` syntax is shared with `GRANT`, so the description here applies to `GRANT` as well.

Each account name uses the format described in Section 6.2.4, “Specifying Account Names”. For example:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'password';
```

The host name part of the account name, if omitted, defaults to `'%'`.

The server assigns an authentication plugin and password to each account as follows, depending on whether the user specification clause includes `IDENTIFIED BY` or `IDENTIFIED WITH` to specify authentication information:

- With `IDENTIFIED BY`, the server assigns the plugin implicitly. The `'auth_string'` value is cleartext and is passed to the plugin for possible hashing. The result returned by the plugin is stored as the password. A plugin may use the value as specified, in which case no hashing occurs. If the optional `PASSWORD` keyword is given, the behavior is the same except that if the plugin requires a hashed string, the string is assumed to be already hashed in the format the plugin requires, and is stored as is as the password.

- With `IDENTIFIED WITH`, the server assigns the specified plugin and the account has no password. If the optional `AS 'auth_string'` clause is given, the string is stored as is as the password. If the plugin requires a hashed string, the string is assumed to be already hashed in the format the plugin requires.

- With neither `IDENTIFIED BY` nor `IDENTIFIED WITH`, the server assigns the plugin implicitly and the account has no password.

If the account has no password, the password information in the account row in the `mysql.user` table remains empty, which is insecure. To set the password, use `SET PASSWORD`. See Section 13.7.1.7, “SET PASSWORD Statement”.

For implicit plugin assignment, the default plugin becomes the value of the `plugin` column in the account's `mysql.user` system table row. The default plugin is `mysql_native_password` unless the `--default-authentication-plugin` option is set otherwise at server startup.

For client connections that use a given account, the server invokes the authentication plugin assigned to the account and the client must provide credentials as required by the authentication method that the plugin implements. If the server cannot find the plugin, either at account-creation time or connect time, an error occurs.

If an account's `mysql.user` table row has a nonempty `plugin` column:

- The server authenticates client connection attempts using the named plugin.

- Changes to the account password using `SET PASSWORD` with `PASSWORD()` must be made with the `old_passwords` system variable set to the value required by the authentication plugin, so that `PASSWORD()` uses the appropriate password hashing method. If the plugin is `mysql_old_password`, the password can also be changed using `SET PASSWORD`

with OLD_PASSWORD(), which uses pre-4.1 password hashing regardless of the value of old_passwords.

If an account's mysql.user table row has an empty plugin column:

- The server authenticates client connection attempts using the mysql_native_password or mysql_old_password authentication plugin, depending on the hash format of the password stored in the Password column.
- Changes to the account password using SET PASSWORD can be made with PASSWORD(), with old_passwords set to 0 or 1 for 4.1 or pre-4.1 password hashing, respectively, or with OLD_PASSWORD(), which uses pre-4.1 password hashing regardless of the value of old_passwords.

CREATE USER examples:

- To specify an authentication plugin for an account, use IDENTIFIED WITH *auth_plugin*. The plugin name can be a quoted string literal or an unquoted name. '*auth_string*' is an optional quoted string literal to pass to the plugin. The plugin interprets the meaning of the string, so its format is plugin specific and it is stored in the authentication_string column as given. (This value is meaningful only for plugins that use that column.) Consult the documentation for a given plugin for information about the authentication string values it accepts, if any.

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED WITH mysql_native_password;
```

The server assigns the given authentication plugin to the account but no password. Clients must provide no password when they connect. However, an account with no password is insecure. To ensure that an account uses a specific authentication plugin and has a password with the corresponding hash format, specify the plugin explicitly with IDENTIFIED WITH, then use SET PASSWORD to set the password:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED WITH mysql_native_password;
SET old_passwords = 0;
SET PASSWORD FOR 'jeffrey'@'localhost' = PASSWORD('password');
```

Changes to the account password using SET PASSWORD with PASSWORD() must be made with the old_passwords system variable set to the value required by the account's authentication plugin, so that PASSWORD() uses the appropriate password hashing method. Therefore, to use the sha256_password or mysql_old_password plugin instead, name that plugin in the CREATE USER statement and set old_passwords to 2 or 1, respectively, before using SET PASSWORD. (Use of mysql_old_password is not recommended. It is deprecated; expect support for it to removed in a future MySQL release.)

- To specify a password for an account at account-creation time, use IDENTIFIED BY with the literal cleartext password value:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'password';
```

The server assigns an authentication plugin to the account implicitly, as described previously, and assigns the given password. Clients must provide the given password when they connect.

If the implicitly assigned plugin is mysql_native_password, the old_passwords system variable must be set to 0. Otherwise, CREATE USER does not hash the password in the format required by the plugin and an error occurs:

```
mysql> SET old_passwords = 1;
mysql> CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'password';
ERROR 1827 (HY000): The password hash doesn't have the expected
format. Check if the correct password algorithm is being used with
the PASSWORD() function.

mysql> SET old_passwords = 0;
mysql> CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'password';
```

```
Query OK, 0 rows affected (0.00 sec)
```

- To avoid specifying the cleartext password if you know its hash value (the value that `PASSWORD()` would return for the password), specify the hash value preceded by the keyword `PASSWORD`:

  ```
  CREATE USER 'jeffrey'@'localhost'
  IDENTIFIED BY PASSWORD '*90E462C37378CED12064BB3388827D2BA3A9B689';
  ```

  The server assigns an authentication plugin to the account implicitly, as described previously, and assigns the given password. The password hash must be in the format required by the assigned plugin. Clients must provide the password when they connect.

- To enable the user to connect with no password, include no `IDENTIFIED BY` clause:

  ```
  CREATE USER 'jeffrey'@'localhost';
  ```

  The server assigns an authentication plugin to the account implicitly, as described previously, but no password. Clients must provide no password when they connect. However, an account with no password is insecure. To avoid this, use `SET PASSWORD` to set the account password.

As mentioned previously, implicit plugin assignment depends on the default authentication plugin. Permitted values of `--default-authentication-plugin` are `mysql_native_plugin` and `sha256_password`, but not `mysql_old_password`. This means it is not possible to set the default plugin so as to be able to create an account that uses `mysql_old_password` with `CREATE USER ... IDENTIFIED BY` syntax. To create an account that uses `mysql_old_password`, use `CREATE USER ... IDENTIFIED WITH` to name the plugin explicitly, then set the password:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED WITH mysql_old_password;
SET old_passwords = 1;
SET PASSWORD FOR 'jeffrey'@'localhost' = PASSWORD('password');
```

However, the preceding procedure is not recommended because `mysql_old_password` is deprecated.

For additional information about setting passwords and authentication plugins, see Section 6.2.9, “Assigning Account Passwords”, and Section 6.2.11, “Pluggable Authentication”.

13.7.1.3 DROP USER Statement

```
DROP USER user [, user] ...
```

The `DROP USER` statement removes one or more MySQL accounts and their privileges. It removes privilege rows for the account from all grant tables. An error occurs for accounts that do not exist.

To use `DROP USER`, you must have the global `CREATE USER` privilege, or the `DELETE` privilege for the `mysql` system database. When the `read_only` system variable is enabled, `DROP USER` additionally requires the `SUPER` privilege.

Each account name uses the format described in Section 6.2.4, “Specifying Account Names”. For example:

```
DROP USER 'jeffrey'@'localhost';
```

The host name part of the account name, if omitted, defaults to `'%'`.

Important

`DROP USER` does not automatically close any open user sessions. Rather, in the event that a user with an open session is dropped, the statement does not take effect until that user's session is closed. Once the session is closed, the user is dropped, and that user's next attempt to log in fails. *This is by design*.

`DROP USER` does not automatically drop or invalidate databases or objects within them that the old user created. This includes stored programs or views for which the `DEFINER` attribute names the

dropped user. Attempts to access such objects may produce an error if they execute in definer security context. (For information about security context, see Section 20.6, “Stored Object Access Control”.)

13.7.1.4 GRANT Statement

```
GRANT
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    ON [object_type] priv_level
    TO user [auth_option] [, user [auth_option]] ...
    [REQUIRE {NONE | tls_option [[AND] tls_option] ...}]
    [WITH {GRANT OPTION | resource_option} ...]

GRANT PROXY ON user
    TO user [, user] ...
    [WITH GRANT OPTION]

object_type: {
    TABLE
  | FUNCTION
  | PROCEDURE
}

priv_level: {
    *
  | *.*
  | db_name.*
  | db_name.tbl_name
  | tbl_name
  | db_name.routine_name
}

user:
    (see Section 6.2.4, “Specifying Account Names”)

auth_option: {
    IDENTIFIED BY [PASSWORD] 'auth_string'
  | IDENTIFIED WITH auth_plugin
  | IDENTIFIED WITH auth_plugin AS 'auth_string'
}

tls_option: {
    SSL
  | X509
  | CIPHER 'cipher'
  | ISSUER 'issuer'
  | SUBJECT 'subject'
}

resource_option: {
  | MAX_QUERIES_PER_HOUR count
  | MAX_UPDATES_PER_HOUR count
  | MAX_CONNECTIONS_PER_HOUR count
  | MAX_USER_CONNECTIONS count
}
```

The `GRANT` statement grants privileges to MySQL user accounts. There are several aspects to the `GRANT` statement, described under the following topics:

- GRANT General Overview
- Object Quoting Guidelines
- Account Names and Passwords
- Privileges Supported by MySQL
- Global Privileges
- Database Privileges

- Table Privileges
- Column Privileges
- Stored Routine Privileges
- Proxy User Privileges
- Implicit Account Creation
- Other Account Characteristics
- MySQL and Standard SQL Versions of GRANT

GRANT General Overview

The `GRANT` statement grants privileges to MySQL user accounts. `GRANT` also serves to specify other account characteristics such as use of secure connections and limits on access to server resources.

To grant a privilege with `GRANT`, you must have the `GRANT OPTION` privilege, and you must have the privileges that you are granting. (Alternatively, if you have the `UPDATE` privilege for the grant tables in the `mysql` system database, you can grant any account any privilege.) When the `read_only` system variable is enabled, `GRANT` additionally requires the `SUPER` privilege.

The `REVOKE` statement is related to `GRANT` and enables administrators to remove account privileges. See Section 13.7.1.6, “REVOKE Statement”.

Each account name uses the format described in Section 6.2.4, “Specifying Account Names”. For example:

```
GRANT ALL ON db1.* TO 'jeffrey'@'localhost';
```

The host name part of the account, if omitted, defaults to `'%'`.

Normally, a database administrator first uses `CREATE USER` to create an account, then `GRANT` to define its privileges and characteristics. For example:

```
CREATE USER 'jeffrey'@'localhost' IDENTIFIED BY 'password';
GRANT ALL ON db1.* TO 'jeffrey'@'localhost';
GRANT SELECT ON db2.invoice TO 'jeffrey'@'localhost';
GRANT USAGE ON *.* TO 'jeffrey'@'localhost' WITH MAX_QUERIES_PER_HOUR 90;
```

Note

Examples shown here include no `IDENTIFIED` clause. It is assumed that you establish passwords with `CREATE USER` at account-creation time to avoid creating insecure accounts.

If an account named in a `GRANT` statement does not already exist, `GRANT` may create it under the conditions described later in the discussion of the `NO_AUTO_CREATE_USER` SQL mode.

From the `mysql` program, `GRANT` responds with `Query OK, 0 rows affected` when executed successfully. To determine what privileges result from the operation, use `SHOW GRANTS`. See Section 13.7.5.22, “SHOW GRANTS Statement”.

Important

Under some circumstances, `GRANT` may be recorded in server logs or on the client side in a history file such as `~/.mysql_history`, which means that cleartext passwords may be read by anyone having read access to that information. For information about the conditions under which this occurs for the server logs and how to control it, see Section 6.1.2.3, “Passwords and Logging”.

> For similar information about client-side logging, see Section 4.5.1.3, “mysql Client Logging”.

`GRANT` supports host names up to 60 characters long. User names can be up to 16 characters. Database, table, column, and routine names can be up to 64 characters.

> **Warning**
>
> *Do not attempt to change the permissible length for user names by altering the `mysql.user` system table. Doing so results in unpredictable behavior which may even make it impossible for users to log in to the MySQL server*. Never alter the structure of tables in the `mysql` system database in any manner except by means of the procedure described in Section 2.11, “Upgrading MySQL”.

Object Quoting Guidelines

Several objects within `GRANT` statements are subject to quoting, although quoting is optional in many cases: Account, database, table, column, and routine names. For example, if a *`user_name`* or *`host_name`* value in an account name is legal as an unquoted identifier, you need not quote it. However, quotation marks are necessary to specify a *`user_name`* string containing special characters (such as `-`), or a *`host_name`* string containing special characters or wildcard characters such as `%` (for example, `'test-user'@'%.com'`). Quote the user name and host name separately.

To specify quoted values:

- Quote database, table, column, and routine names as identifiers.
- Quote user names and host names as identifiers or as strings.
- Quote passwords as strings.

For string-quoting and identifier-quoting guidelines, see Section 9.1.1, “String Literals”, and Section 9.2, “Schema Object Names”.

The `_` and `%` wildcards are permitted when specifying database names in `GRANT` statements that grant privileges at the database level (`GRANT ... ON db_name.*`). This means, for example, that to use a `_` character as part of a database name, specify it as `\_` in the `GRANT` statement, to prevent the user from being able to access additional databases matching the wildcard pattern (for example, `GRANT ... ON \`foo_bar\`.* TO ...`).

When a database name not is used to grant privileges at the database level, but as a qualifier for granting privileges to some other object such as a table or routine (for example, `GRANT ... ON db_name.tbl_name`), wildcard characters are treated as normal characters.

Account Names and Passwords

A *`user`* value in a `GRANT` statement indicates a MySQL account to which the statement applies. To accommodate granting rights to users from arbitrary hosts, MySQL supports specifying the *`user`* value in the form `'user_name'@'host_name'`.

You can specify wildcards in the host name. For example, `'user_name'@'%.example.com'` applies to *`user_name`* for any host in the `example.com` domain, and `'user_name'@'198.51.100.%'` applies to *`user_name`* for any host in the `198.51.100` class C subnet.

The simple form `'user_name'` is a synonym for `'user_name'@'%'`.

MySQL does not support wildcards in user names. To refer to an anonymous user, specify an account with an empty user name with the `GRANT` statement:

```
GRANT ALL ON test.* TO ''@'localhost' ...;
```

In this case, any user who connects from the local host with the correct password for the anonymous user is permitted access, with the privileges associated with the anonymous-user account.

For additional information about user name and host name values in account names, see Section 6.2.4, “Specifying Account Names”.

Warning

If you permit local anonymous users to connect to the MySQL server, you should also grant privileges to all local users as `'user_name'@'localhost'`. Otherwise, the anonymous user account for `localhost` in the `mysql.user` system table (created during MySQL installation) is used when named users try to log in to the MySQL server from the local machine. For details, see Section 6.2.5, “Access Control, Stage 1: Connection Verification”.

To determine whether this issue applies to you, execute the following query, which lists any anonymous users:

```
SELECT Host, User FROM mysql.user WHERE User='';
```

To avoid the problem just described, delete the local anonymous user account using this statement:

```
DROP USER ''@'localhost';
```

For `GRANT` syntax that permits an *`auth_option`* value to follow a *`user`* value, *`auth_option`* begins with `IDENTIFIED` and indicates how the account authenticates by specifying an account authentication plugin, credentials (for example, a password), or both. Syntax of the *`auth_option`* clause is the same as for the `CREATE USER` statement. For details, see Section 13.7.1.2, “CREATE USER Statement”.

When `IDENTIFIED` is present and you have the global grant privilege (`GRANT OPTION`), any password specified becomes the new password for the account, even if the account exists and already has a password. Without `IDENTIFIED`, the account password remains unchanged.

Privileges Supported by MySQL

The following table summarizes the permissible *`priv_type`* privilege types that can be specified for the `GRANT` and `REVOKE` statements, and the levels at which each privilege can be granted. For additional information about each privilege, see Section 6.2.2, “Privileges Provided by MySQL”.

Table 13.8 Permissible Privileges for GRANT and REVOKE

Privilege	Meaning and Grantable Levels
`ALL [PRIVILEGES]`	Grant all privileges at specified access level except `GRANT OPTION` and `PROXY`.
`ALTER`	Enable use of `ALTER TABLE`. Levels: Global, database, table.
`ALTER ROUTINE`	Enable stored routines to be altered or dropped. Levels: Global, database, routine.
`CREATE`	Enable database and table creation. Levels: Global, database, table.
`CREATE ROUTINE`	Enable stored routine creation. Levels: Global, database.
`CREATE TABLESPACE`	Enable tablespaces and log file groups to be created, altered, or dropped. Level: Global.
`CREATE TEMPORARY TABLES`	Enable use of `CREATE TEMPORARY TABLE`. Levels: Global, database.
`CREATE USER`	Enable use of `CREATE USER`, `DROP USER`, `RENAME USER`, and `REVOKE ALL PRIVILEGES`. Level: Global.

Privilege	Meaning and Grantable Levels
CREATE VIEW	Enable views to be created or altered. Levels: Global, database, table.
DELETE	Enable use of DELETE. Level: Global, database, table.
DROP	Enable databases, tables, and views to be dropped. Levels: Global, database, table.
EVENT	Enable use of events for the Event Scheduler. Levels: Global, database.
EXECUTE	Enable the user to execute stored routines. Levels: Global, database, routine.
FILE	Enable the user to cause the server to read or write files. Level: Global.
GRANT OPTION	Enable privileges to be granted to or removed from other accounts. Levels: Global, database, table, routine, proxy.
INDEX	Enable indexes to be created or dropped. Levels: Global, database, table.
INSERT	Enable use of INSERT. Levels: Global, database, table, column.
LOCK TABLES	Enable use of LOCK TABLES on tables for which you have the SELECT privilege. Levels: Global, database.
PROCESS	Enable the user to see all processes with SHOW PROCESSLIST. Level: Global.
PROXY	Enable user proxying. Level: From user to user.
REFERENCES	Enable foreign key creation. Levels: Global, database, table, column.
RELOAD	Enable use of FLUSH operations. Level: Global.
REPLICATION CLIENT	Enable the user to ask where source or replica servers are. Level: Global.
REPLICATION SLAVE	Enable replicas to read binary log events from the source. Level: Global.
SELECT	Enable use of SELECT. Levels: Global, database, table, column.
SHOW DATABASES	Enable SHOW DATABASES to show all databases. Level: Global.
SHOW VIEW	Enable use of SHOW CREATE VIEW. Levels: Global, database, table.
SHUTDOWN	Enable use of mysqladmin shutdown. Level: Global.
SUPER	Enable use of other administrative operations such as CHANGE MASTER TO, KILL, PURGE BINARY LOGS, SET GLOBAL, and mysqladmin debug command. Level: Global.
TRIGGER	Enable trigger operations. Levels: Global, database, table.
UPDATE	Enable use of UPDATE. Levels: Global, database, table, column.
USAGE	Synonym for “no privileges”

A trigger is associated with a table. To create or drop a trigger, you must have the TRIGGER privilege for the table, not the trigger.

In GRANT statements, the ALL [PRIVILEGES] or PROXY privilege must be named by itself and cannot be specified along with other privileges. ALL [PRIVILEGES] stands for all privileges available for the level at which privileges are to be granted except for the GRANT OPTION and PROXY privileges.

USAGE can be specified to create a user that has no privileges, or to specify the REQUIRE or WITH clauses for an account without changing its existing privileges.

MySQL account information is stored in the tables of the mysql system database. For additional details, consult Section 6.2, “Access Control and Account Management”, which discusses the mysql system database and the access control system extensively.

If the grant tables hold privilege rows that contain mixed-case database or table names and the `lower_case_table_names` system variable is set to a nonzero value, `REVOKE` cannot be used to revoke these privileges. It is necessary to manipulate the grant tables directly. (`GRANT` does not create such rows when `lower_case_table_names` is set, but such rows might have been created prior to setting that variable.)

Privileges can be granted at several levels, depending on the syntax used for the `ON` clause. For `REVOKE`, the same `ON` syntax specifies which privileges to remove.

For the global, database, table, and routine levels, `GRANT ALL` assigns only the privileges that exist at the level you are granting. For example, `GRANT ALL ON *db_name*.*` is a database-level statement, so it does not grant any global-only privileges such as `FILE`. Granting `ALL` does not assign the `GRANT OPTION` or `PROXY` privilege.

The *`object_type`* clause, if present, should be specified as `TABLE`, `FUNCTION`, or `PROCEDURE` when the following object is a table, a stored function, or a stored procedure.

The privileges that a user holds for a database, table, column, or routine are formed additively as the logical `OR` of the account privileges at each of the privilege levels, including the global level. It is not possible to deny a privilege granted at a higher level by absence of that privilege at a lower level. For example, this statement grants the `SELECT` and `INSERT` privileges globally:

```
GRANT SELECT, INSERT ON *.* TO u1;
```

The globally granted privileges apply to all databases, tables, and columns, even though not granted at any of those lower levels.

Details of the privilege-checking procedure are presented in Section 6.2.6, “Access Control, Stage 2: Request Verification”.

If you are using table, column, or routine privileges for even one user, the server examines table, column, and routine privileges for all users and this slows down MySQL a bit. Similarly, if you limit the number of queries, updates, or connections for any users, the server must monitor these values.

MySQL enables you to grant privileges on databases or tables that do not exist. For tables, the privileges to be granted must include the `CREATE` privilege. *This behavior is by design*, and is intended to enable the database administrator to prepare user accounts and privileges for databases or tables that are to be created at a later time.

Important

MySQL does not automatically revoke any privileges when you drop a database or table. However, if you drop a routine, any routine-level privileges granted for that routine are revoked.

Global Privileges

Global privileges are administrative or apply to all databases on a given server. To assign global privileges, use `ON *.*` syntax:

```
GRANT ALL ON *.* TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON *.* TO 'someuser'@'somehost';
```

The `CREATE TABLESPACE`, `CREATE USER`, `FILE`, `PROCESS`, `RELOAD`, `REPLICATION CLIENT`, `REPLICATION SLAVE`, `SHOW DATABASES`, `SHUTDOWN`, and `SUPER` privileges are administrative and can only be granted globally.

Other privileges can be granted globally or at more specific levels.

`GRANT OPTION` granted at the global level for any global privilege applies to all global privileges.

MySQL stores global privileges in the `mysql.user` system table.

Database Privileges

Database privileges apply to all objects in a given database. To assign database-level privileges, use `ON db_name.*` syntax:

```
GRANT ALL ON mydb.* TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON mydb.* TO 'someuser'@'somehost';
```

If you use `ON *` syntax (rather than `ON *.*`), privileges are assigned at the database level for the default database. An error occurs if there is no default database.

The `CREATE`, `DROP`, `EVENT`, `GRANT OPTION`, `LOCK TABLES`, and `REFERENCES` privileges can be specified at the database level. Table or routine privileges also can be specified at the database level, in which case they apply to all tables or routines in the database.

MySQL stores database privileges in the `mysql.db` system table.

Table Privileges

Table privileges apply to all columns in a given table. To assign table-level privileges, use `ON db_name.tbl_name` syntax:

```
GRANT ALL ON mydb.mytbl TO 'someuser'@'somehost';
GRANT SELECT, INSERT ON mydb.mytbl TO 'someuser'@'somehost';
```

If you specify *`tbl_name`* rather than *`db_name.tbl_name`*, the statement applies to *`tbl_name`* in the default database. An error occurs if there is no default database.

The permissible *`priv_type`* values at the table level are `ALTER`, `CREATE VIEW`, `CREATE`, `DELETE`, `DROP`, `GRANT OPTION`, `INDEX`, `INSERT`, `REFERENCES`, `SELECT`, `SHOW VIEW`, `TRIGGER`, and `UPDATE`.

Table-level privileges apply to base tables and views. They do not apply to tables created with `CREATE TEMPORARY TABLE`, even if the table names match. For information about `TEMPORARY` table privileges, see Section 13.1.17.2, “CREATE TEMPORARY TABLE Statement”.

MySQL stores table privileges in the `mysql.tables_priv` system table.

Column Privileges

Column privileges apply to single columns in a given table. Each privilege to be granted at the column level must be followed by the column or columns, enclosed within parentheses.

```
GRANT SELECT (col1), INSERT (col1, col2) ON mydb.mytbl TO 'someuser'@'somehost';
```

The permissible *`priv_type`* values for a column (that is, when you use a *`column_list`* clause) are `INSERT`, `REFERENCES`, `SELECT`, and `UPDATE`.

MySQL stores column privileges in the `mysql.columns_priv` system table.

Stored Routine Privileges

The `ALTER ROUTINE`, `CREATE ROUTINE`, `EXECUTE`, and `GRANT OPTION` privileges apply to stored routines (procedures and functions). They can be granted at the global and database levels. Except for `CREATE ROUTINE`, these privileges can be granted at the routine level for individual routines.

```
GRANT CREATE ROUTINE ON mydb.* TO 'someuser'@'somehost';
```

```
GRANT EXECUTE ON PROCEDURE mydb.myproc TO 'someuser'@'somehost';
```

The permissible *`priv_type`* values at the routine level are `ALTER ROUTINE`, `EXECUTE`, and `GRANT OPTION`. `CREATE ROUTINE` is not a routine-level privilege because you must have the privilege at the global or database level to create a routine in the first place.

MySQL stores routine-level privileges in the `mysql.procs_priv` system table.

Proxy User Privileges

The `PROXY` privilege enables one user to be a proxy for another. The proxy user impersonates or takes the identity of the proxied user; that is, it assumes the privileges of the proxied user.

```
GRANT PROXY ON 'localuser'@'localhost' TO 'externaluser'@'somehost';
```

When `PROXY` is granted, it must be the only privilege named in the `GRANT` statement, the `REQUIRE` clause cannot be given, and the only permitted `WITH` option is `WITH GRANT OPTION`.

Proxying requires that the proxy user authenticate through a plugin that returns the name of the proxied user to the server when the proxy user connects, and that the proxy user have the `PROXY` privilege for the proxied user. For details and examples, see Section 6.2.12, “Proxy Users”.

MySQL stores proxy privileges in the `mysql.proxies_priv` system table.

Implicit Account Creation

If an account named in a `GRANT` statement does not exist, the action taken depends on the `NO_AUTO_CREATE_USER` SQL mode:

- If `NO_AUTO_CREATE_USER` is not enabled, `GRANT` creates the account. *This is very insecure* unless you specify a nonempty password using `IDENTIFIED BY`.
- If `NO_AUTO_CREATE_USER` is enabled, `GRANT` fails and does not create the account, unless you specify a nonempty password using `IDENTIFIED BY` or name an authentication plugin using `IDENTIFIED WITH`.

As of MySQL 5.6.12, if the account already exists, `IDENTIFIED WITH` is prohibited because it is intended only for use when creating new accounts.

Other Account Characteristics

MySQL can check X.509 certificate attributes in addition to the usual authentication that is based on the user name and credentials. For background information on the use of SSL with MySQL, see Section 6.3, “Using Encrypted Connections”.

The optional `REQUIRE` clause specifies SSL-related options for a MySQL account, using one or more *`tls_option`* values.

Order of `REQUIRE` options does not matter, but no option can be specified twice. The `AND` keyword is optional between `REQUIRE` options.

`GRANT` permits these *`tls_option`* values:

- `NONE`

  Indicates that the account has no SSL or X.509 requirements.

  ```
  GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
    REQUIRE NONE;
  ```

  Unencrypted connections are permitted if the user name and password are valid. Encrypted connections can be used, at the client's option, if the client has the proper certificate and key files.

That is, the client need not specify any SSL command options, in which case the connection is unencrypted. To use an encrypted connection, the client must specify either the `--ssl-ca` option, or all three of the `--ssl-ca`, `--ssl-key`, and `--ssl-cert` options.

`NONE` is the default if no SSL-related `REQUIRE` options are specified.

- `SSL`

  Tells the server to permit only encrypted connections for the account.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  REQUIRE SSL;
```

  To connect, the client must specify the `--ssl-ca` option to authenticate the server certificate, and may additionally specify the `--ssl-key` and `--ssl-cert` options. If neither the `--ssl-ca` option nor `--ssl-capath` option is specified, the client does not authenticate the server certificate.

- `X509`

  Requires that clients present a valid certificate, but the exact certificate, issuer, and subject do not matter. The only requirement is that it should be possible to verify its signature with one of the CA certificates. Use of X.509 certificates always implies encryption, so the `SSL` option is unnecessary in this case.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  REQUIRE X509;
```

  For accounts with `REQUIRE X509`, clients must specify the `--ssl-key` and `--ssl-cert` options to connect. (It is recommended but not required that `--ssl-ca` also be specified so that the public certificate provided by the server can be verified.) This is true for `ISSUER` and `SUBJECT` as well because those `REQUIRE` options imply the requirements of `X509`.

- `ISSUER 'issuer'`

  Requires that clients present a valid X.509 certificate issued by CA `'issuer'`. If a client presents a certificate that is valid but has a different issuer, the server rejects the connection. Use of X.509 certificates always implies encryption, so the `SSL` option is unnecessary in this case.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  REQUIRE ISSUER '/C=SE/ST=Stockholm/L=Stockholm/
    O=MySQL/CN=CA/emailAddress=ca@example.com';
```

  Because `ISSUER` implies the requirements of `X509`, clients must specify the `--ssl-key` and `--ssl-cert` options to connect. (It is recommended but not required that `--ssl-ca` also be specified so that the public certificate provided by the server can be verified.)

Note

If MySQL is linked against a version of OpenSSL older than 0.9.6h, use `Email` rather than `emailAddress` in the `'issuer'` value.

- `SUBJECT 'subject'`

  Requires that clients present a valid X.509 certificate containing the subject `subject`. If a client presents a certificate that is valid but has a different subject, the server rejects the connection. Use of X.509 certificates always implies encryption, so the `SSL` option is unnecessary in this case.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  REQUIRE SUBJECT '/C=SE/ST=Stockholm/L=Stockholm/
    O=MySQL demo client certificate/
    CN=client/emailAddress=client@example.com';
```

  MySQL does a simple string comparison of the `'subject'` value to the value in the certificate, so lettercase and component ordering must be given exactly as present in the certificate.

Note

Regarding `emailAddress`, see the note in the description of `REQUIRE ISSUER`.

Because `SUBJECT` implies the requirements of `X509`, clients must specify the `--ssl-key` and `--ssl-cert` options to connect. (It is recommended but not required that `--ssl-ca` also be specified so that the public certificate provided by the server can be verified.)

- `CIPHER 'cipher'`

  Requires a specific cipher method for encrypting connections. This option is needed to ensure that ciphers and key lengths of sufficient strength are used. SSL itself can be weak if old algorithms using short encryption keys are used.

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  REQUIRE CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

The `SUBJECT`, `ISSUER`, and `CIPHER` options can be combined in the `REQUIRE` clause like this:

```
GRANT ALL PRIVILEGES ON test.* TO 'root'@'localhost'
  REQUIRE SUBJECT '/C=SE/ST=Stockholm/L=Stockholm/
    O=MySQL demo client certificate/
    CN=client/emailAddress=client@example.com'
  AND ISSUER '/C=SE/ST=Stockholm/L=Stockholm/
    O=MySQL/CN=CA/emailAddress=ca@example.com'
  AND CIPHER 'EDH-RSA-DES-CBC3-SHA';
```

The optional `WITH` clause is used for these purposes:

- To enable a user to grant privileges to other users
- To specify resource limits for a user

The `WITH GRANT OPTION` clause gives the user the ability to give to other users any privileges the user has at the specified privilege level.

To grant the `GRANT OPTION` privilege to an account without otherwise changing its privileges, do this:

```
GRANT USAGE ON *.* TO 'someuser'@'somehost' WITH GRANT OPTION;
```

Be careful to whom you give the `GRANT OPTION` privilege because two users with different privileges may be able to combine privileges!

You cannot grant another user a privilege which you yourself do not have; the `GRANT OPTION` privilege enables you to assign only those privileges which you yourself possess.

Be aware that when you grant a user the `GRANT OPTION` privilege at a particular privilege level, any privileges the user possesses (or may be given in the future) at that level can also be granted by that user to other users. Suppose that you grant a user the `INSERT` privilege on a database. If you then grant the `SELECT` privilege on the database and specify `WITH GRANT OPTION`, that user can give to other users not only the `SELECT` privilege, but also `INSERT`. If you then grant the `UPDATE` privilege to the user on the database, the user can grant `INSERT`, `SELECT`, and `UPDATE`.

For a nonadministrative user, you should not grant the `ALTER` privilege globally or for the `mysql` system database. If you do that, the user can try to subvert the privilege system by renaming tables!

For additional information about security risks associated with particular privileges, see Section 6.2.2, “Privileges Provided by MySQL”.

It is possible to place limits on use of server resources by an account, as discussed in Section 6.2.13, “Setting Account Resource Limits”. To do so, use a `WITH` clause that specifies one or more *`resource_option`* values. Limits not specified retain their current values.

Order of `WITH` options does not matter, except that if a given resource limit is specified multiple times, the last instance takes precedence.

`GRANT` permits these *`resource_option`* values:

- `MAX_QUERIES_PER_HOUR` *`count`*, `MAX_UPDATES_PER_HOUR` *`count`*, `MAX_CONNECTIONS_PER_HOUR` *`count`*

  These options restrict how many queries, updates, and connections to the server are permitted to this account during any given one-hour period. (Queries for which results are served from the query cache do not count against the `MAX_QUERIES_PER_HOUR` limit.) If *`count`* is `0` (the default), this means that there is no limitation for the account.

- `MAX_USER_CONNECTIONS` *`count`*

  Restricts the maximum number of simultaneous connections to the server by the account. A nonzero *`count`* specifies the limit for the account explicitly. If *`count`* is `0` (the default), the server determines the number of simultaneous connections for the account from the global value of the `max_user_connections` system variable. If `max_user_connections` is also zero, there is no limit for the account.

To specify resource limits for an existing user without affecting existing privileges, use `GRANT USAGE` at the global level (`ON *.*`) and name the limits to be changed. For example:

```
GRANT USAGE ON *.* TO ...
  WITH MAX_QUERIES_PER_HOUR 500 MAX_UPDATES_PER_HOUR 100;
```

MySQL and Standard SQL Versions of GRANT

The biggest differences between the MySQL and standard SQL versions of `GRANT` are:

- MySQL associates privileges with the combination of a host name and user name and not with only a user name.
- Standard SQL does not have global or database-level privileges, nor does it support all the privilege types that MySQL supports.
- MySQL does not support the standard SQL `UNDER` privilege.
- Standard SQL privileges are structured in a hierarchical manner. If you remove a user, all privileges the user has been granted are revoked. This is also true in MySQL if you use `DROP USER`. See Section 13.7.1.3, “DROP USER Statement”.
- In standard SQL, when you drop a table, all privileges for the table are revoked. In standard SQL, when you revoke a privilege, all privileges that were granted based on that privilege are also revoked. In MySQL, privileges can be dropped with `DROP USER` or `REVOKE` statements.
- In MySQL, it is possible to have the `INSERT` privilege for only some of the columns in a table. In this case, you can still execute `INSERT` statements on the table, provided that you insert values only for those columns for which you have the `INSERT` privilege. The omitted columns are set to their implicit default values if strict SQL mode is not enabled. In strict mode, the statement is rejected if any of the omitted columns have no default value. (Standard SQL requires you to have the `INSERT` privilege on all columns.) For information about strict SQL mode and implicit default values, see Section 5.1.10, “Server SQL Modes”, and Section 11.5, “Data Type Default Values”.

13.7.1.5 RENAME USER Statement

```
RENAME USER old_user TO new_user
    [, old_user TO new_user] ...
```

The `RENAME USER` statement renames existing MySQL accounts. An error occurs for old accounts that do not exist or new accounts that already exist.

To use RENAME USER, you must have the global CREATE USER privilege, or the UPDATE privilege for the mysql system database. When the read_only system variable is enabled, RENAME USER additionally requires the SUPER privilege.

Each account name uses the format described in Section 6.2.4, “Specifying Account Names”. For example:

```
RENAME USER 'jeffrey'@'localhost' TO 'jeff'@'127.0.0.1';
```

The host name part of the account name, if omitted, defaults to '%'.

RENAME USER causes the privileges held by the old user to be those held by the new user. However, RENAME USER does not automatically drop or invalidate databases or objects within them that the old user created. This includes stored programs or views for which the DEFINER attribute names the old user. Attempts to access such objects may produce an error if they execute in definer security context. (For information about security context, see Section 20.6, “Stored Object Access Control”.)

The privilege changes take effect as indicated in Section 6.2.8, “When Privilege Changes Take Effect”.

13.7.1.6 REVOKE Statement

```
REVOKE
    priv_type [(column_list)]
      [, priv_type [(column_list)]] ...
    ON [object_type] priv_level
    FROM user [, user] ...

REVOKE ALL [PRIVILEGES], GRANT OPTION
    FROM user [, user] ...

REVOKE PROXY ON user
    FROM user [, user] ...
```

The REVOKE statement enables system administrators to revoke privileges from MySQL accounts.

For details on the levels at which privileges exist, the permissible *priv_type*, *priv_level*, and *object_type* values, and the syntax for specifying users and passwords, see Section 13.7.1.4, “GRANT Statement”.

When the read_only system variable is enabled, REVOKE requires the SUPER privilege in addition to any other required privileges described in the following discussion.

Each account name uses the format described in Section 6.2.4, “Specifying Account Names”. For example:

```
REVOKE INSERT ON *.* FROM 'jeffrey'@'localhost';
```

The host name part of the account name, if omitted, defaults to '%'.

To use the first REVOKE syntax, you must have the GRANT OPTION privilege, and you must have the privileges that you are revoking.

To revoke all privileges, use the second syntax, which drops all global, database, table, column, and routine privileges for the named user or users:

```
REVOKE ALL PRIVILEGES, GRANT OPTION FROM user [, user] ...
```

To use this REVOKE syntax, you must have the global CREATE USER privilege, or the UPDATE privilege for the mysql system database.

User accounts from which privileges are to be revoked must exist, but the privileges to be revoked need not be currently granted to them.

`REVOKE` removes privileges, but does not remove rows from the `mysql.user` system table. To remove a user account entirely, use `DROP USER`. See Section 13.7.1.3, “DROP USER Statement”.

If the grant tables hold privilege rows that contain mixed-case database or table names and the `lower_case_table_names` system variable is set to a nonzero value, `REVOKE` cannot be used to revoke these privileges. It is necessary to manipulate the grant tables directly. (`GRANT` does not create such rows when `lower_case_table_names` is set, but such rows might have been created prior to setting the variable.)

When successfully executed from the `mysql` program, `REVOKE` responds with `Query OK, 0 rows affected`. To determine what privileges remain after the operation, use `SHOW GRANTS`. See Section 13.7.5.22, “SHOW GRANTS Statement”.

13.7.1.7 SET PASSWORD Statement

```
SET PASSWORD [FOR user] = password_option

password_option: {
    PASSWORD('auth_string')
  | OLD_PASSWORD('auth_string')
  | 'hash_string'
}
```

The `SET PASSWORD` statement assigns a password to a MySQL user account, specified as either a cleartext (unencrypted) or encrypted value:

- *`'auth_string'`* represents a cleartext password.
- *`'hash_string'`* represents an encrypted password.

Important

Under some circumstances, `SET PASSWORD` may be recorded in server logs or on the client side in a history file such as `~/.mysql_history`, which means that cleartext passwords may be read by anyone having read access to that information. For information about the conditions under which this occurs for the server logs and how to control it, see Section 6.1.2.3, “Passwords and Logging”. For similar information about client-side logging, see Section 4.5.1.3, “mysql Client Logging”.

`SET PASSWORD` can be used with or without a `FOR` clause that explicitly names a user account:

- With a `FOR user` clause, the statement sets the password for the named account, which must exist:

  ```
  SET PASSWORD FOR 'jeffrey'@'localhost' = password_option;
  ```

- With no `FOR user` clause, the statement sets the password for the current user:

  ```
  SET PASSWORD = password_option;
  ```

  Any client who connects to the server using a nonanonymous account can change the password for that account. (In particular, you can change your own password.) To see which account the server authenticated you as, invoke the `CURRENT_USER()` function:

  ```
  SELECT CURRENT_USER();
  ```

If a `FOR user` clause is given, the account name uses the format described in Section 6.2.4, “Specifying Account Names”. For example:

```
SET PASSWORD FOR 'bob'@'%.example.org' = PASSWORD('auth_string');
```

The host name part of the account name, if omitted, defaults to `'%'`.

Setting the password for a named account (with a `FOR` clause) requires the `UPDATE` privilege for the `mysql` system database. Setting the password for yourself (for a nonanonymous account with no `FOR` clause) requires no special privileges. When the `read_only` system variable is enabled, `SET PASSWORD` requires the `SUPER` privilege in addition to any other required privileges.

The password can be specified in these ways:

- Use the `PASSWORD()` function

  The `PASSWORD()` argument is the cleartext (unencrypted) password. `PASSWORD()` hashes the password and returns the encrypted password string for storage in the account row in the `mysql.user` system table.

  The `PASSWORD()` function hashes the password using the hashing method determined by the value of the `old_passwords` system variable value. If `SET PASSWORD` rejects the hashed password value returned by `PASSWORD()` as not being in the correct format, it may be necessary to change `old_passwords` to change the hashing method. For example, if the account uses the `mysql_native_password` plugin, the `old_passwords` value must be 0:

  ```
  SET old_passwords = 0;
  SET PASSWORD FOR 'jeffrey'@'localhost' = PASSWORD('password');
  ```

  If the `old_passwords` value differs from that required by the authentication plugin, the hashed password value returned by `PASSWORD()` is not acceptable for that plugin, and attempts to set the password produce an error. For example:

  ```
  mysql> SET old_passwords = 1;
  mysql> SET PASSWORD FOR 'jeffrey'@'localhost' = PASSWORD('password');
  ERROR 1372 (HY000): Password hash should be a 41-digit hexadecimal number
  ```

  Permitted `old_passwords` values are described later in this section.

- Use the `OLD_PASSWORD()` function:

  The `'auth_string'` function argument is the cleartext (unencrypted) password. `OLD_PASSWORD()` hashes the password using pre-4.1 hashing and returns the encrypted password string for storage in the account row in the `mysql.user` system table. This hashing method is appropriate only for accounts that use the `mysql_old_password` authentication plugin.

- Use an already encrypted password string

  The password is specified as a string literal. It must represent the already encrypted password value, in the hash format required by the authentication method used for the account.

The following table shows, for each password hashing method, the permitted value of `old_passwords` and which authentication plugins use the hashing method.

Password Hashing Method	old_passwords Value	Associated Authentication Plugin
MySQL 4.1 native hashing	0	`mysql_native_password`
Pre-4.1 (“old”) hashing	1	`mysql_old_password`
SHA-256 hashing	2	`sha256_password`

Caution

If you are connecting to a MySQL 4.1 or later server using a pre-4.1 client program, do not change your password without first reading Section 6.1.2.4, “Password Hashing in MySQL”. The default password hashing format changed in MySQL 4.1, and if you change your password, it might be stored using a

hashing format that pre-4.1 clients cannot generate, thus preventing you from connecting to the server afterward.

For additional information about setting passwords and authentication plugins, see Section 6.2.9, “Assigning Account Passwords”, and Section 6.2.11, “Pluggable Authentication”.

13.7.2 Table Maintenance Statements

13.7.2.1 ANALYZE TABLE Statement

```
ANALYZE [NO_WRITE_TO_BINLOG | LOCAL]
    TABLE tbl_name [, tbl_name] ...
```

`ANALYZE TABLE` performs a key distribution analysis and stores the distribution for the named table or tables. For `MyISAM` tables, this statement is equivalent to using `myisamchk --analyze`.

This statement requires `SELECT` and `INSERT` privileges for the table.

`ANALYZE TABLE` works with `InnoDB`, `NDB`, and `MyISAM` tables. It does not work with views.

`ANALYZE TABLE` is supported for partitioned tables, and you can use `ALTER TABLE ... ANALYZE PARTITION` to analyze one or more partitions; for more information, see Section 13.1.7, “ALTER TABLE Statement”, and Section 19.3.4, “Maintenance of Partitions”.

During the analysis, the table is locked with a read lock for `InnoDB` and `MyISAM`.

`ANALYZE TABLE` removes the table from the table definition cache, which requires a flush lock. If there are long running statements or transactions still using the table, subsequent statements and transactions must wait for those operations to finish before the flush lock is released. Because `ANALYZE TABLE` itself typically finishes quickly, it may not be apparent that delayed transactions or statements involving the same table are due to the remaining flush lock.

By default, the server writes `ANALYZE TABLE` statements to the binary log so that they replicate to replicas. To suppress logging, specify the optional `NO_WRITE_TO_BINLOG` keyword or its alias `LOCAL`.

- ANALYZE TABLE Output
- Key Distribution Analysis
- Other Considerations

ANALYZE TABLE Output

`ANALYZE TABLE` returns a result set with the columns shown in the following table.

Column	Value
`Table`	The table name
`Op`	Always `analyze`
`Msg_type`	`status`, `error`, `info`, `note`, or `warning`
`Msg_text`	An informational message

Key Distribution Analysis

If the table has not changed since the last key distribution analysis, the table is not analyzed again.

MySQL uses the stored key distribution to decide the table join order for joins on something other than a constant. In addition, key distributions can be used when deciding which indexes to use for a specific table within a query.

To check the stored key distribution cardinality, use the `SHOW INDEX` statement or the `INFORMATION_SCHEMA STATISTICS` table. See Section 13.7.5.23, “SHOW INDEX Statement”, and Section 21.22, “The INFORMATION_SCHEMA STATISTICS Table”.

For `InnoDB` tables, `ANALYZE TABLE` determines index cardinality by performing random dives on each of the index trees and updating index cardinality estimates accordingly. Because these are only estimates, repeated runs of `ANALYZE TABLE` could produce different numbers. This makes `ANALYZE TABLE` fast on `InnoDB` tables but not 100% accurate because it does not take all rows into account.

You can make the statistics collected by `ANALYZE TABLE` more precise and more stable by enabling `innodb_stats_persistent`, as explained in Section 14.8.11.1, “Configuring Persistent Optimizer Statistics Parameters”. When `innodb_stats_persistent` is enabled, it is important to run `ANALYZE TABLE` after major changes to index column data, as statistics are not recalculated periodically (such as after a server restart).

If `innodb_stats_persistent` is enabled, you can change the number of random dives by modifying the `innodb_stats_persistent_sample_pages` system variable. If `innodb_stats_persistent` is disabled, modify `innodb_stats_transient_sample_pages` instead.

For more information about key distribution analysis in `InnoDB`, see Section 14.8.11.1, “Configuring Persistent Optimizer Statistics Parameters”, and Section 14.8.11.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”.

MySQL uses index cardinality estimates in join optimization. If a join is not optimized in the right way, try running `ANALYZE TABLE`. In the few cases that `ANALYZE TABLE` does not produce values good enough for your particular tables, you can use `FORCE INDEX` with your queries to force the use of a particular index, or set the `max_seeks_for_key` system variable to ensure that MySQL prefers index lookups over table scans. See Section B.3.5, “Optimizer-Related Issues”.

Other Considerations

`ANALYZE TABLE` clears table statistics from the `INFORMATION_SCHEMA.INNODB_SYS_TABLESTATS` table and sets the `STATS_INITIALIZED` column to `Uninitialized`. Statistics are collected again the next time the table is accessed.

In MySQL 5.6.11 only, `gtid_next` must be set to `AUTOMATIC` before issuing this statement. (Bug #16062608, Bug #16715809, Bug #69045)

13.7.2.2 CHECK TABLE Statement

```
CHECK TABLE tbl_name [, tbl_name] ... [option] ...

option: {
    FOR UPGRADE
  | QUICK
  | FAST
  | MEDIUM
  | EXTENDED
  | CHANGED
}
```

`CHECK TABLE` checks a table or tables for errors. For `MyISAM` tables, the key statistics are updated as well. `CHECK TABLE` can also check views for problems, such as tables that are referenced in the view definition that no longer exist.

To check a table, you must have some privilege for it.

`CHECK TABLE` works for `InnoDB`, `MyISAM`, `ARCHIVE`, and `CSV` tables.

Before running `CHECK TABLE` on `InnoDB` tables, see CHECK TABLE Usage Notes for InnoDB Tables.

`CHECK TABLE` is supported for partitioned tables, and you can use `ALTER TABLE ... CHECK PARTITION` to check one or more partitions; for more information, see Section 13.1.7, “ALTER TABLE Statement”, and Section 19.3.4, “Maintenance of Partitions”.

In MySQL 5.6.11 only, `gtid_next` must be set to `AUTOMATIC` before issuing this statement. (Bug #16062608, Bug #16715809, Bug #69045)

- CHECK TABLE Output
- Checking Version Compatibility
- Checking Data Consistency
- CHECK TABLE Usage Notes for InnoDB Tables

CHECK TABLE Output

`CHECK TABLE` returns a result set with the columns shown in the following table.

Column	Value
`Table`	The table name
`Op`	Always `check`
`Msg_type`	`status`, `error`, `info`, `note`, or `warning`
`Msg_text`	An informational message

The statement might produce many rows of information for each checked table. The last row has a `Msg_type` value of `status` and the `Msg_text` normally should be `OK`. For a `MyISAM` table, if you don't get `OK` or `Table is already up to date`, you should normally run a repair of the table. See Section 7.6, “MyISAM Table Maintenance and Crash Recovery”. `Table is already up to date` means that the storage engine for the table indicated that there was no need to check the table.

Checking Version Compatibility

The `FOR UPGRADE` option checks whether the named tables are compatible with the current version of MySQL. With `FOR UPGRADE`, the server checks each table to determine whether there have been any incompatible changes in any of the table's data types or indexes since the table was created. If not, the check succeeds. Otherwise, if there is a possible incompatibility, the server runs a full check on the table (which might take some time). If the full check succeeds, the server marks the table's `.frm` file with the current MySQL version number. Marking the `.frm` file ensures that further checks for the table with the same version of the server are fast.

Incompatibilities might occur because the storage format for a data type has changed or because its sort order has changed. Our aim is to avoid these changes, but occasionally they are necessary to correct problems that would be worse than an incompatibility between releases.

`FOR UPGRADE` discovers these incompatibilities:

- The indexing order for end-space in `TEXT` columns for `InnoDB` and `MyISAM` tables changed between MySQL 4.1 and 5.0.
- The storage method of the new `DECIMAL` data type changed between MySQL 5.0.3 and 5.0.5.
- If your table was created by a different version of the MySQL server than the one you are currently running, `FOR UPGRADE` indicates that the table has an `.frm` file with an incompatible version. In this case, the result set returned by `CHECK TABLE` contains a line with a `Msg_type` value of `error` and a `Msg_text` value of `Table upgrade required. Please do "REPAIR TABLE` `` `tbl_name` `` `" to fix it!`

- Changes are sometimes made to character sets or collations that require table indexes to be rebuilt. For details about such changes, see Section 2.11.3, “Changes in MySQL 5.6”. For information about rebuilding tables, see Section 2.11.10, “Rebuilding or Repairing Tables or Indexes”.
- The `YEAR(2)` data type is deprecated as of MySQL 5.6.6. For tables containing `YEAR(2)` columns, `CHECK TABLE` recommends `REPAIR TABLE`, which converts 2-digit `YEAR(2)` columns to 4-digit `YEAR` columns.
- As of MySQL 5.6.4, MySQL permits fractional seconds for `TIME`, `DATETIME`, and `TIMESTAMP` column values. As a result, encoding and storage requirements for these temporal column types differ in tables created in MySQL 5.6.4 and later. This incompatible change is described in Section 2.11.3, “Changes in MySQL 5.6”. When upgrading to MySQL 5.6.4 or later, be aware that `CHECK TABLE ... FOR UPGRADE` does not report temporal columns that use the pre-MySQL 5.6.4 format (Bug #73008, Bug #18985579). In MySQL 5.6.24, two new system variables, `avoid_temporal_upgrade` and `show_old_temporals`, were added to provide control over temporal column upgrades (Bug #72997, Bug #18985760).

Checking Data Consistency

The following table shows the other check options that can be given. These options are passed to the storage engine, which may use or ignore them.

Type	Meaning
`QUICK`	Do not scan the rows to check for incorrect links. Applies to `InnoDB` and `MyISAM` tables and views.
`FAST`	Check only tables that have not been closed properly. Applies only to `MyISAM` tables and views; ignored for `InnoDB`.
`CHANGED`	Check only tables that have been changed since the last check or that have not been closed properly. Applies only to `MyISAM` tables and views; ignored for `InnoDB`.
`MEDIUM`	Scan rows to verify that deleted links are valid. This also calculates a key checksum for the rows and verifies this with a calculated checksum for the keys. Applies only to `MyISAM` tables and views; ignored for `InnoDB`.
`EXTENDED`	Do a full key lookup for all keys for each row. This ensures that the table is 100% consistent, but takes a long time. Applies only to `MyISAM` tables and views; ignored for `InnoDB`.

If none of the options `QUICK`, `MEDIUM`, or `EXTENDED` are specified, the default check type for dynamic-format `MyISAM` tables is `MEDIUM`. This has the same result as running `myisamchk --medium-check` *`tbl_name`* on the table. The default check type also is `MEDIUM` for static-format `MyISAM` tables, unless `CHANGED` or `FAST` is specified. In that case, the default is `QUICK`. The row scan is skipped for `CHANGED` and `FAST` because the rows are very seldom corrupted.

You can combine check options, as in the following example that does a quick check on the table to determine whether it was closed properly:

```
CHECK TABLE test_table FAST QUICK;
```

Note

If `CHECK TABLE` finds no problems with a table that is marked as “corrupted” or “not closed properly”, `CHECK TABLE` may remove the mark.

If a table is corrupted, the problem is most likely in the indexes and not in the data part. All of the preceding check types check the indexes thoroughly and should thus find most errors.

To check a table that you assume is okay, use no check options or the `QUICK` option. The latter should be used when you are in a hurry and can take the very small risk that `QUICK` does not find an error in

the data file. (In most cases, under normal usage, MySQL should find any error in the data file. If this happens, the table is marked as “corrupted” and cannot be used until it is repaired.)

`FAST` and `CHANGED` are mostly intended to be used from a script (for example, to be executed from `cron`) to check tables periodically. In most cases, `FAST` is to be preferred over `CHANGED`. (The only case when it is not preferred is when you suspect that you have found a bug in the `MyISAM` code.)

`EXTENDED` is to be used only after you have run a normal check but still get errors from a table when MySQL tries to update a row or find a row by key. This is very unlikely if a normal check has succeeded.

Use of `CHECK TABLE ... EXTENDED` might influence execution plans generated by the query optimizer.

Some problems reported by `CHECK TABLE` cannot be corrected automatically:

- `Found row where the auto_increment column has the value 0`.

  This means that you have a row in the table where the `AUTO_INCREMENT` index column contains the value 0. (It is possible to create a row where the `AUTO_INCREMENT` column is 0 by explicitly setting the column to 0 with an `UPDATE` statement.)

  This is not an error in itself, but could cause trouble if you decide to dump the table and restore it or do an `ALTER TABLE` on the table. In this case, the `AUTO_INCREMENT` column changes value according to the rules of `AUTO_INCREMENT` columns, which could cause problems such as a duplicate-key error.

  To get rid of the warning, execute an `UPDATE` statement to set the column to some value other than 0.

CHECK TABLE Usage Notes for InnoDB Tables

The following notes apply to `InnoDB` tables:

- If `CHECK TABLE` encounters a corrupt page, the server exits to prevent error propagation (Bug #10132). If the corruption occurs in a secondary index but table data is readable, running `CHECK TABLE` can still cause a server exit.
- If `CHECK TABLE` encounters a corrupted `DB_TRX_ID` or `DB_ROLL_PTR` field in a clustered index, `CHECK TABLE` can cause `InnoDB` to access an invalid undo log record, resulting in an MVCC-related server exit.
- If `CHECK TABLE` encounters errors in `InnoDB` tables or indexes, it reports an error, and usually marks the index and sometimes marks the table as corrupted, preventing further use of the index or table. Such errors include an incorrect number of entries in a secondary index or incorrect links.
- If `CHECK TABLE` finds an incorrect number of entries in a secondary index, it reports an error but does not cause a server exit or prevent access to the file.
- `CHECK TABLE` surveys the index page structure, then surveys each key entry. It does not validate the key pointer to a clustered record or follow the path for `BLOB` pointers.
- When an `InnoDB` table is stored in its own `.ibd` file, the first 3 pages of the `.ibd` file contain header information rather than table or index data. The `CHECK TABLE` statement does not detect inconsistencies that affect only the header data. To verify the entire contents of an `InnoDB .ibd` file, use the `innochecksum` command.
- When running `CHECK TABLE` on large `InnoDB` tables, other threads may be blocked during `CHECK TABLE` execution. To avoid timeouts, the semaphore wait threshold (600 seconds) is extended by 2 hours (7200 seconds) for `CHECK TABLE` operations. If `InnoDB` detects semaphore waits of 240 seconds or more, it starts printing `InnoDB` monitor output to the error log. If a lock request extends

beyond the semaphore wait threshold, InnoDB aborts the process. To avoid the possibility of a semaphore wait timeout entirely, run CHECK TABLE QUICK instead of CHECK TABLE.

13.7.2.3 CHECKSUM TABLE Statement

```
CHECKSUM TABLE tbl_name [, tbl_name] ... [QUICK | EXTENDED]
```

CHECKSUM TABLE reports a checksum for the contents of a table. You can use this statement to verify that the contents are the same before and after a backup, rollback, or other operation that is intended to put the data back to a known state.

This statement requires the SELECT privilege for the table.

This statement is not supported for views. If you run CHECKSUM TABLE against a view, the Checksum value is always NULL, and a warning is returned.

For a nonexistent table, CHECKSUM TABLE returns NULL and generates a warning.

During the checksum operation, the table is locked with a read lock for InnoDB and MyISAM.

Performance Considerations

By default, the entire table is read row by row and the checksum is calculated. For large tables, this could take a long time, thus you would only perform this operation occasionally. This row-by-row calculation is what you get with the EXTENDED clause, with InnoDB and all other storage engines other than MyISAM, and with MyISAM tables not created with the CHECKSUM=1 clause.

For MyISAM tables created with the CHECKSUM=1 clause, CHECKSUM TABLE or CHECKSUM TABLE ... QUICK returns the "live" table checksum that can be returned very fast. If the table does not meet all these conditions, the QUICK method returns NULL. The QUICK method is not supported with InnoDB tables. See Section 13.1.17, "CREATE TABLE Statement" for the syntax of the CHECKSUM clause.

The checksum value depends on the table row format. If the row format changes, the checksum also changes. For example, the change in storage format for temporal types such as TIME, DATETIME, and TIMESTAMP just described mean that, if a 5.5 table is upgraded to MySQL 5.6, the checksum value may change.

Important

If the checksums for two tables are different, then it is almost certain that the tables are different in some way. However, because the hashing function used by CHECKSUM TABLE is not guaranteed to be collision-free, there is a slight chance that two tables which are not identical can produce the same checksum.

13.7.2.4 OPTIMIZE TABLE Statement

```
OPTIMIZE [NO_WRITE_TO_BINLOG | LOCAL]
    TABLE tbl_name [, tbl_name] ...
```

OPTIMIZE TABLE reorganizes the physical storage of table data and associated index data, to reduce storage space and improve I/O efficiency when accessing the table. The exact changes made to each table depend on the storage engine used by that table.

Use OPTIMIZE TABLE in these cases, depending on the type of table:

- After doing substantial insert, update, or delete operations on an InnoDB table that has its own .ibd file because it was created with the innodb_file_per_table option enabled. The table and indexes are reorganized, and disk space can be reclaimed for use by the operating system.
- After doing substantial insert, update, or delete operations on columns that are part of a FULLTEXT index in an InnoDB table. Set the configuration option

innodb_optimize_fulltext_only=1 first. To keep the index maintenance period to a reasonable time, set the innodb_ft_num_word_optimize option to specify how many words to update in the search index, and run a sequence of OPTIMIZE TABLE statements until the search index is fully updated.

- After deleting a large part of a MyISAM or ARCHIVE table, or making many changes to a MyISAM or ARCHIVE table with variable-length rows (tables that have VARCHAR, VARBINARY, BLOB, or TEXT columns). Deleted rows are maintained in a linked list and subsequent INSERT operations reuse old row positions. You can use OPTIMIZE TABLE to reclaim the unused space and to defragment the data file. After extensive changes to a table, this statement may also improve performance of statements that use the table, sometimes significantly.

This statement requires SELECT and INSERT privileges for the table.

OPTIMIZE TABLE works for InnoDB, MyISAM, and ARCHIVE tables. OPTIMIZE TABLE is also supported for dynamic columns of in-memory NDB tables. It does not work for fixed-width columns of in-memory tables, nor does it work for Disk Data tables. The performance of OPTIMIZE on NDB Cluster tables can be tuned using --ndb-optimization-delay, which controls the length of time to wait between processing batches of rows by OPTIMIZE TABLE. For more information, see Section 18.1.7.11, “Previous NDB Cluster Issues Resolved in NDB Cluster 7.3”.

For NDB Cluster tables, OPTIMIZE TABLE can be interrupted by (for example) killing the SQL thread performing the OPTIMIZE operation.

By default, OPTIMIZE TABLE does *not* work for tables created using any other storage engine and returns a result indicating this lack of support. You can make OPTIMIZE TABLE work for other storage engines by starting mysqld with the --skip-new option. In this case, OPTIMIZE TABLE is just mapped to ALTER TABLE.

This statement does not work with views.

OPTIMIZE TABLE is supported for partitioned tables. For information about using this statement with partitioned tables and table partitions, see Section 19.3.4, “Maintenance of Partitions”.

By default, the server writes OPTIMIZE TABLE statements to the binary log so that they replicate to replicas. To suppress logging, specify the optional NO_WRITE_TO_BINLOG keyword or its alias LOCAL.

In MySQL 5.6.11 only, gtid_next must be set to AUTOMATIC before issuing this statement. (Bug #16062608, Bug #16715809, Bug #69045)

- OPTIMIZE TABLE Output
- InnoDB Details
- MyISAM Details
- Other Considerations

OPTIMIZE TABLE Output

OPTIMIZE TABLE returns a result set with the columns shown in the following table.

Column	Value
Table	The table name
Op	Always optimize
Msg_type	status, error, info, note, or warning
Msg_text	An informational message

`OPTIMIZE TABLE` table catches and throws any errors that occur while copying table statistics from the old file to the newly created file. For example. if the user ID of the owner of the `.frm`, `.MYD`, or `.MYI` file is different from the user ID of the `mysqld` process, `OPTIMIZE TABLE` generates a "cannot change ownership of the file" error unless `mysqld` is started by the `root` user.

InnoDB Details

For `InnoDB` tables, `OPTIMIZE TABLE` is mapped to `ALTER TABLE ... FORCE`, which rebuilds the table to update index statistics and free unused space in the clustered index. This is displayed in the output of `OPTIMIZE TABLE` when you run it on an `InnoDB` table, as shown here:

```
mysql> OPTIMIZE TABLE foo;
+----------+----------+----------+-------------------------------------------------------------------+
| Table    | Op       | Msg_type | Msg_text                                                          |
+----------+----------+----------+-------------------------------------------------------------------+
| test.foo | optimize | note     | Table does not support optimize, doing recreate + analyze instead |
| test.foo | optimize | status   | OK                                                                |
+----------+----------+----------+-------------------------------------------------------------------+
```

Prior to Mysql 5.6.17, `OPTIMIZE TABLE` does not use online DDL. Consequently, concurrent DML (`INSERT`, `UPDATE`, `DELETE`) is not permitted on a table while `OPTIMIZE TABLE` is running, and secondary indexes are not created as efficiently.

As of MySQL 5.6.17, `OPTIMIZE TABLE` uses online DDL for regular and partitioned `InnoDB` tables, which reduces downtime for concurrent DML operations. The table rebuild triggered by `OPTIMIZE TABLE` is completed in place. An exclusive table lock is only taken briefly during the prepare phase and the commit phase of the operation. During the prepare phase, metadata is updated and an intermediate table is created. During the commit phase, table metadata changes are committed.

`OPTIMIZE TABLE` rebuilds the table using the table copy method under the following conditions:

- When the `old_alter_table` system variable is enabled.
- When the server is started with the `--skip-new` option.

`OPTIMIZE TABLE` using online DDL is not supported for `InnoDB` tables that contain `FULLTEXT` indexes. The table copy method is used instead.

`InnoDB` stores data using a page-allocation method and does not suffer from fragmentation in the same way that legacy storage engines (such as `MyISAM`) do. When considering whether or not to run optimize, consider the workload of transactions that your server is expected to process:

- Some level of fragmentation is expected. `InnoDB` only fills pages 93% full, to leave room for updates without having to split pages.
- Delete operations might leave gaps that leave pages less filled than desired, which could make it worthwhile to optimize the table.
- Updates to rows usually rewrite the data within the same page, depending on the data type and row format, when sufficient space is available. See Section 14.9.5, “How Compression Works for InnoDB Tables” and Section 14.11, “InnoDB Row Formats”.
- High-concurrency workloads might leave gaps in indexes over time, as `InnoDB` retains multiple versions of the same data due through its MVCC mechanism. See Section 14.3, “InnoDB Multi-Versioning”.

MyISAM Details

For `MyISAM` tables, `OPTIMIZE TABLE` works as follows:

1. If the table has deleted or split rows, repair the table.

2. If the index pages are not sorted, sort them.

3. If the table's statistics are not up to date (and the repair could not be accomplished by sorting the index), update them.

Other Considerations

For `InnoDB` tables prior to 5.6.17 and other table types, MySQL locks the table during the time `OPTIMIZE TABLE` is running. As of MySQL 5.6.17, `OPTIMIZE TABLE` is performed online for regular and partitioned `InnoDB` tables.

`OPTIMIZE TABLE` does not sort R-tree indexes, such as spatial indexes on `POINT` columns. (Bug #23578)

13.7.2.5 REPAIR TABLE Statement

```
REPAIR [NO_WRITE_TO_BINLOG | LOCAL]
    TABLE tbl_name [, tbl_name] ...
    [QUICK] [EXTENDED] [USE_FRM]
```

`REPAIR TABLE` repairs a possibly corrupted table, for certain storage engines only.

This statement requires `SELECT` and `INSERT` privileges for the table.

Although normally you should never have to run `REPAIR TABLE`, if disaster strikes, this statement is very likely to get back all your data from a `MyISAM` table. If your tables become corrupted often, try to find the reason for it, to eliminate the need to use `REPAIR TABLE`. See Section B.3.3.3, “What to Do If MySQL Keeps Crashing”, and Section 15.2.4, “MyISAM Table Problems”.

`REPAIR TABLE` checks the table to see whether an upgrade is required. If so, it performs the upgrade, following the same rules as `CHECK TABLE ... FOR UPGRADE`. See Section 13.7.2.2, “CHECK TABLE Statement”, for more information.

Important

- Make a backup of a table before performing a table repair operation; under some circumstances the operation might cause data loss. Possible causes include but are not limited to file system errors. See Chapter 7, *Backup and Recovery*.
- If the server exits during a `REPAIR TABLE` operation, it is essential after restarting it that you immediately execute another `REPAIR TABLE` statement for the table before performing any other operations on it. In the worst case, you might have a new clean index file without information about the data file, and then the next operation you perform could overwrite the data file. This is an unlikely but possible scenario that underscores the value of making a backup first.
- In the event that a table on the source becomes corrupted and you run `REPAIR TABLE` on it, any resulting changes to the original table are *not* propagated to replicas.
- In MySQL 5.6.11 only, `gtid_next` must be set to `AUTOMATIC` before issuing this statement. (Bug #16062608, Bug #16715809, Bug #69045)

- REPAIR TABLE Storage Engine and Partitioning Support
- REPAIR TABLE Options
- REPAIR TABLE Output

- Table Repair Considerations

REPAIR TABLE Storage Engine and Partitioning Support

`REPAIR TABLE` works for `MyISAM`, `ARCHIVE`, and `CSV` tables. For `MyISAM` tables, it has the same effect as `myisamchk --recover tbl_name` by default. This statement does not work with views.

`REPAIR TABLE` is supported for partitioned tables. However, the `USE_FRM` option cannot be used with this statement on a partitioned table.

You can use `ALTER TABLE ... REPAIR PARTITION` to repair one or more partitions; for more information, see Section 13.1.7, “ALTER TABLE Statement”, and Section 19.3.4, “Maintenance of Partitions”.

REPAIR TABLE Options

- `NO_WRITE_TO_BINLOG` or `LOCAL`

  By default, the server writes `REPAIR TABLE` statements to the binary log so that they replicate to replicas. To suppress logging, specify the optional `NO_WRITE_TO_BINLOG` keyword or its alias `LOCAL`.

- `QUICK`

  If you use the `QUICK` option, `REPAIR TABLE` tries to repair only the index file, and not the data file. This type of repair is like that done by `myisamchk --recover --quick`.

- `EXTENDED`

  If you use the `EXTENDED` option, MySQL creates the index row by row instead of creating one index at a time with sorting. This type of repair is like that done by `myisamchk --safe-recover`.

- `USE_FRM`

  The `USE_FRM` option is available for use if the `.MYI` index file is missing or if its header is corrupted. This option tells MySQL not to trust the information in the `.MYI` file header and to re-create it using information from the `.frm` file. This kind of repair cannot be done with `myisamchk`.

Caution

Use the `USE_FRM` option *only* if you cannot use regular `REPAIR` modes. Telling the server to ignore the `.MYI` file makes important table metadata stored in the `.MYI` unavailable to the repair process, which can have deleterious consequences:

- The current `AUTO_INCREMENT` value is lost.
- The link to deleted records in the table is lost, which means that free space for deleted records remains unoccupied thereafter.
- The `.MYI` header indicates whether the table is compressed. If the server ignores this information, it cannot tell that a table is compressed and repair can cause change or loss of table contents. This means that `USE_FRM` should not be used with compressed tables. That should not be necessary, anyway: Compressed tables are read only, so they should not become corrupt.

If you use `USE_FRM` for a table that was created by a different version of the MySQL server than the one you are currently running, `REPAIR TABLE` does not attempt to repair the table. In this case, the result set returned by `REPAIR TABLE` contains a line with a `Msg_type` value of `error` and a `Msg_text` value of `Failed repairing incompatible .FRM file`.

> If `USE_FRM` is used, `REPAIR TABLE` does not check the table to see whether an upgrade is required.

REPAIR TABLE Output

`REPAIR TABLE` returns a result set with the columns shown in the following table.

Column	Value
`Table`	The table name
`Op`	Always `repair`
`Msg_type`	`status`, `error`, `info`, `note`, or `warning`
`Msg_text`	An informational message

The `REPAIR TABLE` statement might produce many rows of information for each repaired table. The last row has a `Msg_type` value of `status` and `Msg_test` normally should be `OK`. For a `MyISAM` table, if you do not get `OK`, you should try repairing it with `myisamchk --safe-recover`. (`REPAIR TABLE` does not implement all the options of `myisamchk`. With `myisamchk --safe-recover`, you can also use options that `REPAIR TABLE` does not support, such as `--max-record-length`.)

`REPAIR TABLE` table catches and throws any errors that occur while copying table statistics from the old corrupted file to the newly created file. For example. if the user ID of the owner of the `.frm`, `.MYD`, or `.MYI` file is different from the user ID of the `mysqld` process, `REPAIR TABLE` generates a "cannot change ownership of the file" error unless `mysqld` is started by the `root` user.

Table Repair Considerations

You may be able to increase `REPAIR TABLE` performance by setting certain system variables. See Section 8.6.3, “Optimizing REPAIR TABLE Statements”.

13.7.3 Plugin and User-Defined Function Statements

13.7.3.1 CREATE FUNCTION Statement for User-Defined Functions

```
CREATE [AGGREGATE] FUNCTION function_name
    RETURNS {STRING|INTEGER|REAL|DECIMAL}
    SONAME shared_library_name
```

This statement loads the user-defined function (UDF) named *`function_name`*. (`CREATE FUNCTION` is also used to created stored functions; see Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”.)

A user-defined function is a way to extend MySQL with a new function that works like a native (built-in) MySQL function such as `ABS()` or `CONCAT()`. See Adding a User-Defined Function.

`function_name` is the name that should be used in SQL statements to invoke the function. The `RETURNS` clause indicates the type of the function's return value. `DECIMAL` is a legal value after `RETURNS`, but currently `DECIMAL` functions return string values and should be written like `STRING` functions.

The `AGGREGATE` keyword, if given, signifies that the UDF is an aggregate (group) function. An aggregate UDF works exactly like a native MySQL aggregate function such as `SUM()` or `COUNT()`.

`shared_library_name` is the base name of the shared library file containing the code that implements the function. The file must be located in the plugin directory. This directory is given by the value of the `plugin_dir` system variable. For more information, see Section 5.6.1, “Installing and Uninstalling User-Defined Functions”.

`CREATE FUNCTION` requires the `INSERT` privilege for the `mysql` system database because it adds a row to the `mysql.func` system table to register the function.

During the normal startup sequence, the server loads UDFs registered in the `mysql.func` table. If the server is started with the `--skip-grant-tables` option, UDFs registered in the table are not loaded and are unavailable.

Note

To upgrade the shared library associated with a UDF, issue a `DROP FUNCTION` statement, upgrade the shared library, and then issue a `CREATE FUNCTION` statement. If you upgrade the shared library first and then use `DROP FUNCTION`, the server may unexpectedly shut down.

13.7.3.2 DROP FUNCTION Statement for User-Defined Functions

```
DROP FUNCTION [IF EXISTS] function_name
```

This statement drops the user-defined function (UDF) named *`function_name`*. (`DROP FUNCTION` is also used to drop stored functions; see Section 13.1.26, “DROP PROCEDURE and DROP FUNCTION Statements”.)

`DROP FUNCTION` is the complement of `CREATE FUNCTION`. It requires the `DELETE` privilege for the `mysql` system database because it removes the row from the `mysql.func` system table that registers the function.

During the normal startup sequence, the server loads UDFs registered in the `mysql.func` table. Because `DROP FUNCTION` removes the `mysql.func` row for the dropped function, the server does not load the function during subsequent restarts.

Note

To upgrade the shared library associated with a UDF, issue a `DROP FUNCTION` statement, upgrade the shared library, and then issue a `CREATE FUNCTION` statement. If you upgrade the shared library first and then use `DROP FUNCTION`, the server may unexpectedly shut down.

13.7.3.3 INSTALL PLUGIN Statement

```
INSTALL PLUGIN plugin_name SONAME 'shared_library_name'
```

This statement installs a server plugin. It requires the `INSERT` privilege for the `mysql.plugin` system table because it adds a row to that table to register the plugin.

`plugin_name` is the name of the plugin as defined in the plugin descriptor structure contained in the library file (see Plugin Data Structures). Plugin names are not case-sensitive. For maximal compatibility, plugin names should be limited to ASCII letters, digits, and underscore because they are used in C source files, shell command lines, M4 and Bourne shell scripts, and SQL environments.

`shared_library_name` is the name of the shared library that contains the plugin code. The name includes the file name extension (for example, `libmyplugin.so`, `libmyplugin.dll`, or `libmyplugin.dylib`).

The shared library must be located in the plugin directory (the directory named by the `plugin_dir` system variable). The library must be in the plugin directory itself, not in a subdirectory. By default, `plugin_dir` is the `plugin` directory under the directory named by the `pkglibdir` configuration variable, but it can be changed by setting the value of `plugin_dir` at server startup. For example, set its value in a `my.cnf` file:

```
[mysqld]
```

```
plugin_dir=/path/to/plugin/directory
```

If the value of `plugin_dir` is a relative path name, it is taken to be relative to the MySQL base directory (the value of the `basedir` system variable).

`INSTALL PLUGIN` loads and initializes the plugin code to make the plugin available for use. A plugin is initialized by executing its initialization function, which handles any setup that the plugin must perform before it can be used. When the server shuts down, it executes the deinitialization function for each plugin that is loaded so that the plugin has a chance to perform any final cleanup.

`INSTALL PLUGIN` also registers the plugin by adding a line that indicates the plugin name and library file name to the `mysql.plugin` system table. During the normal startup sequence, the server loads and initializes plugins registered in `mysql.plugin`. This means that a plugin is installed with `INSTALL PLUGIN` only once, not every time the server starts. If the server is started with the `--skip-grant-tables` option, plugins registered in the `mysql.plugin` table are not loaded and are unavailable.

A plugin library can contain multiple plugins. For each of them to be installed, use a separate `INSTALL PLUGIN` statement. Each statement names a different plugin, but all of them specify the same library name.

`INSTALL PLUGIN` causes the server to read option (`my.cnf`) files just as during server startup. This enables the plugin to pick up any relevant options from those files. It is possible to add plugin options to an option file even before loading a plugin (if the `loose` prefix is used). It is also possible to uninstall a plugin, edit `my.cnf`, and install the plugin again. Restarting the plugin this way enables it to the new option values without a server restart.

For options that control individual plugin loading at server startup, see Section 5.5.1, “Installing and Uninstalling Plugins”. If you need to load plugins for a single server startup when the `--skip-grant-tables` option is given (which tells the server not to read system tables), use the `--plugin-load` option. See Section 5.1.6, “Server Command Options”.

To remove a plugin, use the `UNINSTALL PLUGIN` statement.

For additional information about plugin loading, see Section 5.5.1, “Installing and Uninstalling Plugins”.

To see what plugins are installed, use the `SHOW PLUGINS` statement or query the `INFORMATION_SCHEMA` the `PLUGINS` table.

If you recompile a plugin library and need to reinstall it, you can use either of the following methods:

- Use `UNINSTALL PLUGIN` to uninstall all plugins in the library, install the new plugin library file in the plugin directory, and then use `INSTALL PLUGIN` to install all plugins in the library. This procedure has the advantage that it can be used without stopping the server. However, if the plugin library contains many plugins, you must issue many `INSTALL PLUGIN` and `UNINSTALL PLUGIN` statements.
- Stop the server, install the new plugin library file in the plugin directory, and restart the server.

13.7.3.4 UNINSTALL PLUGIN Statement

```
UNINSTALL PLUGIN plugin_name
```

This statement removes an installed server plugin. `UNINSTALL PLUGIN` is the complement of `INSTALL PLUGIN`. It requires the `DELETE` privilege for the `mysql.plugin` system table because it removes the row from that table that registers the plugin.

`plugin_name` must be the name of some plugin that is listed in the `mysql.plugin` table. The server executes the plugin's deinitialization function and removes the row for the plugin from the `mysql.plugin` system table, so that subsequent server restarts do not load and initialize the plugin. `UNINSTALL PLUGIN` does not remove the plugin's shared library file.

You cannot uninstall a plugin if any table that uses it is open.

Plugin removal has implications for the use of associated tables. For example, if a full-text parser plugin is associated with a `FULLTEXT` index on the table, uninstalling the plugin makes the table unusable. Any attempt to access the table results in an error. The table cannot even be opened, so you cannot drop an index for which the plugin is used. This means that uninstalling a plugin is something to do with care unless you do not care about the table contents. If you are uninstalling a plugin with no intention of reinstalling it later and you care about the table contents, you should dump the table with `mysqldump` and remove the `WITH PARSER` clause from the dumped `CREATE TABLE` statement so that you can reload the table later. If you do not care about the table, `DROP TABLE` can be used even if any plugins associated with the table are missing.

For additional information about plugin loading, see Section 5.5.1, “Installing and Uninstalling Plugins”.

13.7.4 SET Statements

The `SET` statement has several forms. Descriptions for those forms that are not associated with a specific server capability appear in subsections of this section:

- `SET var_name = value` enables you to assign values to variables that affect the operation of the server or clients. See Section 13.7.4.1, “SET Syntax for Variable Assignment”.
- `SET CHARACTER SET` and `SET NAMES` assign values to character set and collation variables associated with the current connection to the server. See Section 13.7.4.2, “SET CHARACTER SET Statement”, and Section 13.7.4.3, “SET NAMES Statement”.

Descriptions for the other forms appear elsewhere, grouped with other statements related to the capability they help implement:

- `SET PASSWORD` assigns account passwords. See Section 13.7.1.7, “SET PASSWORD Statement”.
- `SET TRANSACTION ISOLATION LEVEL` sets the isolation level for transaction processing. See Section 13.3.6, “SET TRANSACTION Statement”.

13.7.4.1 SET Syntax for Variable Assignment

```
SET variable = expr [, variable = expr] ...

variable: {
    user_var_name
  | param_name
  | local_var_name
  | {GLOBAL | @@GLOBAL.} system_var_name
  | [SESSION | @@SESSION. | @@] system_var_name
}

SET ONE_SHOT system_var_name = expr
```

`SET` syntax for variable assignment enables you to assign values to different types of variables that affect the operation of the server or clients:

- User-defined variables. See Section 9.4, “User-Defined Variables”.
- Stored procedure and function parameters, and stored program local variables. See Section 13.6.4, “Variables in Stored Programs”.
- System variables. See Section 5.1.7, “Server System Variables”. System variables also can be set at server startup, as described in Section 5.1.8, “Using System Variables”.

A `SET` statement that assigns variable values is not written to the binary log, so in replication scenarios it affects only the host on which you execute it. To affect all replication hosts, execute the statement on each host.

The following sections describe SET syntax for setting variables. They use the = assignment operator, but the := assignment operator is also permitted for this purpose.

- User-Defined Variable Assignment
- Parameter and Local Variable Assignment
- System Variable Assignment
- SET Error Handling
- Multiple Variable Assignment
- System Variable References in Expressions
- ONE_SHOT Assignment

User-Defined Variable Assignment

User-defined variables are created locally within a session and exist only within the context of that session; see Section 9.4, “User-Defined Variables”.

A user-defined variable is written as *@var_name* and is assigned an expression value as follows:

```
SET @var_name = expr;
```

Examples:

```
SET @name = 43;
SET @total_tax = (SELECT SUM(tax) FROM taxable_transactions);
```

As demonstrated by those statements, *expr* can range from simple (a literal value) to more complex (the value returned by a scalar subquery).

Parameter and Local Variable Assignment

SET applies to parameters and local variables in the context of the stored object within which they are defined. The following procedure uses the `increment` procedure parameter and `counter` local variable:

```
CREATE PROCEDURE p(increment INT)
BEGIN
  DECLARE counter INT DEFAULT 0;
  WHILE counter < 10 DO
    -- ... do work ...
    SET counter = counter + increment;
  END WHILE;
END;
```

System Variable Assignment

The MySQL server maintains system variables that configure its operation. A system variable can have a global value that affects server operation as a whole, a session value that affects the current session, or both. Many system variables are dynamic and can be changed at runtime using the SET statement to affect operation of the current server instance. (To make a global system variable setting permanent so that it applies across server restarts, you should also set it in an option file.)

If you change a session system variable, the value remains in effect within your session until you change the variable to a different value or the session ends. The change has no effect on other sessions.

If you change a global system variable, the value is remembered and used to initialize the session value for new sessions until you change the variable to a different value or the server exits. The change

is visible to any client that accesses the global value. However, the change affects the corresponding session value only for clients that connect after the change. The global variable change does not affect the session value for any current client sessions (not even the session within which the global value change occurs).

Note

Setting a global system variable value always requires special privileges. Setting a session system variable value normally requires no special privileges and can be done by any user, although there are exceptions. For more information, see Section 5.1.8.1, “System Variable Privileges”.

The following discussion describes the syntax options for setting system variables:

- To assign a value to a global system variable, precede the variable name by the `GLOBAL` keyword or the `@@GLOBAL.` qualifier:

```
SET GLOBAL max_connections = 1000;
SET @@GLOBAL.max_connections = 1000;
```

- To assign a value to a session system variable, precede the variable name by the `SESSION` or `LOCAL` keyword, by the `@@SESSION.`, `@@LOCAL.`, or `@@` qualifier, or by no keyword or no modifier at all:

```
SET SESSION sql_mode = 'TRADITIONAL';
SET LOCAL sql_mode = 'TRADITIONAL';
SET @@SESSION.sql_mode = 'TRADITIONAL';
SET @@LOCAL.sql_mode = 'TRADITIONAL';
SET @@sql_mode = 'TRADITIONAL';
SET sql_mode = 'TRADITIONAL';
```

  A client can change its own session variables, but not those of any other client.

To set a global system variable value to the compiled-in MySQL default value or a session system variable to the current corresponding global value, set the variable to the value `DEFAULT`. For example, the following two statements are identical in setting the session value of `max_join_size` to the current global value:

```
SET @@SESSION.max_join_size = DEFAULT;
SET @@SESSION.max_join_size = @@GLOBAL.max_join_size;
```

To display system variable names and values:

- Use the `SHOW VARIABLES` statement; see Section 13.7.5.40, “SHOW VARIABLES Statement”.
- Several `INFORMATION_SCHEMA` tables provide system variable information. See Section 21.10, “The INFORMATION_SCHEMA GLOBAL_VARIABLES and SESSION_VARIABLES Tables”.

SET Error Handling

If any variable assignment in a `SET` statement fails, the entire statement fails and no variables are changed.

`SET` produces an error under the circumstances described here. Most of the examples show `SET` statements that use keyword syntax (for example, `GLOBAL` or `SESSION`), but the principles are also true for statements that use the corresponding modifiers (for example, `@@GLOBAL.` or `@@SESSION.`).

- Use of `SET` (any variant) to set a read-only variable:

```
mysql> SET GLOBAL version = 'abc';
ERROR 1238 (HY000): Variable 'version' is a read only variable
```

- Use of `GLOBAL` to set a variable that has only a session value:

```
mysql> SET GLOBAL sql_log_bin = ON;
ERROR 1231 (42000): Variable 'sql_log_bin' can't be
set to the value of 'ON'
```

- Use of SESSION to set a variable that has only a global value:

```
mysql> SET SESSION max_connections = 1000;
ERROR 1229 (HY000): Variable 'max_connections' is a
GLOBAL variable and should be set with SET GLOBAL
```

- Omission of GLOBAL to set a variable that has only a global value:

```
mysql> SET max_connections = 1000;
ERROR 1229 (HY000): Variable 'max_connections' is a
GLOBAL variable and should be set with SET GLOBAL
```

- The @@GLOBAL., @@SESSION., and @@ modifiers apply only to system variables. An error occurs for attempts to apply them to user-defined variables, stored procedure or function parameters, or stored program local variables.
- Not all system variables can be set to DEFAULT. In such cases, assigning DEFAULT results in an error.
- An error occurs for attempts to assign DEFAULT to user-defined variables, stored procedure or function parameters, or stored program local variables.

Multiple Variable Assignment

A SET statement can contain multiple variable assignments, separated by commas. This statement assigns a value to a user-defined variable and a system variable:

```
SET @x = 1, SESSION sql_mode = '';
```

If you set multiple system variables in a single statement, the most recent GLOBAL or SESSION keyword in the statement is used for following assignments that have no keyword specified.

Examples of multiple-variable assignment:

```
SET GLOBAL sort_buffer_size = 1000000, SESSION sort_buffer_size = 1000000;
SET @@GLOBAL.sort_buffer_size = 1000000, @@LOCAL.sort_buffer_size = 1000000;
SET GLOBAL max_connections = 1000, sort_buffer_size = 1000000;
```

The @@GLOBAL., @@SESSION., and @@ modifiers apply only to the immediately following system variable, not any remaining system variables. This statement sets the sort_buffer_size global value to 50000 and the session value to 1000000:

```
SET @@GLOBAL.sort_buffer_size = 50000, sort_buffer_size = 1000000;
```

System Variable References in Expressions

To refer to the value of a system variable in expressions, use one of the @@-modifiers. For example, you can retrieve system variable values in a SELECT statement like this:

```
SELECT @@GLOBAL.sql_mode, @@SESSION.sql_mode, @@sql_mode;
```

Note

A reference to a system variable in an expression as *@@var_name* (with @@ rather than @@GLOBAL. or @@SESSION.) returns the session value if it exists and the global value otherwise. This differs from *SET @@var_name = expr*, which always refers to the session value.

ONE_SHOT Assignment

The `SET ONE_SHOT` syntax is for internal use only, has been deprecated since MySQL 5.0, and was removed in MySQL 5.6.1.

13.7.4.2 SET CHARACTER SET Statement

```
SET {CHARACTER SET | CHARSET}
    {'charset_name' | DEFAULT}
```

This statement maps all strings sent between the server and the current client with the given mapping. `SET CHARACTER SET` sets three session system variables: `character_set_client` and `character_set_results` are set to the given character set, and `character_set_connection` to the value of `character_set_database`. See Section 10.4, “Connection Character Sets and Collations”.

`charset_name` may be quoted or unquoted.

The default character set mapping can be restored by using the value `DEFAULT`. The default depends on the server configuration.

Some character sets cannot be used as the client character set. Attempting to use them with `SET CHARACTER SET` produces an error. See Impermissible Client Character Sets.

13.7.4.3 SET NAMES Statement

```
SET NAMES {'charset_name'
    [COLLATE 'collation_name'] | DEFAULT}
```

This statement sets the three session system variables `character_set_client`, `character_set_connection`, and `character_set_results` to the given character set. Setting `character_set_connection` to `charset_name` also sets `collation_connection` to the default collation for `charset_name`. See Section 10.4, “Connection Character Sets and Collations”.

The optional `COLLATE` clause may be used to specify a collation explicitly. If given, the collation must one of the permitted collations for *`charset_name`*.

`charset_name` and *`collation_name`* may be quoted or unquoted.

The default mapping can be restored by using a value of `DEFAULT`. The default depends on the server configuration.

Some character sets cannot be used as the client character set. Attempting to use them with `SET NAMES` produces an error. See Impermissible Client Character Sets.

13.7.5 SHOW Statements

`SHOW` has many forms that provide information about databases, tables, columns, or status information about the server. This section describes those following:

```
SHOW AUTHORS
SHOW {BINARY | MASTER} LOGS
SHOW BINLOG EVENTS [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
SHOW CHARACTER SET [like_or_where]
SHOW COLLATION [like_or_where]
SHOW [FULL] COLUMNS FROM tbl_name [FROM db_name] [like_or_where]
SHOW CONTRIBUTORS
SHOW CREATE DATABASE db_name
SHOW CREATE EVENT event_name
SHOW CREATE FUNCTION func_name
SHOW CREATE PROCEDURE proc_name
SHOW CREATE TABLE tbl_name
SHOW CREATE TRIGGER trigger_name
SHOW CREATE VIEW view_name
SHOW DATABASES [like_or_where]
```

```
SHOW ENGINE engine_name {STATUS | MUTEX}
SHOW [STORAGE] ENGINES
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW EVENTS
SHOW FUNCTION CODE func_name
SHOW FUNCTION STATUS [like_or_where]
SHOW GRANTS FOR user
SHOW INDEX FROM tbl_name [FROM db_name]
SHOW MASTER STATUS
SHOW OPEN TABLES [FROM db_name] [like_or_where]
SHOW PLUGINS
SHOW PROCEDURE CODE proc_name
SHOW PROCEDURE STATUS [like_or_where]
SHOW PRIVILEGES
SHOW [FULL] PROCESSLIST
SHOW PROFILE [types] [FOR QUERY n] [OFFSET n] [LIMIT n]
SHOW PROFILES
SHOW RELAYLOG EVENTS [IN 'log_name'] [FROM pos] [LIMIT [offset,] row_count]
SHOW SLAVE HOSTS
SHOW SLAVE STATUS
SHOW [GLOBAL | SESSION] STATUS [like_or_where]
SHOW TABLE STATUS [FROM db_name] [like_or_where]
SHOW [FULL] TABLES [FROM db_name] [like_or_where]
SHOW TRIGGERS [FROM db_name] [like_or_where]
SHOW [GLOBAL | SESSION] VARIABLES [like_or_where]
SHOW WARNINGS [LIMIT [offset,] row_count]

like_or_where: {
    LIKE 'pattern'
  | WHERE expr
}
```

If the syntax for a given `SHOW` statement includes a `LIKE 'pattern'` part, `'pattern'` is a string that can contain the SQL `%` and `_` wildcard characters. The pattern is useful for restricting statement output to matching values.

Several `SHOW` statements also accept a `WHERE` clause that provides more flexibility in specifying which rows to display. See Section 21.35, “Extensions to SHOW Statements”.

Many MySQL APIs (such as PHP) enable you to treat the result returned from a `SHOW` statement as you would a result set from a `SELECT`; see Chapter 23, *Connectors and APIs*, or your API documentation for more information. In addition, you can work in SQL with results from queries on tables in the `INFORMATION_SCHEMA` database, which you cannot easily do with results from `SHOW` statements. See Chapter 21, *INFORMATION_SCHEMA Tables*.

13.7.5.1 SHOW AUTHORS Statement

```
SHOW AUTHORS
```

The `SHOW AUTHORS` statement displays information about the people who work on MySQL. For each author, it displays `Name`, `Location`, and `Comment` values.

This statement is removed as of MySQL 5.6.8.

13.7.5.2 SHOW BINARY LOGS Statement

```
SHOW BINARY LOGS
SHOW MASTER LOGS
```

Lists the binary log files on the server. This statement is used as part of the procedure described in Section 13.4.1.1, “PURGE BINARY LOGS Statement”, that shows how to determine which logs can be purged.

```
mysql> SHOW BINARY LOGS;
+---------------+-----------+
| Log_name      | File_size |
```

```
+---------------+-----------+
| binlog.000015 |    724935 |
| binlog.000016 |    733481 |
+---------------+-----------+
```

`SHOW MASTER LOGS` is equivalent to `SHOW BINARY LOGS`.

A user with the `SUPER` or `REPLICATION CLIENT` privilege may execute this statement.

13.7.5.3 SHOW BINLOG EVENTS Statement

```
SHOW BINLOG EVENTS
   [IN 'log_name']
   [FROM pos]
   [LIMIT [offset,] row_count]
```

Shows the events in the binary log. If you do not specify `'log_name'`, the first binary log is displayed.

The `LIMIT` clause has the same syntax as for the `SELECT` statement. See Section 13.2.9, "SELECT Statement".

Note

Issuing a `SHOW BINLOG EVENTS` with no `LIMIT` clause could start a very time- and resource-consuming process because the server returns to the client the complete contents of the binary log (which includes all statements executed by the server that modify data). As an alternative to `SHOW BINLOG EVENTS`, use the `mysqlbinlog` utility to save the binary log to a text file for later examination and analysis. See Section 4.6.8, "`mysqlbinlog` — Utility for Processing Binary Log Files".

`SHOW BINLOG EVENTS` displays the following fields for each event in the binary log:

- `Log_name`

  The name of the file that is being listed.

- `Pos`

  The position at which the event occurs.

- `Event_type`

  An identifier that describes the event type.

- `Server_id`

  The server ID of the server on which the event originated.

- `End_log_pos`

  The position at which the next event begins, which is equal to `Pos` plus the size of the event.

- `Info`

  More detailed information about the event type. The format of this information depends on the event type.

Note

Some events relating to the setting of user and system variables are not included in the output from `SHOW BINLOG EVENTS`. To get complete coverage of events within a binary log, use `mysqlbinlog`.

Note

`SHOW BINLOG EVENTS` does *not* work with relay log files. You can use `SHOW RELAYLOG EVENTS` for this purpose.

13.7.5.4 SHOW CHARACTER SET Statement

```
SHOW CHARACTER SET
    [LIKE 'pattern' | WHERE expr]
```

The `SHOW CHARACTER SET` statement shows all available character sets. The `LIKE` clause, if present, indicates which character set names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 21.35, “Extensions to SHOW Statements”. For example:

```
mysql> SHOW CHARACTER SET LIKE 'latin%';
+---------+-----------------------------+-------------------+--------+
| Charset | Description                 | Default collation | Maxlen |
+---------+-----------------------------+-------------------+--------+
| latin1  | cp1252 West European        | latin1_swedish_ci |      1 |
| latin2  | ISO 8859-2 Central European | latin2_general_ci |      1 |
| latin5  | ISO 8859-9 Turkish          | latin5_turkish_ci |      1 |
| latin7  | ISO 8859-13 Baltic          | latin7_general_ci |      1 |
+---------+-----------------------------+-------------------+--------+
```

`SHOW CHARACTER SET` output has these columns:

- `Charset`

  The character set name.

- `Description`

  A description of the character set.

- `Default collation`

  The default collation for the character set.

- `Maxlen`

  The maximum number of bytes required to store one character.

The `filename` character set is for internal use only; consequently, `SHOW CHARACTER SET` does not display it.

Character set information is also available from the `INFORMATION_SCHEMA CHARACTER_SETS` table. See Section 21.2, “The INFORMATION_SCHEMA CHARACTER_SETS Table”.

13.7.5.5 SHOW COLLATION Statement

```
SHOW COLLATION
    [LIKE 'pattern' | WHERE expr]
```

This statement lists collations supported by the server. By default, the output from `SHOW COLLATION` includes all available collations. The `LIKE` clause, if present, indicates which collation names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 21.35, “Extensions to SHOW Statements”. For example:

```
mysql> SHOW COLLATION WHERE Charset = 'latin1';
+-------------------+---------+----+---------+----------+---------+
| Collation         | Charset | Id | Default | Compiled | Sortlen |
+-------------------+---------+----+---------+----------+---------+
| latin1_german1_ci | latin1  |  5 |         | Yes      |       1 |
| latin1_swedish_ci | latin1  |  8 | Yes     | Yes      |       1 |
```

```
| latin1_danish_ci  | latin1  | 15 |         | Yes      |       1 |
| latin1_german2_ci | latin1  | 31 |         | Yes      |       2 |
| latin1_bin        | latin1  | 47 |         | Yes      |       1 |
| latin1_general_ci | latin1  | 48 |         | Yes      |       1 |
| latin1_general_cs | latin1  | 49 |         | Yes      |       1 |
| latin1_spanish_ci | latin1  | 94 |         | Yes      |       1 |
+-------------------+---------+----+---------+----------+---------+
```

`SHOW COLLATION` output has these columns:

- `Collation`

  The collation name.

- `Charset`

  The name of the character set with which the collation is associated.

- `Id`

  The collation ID.

- `Default`

  Whether the collation is the default for its character set.

- `Compiled`

  Whether the character set is compiled into the server.

- `Sortlen`

  This is related to the amount of memory required to sort strings expressed in the character set.

To see the default collation for each character set, use the following statement. `Default` is a reserved word, so to use it as an identifier, it must be quoted as such:

```
mysql> SHOW COLLATION WHERE `Default` = 'Yes';
+---------------------+----------+----+---------+----------+---------+
| Collation           | Charset  | Id | Default | Compiled | Sortlen |
+---------------------+----------+----+---------+----------+---------+
| big5_chinese_ci     | big5     |  1 | Yes     | Yes      |       1 |
| dec8_swedish_ci     | dec8     |  3 | Yes     | Yes      |       1 |
| cp850_general_ci    | cp850    |  4 | Yes     | Yes      |       1 |
| hp8_english_ci      | hp8      |  6 | Yes     | Yes      |       1 |
| koi8r_general_ci    | koi8r    |  7 | Yes     | Yes      |       1 |
| latin1_swedish_ci   | latin1   |  8 | Yes     | Yes      |       1 |
...
```

Collation information is also available from the `INFORMATION_SCHEMA COLLATIONS` table. See Section 21.3, “The INFORMATION_SCHEMA COLLATIONS Table”.

13.7.5.6 SHOW COLUMNS Statement

```
SHOW [FULL] {COLUMNS | FIELDS}
    {FROM | IN} tbl_name
    [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

`SHOW COLUMNS` displays information about the columns in a given table. It also works for views. `SHOW COLUMNS` displays information only for those columns for which you have some privilege.

```
mysql> SHOW COLUMNS FROM City;
+-------------+----------+------+-----+---------+----------------+
| Field       | Type     | Null | Key | Default | Extra          |
+-------------+----------+------+-----+---------+----------------+
| ID          | int(11)  | NO   | PRI | NULL    | auto_increment |
```

```
| Name        | char(35) | NO   |     |         |                |
| CountryCode | char(3)  | NO   | MUL |         |                |
| District    | char(20) | NO   |     |         |                |
| Population  | int(11)  | NO   |     | 0       |                |
+-------------+----------+------+-----+---------+----------------+
```

An alternative to *`tbl_name FROM db_name`* syntax is *`db_name.tbl_name`*. These two statements are equivalent:

```
SHOW COLUMNS FROM mytable FROM mydb;
SHOW COLUMNS FROM mydb.mytable;
```

The optional `FULL` keyword causes the output to include the column collation and comments, as well as the privileges you have for each column.

The `LIKE` clause, if present, indicates which column names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 21.35, “Extensions to SHOW Statements”.

The data types may differ from what you expect them to be based on a `CREATE TABLE` statement because MySQL sometimes changes data types when you create or alter a table. The conditions under which this occurs are described in Section 13.1.17.6, “Silent Column Specification Changes”.

`SHOW COLUMNS` displays the following values for each table column:

- `Field`

  The column name.

- `Type`

  The column data type.

- `Collation`

  The collation for nonbinary string columns, or `NULL` for other columns. This value is displayed only if you use the `FULL` keyword.

- `Null`

  The column nullability. The value is `YES` if `NULL` values can be stored in the column, `NO` if not.

- `Key`

  Whether the column is indexed:

  - If `Key` is empty, the column either is not indexed or is indexed only as a secondary column in a multiple-column, nonunique index.
  - If `Key` is `PRI`, the column is a `PRIMARY KEY` or is one of the columns in a multiple-column `PRIMARY KEY`.
  - If `Key` is `UNI`, the column is the first column of a `UNIQUE` index. (A `UNIQUE` index permits multiple `NULL` values, but you can tell whether the column permits `NULL` by checking the `Null` field.)
  - If `Key` is `MUL`, the column is the first column of a nonunique index in which multiple occurrences of a given value are permitted within the column.

  If more than one of the `Key` values applies to a given column of a table, `Key` displays the one with the highest priority, in the order `PRI`, `UNI`, `MUL`.

  A `UNIQUE` index may be displayed as `PRI` if it cannot contain `NULL` values and there is no `PRIMARY KEY` in the table. A `UNIQUE` index may display as `MUL` if several columns form a composite `UNIQUE`

index; although the combination of the columns is unique, each column can still hold multiple occurrences of a given value.

- `Default`

  The default value for the column. This is `NULL` if the column has an explicit default of `NULL`, or if the column definition includes no `DEFAULT` clause.

- `Extra`

  Any additional information that is available about a given column. The value is nonempty in these cases: `auto_increment` for columns that have the `AUTO_INCREMENT` attribute; `on update CURRENT_TIMESTAMP` for `TIMESTAMP` or `DATETIME` columns that have the `ON UPDATE CURRENT_TIMESTAMP` attribute.

- `Privileges`

  The privileges you have for the column. This value is displayed only if you use the `FULL` keyword.

- `Comment`

  Any comment included in the column definition. This value is displayed only if you use the `FULL` keyword.

Table column information is also available from the `INFORMATION_SCHEMA COLUMNS` table. See Section 21.5, “The INFORMATION_SCHEMA COLUMNS Table”.

You can list a table's columns with the `mysqlshow` *`db_name tbl_name`* command.

The `DESCRIBE` statement provides information similar to `SHOW COLUMNS`. See Section 13.8.1, “DESCRIBE Statement”.

The `SHOW CREATE TABLE`, `SHOW TABLE STATUS`, and `SHOW INDEX` statements also provide information about tables. See Section 13.7.5, “SHOW Statements”.

13.7.5.7 SHOW CONTRIBUTORS Statement

```
SHOW CONTRIBUTORS
```

The `SHOW CONTRIBUTORS` statement displays information about the people who contribute to MySQL source or to causes that we support. For each contributor, it displays `Name`, `Location`, and `Comment` values.

This statement is removed as of MySQL 5.6.8.

13.7.5.8 SHOW CREATE DATABASE Statement

```
SHOW CREATE {DATABASE | SCHEMA} [IF NOT EXISTS] db_name
```

Shows the `CREATE DATABASE` statement that creates the named database. If the `SHOW` statement includes an `IF NOT EXISTS` clause, the output too includes such a clause. `SHOW CREATE SCHEMA` is a synonym for `SHOW CREATE DATABASE`.

```
mysql> SHOW CREATE DATABASE test\G
*************************** 1. row ***************************
       Database: test
Create Database: CREATE DATABASE `test`
                 /*!40100 DEFAULT CHARACTER SET latin1 */

mysql> SHOW CREATE SCHEMA test\G
*************************** 1. row ***************************
       Database: test
```

```
Create Database: CREATE DATABASE `test`
                 /*!40100 DEFAULT CHARACTER SET latin1 */
```

`SHOW CREATE DATABASE` quotes table and column names according to the value of the `sql_quote_show_create` option. See Section 5.1.7, “Server System Variables”.

13.7.5.9 SHOW CREATE EVENT Statement

```
SHOW CREATE EVENT event_name
```

This statement displays the `CREATE EVENT` statement needed to re-create a given event. It requires the `EVENT` privilege for the database from which the event is to be shown. For example (using the same event `e_daily` defined and then altered in Section 13.7.5.19, “SHOW EVENTS Statement”):

```
mysql> SHOW CREATE EVENT myschema.e_daily\G
*************************** 1. row ***************************
               Event: e_daily
            sql_mode: NO_ENGINE_SUBSTITUTION
           time_zone: SYSTEM
       Create Event: CREATE DEFINER=`jon`@`ghidora` EVENT `e_daily`
                       ON SCHEDULE EVERY 1 DAY
                       STARTS CURRENT_TIMESTAMP + INTERVAL 6 HOUR
                       ON COMPLETION NOT PRESERVE
                       ENABLE
                       COMMENT 'Saves total number of sessions then
                               clears the table each day'
                       DO BEGIN
                         INSERT INTO site_activity.totals (time, total)
                           SELECT CURRENT_TIMESTAMP, COUNT(*)
                             FROM site_activity.sessions;
                         DELETE FROM site_activity.sessions;
                       END
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

`character_set_client` is the session value of the `character_set_client` system variable when the event was created. `collation_connection` is the session value of the `collation_connection` system variable when the event was created. `Database Collation` is the collation of the database with which the event is associated.

The output reflects the current status of the event (`ENABLE`) rather than the status with which it was created.

13.7.5.10 SHOW CREATE FUNCTION Statement

```
SHOW CREATE FUNCTION func_name
```

This statement is similar to `SHOW CREATE PROCEDURE` but for stored functions. See Section 13.7.5.11, “SHOW CREATE PROCEDURE Statement”.

13.7.5.11 SHOW CREATE PROCEDURE Statement

```
SHOW CREATE PROCEDURE proc_name
```

This statement is a MySQL extension. It returns the exact string that can be used to re-create the named stored procedure. A similar statement, `SHOW CREATE FUNCTION`, displays information about stored functions (see Section 13.7.5.10, “SHOW CREATE FUNCTION Statement”).

To use either statement, you must be the user named in the routine `DEFINER` clause or have `SELECT` access to the `mysql.proc` table. If you do not have privileges for the routine itself, the value displayed for the `Create Procedure` or `Create Function` column is `NULL`.

```
mysql> SHOW CREATE PROCEDURE test.citycount\G
```

```
*************************** 1. row ***************************
           Procedure: citycount
            sql_mode: STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION
    Create Procedure: CREATE DEFINER=`me`@`localhost`
                      PROCEDURE `citycount`(IN country CHAR(3), OUT cities INT)
                      BEGIN
                        SELECT COUNT(*) INTO cities FROM world.city
                        WHERE CountryCode = country;
                      END
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci

mysql> SHOW CREATE FUNCTION test.hello\G
*************************** 1. row ***************************
            Function: hello
            sql_mode: STRICT_TRANS_TABLES,NO_ENGINE_SUBSTITUTION
     Create Function: CREATE DEFINER=`me`@`localhost`
                      FUNCTION `hello`(s CHAR(20))
                      RETURNS char(50) CHARSET latin1
                      DETERMINISTIC
                      RETURN CONCAT('Hello, ',s,'!')
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

`character_set_client` is the session value of the `character_set_client` system variable when the routine was created. `collation_connection` is the session value of the `collation_connection` system variable when the routine was created. `Database Collation` is the collation of the database with which the routine is associated.

13.7.5.12 SHOW CREATE TABLE Statement

```
SHOW CREATE TABLE tbl_name
```

Shows the `CREATE TABLE` statement that creates the named table. To use this statement, you must have some privilege for the table. This statement also works with views.

```
mysql> SHOW CREATE TABLE t\G
*************************** 1. row ***************************
       Table: t
Create Table: CREATE TABLE `t` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `s` char(60) DEFAULT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

`SHOW CREATE TABLE` quotes table and column names according to the value of the `sql_quote_show_create` option. See Section 5.1.7, “Server System Variables”.

When altering the storage engine of a table, table options that are not applicable to the new storage engine are retained in the table definition to enable reverting the table with its previously defined options to the original storage engine, if necessary. For example, when changing the storage engine from InnoDB to MyISAM, InnoDB-specific options such as `ROW_FORMAT=COMPACT` are retained.

```
mysql> CREATE TABLE t1 (c1 INT PRIMARY KEY) ROW_FORMAT=COMPACT ENGINE=InnoDB;
mysql> ALTER TABLE t1 ENGINE=MyISAM;
mysql> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `c1` int(11) NOT NULL,
  PRIMARY KEY (`c1`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1 ROW_FORMAT=COMPACT
```

When creating a table with strict mode disabled, the storage engine's default row format is used if the specified row format is not supported. The actual row format of the table is reported in the

`Row_format` column in response to `SHOW TABLE STATUS`. `SHOW CREATE TABLE` shows the row format that was specified in the `CREATE TABLE` statement.

13.7.5.13 SHOW CREATE TRIGGER Statement

```
SHOW CREATE TRIGGER trigger_name
```

This statement shows the `CREATE TRIGGER` statement that creates the named trigger. This statement requires the `TRIGGER` privilege for the table associated with the trigger.

```
mysql> SHOW CREATE TRIGGER ins_sum\G
*************************** 1. row ***************************
               Trigger: ins_sum
              sql_mode: NO_ENGINE_SUBSTITUTION
SQL Original Statement: CREATE DEFINER=`me`@`localhost` TRIGGER ins_sum
                        BEFORE INSERT ON account
                        FOR EACH ROW SET @sum = @sum + NEW.amount
  character_set_client: utf8
  collation_connection: utf8_general_ci
    Database Collation: latin1_swedish_ci
```

`SHOW CREATE TRIGGER` output has these columns:

- `Trigger`: The trigger name.
- `sql_mode`: The SQL mode in effect when the trigger executes.
- `SQL Original Statement`: The `CREATE TRIGGER` statement that defines the trigger.
- `character_set_client`: The session value of the `character_set_client` system variable when the trigger was created.
- `collation_connection`: The session value of the `collation_connection` system variable when the trigger was created.
- `Database Collation`: The collation of the database with which the trigger is associated.

Trigger information is also available from the `INFORMATION_SCHEMA TRIGGERS` table. See Section 21.27, “The INFORMATION_SCHEMA TRIGGERS Table”.

13.7.5.14 SHOW CREATE VIEW Statement

```
SHOW CREATE VIEW view_name
```

This statement shows the `CREATE VIEW` statement that creates the named view.

```
mysql> SHOW CREATE VIEW v\G
*************************** 1. row ***************************
                View: v
         Create View: CREATE ALGORITHM=UNDEFINED
                      DEFINER=`bob`@`localhost`
                      SQL SECURITY DEFINER VIEW
                      `v` AS select 1 AS `a`,2 AS `b`
character_set_client: utf8
collation_connection: utf8_general_ci
```

`character_set_client` is the session value of the `character_set_client` system variable when the view was created. `collation_connection` is the session value of the `collation_connection` system variable when the view was created.

Use of `SHOW CREATE VIEW` requires the `SHOW VIEW` privilege, and the `SELECT` privilege for the view in question.

View information is also available from the `INFORMATION_SCHEMA VIEWS` table. See Section 21.29, “The INFORMATION_SCHEMA VIEWS Table”.

MySQL lets you use different `sql_mode` settings to tell the server the type of SQL syntax to support. For example, you might use the `ANSI` SQL mode to ensure MySQL correctly interprets the standard SQL concatenation operator, the double bar (`||`), in your queries. If you then create a view that concatenates items, you might worry that changing the `sql_mode` setting to a value different from `ANSI` could cause the view to become invalid. But this is not the case. No matter how you write out a view definition, MySQL always stores it the same way, in a canonical form. Here is an example that shows how the server changes a double bar concatenation operator to a `CONCAT()` function:

```
mysql> SET sql_mode = 'ANSI';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE VIEW test.v AS SELECT 'a' || 'b' as col1;
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW CREATE VIEW test.v\G
*************************** 1. row ***************************
                View: v
         Create View: CREATE VIEW "v" AS select concat('a','b') AS "col1"
...
1 row in set (0.00 sec)
```

The advantage of storing a view definition in canonical form is that changes made later to the value of `sql_mode` does not affect the results from the view. However an additional consequence is that comments prior to `SELECT` are stripped from the definition by the server.

13.7.5.15 SHOW DATABASES Statement

```
SHOW {DATABASES | SCHEMAS}
    [LIKE 'pattern' | WHERE expr]
```

`SHOW DATABASES` lists the databases on the MySQL server host. `SHOW SCHEMAS` is a synonym for `SHOW DATABASES`. The `LIKE` clause, if present, indicates which database names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 21.35, “Extensions to SHOW Statements”.

You see only those databases for which you have some kind of privilege, unless you have the global `SHOW DATABASES` privilege. You can also get this list using the `mysqlshow` command.

If the server was started with the `--skip-show-database` option, you cannot use this statement at all unless you have the `SHOW DATABASES` privilege.

MySQL implements databases as directories in the data directory, so this statement simply lists directories in that location. However, the output may include names of directories that do not correspond to actual databases.

Database information is also available from the `INFORMATION_SCHEMA SCHEMATA` table. See Section 21.20, “The INFORMATION_SCHEMA SCHEMATA Table”.

Caution

Because a global privilege is considered a privilege for all databases, *any* global privilege enables a user to see all database names with `SHOW DATABASES` or by examining the `INFORMATION_SCHEMA SCHEMATA` table.

13.7.5.16 SHOW ENGINE Statement

```
SHOW ENGINE engine_name {STATUS | MUTEX}
```

`SHOW ENGINE` displays operational information about a storage engine. It requires the `PROCESS` privilege. The statement has these variants:

```
SHOW ENGINE INNODB STATUS
```

```
SHOW ENGINE INNODB MUTEX
SHOW ENGINE {NDB | NDBCLUSTER} STATUS
SHOW ENGINE PERFORMANCE_SCHEMA STATUS
```

`SHOW ENGINE INNODB STATUS` displays extensive information from the standard `InnoDB` Monitor about the state of the `InnoDB` storage engine. For information about the standard monitor and other `InnoDB` Monitors that provide information about `InnoDB` processing, see Section 14.17, "InnoDB Monitors".

`SHOW ENGINE INNODB MUTEX` displays `InnoDB` mutex and rw-lock statistics.

Note

Most `SHOW ENGINE INNODB MUTEX` output is removed in 5.6.14. `SHOW ENGINE INNODB MUTEX` output is removed entirely in MySQL 5.7.2. `InnoDB` mutexes can be monitored using Performance Schema tables. For an example, see Section 14.16.1, "Monitoring InnoDB Mutex Waits Using Performance Schema".

- `Type`

  Always `InnoDB`.

- `Name`

  The source file where the mutex is implemented, and the line number in the file where the mutex is created. The line number is specific to your version of MySQL.

- `Status`

  The mutex status. This field displays several values if `WITH_DEBUG` was defined at MySQL compilation time. If `WITH_DEBUG` was not defined, the statement displays only the `os_waits` value. In the latter case (without `WITH_DEBUG`), the information on which the output is based is insufficient to distinguish regular mutexes and mutexes that protect rw-locks (which permit multiple readers or a single writer). Consequently, the output may appear to contain multiple rows for the same mutex.

  - `count` indicates how many times the mutex was requested.
  - `spin_waits` indicates how many times the spinlock had to run.
  - `spin_rounds` indicates the number of spinlock rounds. (`spin_rounds` divided by `spin_waits` provides the average round count.)
  - `os_waits` indicates the number of operating system waits. This occurs when the spinlock did not work (the mutex was not locked during the spinlock and it was necessary to yield to the operating system and wait).
  - `os_yields` indicates the number of times a thread trying to lock a mutex gave up its timeslice and yielded to the operating system (on the presumption that permitting other threads to run frees the mutex so that it can be locked).
  - `os_wait_times` indicates the amount of time (in ms) spent in operating system waits. In MySQL 5.6 timing is disabled and this value is always 0.

`SHOW ENGINE INNODB MUTEX` does not list mutexes and rw-locks for each buffer pool block, as the amount of output would be overwhelming on systems with a large buffer pool. `SHOW ENGINE INNODB MUTEX` does, however, print aggregate `BUF_BLOCK_MUTEX` spin, wait, and call values for buffer pool block mutexes and rw-locks. `SHOW ENGINE INNODB MUTEX` also does not list any mutexes or rw-locks that have never been waited on (`os_waits=0`). Thus, `SHOW ENGINE INNODB MUTEX` only displays information about mutexes and rw-locks outside of the buffer pool that have caused at least one OS-level wait.

`SHOW ENGINE INNODB MUTEX` information can be used to diagnose system problems. For example, large values of `spin_waits` and `spin_rounds` may indicate scalability problems.

Use `SHOW ENGINE PERFORMANCE_SCHEMA STATUS` to inspect the internal operation of the Performance Schema code:

```
mysql> SHOW ENGINE PERFORMANCE_SCHEMA STATUS\G
...
*************************** 3. row ***************************
  Type: performance_schema
  Name: events_waits_history.row_size
Status: 76
*************************** 4. row ***************************
  Type: performance_schema
  Name: events_waits_history.row_count
Status: 10000
*************************** 5. row ***************************
  Type: performance_schema
  Name: events_waits_history.memory
Status: 760000
...
*************************** 57. row ***************************
  Type: performance_schema
  Name: performance_schema.memory
Status: 26459600
...
```

This statement is intended to help the DBA understand the effects that different Performance Schema options have on memory requirements.

`Name` values consist of two parts, which name an internal buffer and a buffer attribute, respectively. Interpret buffer names as follows:

- An internal buffer that is not exposed as a table is named within parentheses. Examples: `(pfs_cond_class).row_size`, `(pfs_mutex_class).memory`.
- An internal buffer that is exposed as a table in the `performance_schema` database is named after the table, without parentheses. Examples: `events_waits_history.row_size`, `mutex_instances.row_count`.
- A value that applies to the Performance Schema as a whole begins with `performance_schema`. Example: `performance_schema.memory`.

Buffer attributes have these meanings:

- `row_size` is the size of the internal record used by the implementation, such as the size of a row in a table. `row_size` values cannot be changed.
- `row_count` is the number of internal records, such as the number of rows in a table. `row_count` values can be changed using Performance Schema configuration options.
- For a table, *`tbl_name`*`.memory` is the product of `row_size` and `row_count`. For the Performance Schema as a whole, `performance_schema.memory` is the sum of all the memory used (the sum of all other `memory` values).

In some cases, there is a direct relationship between a Performance Schema configuration parameter and a `SHOW ENGINE` value. For example, `events_waits_history_long.row_count` corresponds to `performance_schema_events_waits_history_long_size`. In other cases, the relationship is more complex. For example, `events_waits_history.row_count` corresponds to `performance_schema_events_waits_history_size` (the number of rows per thread) multiplied by `performance_schema_max_thread_instances` (the number of threads).

SHOW ENGINE NDB STATUS. If the server has the `NDB` storage engine enabled, `SHOW ENGINE NDB STATUS` displays cluster status information such as the number of connected data nodes, the

cluster connectstring, and cluster binary log epochs, as well as counts of various Cluster API objects created by the MySQL Server when connected to the cluster. Sample output from this statement is shown here:

```
mysql> SHOW ENGINE NDB STATUS;
+------------+----------------------+-------------------------------------------------+
| Type       | Name                 | Status                                          |
+------------+----------------------+-------------------------------------------------+
| ndbcluster | connection           | cluster_node_id=7,
  connected_host=198.51.100.103, connected_port=1186, number_of_data_nodes=4,
  number_of_ready_data_nodes=3, connect_count=0                                       |
| ndbcluster | NdbTransaction       | created=6, free=0, sizeof=212                   |
| ndbcluster | NdbOperation         | created=8, free=8, sizeof=660                   |
| ndbcluster | NdbIndexScanOperation | created=1, free=1, sizeof=744                  |
| ndbcluster | NdbIndexOperation    | created=0, free=0, sizeof=664                   |
| ndbcluster | NdbRecAttr           | created=1285, free=1285, sizeof=60              |
| ndbcluster | NdbApiSignal         | created=16, free=16, sizeof=136                 |
| ndbcluster | NdbLabel             | created=0, free=0, sizeof=196                   |
| ndbcluster | NdbBranch            | created=0, free=0, sizeof=24                    |
| ndbcluster | NdbSubroutine        | created=0, free=0, sizeof=68                    |
| ndbcluster | NdbCall              | created=0, free=0, sizeof=16                    |
| ndbcluster | NdbBlob              | created=1, free=1, sizeof=264                   |
| ndbcluster | NdbReceiver          | created=4, free=0, sizeof=68                    |
| ndbcluster | binlog               | latest_epoch=155467, latest_trans_epoch=148126,
  latest_received_binlog_epoch=0, latest_handled_binlog_epoch=0,
  latest_applied_binlog_epoch=0                                                       |
+------------+----------------------+-------------------------------------------------+
```

The `Status` column in each of these rows provides information about the MySQL server's connection to the cluster and about the cluster binary log's status, respectively. The `Status` information is in the form of comma-delimited set of name/value pairs.

The `connection` row's `Status` column contains the name/value pairs described in the following table.

Name	Value
`cluster_node_id`	The node ID of the MySQL server in the cluster
`connected_host`	The host name or IP address of the cluster management server to which the MySQL server is connected
`connected_port`	The port used by the MySQL server to connect to the management server (`connected_host`)
`number_of_data_nodes`	The number of data nodes configured for the cluster (that is, the number of `[ndbd]` sections in the cluster `config.ini` file)
`number_of_ready_data_nodes`	The number of data nodes in the cluster that are actually running
`connect_count`	The number of times this `mysqld` has connected or reconnected to cluster data nodes

The `binlog` row's `Status` column contains information relating to NDB Cluster Replication. The name/value pairs it contains are described in the following table.

Name	Value
`latest_epoch`	The most recent epoch most recently run on this MySQL server (that is, the sequence number of the most recent transaction run on the server)
`latest_trans_epoch`	The most recent epoch processed by the cluster's data nodes
`latest_received_binlog_epoch`	The most recent epoch received by the binary log thread

Name	Value
`latest_handled_binlog_epoch`	The most recent epoch processed by the binary log thread (for writing to the binary log)
`latest_applied_binlog_epoch`	The most recent epoch actually written to the binary log

See Section 18.6, “NDB Cluster Replication”, for more information.

The remaining rows from the output of `SHOW ENGINE NDB STATUS` which are most likely to prove useful in monitoring the cluster are listed here by `Name`:

- `NdbTransaction`: The number and size of `NdbTransaction` objects that have been created. An `NdbTransaction` is created each time a table schema operation (such as `CREATE TABLE` or `ALTER TABLE`) is performed on an `NDB` table.
- `NdbOperation`: The number and size of `NdbOperation` objects that have been created.
- `NdbIndexScanOperation`: The number and size of `NdbIndexScanOperation` objects that have been created.
- `NdbIndexOperation`: The number and size of `NdbIndexOperation` objects that have been created.
- `NdbRecAttr`: The number and size of `NdbRecAttr` objects that have been created. In general, one of these is created each time a data manipulation statement is performed by an SQL node.
- `NdbBlob`: The number and size of `NdbBlob` objects that have been created. An `NdbBlob` is created for each new operation involving a `BLOB` column in an `NDB` table.
- `NdbReceiver`: The number and size of any `NdbReceiver` object that have been created. The number in the `created` column is the same as the number of data nodes in the cluster to which the MySQL server has connected.

Note

`SHOW ENGINE NDB STATUS` returns an empty result if no operations involving `NDB` tables have been performed during the current session by the MySQL client accessing the SQL node on which this statement is run.

13.7.5.17 SHOW ENGINES Statement

```
SHOW [STORAGE] ENGINES
```

`SHOW ENGINES` displays status information about the server's storage engines. This is particularly useful for checking whether a storage engine is supported, or to see what the default engine is.

For information about MySQL storage engines, see Chapter 14, *The InnoDB Storage Engine*, and Chapter 15, *Alternative Storage Engines*.

```
mysql> SHOW ENGINES\G
*************************** 1. row ***************************
      Engine: FEDERATED
     Support: NO
     Comment: Federated MySQL storage engine
Transactions: NULL
          XA: NULL
  Savepoints: NULL
*************************** 2. row ***************************
      Engine: MRG_MYISAM
     Support: YES
     Comment: Collection of identical MyISAM tables
Transactions: NO
          XA: NO
  Savepoints: NO
```

```
*************************** 3. row ***************************
      Engine: MyISAM
     Support: YES
     Comment: MyISAM storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 4. row ***************************
      Engine: BLACKHOLE
     Support: YES
     Comment: /dev/null storage engine (anything you write to it disappears)
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 5. row ***************************
      Engine: CSV
     Support: YES
     Comment: CSV storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 6. row ***************************
      Engine: MEMORY
     Support: YES
     Comment: Hash based, stored in memory, useful for temporary tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 7. row ***************************
      Engine: ARCHIVE
     Support: YES
     Comment: Archive storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 8. row ***************************
      Engine: InnoDB
     Support: DEFAULT
     Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
          XA: YES
  Savepoints: YES
*************************** 9. row ***************************
      Engine: PERFORMANCE_SCHEMA
     Support: YES
     Comment: Performance Schema
Transactions: NO
          XA: NO
  Savepoints: NO
```

The output from `SHOW ENGINES` may vary according to the MySQL version used and other factors.

`SHOW ENGINES` output has these columns:

- `Engine`

  The name of the storage engine.

- `Support`

  The server's level of support for the storage engine, as shown in the following table.

Value	Meaning
`YES`	The engine is supported and is active
`DEFAULT`	Like `YES`, plus this is the default engine
`NO`	The engine is not supported
`DISABLED`	The engine is supported but has been disabled

A value of `NO` means that the server was compiled without support for the engine, so it cannot be enabled at runtime.

A value of `DISABLED` occurs either because the server was started with an option that disables the engine, or because not all options required to enable it were given. In the latter case, the error log should contain a reason indicating why the option is disabled. See Section 5.4.2, "The Error Log".

You might also see `DISABLED` for a storage engine if the server was compiled to support it, but was started with a `--skip-engine_name` option. For the `NDB` storage engine, `DISABLED` means the server was compiled with support for NDB Cluster, but was not started with the `--ndbcluster` option.

All MySQL servers support `MyISAM` tables. It is not possible to disable `MyISAM`.

- `Comment`

  A brief description of the storage engine.

- `Transactions`

  Whether the storage engine supports transactions.

- `XA`

  Whether the storage engine supports XA transactions.

- `Savepoints`

  Whether the storage engine supports savepoints.

Storage engine information is also available from the `INFORMATION_SCHEMA ENGINES` table. See Section 21.7, "The INFORMATION_SCHEMA ENGINES Table".

13.7.5.18 SHOW ERRORS Statement

```
SHOW ERRORS [LIMIT [offset,] row_count]
SHOW COUNT(*) ERRORS
```

`SHOW ERRORS` is a diagnostic statement that is similar to `SHOW WARNINGS`, except that it displays information only for errors, rather than for errors, warnings, and notes.

The `LIMIT` clause has the same syntax as for the `SELECT` statement. See Section 13.2.9, "SELECT Statement".

The `SHOW COUNT(*) ERRORS` statement displays the number of errors. You can also retrieve this number from the `error_count` variable:

```
SHOW COUNT(*) ERRORS;
SELECT @@error_count;
```

`SHOW ERRORS` and `error_count` apply only to errors, not warnings or notes. In other respects, they are similar to `SHOW WARNINGS` and `warning_count`. In particular, `SHOW ERRORS` cannot display information for more than `max_error_count` messages, and `error_count` can exceed the value of `max_error_count` if the number of errors exceeds `max_error_count`.

For more information, see Section 13.7.5.41, "SHOW WARNINGS Statement".

13.7.5.19 SHOW EVENTS Statement

```
SHOW EVENTS
    [{FROM | IN} schema_name]
```

```
    [LIKE 'pattern' | WHERE expr]
```

This statement displays information about Event Manager events, which are discussed in Section 20.4, “Using the Event Scheduler”. It requires the `EVENT` privilege for the database from which the events are to be shown.

In its simplest form, `SHOW EVENTS` lists all of the events in the current schema:

```
mysql> SELECT CURRENT_USER(), SCHEMA();
+----------------+----------+
| CURRENT_USER() | SCHEMA() |
+----------------+----------+
| jon@ghidora    | myschema |
+----------------+----------+
1 row in set (0.00 sec)

mysql> SHOW EVENTS\G
*************************** 1. row ***************************
                  Db: myschema
                Name: e_daily
             Definer: jon@ghidora
           Time zone: SYSTEM
                Type: RECURRING
          Execute at: NULL
      Interval value: 1
      Interval field: DAY
              Starts: 2018-08-08 11:06:34
                Ends: NULL
              Status: ENABLED
          Originator: 1
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

To see events for a specific schema, use the `FROM` clause. For example, to see events for the `test` schema, use the following statement:

```
SHOW EVENTS FROM test;
```

The `LIKE` clause, if present, indicates which event names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 21.35, “Extensions to SHOW Statements”.

`SHOW EVENTS` output has these columns:

- `Db`

  The name of the schema (database) to which the event belongs.

- `Name`

  The name of the event.

- `Definer`

  The account of the user who created the event, in `'user_name'@'host_name'` format.

- `Time zone`

  The event time zone, which is the time zone used for scheduling the event and that is in effect within the event as it executes. The default value is `SYSTEM`.

- `Type`

  The event repetition type, either `ONE TIME` (transient) or `RECURRING` (repeating).

- `Execute At`

For a one-time event, this is the `DATETIME` value specified in the `AT` clause of the `CREATE EVENT` statement used to create the event, or of the last `ALTER EVENT` statement that modified the event. The value shown in this column reflects the addition or subtraction of any `INTERVAL` value included in the event's `AT` clause. For example, if an event is created using `ON SCHEDULE AT CURRENT_TIMESTAMP + '1:6' DAY_HOUR`, and the event was created at 2018-02-09 14:05:30, the value shown in this column would be `'2018-02-10 20:05:30'`. If the event's timing is determined by an `EVERY` clause instead of an `AT` clause (that is, if the event is recurring), the value of this column is `NULL`.

- `Interval Value`

  For a recurring event, the number of intervals to wait between event executions. For a transient event, the value of this column is always `NULL`.

- `Interval Field`

  The time units used for the interval which a recurring event waits before repeating. For a transient event, the value of this column is always `NULL`.

- `Starts`

  The start date and time for a recurring event. This is displayed as a `DATETIME` value, and is `NULL` if no start date and time are defined for the event. For a transient event, this column is always `NULL`. For a recurring event whose definition includes a `STARTS` clause, this column contains the corresponding `DATETIME` value. As with the `Execute At` column, this value resolves any expressions used. If there is no `STARTS` clause affecting the timing of the event, this column is `NULL`

- `Ends`

  For a recurring event whose definition includes a `ENDS` clause, this column contains the corresponding `DATETIME` value. As with the `Execute At` column, this value resolves any expressions used. If there is no `ENDS` clause affecting the timing of the event, this column is `NULL`.

- `Status`

  The event status. One of `ENABLED`, `DISABLED`, or `SLAVESIDE_DISABLED`. `SLAVESIDE_DISABLED` indicates that the creation of the event occurred on another MySQL server acting as a replication source and replicated to the current MySQL server which is acting as a replica, but the event is not presently being executed on the replica. For more information, see Section 17.4.1.16, “Replication of Invoked Features”. information.

- `Originator`

  The server ID of the MySQL server on which the event was created; used in replication. This value may be updated by `ALTER EVENT` to the server ID of the server on which that statement occurs, if executed on a source server. The default value is 0.

- `character_set_client`

  The session value of the `character_set_client` system variable when the event was created.

- `collation_connection`

  The session value of the `collation_connection` system variable when the event was created.

- `Database Collation`

  The collation of the database with which the event is associated.

For more information about `SLAVESIDE_DISABLED` and the `Originator` column, see Section 17.4.1.16, “Replication of Invoked Features”.

Times displayed by `SHOW EVENTS` are given in the event time zone, as discussed in Section 20.4.4, “Event Metadata”.

Event information is also available from the `INFORMATION_SCHEMA EVENTS` table. See Section 21.8, “The INFORMATION_SCHEMA EVENTS Table”.

The event action statement is not shown in the output of `SHOW EVENTS`. Use `SHOW CREATE EVENT` or the `INFORMATION_SCHEMA EVENTS` table.

13.7.5.20 SHOW FUNCTION CODE Statement

```
SHOW FUNCTION CODE func_name
```

This statement is similar to `SHOW PROCEDURE CODE` but for stored functions. See Section 13.7.5.28, “SHOW PROCEDURE CODE Statement”.

13.7.5.21 SHOW FUNCTION STATUS Statement

```
SHOW FUNCTION STATUS
    [LIKE 'pattern' | WHERE expr]
```

This statement is similar to `SHOW PROCEDURE STATUS` but for stored functions. See Section 13.7.5.29, “SHOW PROCEDURE STATUS Statement”.

13.7.5.22 SHOW GRANTS Statement

```
SHOW GRANTS [FOR user]
```

This statement displays the privileges that are assigned to a MySQL user account, in the form of `GRANT` statements that must be executed to duplicate the privilege assignments.

`SHOW GRANTS` requires the `SELECT` privilege for the `mysql` system database, except to display privileges for the current user. For output that includes an `IDENTIFIED BY PASSWORD` clause displaying an account password hash value, the `SUPER` privilege is required to see the actual hash value. Otherwise, the value displays as `<secret>`.

To name the account for `SHOW GRANTS`, use the same format as for the `GRANT` statement (for example, `'jeffrey'@'localhost'`):

```
mysql> SHOW GRANTS FOR 'jeffrey'@'localhost';
+-----------------------------------------------------------------------+
| Grants for jeffrey@localhost                                          |
+-----------------------------------------------------------------------+
| GRANT USAGE ON *.* TO `jeffrey`@`localhost`                           |
| GRANT SELECT, INSERT, UPDATE ON `db1`.* TO `jeffrey`@`localhost` |
+-----------------------------------------------------------------------+
```

The host part, if omitted, defaults to `'%'`. For additional information about specifying account names, see Section 6.2.4, “Specifying Account Names”.

To display the privileges granted to the current user (the account you are using to connect to the server), you can use any of the following statements:

```
SHOW GRANTS;
SHOW GRANTS FOR CURRENT_USER;
SHOW GRANTS FOR CURRENT_USER();
```

If `SHOW GRANTS FOR CURRENT_USER` (or any equivalent syntax) is used in definer context, such as within a stored procedure that executes with definer rather than invoker privileges, the grants displayed are those of the definer and not the invoker.

`SHOW GRANTS` does not display privileges that are available to the named account but are granted to a different account. For example, if an anonymous account exists, the named account might be able to use its privileges, but `SHOW GRANTS` does not display them.

13.7.5.23 SHOW INDEX Statement

```
SHOW {INDEX | INDEXES | KEYS}
    {FROM | IN} tbl_name
    [{FROM | IN} db_name]
    [WHERE expr]
```

`SHOW INDEX` returns table index information. The format resembles that of the `SQLStatistics` call in ODBC. This statement requires some privilege for any column in the table.

```
mysql> SHOW INDEX FROM City\G
*************************** 1. row ***************************
        Table: city
   Non_unique: 0
     Key_name: PRIMARY
 Seq_in_index: 1
  Column_name: ID
    Collation: A
  Cardinality: 4188
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
*************************** 2. row ***************************
        Table: city
   Non_unique: 1
     Key_name: CountryCode
 Seq_in_index: 1
  Column_name: CountryCode
    Collation: A
  Cardinality: 4188
     Sub_part: NULL
       Packed: NULL
         Null:
   Index_type: BTREE
      Comment:
Index_comment:
```

An alternative to *`tbl_name` `FROM` `db_name`* syntax is *`db_name`.`tbl_name`*. These two statements are equivalent:

```
SHOW INDEX FROM mytable FROM mydb;
SHOW INDEX FROM mydb.mytable;
```

The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 21.35, “Extensions to SHOW Statements”.

`SHOW INDEX` returns the following fields:

- `Table`

  The name of the table.

- `Non_unique`

  0 if the index cannot contain duplicates, 1 if it can.

- `Key_name`

  The name of the index. If the index is the primary key, the name is always `PRIMARY`.

- `Seq_in_index`

  The column sequence number in the index, starting with 1.

- `Column_name`

The name of the column.

- `Collation`

  How the column is sorted in the index. This can have values `A` (ascending) or `NULL` (not sorted).

- `Cardinality`

  An estimate of the number of unique values in the index. To update this number, run `ANALYZE TABLE` or (for `MyISAM` tables) `myisamchk -a`.

  `Cardinality` is counted based on statistics stored as integers, so the value is not necessarily exact even for small tables. The higher the cardinality, the greater the chance that MySQL uses the index when doing joins.

- `Sub_part`

  The index prefix. That is, the number of indexed characters if the column is only partly indexed, `NULL` if the entire column is indexed.

  Note

  Prefix *limits* are measured in bytes. However, prefix *lengths* for index specifications in `CREATE TABLE`, `ALTER TABLE`, and `CREATE INDEX` statements are interpreted as number of characters for nonbinary string types (`CHAR`, `VARCHAR`, `TEXT`) and number of bytes for binary string types (`BINARY`, `VARBINARY`, `BLOB`). Take this into account when specifying a prefix length for a nonbinary string column that uses a multibyte character set.

  For additional information about index prefixes, see Section 8.3.4, “Column Indexes”, and Section 13.1.13, “CREATE INDEX Statement”.

- `Packed`

  Indicates how the key is packed. `NULL` if it is not.

- `Null`

  Contains `YES` if the column may contain `NULL` values and `''` if not.

- `Index_type`

  The index method used (`BTREE`, `FULLTEXT`, `HASH`, `RTREE`).

- `Comment`

  Information about the index not described in its own column, such as `disabled` if the index is disabled.

- `Index_comment`

  Any comment provided for the index with a `COMMENT` attribute when the index was created.

Information about table indexes is also available from the `INFORMATION_SCHEMA STATISTICS` table. See Section 21.22, “The INFORMATION_SCHEMA STATISTICS Table”.

You can list a table's indexes with the `mysqlshow -k db_name tbl_name` command.

13.7.5.24 SHOW MASTER STATUS Statement

```
SHOW MASTER STATUS
```

This statement provides status information about the binary log files of the source. It requires either the `SUPER` or `REPLICATION CLIENT` privilege.

Example:

```
mysql> SHOW MASTER STATUS\G
*************************** 1. row ***************************
             File: source-bin.000002
         Position: 1307
     Binlog_Do_DB: test
 Binlog_Ignore_DB: manual, mysql
Executed_Gtid_Set: 3E11FA47-71CA-11E1-9E33-C80AA9429562:1-5
1 row in set (0.00 sec)
```

When global transaction IDs are in use, this column shows the set of GTIDs for transactions that have been executed on the source. This is the same as the value for the `gtid_executed` system variable on this server, as well as the value for `Executed_Gtid_Set` in the output of `SHOW SLAVE STATUS` on this server.

13.7.5.25 SHOW OPEN TABLES Statement

```
SHOW OPEN TABLES
    [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

`SHOW OPEN TABLES` lists the non-`TEMPORARY` tables that are currently open in the table cache. See Section 8.4.3.1, “How MySQL Opens and Closes Tables”. The `FROM` clause, if present, restricts the tables shown to those present in the *`db_name`* database. The `LIKE` clause, if present, indicates which table names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 21.35, “Extensions to SHOW Statements”.

`SHOW OPEN TABLES` output has these columns:

- `Database`

  The database containing the table.

- `Table`

  The table name.

- `In_use`

  The number of table locks or lock requests there are for the table. For example, if one client acquires a lock for a table using `LOCK TABLE t1 WRITE`, `In_use` is 1. If another client issues `LOCK TABLE t1 WRITE` while the table remains locked, the client blocks waiting for the lock, but the lock request causes `In_use` to be 2. If the count is zero, the table is open but not currently being used. `In_use` is also increased by the `HANDLER ... OPEN` statement and decreased by `HANDLER ... CLOSE`.

- `Name_locked`

  Whether the table name is locked. Name locking is used for operations such as dropping or renaming tables.

If you have no privileges for a table, it does not show up in the output from `SHOW OPEN TABLES`.

13.7.5.26 SHOW PLUGINS Statement

```
SHOW PLUGINS
```

`SHOW PLUGINS` displays information about server plugins.

Example of `SHOW PLUGINS` output:

```
mysql> SHOW PLUGINS\G
*************************** 1. row ***************************
   Name: binlog
 Status: ACTIVE
   Type: STORAGE ENGINE
Library: NULL
License: GPL
*************************** 2. row ***************************
   Name: CSV
 Status: ACTIVE
   Type: STORAGE ENGINE
Library: NULL
License: GPL
*************************** 3. row ***************************
   Name: MEMORY
 Status: ACTIVE
   Type: STORAGE ENGINE
Library: NULL
License: GPL
*************************** 4. row ***************************
   Name: MyISAM
 Status: ACTIVE
   Type: STORAGE ENGINE
Library: NULL
License: GPL
...
```

`SHOW PLUGINS` output has these columns:

- `Name`

  The name used to refer to the plugin in statements such as `INSTALL PLUGIN` and `UNINSTALL PLUGIN`.

- `Status`

  The plugin status, one of `ACTIVE`, `INACTIVE`, `DISABLED`, or `DELETED`.

- `Type`

  The type of plugin, such as `STORAGE ENGINE`, `INFORMATION_SCHEMA`, or `AUTHENTICATION`.

- `Library`

  The name of the plugin shared library file. This is the name used to refer to the plugin file in statements such as `INSTALL PLUGIN` and `UNINSTALL PLUGIN`. This file is located in the directory named by the `plugin_dir` system variable. If the library name is `NULL`, the plugin is compiled in and cannot be uninstalled with `UNINSTALL PLUGIN`.

- `License`

  How the plugin is licensed (for example, `GPL`).

For plugins installed with `INSTALL PLUGIN`, the `Name` and `Library` values are also registered in the `mysql.plugin` system table.

For information about plugin data structures that form the basis of the information displayed by `SHOW PLUGINS`, see The MySQL Plugin API.

Plugin information is also available from the `INFORMATION_SCHEMA .PLUGINS` table. See Section 21.15, “The INFORMATION_SCHEMA PLUGINS Table”.

13.7.5.27 SHOW PRIVILEGES Statement

```
SHOW PRIVILEGES
```

`SHOW PRIVILEGES` shows the list of system privileges that the MySQL server supports. The exact list of privileges depends on the version of your server.

```
mysql> SHOW PRIVILEGES\G
*************************** 1. row ***************************
Privilege: Alter
  Context: Tables
  Comment: To alter the table
*************************** 2. row ***************************
Privilege: Alter routine
  Context: Functions,Procedures
  Comment: To alter or drop stored functions/procedures
*************************** 3. row ***************************
Privilege: Create
  Context: Databases,Tables,Indexes
  Comment: To create new databases and tables
*************************** 4. row ***************************
Privilege: Create routine
  Context: Databases
  Comment: To use CREATE FUNCTION/PROCEDURE
*************************** 5. row ***************************
Privilege: Create temporary tables
  Context: Databases
  Comment: To use CREATE TEMPORARY TABLE
...
```

Privileges belonging to a specific user are displayed by the `SHOW GRANTS` statement. See Section 13.7.5.22, “SHOW GRANTS Statement”, for more information.

13.7.5.28 SHOW PROCEDURE CODE Statement

```
SHOW PROCEDURE CODE proc_name
```

This statement is a MySQL extension that is available only for servers that have been built with debugging support. It displays a representation of the internal implementation of the named stored procedure. A similar statement, `SHOW FUNCTION CODE`, displays information about stored functions (see Section 13.7.5.20, “SHOW FUNCTION CODE Statement”).

To use either statement, you must be the owner of the routine or have `SELECT` access to the `mysql.proc` table.

If the named routine is available, each statement produces a result set. Each row in the result set corresponds to one “instruction” in the routine. The first column is `Pos`, which is an ordinal number beginning with 0. The second column is `Instruction`, which contains an SQL statement (usually changed from the original source), or a directive which has meaning only to the stored-routine handler.

```
mysql> DELIMITER //
mysql> CREATE PROCEDURE p1 ()
       BEGIN
         DECLARE fanta INT DEFAULT 55;
         DROP TABLE t2;
         LOOP
           INSERT INTO t3 VALUES (fanta);
           END LOOP;
         END//
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW PROCEDURE CODE p1//
+-----+--------------------------------------+
| Pos | Instruction                          |
+-----+--------------------------------------+
|   0 | set fanta@0 55                       |
|   1 | stmt 9 "DROP TABLE t2"               |
|   2 | stmt 5 "INSERT INTO t3 VALUES (fanta)" |
|   3 | jump 2                               |
+-----+--------------------------------------+
4 rows in set (0.00 sec)
```

```
mysql> CREATE FUNCTION test.hello (s CHAR(20))
       RETURNS CHAR(50) DETERMINISTIC
       RETURN CONCAT('Hello, ',s,'!');
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW FUNCTION CODE test.hello;
+-----+--------------------------------------+
| Pos | Instruction                          |
+-----+--------------------------------------+
|   0 | freturn 254 concat('Hello, ',s@0,'!') |
+-----+--------------------------------------+
1 row in set (0.00 sec)
```

In this example, the nonexecutable `BEGIN` and `END` statements have disappeared, and for the `DECLARE` *`variable_name`* statement, only the executable part appears (the part where the default is assigned). For each statement that is taken from source, there is a code word `stmt` followed by a type (9 means `DROP`, 5 means `INSERT`, and so on). The final row contains an instruction `jump 2`, meaning `GOTO instruction #2`.

13.7.5.29 SHOW PROCEDURE STATUS Statement

```
SHOW PROCEDURE STATUS
    [LIKE 'pattern' | WHERE expr]
```

This statement is a MySQL extension. It returns characteristics of a stored procedure, such as the database, name, type, creator, creation and modification dates, and character set information. A similar statement, `SHOW FUNCTION STATUS`, displays information about stored functions (see Section 13.7.5.21, “SHOW FUNCTION STATUS Statement”).

To use either statement, you must be the owner of the routine or have `SELECT` access to the `mysql.proc` table.

The `LIKE` clause, if present, indicates which procedure or function names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 21.35, “Extensions to SHOW Statements”.

```
mysql> SHOW PROCEDURE STATUS LIKE 'sp1'\G
*************************** 1. row ***************************
                  Db: test
                Name: sp1
                Type: PROCEDURE
             Definer: testuser@localhost
            Modified: 2018-08-08 13:54:11
             Created: 2018-08-08 13:54:11
       Security_type: DEFINER
             Comment:
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci

mysql> SHOW FUNCTION STATUS LIKE 'hello'\G
*************************** 1. row ***************************
                  Db: test
                Name: hello
                Type: FUNCTION
             Definer: testuser@localhost
            Modified: 2020-03-10 11:14:27
             Created: 2020-03-10 11:14:27
       Security_type: DEFINER
             Comment:
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

`character_set_client` is the session value of the `character_set_client` system variable when the routine was created. `collation_connection` is the session value of the

`collation_connection` system variable when the routine was created. `Database Collation` is the collation of the database with which the routine is associated.

Stored routine information is also available from the `INFORMATION_SCHEMA PARAMETERS` and `ROUTINES` tables. See Section 21.13, “The INFORMATION_SCHEMA PARAMETERS Table”, and Section 21.19, “The INFORMATION_SCHEMA ROUTINES Table”.

13.7.5.30 SHOW PROCESSLIST Statement

```
SHOW [FULL] PROCESSLIST
```

The MySQL process list indicates the operations currently being performed by the set of threads executing within the server. The `SHOW PROCESSLIST` statement is one source of process information. For a comparison of this statement with other sources, see Sources of Process Information.

If you have the `PROCESS` privilege, you can see all threads, even those belonging to other users. Otherwise (without the `PROCESS` privilege), nonanonymous users have access to information about their own threads but not threads for other users, and anonymous users have no access to thread information.

Without the `FULL` keyword, `SHOW PROCESSLIST` displays only the first 100 characters of each statement in the `Info` field.

The `SHOW PROCESSLIST` statement is very useful if you get the “too many connections” error message and want to find out what is going on. MySQL reserves one extra connection to be used by accounts that have the `SUPER` privilege, to ensure that administrators should always be able to connect and check the system (assuming that you are not giving this privilege to all your users).

Threads can be killed with the `KILL` statement. See Section 13.7.6.4, “KILL Statement”.

Example of `SHOW PROCESSLIST` output:

```
mysql> SHOW FULL PROCESSLIST\G
*************************** 1. row ***************************
     Id: 1
   User: system user
   Host:
     db: NULL
Command: Connect
   Time: 1030455
  State: Waiting for master to send event
   Info: NULL
*************************** 2. row ***************************
     Id: 2
   User: system user
   Host:
     db: NULL
Command: Connect
   Time: 1004
  State: Has read all relay log; waiting for the slave
         I/O thread to update it
   Info: NULL
*************************** 3. row ***************************
     Id: 3112
   User: replikator
   Host: artemis:2204
     db: NULL
Command: Binlog Dump
   Time: 2144
  State: Has sent all binlog to slave; waiting for binlog to be updated
   Info: NULL
*************************** 4. row ***************************
     Id: 3113
   User: replikator
   Host: iconnect2:45781
     db: NULL
```

```
Command: Binlog Dump
   Time: 2086
  State: Has sent all binlog to slave; waiting for binlog to be updated
   Info: NULL
*************************** 5. row ***************************
     Id: 3123
   User: stefan
   Host: localhost
     db: apollon
Command: Query
   Time: 0
  State: NULL
   Info: SHOW FULL PROCESSLIST
```

`SHOW PROCESSLIST` output has these columns:

- `Id`

  The connection identifier. This is the same value displayed in the `ID` column of the `INFORMATION_SCHEMA PROCESSLIST` table, displayed in the `PROCESSLIST_ID` column of the Performance Schema `threads` table, and returned by the `CONNECTION_ID()` function within the thread.

- `User`

  The MySQL user who issued the statement. A value of `system user` refers to a nonclient thread spawned by the server to handle tasks internally, for example, a delayed-row handler thread or an I/O or SQL thread used on replica hosts. For `system user`, there is no host specified in the `Host` column. `unauthenticated user` refers to a thread that has become associated with a client connection but for which authentication of the client user has not yet occurred. `event_scheduler` refers to the thread that monitors scheduled events (see Section 20.4, “Using the Event Scheduler”).

- `Host`

  The host name of the client issuing the statement (except for `system user`, for which there is no host). The host name for TCP/IP connections is reported in *`host_name:client_port`* format to make it easier to determine which client is doing what.

- `db`

  The default database for the thread, or `NULL` if none has been selected.

- `Command`

  The type of command the thread is executing on behalf of the client, or `Sleep` if the session is idle. For descriptions of thread commands, see Section 8.14, “Examining Server Thread (Process) Information”. The value of this column corresponds to the `COM_`*`xxx`* commands of the client/server protocol and `Com_`*`xxx`* status variables. See Section 5.1.9, “Server Status Variables”.

- `Time`

  The time in seconds that the thread has been in its current state. For a replica SQL thread, the value is the number of seconds between the timestamp of the last replicated event and the real time of the replica host. See Section 17.2.1, “Replication Threads”.

- `State`

  An action, event, or state that indicates what the thread is doing. For descriptions of `State` values, see Section 8.14, “Examining Server Thread (Process) Information”.

  Most states correspond to very quick operations. If a thread stays in a given state for many seconds, there might be a problem that needs to be investigated.

- `Info`

The statement the thread is executing, or `NULL` if it is executing no statement. The statement might be the one sent to the server, or an innermost statement if the statement executes other statements. For example, if a `CALL` statement executes a stored procedure that is executing a `SELECT` statement, the `Info` value shows the `SELECT` statement.

13.7.5.31 SHOW PROFILE Statement

```
SHOW PROFILE [type [, type] ... ]
    [FOR QUERY n]
    [LIMIT row_count [OFFSET offset]]

type: {
    ALL
  | BLOCK IO
  | CONTEXT SWITCHES
  | CPU
  | IPC
  | MEMORY
  | PAGE FAULTS
  | SOURCE
  | SWAPS
}
```

The `SHOW PROFILE` and `SHOW PROFILES` statements display profiling information that indicates resource usage for statements executed during the course of the current session.

Note

These statements are deprecated as of MySQL 5.6.7; expect them to be removed in a future MySQL release. Use the Performance Schema instead; see Section 22.18.1, “Query Profiling Using Performance Schema”.

To control profiling, use the `profiling` session variable, which has a default value of 0 (`OFF`). Enable profiling by setting `profiling` to 1 or `ON`:

```
mysql> SET profiling = 1;
```

`SHOW PROFILES` displays a list of the most recent statements sent to the server. The size of the list is controlled by the `profiling_history_size` session variable, which has a default value of 15. The maximum value is 100. Setting the value to 0 has the practical effect of disabling profiling.

All statements are profiled except `SHOW PROFILE` and `SHOW PROFILES`, so you find neither of those statements in the profile list. Malformed statements are profiled. For example, `SHOW PROFILING` is an illegal statement, and a syntax error occurs if you try to execute it, but it shows up in the profiling list.

`SHOW PROFILE` displays detailed information about a single statement. Without the `FOR QUERY n` clause, the output pertains to the most recently executed statement. If `FOR QUERY n` is included, `SHOW PROFILE` displays information for statement *`n`*. The values of *`n`* correspond to the `Query_ID` values displayed by `SHOW PROFILES`.

The `LIMIT row_count` clause may be given to limit the output to *`row_count`* rows. If `LIMIT` is given, `OFFSET offset` may be added to begin the output *`offset`* rows into the full set of rows.

By default, `SHOW PROFILE` displays `Status` and `Duration` columns. The `Status` values are like the `State` values displayed by `SHOW PROCESSLIST`, although there might be some minor differences in interpretion for the two statements for some status values (see Section 8.14, “Examining Server Thread (Process) Information”).

Optional *`type`* values may be specified to display specific additional types of information:

- `ALL` displays all information
- `BLOCK IO` displays counts for block input and output operations

- `CONTEXT SWITCHES` displays counts for voluntary and involuntary context switches
- `CPU` displays user and system CPU usage times
- `IPC` displays counts for messages sent and received
- `MEMORY` is not currently implemented
- `PAGE FAULTS` displays counts for major and minor page faults
- `SOURCE` displays the names of functions from the source code, together with the name and line number of the file in which the function occurs
- `SWAPS` displays swap counts

Profiling is enabled per session. When a session ends, its profiling information is lost.

```
mysql> SELECT @@profiling;
+-------------+
| @@profiling |
+-------------+
|           0 |
+-------------+
1 row in set (0.00 sec)

mysql> SET profiling = 1;
Query OK, 0 rows affected (0.00 sec)

mysql> DROP TABLE IF EXISTS t1;
Query OK, 0 rows affected, 1 warning (0.00 sec)

mysql> CREATE TABLE T1 (id INT);
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW PROFILES;
+----------+----------+--------------------------+
| Query_ID | Duration | Query                    |
+----------+----------+--------------------------+
|        0 | 0.000088 | SET PROFILING = 1        |
|        1 | 0.000136 | DROP TABLE IF EXISTS t1  |
|        2 | 0.011947 | CREATE TABLE t1 (id INT) |
+----------+----------+--------------------------+
3 rows in set (0.00 sec)

mysql> SHOW PROFILE;
+----------------------+----------+
| Status               | Duration |
+----------------------+----------+
| checking permissions | 0.000040 |
| creating table       | 0.000056 |
| After create         | 0.011363 |
| query end            | 0.000375 |
| freeing items        | 0.000089 |
| logging slow query   | 0.000019 |
| cleaning up          | 0.000005 |
+----------------------+----------+
7 rows in set (0.00 sec)

mysql> SHOW PROFILE FOR QUERY 1;
+--------------------+----------+
| Status             | Duration |
+--------------------+----------+
| query end          | 0.000107 |
| freeing items      | 0.000008 |
| logging slow query | 0.000015 |
| cleaning up        | 0.000006 |
+--------------------+----------+
4 rows in set (0.00 sec)

mysql> SHOW PROFILE CPU FOR QUERY 2;
```

```
+----------------------+----------+----------+------------+
| Status               | Duration | CPU_user | CPU_system |
+----------------------+----------+----------+------------+
| checking permissions | 0.000040 | 0.000038 |   0.000002 |
| creating table       | 0.000056 | 0.000028 |   0.000028 |
| After create         | 0.011363 | 0.000217 |   0.001571 |
| query end            | 0.000375 | 0.000013 |   0.000028 |
| freeing items        | 0.000089 | 0.000010 |   0.000014 |
| logging slow query   | 0.000019 | 0.000009 |   0.000010 |
| cleaning up          | 0.000005 | 0.000003 |   0.000002 |
+----------------------+----------+----------+------------+
7 rows in set (0.00 sec)
```

Note

Profiling is only partially functional on some architectures. For values that depend on the `getrusage()` system call, `NULL` is returned on systems such as Windows that do not support the call. In addition, profiling is per process and not per thread. This means that activity on threads within the server other than your own may affect the timing information that you see.

Profiling information is also available from the `INFORMATION_SCHEMA PROFILING` table. See Section 21.17, “The INFORMATION_SCHEMA PROFILING Table”. For example, the following queries are equivalent:

```
SHOW PROFILE FOR QUERY 2;

SELECT STATE, FORMAT(DURATION, 6) AS DURATION
FROM INFORMATION_SCHEMA.PROFILING
WHERE QUERY_ID = 2 ORDER BY SEQ;
```

13.7.5.32 SHOW PROFILES Statement

```
SHOW PROFILES
```

The `SHOW PROFILES` statement, together with `SHOW PROFILE`, displays profiling information that indicates resource usage for statements executed during the course of the current session. For more information, see Section 13.7.5.31, “SHOW PROFILE Statement”.

Note

These statements are deprecated as of MySQL 5.6.7; expect them to be removed in a future MySQL release. Use the Performance Schema instead; see Chapter 22, *MySQL Performance Schema*.

13.7.5.33 SHOW RELAYLOG EVENTS Statement

```
SHOW RELAYLOG EVENTS
    [IN 'log_name']
    [FROM pos]
    [LIMIT [offset,] row_count]
```

Shows the events in the relay log of a replica. If you do not specify `'log_name'`, the first relay log is displayed. This statement has no effect on the source.

The `LIMIT` clause has the same syntax as for the `SELECT` statement. See Section 13.2.9, “SELECT Statement”.

Note

Issuing a `SHOW RELAYLOG EVENTS` with no `LIMIT` clause could start a very time- and resource-consuming process because the server returns to the client the complete contents of the relay log (including all statements modifying data that have been received by the replica).

`SHOW RELAYLOG EVENTS` displays the following fields for each event in the relay log:

- `Log_name`

  The name of the file that is being listed.

- `Pos`

  The position at which the event occurs.

- `Event_type`

  An identifier that describes the event type.

- `Server_id`

  The server ID of the server on which the event originated.

- `End_log_pos`

  The value of `End_log_pos` for this event in the source's binary log.

- `Info`

  More detailed information about the event type. The format of this information depends on the event type.

Note

Some events relating to the setting of user and system variables are not included in the output from `SHOW RELAYLOG EVENTS`. To get complete coverage of events within a relay log, use `mysqlbinlog`.

13.7.5.34 SHOW SLAVE HOSTS Statement

```
SHOW SLAVE HOSTS
```

Displays a list of replicas currently registered with the source.

`SHOW SLAVE HOSTS` should be executed on a server that acts as a replication source. The statement displays information about servers that are or have been connected as replicas, with each row of the result corresponding to one replica server, as shown here:

```
mysql> SHOW SLAVE HOSTS;
+------------+-----------+------+-----------+--------------------------------------+
| Server_id  | Host      | Port | Master_id | Slave_UUID                           |
+------------+-----------+------+-----------+--------------------------------------+
|  192168010 | iconnect2 | 3306 | 192168011 | 14cb6624-7f93-11e0-b2c0-c80aa9429562 |
| 1921680101 | athena    | 3306 | 192168011 | 07af4990-f41f-11df-a566-7ac56fdaf645 |
+------------+-----------+------+-----------+--------------------------------------+
```

- `Server_id`: The unique server ID of the replica server, as configured in the replica server's option file, or on the command line with `--server-id=value`.

- `Host`: The host name of the replica server as specified on the replica with the `--report-host` option. This can differ from the machine name as configured in the operating system.

- `User`: The replica server user name as, specified on the replica with the `--report-user` option. Statement output includes this column only if the source server is started with the `--show-slave-auth-info` option.

- `Password`: The replica server password as, specified on the replica with the `--report-password` option. Statement output includes this column only if the source server is started with the `--show-slave-auth-info` option.

- `Port`: The port on the source to which the replica server is listening, as specified on the replica with the `--report-port` option.

  A zero in this column means that the replica port (`--report-port`) was not set.

- `Master_id`: The unique server ID of the source server that the replica server is replicating from. This is the server ID of the server on which `SHOW SLAVE HOSTS` is executed, so this same value is listed for each row in the result.

- `Slave_UUID`: The globally unique ID of this replica, as generated on the replica and found in the replica's `auto.cnf` file.

13.7.5.35 SHOW SLAVE STATUS Statement

```
SHOW SLAVE STATUS
```

This statement provides status information on essential parameters of the replica threads. It requires either the `SUPER` or `REPLICATION CLIENT` privilege.

If you issue this statement using the `mysql` client, you can use a `\G` statement terminator rather than a semicolon to obtain a more readable vertical layout:

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: localhost
                  Master_User: root
                  Master_Port: 13000
                Connect_Retry: 60
              Master_Log_File: source-bin.000002
          Read_Master_Log_Pos: 1307
               Relay_Log_File: replica-relay-bin.000003
                Relay_Log_Pos: 1508
        Relay_Master_Log_File: source-bin.000002
             Slave_IO_Running: Yes
            Slave_SQL_Running: Yes
              Replicate_Do_DB:
          Replicate_Ignore_DB:
           Replicate_Do_Table:
       Replicate_Ignore_Table:
      Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 1307
              Relay_Log_Space: 1858
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
        Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
                Last_IO_Errno: 0
                Last_IO_Error:
               Last_SQL_Errno: 0
               Last_SQL_Error:
  Replicate_Ignore_Server_Ids:
             Master_Server_Id: 1
                  Master_UUID: 3e11fa47-71ca-11e1-9e33-c80aa9429562
             Master_Info_File: /var/mysqld.2/data/master.info
                    SQL_Delay: 0
          SQL_Remaining_Delay: NULL
      Slave_SQL_Running_State: Reading event from the relay log
```

```
          Master_Retry_Count: 10
                 Master_Bind:
     Last_IO_Error_Timestamp:
    Last_SQL_Error_Timestamp:
               Master_SSL_Crl:
           Master_SSL_Crlpath:
           Retrieved_Gtid_Set: 3e11fa47-71ca-11e1-9e33-c80aa9429562:1-5
            Executed_Gtid_Set: 3e11fa47-71ca-11e1-9e33-c80aa9429562:1-5
                Auto_Position: 1
```

The following list describes the fields returned by `SHOW SLAVE STATUS`. For additional information about interpreting their meanings, see Section 8.14.7, “Replication Replica I/O Thread States”.

- `Slave_IO_State`

  A copy of the `State` field of the `SHOW PROCESSLIST` output for the replica I/O thread. This tells you what the thread is doing: trying to connect to the source, waiting for events from the source, reconnecting to the source, and so on. For a listing of possible states, see Section 8.14.7, “Replication Replica I/O Thread States”.

- `Master_Host`

  The source host that the replica is connected to.

- `Master_User`

  The user name of the account used to connect to the source.

- `Master_Port`

  The port used to connect to the source.

- `Connect_Retry`

  The number of seconds between connect retries (default 60). This can be set with the `CHANGE MASTER TO` statement.

- `Master_Log_File`

  The name of the source binary log file from which the I/O thread is currently reading.

- `Read_Master_Log_Pos`

  The position in the current source binary log file up to which the I/O thread has read.

- `Relay_Log_File`

  The name of the relay log file from which the SQL thread is currently reading and executing.

- `Relay_Log_Pos`

  The position in the current relay log file up to which the SQL thread has read and executed.

- `Relay_Master_Log_File`

  The name of the source binary log file containing the most recent event executed by the SQL thread.

- `Slave_IO_Running`

  Whether the I/O thread is started and has connected successfully to the source. Internally, the state of this thread is represented by one of the following three values:

  - **MYSQL_SLAVE_NOT_RUN.** The replica I/O thread is not running. For this state, `Slave_IO_Running` is `No`.

  - **MYSQL_SLAVE_RUN_NOT_CONNECT.** The replica I/O thread is running, but is not connected to a replication source. For this state, `Slave_IO_Running` is `Connecting`.
  - **MYSQL_SLAVE_RUN_CONNECT.** The replica I/O thread is running, and is connected to a replication source. For this state, `Slave_IO_Running` is `Yes`.

  The value of the `Slave_running` system status variable corresponds with this value.

- `Slave_SQL_Running`

  Whether the SQL thread is started.

- `Replicate_Do_DB`, `Replicate_Ignore_DB`

  The lists of databases that were specified with the `--replicate-do-db` and `--replicate-ignore-db` options, if any.

- `Replicate_Do_Table`, `Replicate_Ignore_Table`, `Replicate_Wild_Do_Table`, `Replicate_Wild_Ignore_Table`

  The lists of tables that were specified with the `--replicate-do-table`, `--replicate-ignore-table`, `--replicate-wild-do-table`, and `--replicate-wild-ignore-table` options, if any.

- `Last_Errno`, `Last_Error`

  These columns are aliases for `Last_SQL_Errno` and `Last_SQL_Error`.

  Issuing `RESET MASTER` or `RESET SLAVE` resets the values shown in these columns.

  Note

  When the replica SQL thread receives an error, it reports the error first, then stops the SQL thread. This means that there is a small window of time during which `SHOW SLAVE STATUS` shows a nonzero value for `Last_SQL_Errno` even though `Slave_SQL_Running` still displays `Yes`.

- `Skip_Counter`

  The current value of the `sql_slave_skip_counter` system variable. See Section 13.4.2.4, “SET GLOBAL sql_slave_skip_counter Statement”.

- `Exec_Master_Log_Pos`

  The position in the current source binary log file to which the SQL thread has read and executed, marking the start of the next transaction or event to be processed. You can use this value with the `CHANGE MASTER TO` statement's `MASTER_LOG_POS` option when starting a new replica from an existing replica, so that the new replica reads from this point. The coordinates given by (`Relay_Master_Log_File`, `Exec_Master_Log_Pos`) in the source's binary log correspond to the coordinates given by (`Relay_Log_File`, `Relay_Log_Pos`) in the relay log.

  When using a multithreaded replica (by setting `slave_parallel_workers` to a nonzero value), the value in this column actually represents a “low-water” mark, before which no uncommitted transactions remain. Because the current implementation allows execution of transactions on different databases in a different order on the replica than on the source, this is not necessarily the position of the most recently executed transaction.

- `Relay_Log_Space`

  The total combined size of all existing relay log files.

- `Until_Condition`, `Until_Log_File`, `Until_Log_Pos`

The values specified in the `UNTIL` clause of the `START SLAVE` statement.

`Until_Condition` has these values:

- `None` if no `UNTIL` clause was specified
- `Master` if the replica is reading until a given position in the source's binary log
- `Relay` if the replica is reading until a given position in its relay log
- `SQL_BEFORE_GTIDS` if the replica SQL thread is processing transactions until it has reached the first transaction whose GTID is listed in the `gtid_set`.
- `SQL_AFTER_GTIDS` if the replica threads are processing all transactions until the last transaction in the `gtid_set` has been processed by both threads.
- `SQL_AFTER_MTS_GAPS` if a multithreaded replica's SQL threads are running until no more gaps are found in the relay log.

`Until_Log_File` and `Until_Log_Pos` indicate the log file name and position that define the coordinates at which the SQL thread stops executing.

For more information on `UNTIL` clauses, see Section 13.4.2.5, “START SLAVE Statement”.

- `Master_SSL_Allowed`, `Master_SSL_CA_File`, `Master_SSL_CA_Path`, `Master_SSL_Cert`, `Master_SSL_Cipher`, `Master_SSL_CRL_File`, `Master_SSL_CRL_Path`, `Master_SSL_Key`, `Master_SSL_Verify_Server_Cert`

  These fields show the SSL parameters used by the replica to connect to the source, if any.

  `Master_SSL_Allowed` has these values:

  - `Yes` if an SSL connection to the source is permitted
  - `No` if an SSL connection to the source is not permitted
  - `Ignored` if an SSL connection is permitted but the replica server does not have SSL support enabled

  The values of the other SSL-related fields correspond to the values of the `MASTER_SSL_CA`, `MASTER_SSL_CAPATH`, `MASTER_SSL_CERT`, `MASTER_SSL_CIPHER`, `MASTER_SSL_CRL`, `MASTER_SSL_CRLPATH`, `MASTER_SSL_KEY`, and `MASTER_SSL_VERIFY_SERVER_CERT` options to the `CHANGE MASTER TO` statement. See Section 13.4.2.1, “CHANGE MASTER TO Statement”.

- `Seconds_Behind_Master`

  This field is an indication of how “late” the replica is:

  - When the replica is actively processing updates, this field shows the difference between the current timestamp on the replica and the original timestamp logged on the source for the event currently being processed on the replica.
  - When no event is currently being processed on the replica, this value is 0.

  In essence, this field measures the time difference in seconds between the replica SQL thread and the replica I/O thread. If the network connection between source and replica is fast, the replica I/O thread is very close to the source, so this field is a good approximation of how late the replica SQL thread is compared to the source. If the network is slow, this is *not* a good approximation; the replica SQL thread may quite often be caught up with the slow-reading replica I/O thread, so `Seconds_Behind_Master` often shows a value of 0, even if the I/O thread is late compared to the source. In other words, *this column is useful only for fast networks*.

This time difference computation works even if the source and replica do not have identical clock times, provided that the difference, computed when the replica I/O thread starts, remains constant from then on. Any changes—including NTP updates—can lead to clock skews that can make calculation of `Seconds_Behind_Master` less reliable.

In MySQL 5.6.9 and later, this field is `NULL` (undefined or unknown) if the replica SQL thread is not running, or if the SQL thread has consumed all of the relay log and the replica I/O thread is not running. Previously, this field was `NULL` if the replica SQL thread or the replica I/O thread was not running or was not connected to the source. (Bug #12946333) For example, if (prior to MySQL 5.6.9) the replica I/O thread was running but was not connected to the source and was sleeping for the number of seconds given by the `CHANGE MASTER TO` statement or `--master-connect-retry` option (default 60) before reconnecting, the value was `NULL`. Now in such cases, the connection to the source is not tested; instead, if the I/O thread is running but the relay log is exhausted, `Seconds_Behind_Master` is set to 0.

The value of `Seconds_Behind_Master` is based on the timestamps stored in events, which are preserved through replication. This means that if a source M1 is itself a replica of M0, any event from M1's binary log that originates from M0's binary log has M0's timestamp for that event. This enables MySQL to replicate `TIMESTAMP` successfully. However, the problem for `Seconds_Behind_Master` is that if M1 also receives direct updates from clients, the `Seconds_Behind_Master` value randomly fluctuates because sometimes the last event from M1 originates from M0 and sometimes is the result of a direct update on M1.

When using a multithreaded replica, you should keep in mind that this value is based on `Exec_Master_Log_Pos`, and so may not reflect the position of the most recently committed transaction.

- `Last_IO_Errno`, `Last_IO_Error`

  The error number and error message of the most recent error that caused the I/O thread to stop. An error number of 0 and message of the empty string mean “no error.” If the `Last_IO_Error` value is not empty, the error values also appear in the replica's error log.

  I/O error information includes a timestamp showing when the most recent I/O thread error occurred. This timestamp uses the format *`YYMMDD hh:mm:ss`*, and appears in the `Last_IO_Error_Timestamp` column.

  Issuing `RESET MASTER` or `RESET SLAVE` resets the values shown in these columns.

- `Last_SQL_Errno`, `Last_SQL_Error`

  The error number and error message of the most recent error that caused the SQL thread to stop. An error number of 0 and message of the empty string mean “no error.” If the `Last_SQL_Error` value is not empty, the error values also appear in the replica's error log.

  SQL error information includes a timestamp showing when the most recent SQL thread error occurred. This timestamp uses the format *`YYMMDD hh:mm:ss`*, and appears in the `Last_SQL_Error_Timestamp` column.

  Issuing `RESET MASTER` or `RESET SLAVE` resets the values shown in these columns.

- `Replicate_Ignore_Server_Ids`

  In MySQL 5.6, you set a replica to ignore events from 0 or more sources using the `IGNORE_SERVER_IDS` option of the `CHANGE MASTER TO` statement. By default this is blank, and is usually modified only when using a circular or other multi-source replication setup. The message shown for `Replicate_Ignore_Server_Ids` when not blank consists of a comma-delimited list of one or more numbers, indicating the server IDs to be ignored. For example:

  ```
  Replicate_Ignore_Server_Ids: 2, 6, 9
  ```

Note

`Ignored_server_ids` also shows the server IDs to be ignored, but is a space-delimited list, which is preceded by the total number of server IDs to be ignored. For example, if a `CHANGE MASTER TO` statement containing the `IGNORE_SERVER_IDS = (2,6,9)` option has been issued to tell a replica to ignore sources having the server ID 2, 6, or 9, that information appears as:

```
Ignored_server_ids: 3 2 6 9
```

where `3` is the total number of server IDs being ignored.

`Replicate_Ignore_Server_Ids` filtering is performed by the I/O thread, rather than by the SQL thread, which means that events which are filtered out are not written to the relay log. This differs from the filtering actions taken by server options such `--replicate-do-table`, which apply to the SQL thread.

- `Master_Server_Id`

  The `server_id` value from the source.

- `Master_UUID`

  The `server_uuid` value from the source.

- `Master_Info_File`

  The location of the `master.info` file.

- `SQL_Delay`

  The number of seconds that the replica must lag the source.

- `SQL_Remaining_Delay`

  When `Slave_SQL_Running_State` is `Waiting until MASTER_DELAY seconds after master executed event`, this field contains the number of delay seconds remaining. At other times, this field is `NULL`.

- `Slave_SQL_Running_State`

  The state of the SQL thread (analogous to `Slave_IO_State`). The value is identical to the `State` value of the SQL thread as displayed by `SHOW PROCESSLIST`; Section 8.14.8, “Replication Replica SQL Thread States”, provides a listing of possible states.

- `Master_Retry_Count`

  The number of times the replica can attempt to reconnect to the source in the event of a lost connection. This value can be set using the `MASTER_RETRY_COUNT` option of the `CHANGE MASTER TO` statement (preferred) or the older `--master-retry-count` server option (still supported for backward compatibility).

- `Master_Bind`

  The network interface that the replica is bound to, if any. This is set using the `MASTER_BIND` option for the `CHANGE MASTER TO` statement.

- `Last_IO_Error_Timestamp`

  A timestamp in *`YYMMDD hh:mm:ss`* format that shows when the most recent I/O error took place.

- `Last_SQL_Error_Timestamp`

A timestamp in *YYMMDD hh:mm:ss* format that shows when the most recent SQL error occurred.

- `Retrieved_Gtid_Set`

  The set of global transaction IDs corresponding to all transactions received by this replica. Empty if GTIDs are not in use.

  This is the set of all GTIDs that exist or have existed in the relay logs. Each GTID is added as soon as the `Gtid_log_event` is received. This can cause partially transmitted transactions to have their GTIDs included in the set.

  When all relay logs are lost due to executing `RESET SLAVE` or `CHANGE MASTER TO`, or due to the effects of the `--relay-log-recovery` option, the set is cleared. When `relay_log_purge = 1`, the newest relay log is always kept, and the set is not cleared.

- `Executed_Gtid_Set`

  The set of global transaction IDs written in the binary log. This is the same as the value for the global `gtid_executed` system variable on this server, as well as the value for `Executed_Gtid_Set` in the output of `SHOW MASTER STATUS` on this server. Empty if GTIDs are not in use. See GTID Sets for more information.

- `Auto_Position`

  1 if autopositioning is in use; otherwise 0.

13.7.5.36 SHOW STATUS Statement

```
SHOW [GLOBAL | SESSION] STATUS
    [LIKE 'pattern' | WHERE expr]
```

`SHOW STATUS` provides server status information (see Section 5.1.9, “Server Status Variables”). This statement does not require any privilege. It requires only the ability to connect to the server.

Status variable information is also available from these sources:

- The `GLOBAL_STATUS` and `SESSION_STATUS` tables. See Section 21.9, “The INFORMATION_SCHEMA GLOBAL_STATUS and SESSION_STATUS Tables”.
- The `mysqladmin extended-status` command. See Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”.

For `SHOW STATUS`, a `LIKE` clause, if present, indicates which variable names to match. A `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 21.35, “Extensions to SHOW Statements”.

`SHOW STATUS` accepts an optional `GLOBAL` or `SESSION` variable scope modifier:

- With a `GLOBAL` modifier, the statement displays the global status values. A global status variable may represent status for some aspect of the server itself (for example, `Aborted_connects`), or the aggregated status over all connections to MySQL (for example, `Bytes_received` and `Bytes_sent`). If a variable has no global value, the session value is displayed.
- With a `SESSION` modifier, the statement displays the status variable values for the current connection. If a variable has no session value, the global value is displayed. `LOCAL` is a synonym for `SESSION`.
- If no modifier is present, the default is `SESSION`.

The scope for each status variable is listed at Section 5.1.9, “Server Status Variables”.

Each invocation of the `SHOW STATUS` statement uses an internal temporary table and increments the global `Created_tmp_tables` value.

Partial output is shown here. The list of names and values may differ for your server. The meaning of each variable is given in Section 5.1.9, “Server Status Variables”.

```
mysql> SHOW STATUS;
+--------------------------+------------+
| Variable_name            | Value      |
+--------------------------+------------+
| Aborted_clients          | 0          |
| Aborted_connects         | 0          |
| Bytes_received           | 155372598  |
| Bytes_sent               | 1176560426 |
| Connections              | 30023      |
| Created_tmp_disk_tables  | 0          |
| Created_tmp_tables       | 8340       |
| Created_tmp_files        | 60         |
...
| Open_tables              | 1          |
| Open_files               | 2          |
| Open_streams             | 0          |
| Opened_tables            | 44600      |
| Questions                | 2026873    |
...
| Table_locks_immediate    | 1920382    |
| Table_locks_waited       | 0          |
| Threads_cached           | 0          |
| Threads_created          | 30022      |
| Threads_connected        | 1          |
| Threads_running          | 1          |
| Uptime                   | 80380      |
+--------------------------+------------+
```

With a `LIKE` clause, the statement displays only rows for those variables with names that match the pattern:

```
mysql> SHOW STATUS LIKE 'Key%';
+--------------------+----------+
| Variable_name      | Value    |
+--------------------+----------+
| Key_blocks_used    | 14955    |
| Key_read_requests  | 96854827 |
| Key_reads          | 162040   |
| Key_write_requests | 7589728  |
| Key_writes         | 3813196  |
+--------------------+----------+
```

13.7.5.37 SHOW TABLE STATUS Statement

```
SHOW TABLE STATUS
    [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

`SHOW TABLE STATUS` works likes `SHOW TABLES`, but provides a lot of information about each non-`TEMPORARY` table. You can also get this list using the `mysqlshow --status db_name` command. The `LIKE` clause, if present, indicates which table names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 21.35, “Extensions to SHOW Statements”.

This statement also displays information about views.

`SHOW TABLE STATUS` output has these columns:

- `Name`

  The name of the table.

- `Engine`

  The storage engine for the table. See Chapter 14, *The InnoDB Storage Engine*, and Chapter 15, *Alternative Storage Engines*.

  For partitioned tables, `Engine` shows the name of the storage engine used by all partitions.

- `Version`

  The version number of the table's `.frm` file.

- `Row_format`

  The row-storage format (`Fixed`, `Dynamic`, `Compressed`, `Redundant`, `Compact`). For `MyISAM` tables, `Dynamic` corresponds to what `myisamchk -dvv` reports as `Packed`. `InnoDB` table format is either `Redundant` or `Compact` when using the `Antelope` file format, or `Compressed` or `Dynamic` when using the `Barracuda` file format.

- `Rows`

  The number of rows. Some storage engines, such as `MyISAM`, store the exact count. For other storage engines, such as `InnoDB`, this value is an approximation, and may vary from the actual value by as much as 40% to 50%. In such cases, use `SELECT COUNT(*)` to obtain an accurate count.

  The `Rows` value is `NULL` for `INFORMATION_SCHEMA` tables.

  For `InnoDB` tables, the row count is only a rough estimate used in SQL optimization. (This is also true if the `InnoDB` table is partitioned.)

- `Avg_row_length`

  The average row length.

  Refer to the notes at the end of this section for related information.

- `Data_length`

  For `MyISAM`, `Data_length` is the length of the data file, in bytes.

  For `InnoDB`, `Data_length` is the approximate amount of space allocated for the clustered index, in bytes. Specifically, it is the clustered index size, in pages, multiplied by the `InnoDB` page size.

  Refer to the notes at the end of this section for information regarding other storage engines.

- `Max_data_length`

  For `MyISAM`, `Max_data_length` is maximum length of the data file. This is the total number of bytes of data that can be stored in the table, given the data pointer size used.

  Unused for `InnoDB`.

  Refer to the notes at the end of this section for information regarding other storage engines.

- `Index_length`

  For `MyISAM`, `Index_length` is the length of the index file, in bytes.

  For `InnoDB`, `Index_length` is the approximate amount of space allocated for non-clustered indexes, in bytes. Specifically, it is the sum of non-clustered index sizes, in pages, multiplied by the `InnoDB` page size.

  Refer to the notes at the end of this section for information regarding other storage engines.

- `Data_free`

  The number of allocated but unused bytes.

  `InnoDB` tables report the free space of the tablespace to which the table belongs. For a table located in the shared tablespace, this is the free space of the shared tablespace. If you are using multiple tablespaces and the table has its own tablespace, the free space is for only that table. Free space means the number of bytes in completely free extents minus a safety margin. Even if free space displays as 0, it may be possible to insert rows as long as new extents need not be allocated.

  For NDB Cluster, `Data_free` shows the space allocated on disk for, but not used by, a Disk Data table or fragment on disk. (In-memory data resource usage is reported by the `Data_length` column.)

  For partitioned tables, this value is only an estimate and may not be absolutely correct. A more accurate method of obtaining this information in such cases is to query the `INFORMATION_SCHEMA PARTITIONS` table, as shown in this example:

  ```
  SELECT SUM(DATA_FREE)
      FROM  INFORMATION_SCHEMA.PARTITIONS
      WHERE TABLE_SCHEMA = 'mydb'
      AND   TABLE_NAME   = 'mytable';
  ```

  For more information, see Section 21.14, “The INFORMATION_SCHEMA PARTITIONS Table”.

- `Auto_increment`

  The next `AUTO_INCREMENT` value.

- `Create_time`

  When the table was created.

  Prior to MySQL 5.6.25, for partitioned `InnoDB` tables, the `Create_time` column shows `NULL`. This column shows the correct table creation time for such tables in MySQL 5.6.25 and later. (Bug #17299181, Bug #69990)

- `Update_time`

  When the data file was last updated. For some storage engines, this value is `NULL`. For example, `InnoDB` stores multiple tables in its system tablespace and the data file timestamp does not apply. Even with file-per-table mode with each `InnoDB` table in a separate `.ibd` file, change buffering can delay the write to the data file, so the file modification time is different from the time of the last insert, update, or delete. For `MyISAM`, the data file timestamp is used; however, on Windows the timestamp is not updated by updates, so the value is inaccurate.

  For partitioned `InnoDB` tables, `Update_time` is always `NULL`.

- `Check_time`

  When the table was last checked. Not all storage engines update this time, in which case, the value is always `NULL`.

  For partitioned `InnoDB` tables, `Check_time` is always `NULL`.

- `Collation`

  The table default collation. The output does not explicitly list the table default character set, but the collation name begins with the character set name.

- `Checksum`

  The live checksum value, if any.

- `Create_options`

  Extra options used with `CREATE TABLE`.

  `Create_options` shows `partitioned` for partitioned tables.

  When creating a table with strict mode disabled, the storage engine's default row format is used if the specified row format is not supported. The actual row format of the table is reported in the `Row_format` column. `Create_options` shows the row format that was specified in the `CREATE TABLE` statement.

  When altering the storage engine of a table, table options that are not applicable to the new storage engine are retained in the table definition to enable reverting the table with its previously defined options to the original storage engine, if necessary. `Create_options` may show retained options.

- `Comment`

  The comment used when creating the table (or information as to why MySQL could not access the table information).

Notes

- For `InnoDB` tables, `SHOW TABLE STATUS` does not give accurate statistics except for the physical size reserved by the table. The row count is only a rough estimate used in SQL optimization.
- For `NDB` tables, the output of this statement shows appropriate values for the `Avg_row_length` and `Data_length` columns, with the exception that `BLOB` columns are not taken into account.
- For `NDB` tables, `Data_length` includes data stored in main memory only; the `Max_data_length` and `Data_free` columns apply to Disk Data.
- For NDB Cluster Disk Data tables, `Max_data_length` shows the space allocated for the disk part of a Disk Data table or fragment. (In-memory data resource usage is reported by the `Data_length` column.)
- For `MEMORY` tables, the `Data_length`, `Max_data_length`, and `Index_length` values approximate the actual amount of allocated memory. The allocation algorithm reserves memory in large amounts to reduce the number of allocation operations.
- For views, all columns displayed by `SHOW TABLE STATUS` are `NULL` except that `Name` indicates the view name and `Comment` says `VIEW`.

Table information is also available from the `INFORMATION_SCHEMA TABLES` table. See Section 21.23, “The INFORMATION_SCHEMA TABLES Table”.

13.7.5.38 SHOW TABLES Statement

```
SHOW [FULL] TABLES
    [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

`SHOW TABLES` lists the non-`TEMPORARY` tables in a given database. You can also get this list using the `mysqlshow db_name` command. The `LIKE` clause, if present, indicates which table names to match. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 21.35, “Extensions to SHOW Statements”.

Matching performed by the `LIKE` clause is dependent on the setting of the `lower_case_table_names` system variable.

This statement also lists any views in the database. The optional `FULL` modifier causes `SHOW TABLES` to display a second output column with values of `BASE TABLE` for a table, `VIEW` for a view, or `SYSTEM VIEW` for an `INFORMATION_SCHEMA` table.

If you have no privileges for a base table or view, it does not show up in the output from `SHOW TABLES` or `mysqlshow db_name`.

Table information is also available from the `INFORMATION_SCHEMA TABLES` table. See Section 21.23, "The INFORMATION_SCHEMA TABLES Table".

13.7.5.39 SHOW TRIGGERS Statement

```
SHOW TRIGGERS
    [{FROM | IN} db_name]
    [LIKE 'pattern' | WHERE expr]
```

`SHOW TRIGGERS` lists the triggers currently defined for tables in a database (the default database unless a `FROM` clause is given). This statement returns results only for databases and tables for which you have the `TRIGGER` privilege. The `LIKE` clause, if present, indicates which table names (not trigger names) to match and causes the statement to display triggers for those tables. The `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 21.35, "Extensions to SHOW Statements".

For the `ins_sum` trigger defined in Section 20.3, "Using Triggers", the output of `SHOW TRIGGERS` is as shown here:

```
mysql> SHOW TRIGGERS LIKE 'acc%'\G
*************************** 1. row ***************************
             Trigger: ins_sum
               Event: INSERT
               Table: account
           Statement: SET @sum = @sum + NEW.amount
              Timing: BEFORE
             Created: NULL
            sql_mode: NO_ENGINE_SUBSTITUTION
             Definer: me@localhost
character_set_client: utf8
collation_connection: utf8_general_ci
  Database Collation: latin1_swedish_ci
```

`SHOW TRIGGERS` output has these columns:

- `Trigger`

  The name of the trigger.

- `Event`

  The trigger event. This is the type of operation on the associated table for which the trigger activates. The value is `INSERT` (a row was inserted), `DELETE` (a row was deleted), or `UPDATE` (a row was modified).

- `Table`

  The table for which the trigger is defined.

- `Statement`

  The trigger body; that is, the statement executed when the trigger activates.

- `Timing`

  Whether the trigger activates before or after the triggering event. The value is `BEFORE` or `AFTER`.

- `Created`

  The value of this column is always `NULL`.

- `sql_mode`

The SQL mode in effect when the trigger was created, and under which the trigger executes. For the permitted values, see Section 5.1.10, “Server SQL Modes”.

- `Definer`

  The account of the user who created the trigger, in `'user_name'@'host_name'` format.

- `character_set_client`

  The session value of the `character_set_client` system variable when the trigger was created.

- `collation_connection`

  The session value of the `collation_connection` system variable when the trigger was created.

- `Database Collation`

  The collation of the database with which the trigger is associated.

Trigger information is also available from the `INFORMATION_SCHEMA TRIGGERS` table. See Section 21.27, “The INFORMATION_SCHEMA TRIGGERS Table”.

13.7.5.40 SHOW VARIABLES Statement

```
SHOW [GLOBAL | SESSION] VARIABLES
    [LIKE 'pattern' | WHERE expr]
```

`SHOW VARIABLES` shows the values of MySQL system variables (see Section 5.1.7, “Server System Variables”). This statement does not require any privilege. It requires only the ability to connect to the server.

System variable information is also available from these sources:

- The `GLOBAL_VARIABLES` and `SESSION_VARIABLES` tables. See Section 21.10, “The INFORMATION_SCHEMA GLOBAL_VARIABLES and SESSION_VARIABLES Tables”.
- The `mysqladmin variables` command. See Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”.

For `SHOW VARIABLES`, a `LIKE` clause, if present, indicates which variable names to match. A `WHERE` clause can be given to select rows using more general conditions, as discussed in Section 21.35, “Extensions to SHOW Statements”.

`SHOW VARIABLES` accepts an optional `GLOBAL` or `SESSION` variable scope modifier:

- With a `GLOBAL` modifier, the statement displays global system variable values. These are the values used to initialize the corresponding session variables for new connections to MySQL. If a variable has no global value, no value is displayed.
- With a `SESSION` modifier, the statement displays the system variable values that are in effect for the current connection. If a variable has no session value, the global value is displayed. `LOCAL` is a synonym for `SESSION`.
- If no modifier is present, the default is `SESSION`.

The scope for each system variable is listed at Section 5.1.7, “Server System Variables”.

`SHOW VARIABLES` is subject to a version-dependent display-width limit. For variables with very long values that are not completely displayed, use `SELECT` as a workaround. For example:

```
SELECT @@GLOBAL.innodb_data_file_path;
```

Most system variables can be set at server startup (read-only variables such as `version_comment` are exceptions). Many can be changed at runtime with the `SET` statement. See Section 5.1.8, “Using System Variables”, and Section 13.7.4.1, “SET Syntax for Variable Assignment”.

Partial output is shown here. The list of names and values may differ for your server. Section 5.1.7, “Server System Variables”, describes the meaning of each variable, and Section 5.1.1, “Configuring the Server”, provides information about tuning them.

```
mysql> SHOW VARIABLES;
+---------------------------------------+----------------------------+
| Variable_name                         | Value                      |
+---------------------------------------+----------------------------+
| auto_increment_increment              | 1                          |
| auto_increment_offset                 | 1                          |
| autocommit                            | ON                         |
| automatic_sp_privileges               | ON                         |
| back_log                              | 50                         |
| basedir                               | /home/jon/bin/mysql-5.5    |
| big_tables                            | OFF                        |
| binlog_cache_size                     | 32768                      |
| binlog_direct_non_transactional_updates | OFF                        |
| binlog_format                         | STATEMENT                  |
| binlog_stmt_cache_size                | 32768                      |
| bulk_insert_buffer_size               | 8388608                    |
...
| max_allowed_packet                    | 4194304                    |
| max_binlog_cache_size                 | 18446744073709547520       |
| max_binlog_size                       | 1073741824                 |
| max_binlog_stmt_cache_size            | 18446744073709547520       |
| max_connect_errors                    | 100                        |
| max_connections                       | 151                        |
| max_delayed_threads                   | 20                         |
| max_error_count                       | 64                         |
| max_heap_table_size                   | 16777216                   |
| max_insert_delayed_threads            | 20                         |
| max_join_size                         | 18446744073709551615       |
...

| thread_handling                       | one-thread-per-connection  |
| thread_stack                          | 262144                     |
| time_format                           | %H:%i:%s                   |
| time_zone                             | SYSTEM                     |
| timed_mutexes                         | OFF                        |
| timestamp                             | 1316689732                 |
| tmp_table_size                        | 16777216                   |
| tmpdir                                | /tmp                       |
| transaction_alloc_block_size          | 8192                       |
| transaction_prealloc_size             | 4096                       |
| tx_isolation                          | REPEATABLE-READ            |
| unique_checks                         | ON                         |
| updatable_views_with_limit            | YES                        |
| version                               | 5.5.17-log                 |
| version_comment                       | Source distribution        |
| version_compile_machine               | x86_64                     |
| version_compile_os                    | Linux                      |
| wait_timeout                          | 28800                      |
| warning_count                         | 0                          |
+---------------------------------------+----------------------------+
```

With a `LIKE` clause, the statement displays only rows for those variables with names that match the pattern. To obtain the row for a specific variable, use a `LIKE` clause as shown:

```
SHOW VARIABLES LIKE 'max_join_size';
SHOW SESSION VARIABLES LIKE 'max_join_size';
```

To get a list of variables whose name match a pattern, use the `%` wildcard character in a `LIKE` clause:

```
SHOW VARIABLES LIKE '%size%';
```

```
SHOW GLOBAL VARIABLES LIKE '%size%';
```

Wildcard characters can be used in any position within the pattern to be matched. Strictly speaking, because `_` is a wildcard that matches any single character, you should escape it as `\_` to match it literally. In practice, this is rarely necessary.

13.7.5.41 SHOW WARNINGS Statement

```
SHOW WARNINGS [LIMIT [offset,] row_count]
SHOW COUNT(*) WARNINGS
```

`SHOW WARNINGS` is a diagnostic statement that displays information about the conditions (errors, warnings, and notes) resulting from executing a statement in the current session. Warnings are generated for DML statements such as `INSERT`, `UPDATE`, and `LOAD DATA` as well as DDL statements such as `CREATE TABLE` and `ALTER TABLE`.

The `LIMIT` clause has the same syntax as for the `SELECT` statement. See Section 13.2.9, “SELECT Statement”.

`SHOW WARNINGS` is also used following `EXPLAIN EXTENDED`, to display the extra information generated by `EXPLAIN` when the `EXTENDED` keyword is used. See Section 8.8.3, “Extended EXPLAIN Output Format”.

`SHOW WARNINGS` displays information about the conditions resulting from the most recent statement in the current session that generated messages. It shows nothing if the most recent statement used a table and generated no messages. (That is, statements that use a table but generate no messages clear the message list.) Statements that do not use tables and do not generate messages have no effect on the message list.

The `SHOW COUNT(*) WARNINGS` diagnostic statement displays the total number of errors, warnings, and notes. You can also retrieve this number from the `warning_count` system variable:

```
SHOW COUNT(*) WARNINGS;
SELECT @@warning_count;
```

A related diagnostic statement, `SHOW ERRORS`, shows only error conditions (it excludes warnings and notes), and `SHOW COUNT(*) ERRORS` statement displays the total number of errors. See Section 13.7.5.18, “SHOW ERRORS Statement”. `GET DIAGNOSTICS` can be used to examine information for individual conditions. See Section 13.6.7.3, “GET DIAGNOSTICS Statement”.

Here is a simple example that shows data-conversion warnings for `INSERT`. The example assumes that strict SQL mode is disabled. With strict mode enabled, the warnings would become errors and terminate the `INSERT`.

```
mysql> CREATE TABLE t1 (a TINYINT NOT NULL, b CHAR(4));
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO t1 VALUES(10,'mysql'), (NULL,'test'), (300,'xyz');
Query OK, 3 rows affected, 3 warnings (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 3

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1265
Message: Data truncated for column 'b' at row 1
*************************** 2. row ***************************
  Level: Warning
   Code: 1048
Message: Column 'a' cannot be null
*************************** 3. row ***************************
  Level: Warning
   Code: 1264
Message: Out of range value for column 'a' at row 3
```

```
3 rows in set (0.00 sec)
```

The `max_error_count` system variable controls the maximum number of error, warning, and note messages for which the server stores information, and thus the number of messages that `SHOW WARNINGS` displays. To change the number of messages the server can store, change the value of `max_error_count`. The default is 64.

`max_error_count` controls only how many messages are stored, not how many are counted. The value of `warning_count` is not limited by `max_error_count`, even if the number of messages generated exceeds `max_error_count`. The following example demonstrates this. The `ALTER TABLE` statement produces three warning messages (strict SQL mode is disabled for the example to prevent an error from occuring after a single conversion issue). Only one message is stored and displayed because `max_error_count` has been set to 1, but all three are counted (as shown by the value of `warning_count`):

```
mysql> SHOW VARIABLES LIKE 'max_error_count';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| max_error_count | 64    |
+-----------------+-------+
1 row in set (0.00 sec)

mysql> SET max_error_count=1, sql_mode = '';
Query OK, 0 rows affected (0.00 sec)

mysql> ALTER TABLE t1 MODIFY b CHAR;
Query OK, 3 rows affected, 3 warnings (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 3

mysql> SHOW WARNINGS;
+---------+------+----------------------------------------+
| Level   | Code | Message                                |
+---------+------+----------------------------------------+
| Warning | 1263 | Data truncated for column 'b' at row 1 |
+---------+------+----------------------------------------+
1 row in set (0.00 sec)

mysql> SELECT @@warning_count;
+-----------------+
| @@warning_count |
+-----------------+
|               3 |
+-----------------+
1 row in set (0.01 sec)
```

To disable message storage, set `max_error_count` to 0. In this case, `warning_count` still indicates how many warnings occurred, but messages are not stored and cannot be displayed.

The `sql_notes` system variable controls whether note messages increment `warning_count` and whether the server stores them. By default, `sql_notes` is 1, but if set to 0, notes do not increment `warning_count` and the server does not store them:

```
mysql> SET sql_notes = 1;
mysql> DROP TABLE IF EXISTS test.no_such_table;
Query OK, 0 rows affected, 1 warning (0.00 sec)
mysql> SHOW WARNINGS;
+-------+------+-----------------------------------+
| Level | Code | Message                           |
+-------+------+-----------------------------------+
| Note  | 1051 | Unknown table 'test.no_such_table' |
+-------+------+-----------------------------------+
1 row in set (0.00 sec)

mysql> SET sql_notes = 0;
mysql> DROP TABLE IF EXISTS test.no_such_table;
Query OK, 0 rows affected (0.00 sec)
mysql> SHOW WARNINGS;
Empty set (0.00 sec)
```

The MySQL server sends to each client a count indicating the total number of errors, warnings, and notes resulting from the most recent statement executed by that client. From the C API, this value can be obtained by calling `mysql_warning_count()`. See mysql_warning_count().

In the `mysql` client, you can enable and disable automatic warnings display using the `warnings` and `nowarning` commands, respectively, or their shortcuts, `\W` and `\w` (see Section 4.5.1.2, "mysql Client Commands"). For example:

```
mysql> \W
Show warnings enabled.
mysql> SELECT 1/0;
+------+
| 1/0  |
+------+
| NULL |
+------+
1 row in set, 1 warning (0.03 sec)

Warning (Code 1365): Division by 0
mysql> \w
Show warnings disabled.
```

13.7.6 Other Administrative Statements

13.7.6.1 BINLOG Statement

```
BINLOG 'str'
```

`BINLOG` is an internal-use statement. It is generated by the `mysqlbinlog` program as the printable representation of certain events in binary log files. (See Section 4.6.8, "`mysqlbinlog` — Utility for Processing Binary Log Files".) The *`'str'`* value is a base 64-encoded string the that server decodes to determine the data change indicated by the corresponding event. This statement requires the `SUPER` privilege.

This statement can execute only format description events and row events.

13.7.6.2 CACHE INDEX Statement

```
CACHE INDEX {
      tbl_index_list [, tbl_index_list] ...
    | tbl_name PARTITION (partition_list)
  }
  IN key_cache_name

tbl_index_list:
  tbl_name [{INDEX|KEY} (index_name[, index_name] ...)]

partition_list: {
    partition_name[, partition_name] ...
  | ALL
}
```

The `CACHE INDEX` statement assigns table indexes to a specific key cache. It applies only to `MyISAM` tables, including partitioned `MyISAM` tables. After the indexes have been assigned, they can be preloaded into the cache if desired with `LOAD INDEX INTO CACHE`.

The following statement assigns indexes from the tables `t1`, `t2`, and `t3` to the key cache named `hot_cache`:

```
mysql> CACHE INDEX t1, t2, t3 IN hot_cache;
+---------+--------------------+----------+----------+
| Table   | Op                 | Msg_type | Msg_text |
+---------+--------------------+----------+----------+
| test.t1 | assign_to_keycache | status   | OK       |
| test.t2 | assign_to_keycache | status   | OK       |
| test.t3 | assign_to_keycache | status   | OK       |
+---------+--------------------+----------+----------+
```

The syntax of `CACHE INDEX` enables you to specify that only particular indexes from a table should be assigned to the cache. However, the implementation assigns all the table's indexes to the cache, so there is no reason to specify anything other than the table name.

The key cache referred to in a `CACHE INDEX` statement can be created by setting its size with a parameter setting statement or in the server parameter settings. For example:

```
SET GLOBAL keycache1.key_buffer_size=128*1024;
```

Key cache parameters are accessed as members of a structured system variable. See Section 5.1.8.3, “Structured System Variables”.

A key cache must exist before you assign indexes to it, or an error occurs:

```
mysql> CACHE INDEX t1 IN non_existent_cache;
ERROR 1284 (HY000): Unknown key cache 'non_existent_cache'
```

By default, table indexes are assigned to the main (default) key cache created at the server startup. When a key cache is destroyed, all indexes assigned to it are reassigned to the default key cache.

Index assignment affects the server globally: If one client assigns an index to a given cache, this cache is used for all queries involving the index, no matter which client issues the queries.

`CACHE INDEX` is supported for partitioned `MyISAM` tables. You can assign one or more indexes for one, several, or all partitions to a given key cache. For example, you can do the following:

```
CREATE TABLE pt (c1 INT, c2 VARCHAR(50), INDEX i(c1))
    ENGINE=MyISAM
    PARTITION BY HASH(c1)
    PARTITIONS 4;

SET GLOBAL kc_fast.key_buffer_size = 128 * 1024;
SET GLOBAL kc_slow.key_buffer_size = 128 * 1024;

CACHE INDEX pt PARTITION (p0) IN kc_fast;
CACHE INDEX pt PARTITION (p1, p3) IN kc_slow;
```

The previous set of statements performs the following actions:

- Creates a partitioned table with 4 partitions; these partitions are automatically named `p0`, ..., `p3`; this table has an index named `i` on column `c1`.
- Creates 2 key caches named `kc_fast` and `kc_slow`
- Assigns the index for partition `p0` to the `kc_fast` key cache and the index for partitions `p1` and `p3` to the `kc_slow` key cache; the index for the remaining partition (`p2`) uses the server's default key cache.

If you wish instead to assign the indexes for all partitions in table `pt` to a single key cache named `kc_all`, you can use either of the following two statements:

```
CACHE INDEX pt PARTITION (ALL) IN kc_all;

CACHE INDEX pt IN kc_all;
```

The two statements just shown are equivalent, and issuing either one has exactly the same effect. In other words, if you wish to assign indexes for all partitions of a partitioned table to the same key cache, the `PARTITION (ALL)` clause is optional.

When assigning indexes for multiple partitions to a key cache, the partitions need not be contiguous, and you need not list their names in any particular order. Indexes for any partitions not explicitly assigned to a key cache automatically use the server default key cache.

Index preloading is also supported for partitioned `MyISAM` tables. For more information, see Section 13.7.6.5, “LOAD INDEX INTO CACHE Statement”.

In MySQL 5.6.11 only, `gtid_next` must be set to `AUTOMATIC` before issuing this statement. (Bug #16062608, Bug #16715809, Bug #69045)

13.7.6.3 FLUSH Statement

```
FLUSH [NO_WRITE_TO_BINLOG | LOCAL] {
    flush_option [, flush_option] ...
  | tables_option
}

flush_option: {
    BINARY LOGS
  | DES_KEY_FILE
  | ENGINE LOGS
  | ERROR LOGS
  | GENERAL LOGS
  | HOSTS
  | LOGS
  | PRIVILEGES
  | QUERY CACHE
  | RELAY LOGS
  | SLOW LOGS
  | STATUS
  | USER_RESOURCES
}

tables_option: {
    TABLES
  | TABLES tbl_name [, tbl_name] ...
  | TABLES WITH READ LOCK
  | TABLES tbl_name [, tbl_name] ... WITH READ LOCK
  | TABLES tbl_name [, tbl_name] ... FOR EXPORT
}
```

The `FLUSH` statement has several variant forms that clear or reload various internal caches, flush tables, or acquire locks. To execute `FLUSH`, you must have the `RELOAD` privilege. Specific flush options might require additional privileges, as indicated in the option descriptions.

Note

It is not possible to issue `FLUSH` statements within stored functions or triggers. However, you may use `FLUSH` in stored procedures, so long as these are not called from stored functions or triggers. See Section 20.8, “Restrictions on Stored Programs”.

By default, the server writes `FLUSH` statements to the binary log so that they replicate to replicas. To suppress logging, specify the optional `NO_WRITE_TO_BINLOG` keyword or its alias `LOCAL`.

Note

`FLUSH LOGS`, `FLUSH BINARY LOGS`, `FLUSH TABLES WITH READ LOCK` (with or without a table list), and `FLUSH TABLES tbl_name ... FOR EXPORT` are not written to the binary log in any case because they would cause problems if replicated to a replica.

The `FLUSH` statement causes an implicit commit. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

The `mysqladmin` utility provides a command-line interface to some flush operations, using commands such as `flush-hosts`, `flush-logs`, `flush-privileges`, `flush-status`, and `flush-tables`. See Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”.

Sending a `SIGHUP` signal to the server causes several flush operations to occur that are similar to various forms of the `FLUSH` statement. Signals can be sent by the `root` system account or the system account that owns the server process. This enables the flush operations to be performed without

having to connect to the server, which requires a MySQL account that has privileges sufficient for those operations. See Section 4.10, “Unix Signal Handling in MySQL”.

The `RESET` statement is similar to `FLUSH`. See Section 13.7.6.6, “RESET Statement”, for information about using `RESET` with replication.

The following list describes the permitted `FLUSH` statement *`flush_option`* values. For descriptions of the permitted *`tables_option`* values, see FLUSH TABLES Syntax.

- `FLUSH BINARY LOGS`

  Closes and reopens any binary log file to which the server is writing. If binary logging is enabled, the sequence number of the binary log file is incremented by one relative to the previous file.

  This operation has no effect on tables used for the binary and relay logs (as controlled by the `master_info_repository` and `relay_log_info_repository` system variables).

- `FLUSH DES_KEY_FILE`

  Reloads the DES keys from the file that was specified with the `--des-key-file` option at server startup time.

- `FLUSH ENGINE LOGS`

  Closes and reopens any flushable logs for installed storage engines. This causes `InnoDB` to flush its logs to disk.

- `FLUSH ERROR LOGS`

  Closes and reopens any error log file to which the server is writing.

- `FLUSH GENERAL LOGS`

  Closes and reopens any general query log file to which the server is writing.

  This operation has no effect on tables used for the general query log (see Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”).

- `FLUSH HOSTS`

  Empties the host cache and the Performance Schema `host_cache` table that exposes the cache contents, and unblocks any blocked hosts.

  For information about why host cache flushing might be advisable or desirable, see Section 5.1.11.2, “DNS Lookups and the Host Cache”.

- `FLUSH LOGS`

  Closes and reopens any log file to which the server is writing.

  The effect of this operation is equivalent to the combined effects of these operations:

  ```
  FLUSH BINARY LOGS
  FLUSH ENGINE LOGS
  FLUSH ERROR LOGS
  FLUSH GENERAL LOGS
  FLUSH RELAY LOGS
  FLUSH SLOW LOGS
  ```

- `FLUSH PRIVILEGES`

  Rereads the privileges from the grant tables in the `mysql` system database.

  If the `--skip-grant-tables` option was specified at server startup to disable the MySQL privilege system, `FLUSH PRIVILEGES` provides a way to enable the privilege system at runtime.

Frees memory cached by the server as a result of `GRANT`, `CREATE USER`, `CREATE SERVER`, and `INSTALL PLUGIN` statements. This memory is not released by the corresponding `REVOKE`, `DROP USER`, `DROP SERVER`, and `UNINSTALL PLUGIN` statements, so for a server that executes many instances of the statements that cause caching, there is likely to be an increase in cached memory use unless it is freed with `FLUSH PRIVILEGES`.

- `FLUSH QUERY CACHE`

  Defragment the query cache to better utilize its memory. `FLUSH QUERY CACHE` does not remove any queries from the cache, unlike `FLUSH TABLES` or `RESET QUERY CACHE`.

- `FLUSH RELAY LOGS`

  Closes and reopens any relay log file to which the server is writing. If relay logging is enabled, the sequence number of the relay log file is incremented by one relative to the previous file.

  This operation has no effect on tables used for the binary and relay logs (as controlled by the `master_info_repository` and `relay_log_info_repository` system variables).

- `FLUSH SLOW LOGS`

  Closes and reopens any slow query log file to which the server is writing.

  This operation has no effect on tables used for the slow query log (see Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”).

- `FLUSH STATUS`

  Flushes status indicators.

  This operation adds the current thread's session status variable values to the global values and resets the session values to zero. Some global variables may be reset to zero as well. It also resets the counters for key caches (default and named) to zero and sets `Max_used_connections` to the current number of open connections. This information may be of use when debugging a query. See Section 1.6, “How to Report Bugs or Problems”.

- `FLUSH USER_RESOURCES`

  Resets all per-hour user resource indicators to zero.

  Resetting resource indicators enables clients that have reached their hourly connection, query, or update limits to resume activity immediately. `FLUSH USER_RESOURCES` does not apply to the limit on maximum simultaneous connections that is controlled by the `max_user_connections` system variable. See Section 6.2.13, “Setting Account Resource Limits”.

In MySQL 5.6.11 only, `gtid_next` must be set to `AUTOMATIC` before issuing this statement. (Bug #16062608, Bug #16715809, Bug #69045)

FLUSH TABLES Syntax

`FLUSH TABLES` flushes tables, and, depending on the variant used, acquires locks. Any `TABLES` variant used in a `FLUSH` statement must be the only option used. `FLUSH TABLE` is a synonym for `FLUSH TABLES`.

Note

The descriptions here that indicate tables are flushed by closing them apply differently for `InnoDB`, which flushes table contents to disk but leaves them open. This still permits table files to be copied while the tables are open, as long as other activity does not modify them.

- `FLUSH TABLES`

Closes all open tables, forces all tables in use to be closed, and flushes the query cache and prepared statement cache. `FLUSH TABLES` also removes all query results from the query cache, like the `RESET QUERY CACHE` statement. For information about query caching and prepared statement caching, see Section 8.10.3, “The MySQL Query Cache”. and Section 8.10.4, “Caching of Prepared Statements and Stored Programs”.

`FLUSH TABLES` is not permitted when there is an active `LOCK TABLES ... READ`. To flush and lock tables, use `FLUSH TABLES *tbl_name* ... WITH READ LOCK` instead.

- `FLUSH TABLES *tbl_name* [, *tbl_name*] ...`

  With a list of one or more comma-separated table names, this operation is like `FLUSH TABLES` with no names except that the server flushes only the named tables. If a named table does not exist, no error occurs.

- `FLUSH TABLES WITH READ LOCK`

  Closes all open tables and locks all tables for all databases with a global read lock.

  This operation is a very convenient way to get backups if you have a file system such as Veritas or ZFS that can take snapshots in time. Use `UNLOCK TABLES` to release the lock.

  `FLUSH TABLES WITH READ LOCK` acquires a global read lock rather than table locks, so it is not subject to the same behavior as `LOCK TABLES` and `UNLOCK TABLES` with respect to table locking and implicit commits:

  - `UNLOCK TABLES` implicitly commits any active transaction only if any tables currently have been locked with `LOCK TABLES`. The commit does not occur for `UNLOCK TABLES` following `FLUSH TABLES WITH READ LOCK` because the latter statement does not acquire table locks.
  - Beginning a transaction causes table locks acquired with `LOCK TABLES` to be released, as though you had executed `UNLOCK TABLES`. Beginning a transaction does not release a global read lock acquired with `FLUSH TABLES WITH READ LOCK`.

  `FLUSH TABLES WITH READ LOCK` is not compatible with XA transactions.

  `FLUSH TABLES WITH READ LOCK` does not prevent the server from inserting rows into the log tables (see Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”).

- `FLUSH TABLES *tbl_name* [, *tbl_name*] ... WITH READ LOCK`

  Flushes and acquires read locks for the named tables.

  Because this operation acquires table locks, it requires the `LOCK TABLES` privilege for each table, in addition to the `RELOAD` privilege.

  The operation first acquires exclusive metadata locks for the tables, so it waits for transactions that have those tables open to complete. Then the operation flushes the tables from the table cache, reopens the tables, acquires table locks (like `LOCK TABLES ... READ`), and downgrades the

metadata locks from exclusive to shared. After the operation acquires locks and downgrades the metadata locks, other sessions can read but not modify the tables.

This operation applies only to existing base (non-TEMPORARY) tables. If a name refers to a base table, that table is used. If it refers to a TEMPORARY table, it is ignored. If a name applies to a view, an ER_WRONG_OBJECT error occurs. Otherwise, an ER_NO_SUCH_TABLE error occurs.

Use UNLOCK TABLES to release the locks, LOCK TABLES to release the locks and acquire other locks, or START TRANSACTION to release the locks and begin a new transaction.

This FLUSH TABLES variant enables tables to be flushed and locked in a single operation. It provides a workaround for the restriction that FLUSH TABLES is not permitted when there is an active LOCK TABLES ... READ.

This operation does not perform an implicit UNLOCK TABLES, so an error results if you perform the operation while there is any active LOCK TABLES or use it a second time without first releasing the locks acquired.

If a flushed table was opened with HANDLER, the handler is implicitly flushed and loses its position.

- FLUSH TABLES *tbl_name* [, *tbl_name*] ... FOR EXPORT

  This FLUSH TABLES variant applies to InnoDB tables. It ensures that changes to the named tables have been flushed to disk so that binary table copies can be made while the server is running.

  Because the FLUSH TABLES ... FOR EXPORT operation acquires locks on tables in preparation for exporting them, it requires the LOCK TABLES and SELECT privileges for each table, in addition to the RELOAD privilege.

  The operation works like this:

  1. It acquires shared metadata locks for the named tables. The operation blocks as long as other sessions have active transactions that have modified those tables or hold table locks for them. When the locks have been acquired, the operation blocks transactions that attempt to update the tables, while permitting read-only operations to continue.
  2. It checks whether all storage engines for the tables support FOR EXPORT. If any do not, an ER_ILLEGAL_HA error occurs and the operation fails.
  3. The operation notifies the storage engine for each table to make the table ready for export. The storage engine must ensure that any pending changes are written to disk.
  4. The operation puts the session in lock-tables mode so that the metadata locks acquired earlier are not released when the FOR EXPORT operation completes.

  This operation applies only to existing base (non-TEMPORARY) tables. If a name refers to a base table, that table is used. If it refers to a TEMPORARY table, it is ignored. If a name applies to a view, an ER_WRONG_OBJECT error occurs. Otherwise, an ER_NO_SUCH_TABLE error occurs.

  InnoDB supports FOR EXPORT for tables that have their own .ibd file (that is, tables created with the innodb_file_per_table setting enabled). InnoDB ensures when notified by the FOR EXPORT operation that any changes have been flushed to disk. This permits a binary copy of table contents to be made while the FOR EXPORT operation is in effect because the .ibd file is transaction consistent and can be copied while the server is running. FOR EXPORT does not apply to InnoDB system tablespace files, or to InnoDB tables that have FULLTEXT indexes.

  FLUSH TABLES ...FOR EXPORT does not work with partitioned InnoDB tables prior to MySQL 5.6.17, but is supported for such tables in MySQL 5.6.17 and higher.

  When notified by FOR EXPORT, InnoDB writes to disk certain kinds of data that is normally held in memory or in separate disk buffers outside the tablespace files. For each table, InnoDB also

produces a file named *table_name*.cfg in the same database directory as the table. The .cfg file contains metadata needed to reimport the tablespace files later, into the same or different server.

When the FOR EXPORT operation completes, InnoDB has flushed all dirty pages to the table data files. Any change buffer entries are merged prior to flushing. At this point, the tables are locked and quiescent: The tables are in a transactionally consistent state on disk and you can copy the .ibd tablespace files along with the corresponding .cfg files to get a consistent snapshot of those tables.

For the procedure to reimport the copied table data into a MySQL instance, see Section 14.6.1.3, "Importing InnoDB Tables".

After you are done with the tables, use UNLOCK TABLES to release the locks, LOCK TABLES to release the locks and acquire other locks, or START TRANSACTION to release the locks and begin a new transaction.

While any of these statements is in effect within the session, attempts to use FLUSH TABLES ... FOR EXPORT produce an error:

```
FLUSH TABLES ... WITH READ LOCK
FLUSH TABLES ... FOR EXPORT
LOCK TABLES ... READ
LOCK TABLES ... WRITE
```

While FLUSH TABLES ... FOR EXPORT is in effect within the session, attempts to use any of these statements produce an error:

```
FLUSH TABLES WITH READ LOCK
FLUSH TABLES ... WITH READ LOCK
FLUSH TABLES ... FOR EXPORT
```

13.7.6.4 KILL Statement

```
KILL [CONNECTION | QUERY] processlist_id
```

Each connection to mysqld runs in a separate thread. You can kill a thread with the KILL *processlist_id* statement.

Thread processlist identifiers can be determined from the ID column of the INFORMATION_SCHEMA PROCESSLIST table, the Id column of SHOW PROCESSLIST output, and the PROCESSLIST_ID column of the Performance Schema threads table. The value for the current thread is returned by the CONNECTION_ID() function.

KILL permits an optional CONNECTION or QUERY modifier:

- KILL CONNECTION is the same as KILL with no modifier: It terminates the connection associated with the given *processlist_id*, after terminating any statement the connection is executing.
- KILL QUERY terminates the statement the connection is currently executing, but leaves the connection itself intact.

The ability to see which threads are available to be killed depends on the PROCESS privilege:

- Without PROCESS, you can see only your own threads.
- With PROCESS, you can see all threads.

The ability to kill threads and statements depends on the SUPER privilege:

- Without SUPER, you can kill only your own threads and statements.
- With SUPER, you can kill all threads and statements.

You can also use the mysqladmin processlist and mysqladmin kill commands to examine and kill threads.

Note

You cannot use `KILL` with the Embedded MySQL Server library because the embedded server merely runs inside the threads of the host application. It does not create any connection threads of its own.

When you use `KILL`, a thread-specific kill flag is set for the thread. In most cases, it might take some time for the thread to die because the kill flag is checked only at specific intervals:

- During `SELECT` operations, for `ORDER BY` and `GROUP BY` loops, the flag is checked after reading a block of rows. If the kill flag is set, the statement is aborted.
- `ALTER TABLE` operations that make a table copy check the kill flag periodically for each few copied rows read from the original table. If the kill flag was set, the statement is aborted and the temporary table is deleted.

  The `KILL` statement returns without waiting for confirmation, but the kill flag check aborts the operation within a reasonably small amount of time. Aborting the operation to perform any necessary cleanup also takes some time.
- During `UPDATE` or `DELETE` operations, the kill flag is checked after each block read and after each updated or deleted row. If the kill flag is set, the statement is aborted. If you are not using transactions, the changes are not rolled back.
- `GET_LOCK()` aborts and returns `NULL`.
- An `INSERT DELAYED` thread quickly flushes (inserts) all rows it has in memory and then terminates.
- If the thread is in the table lock handler (state: `Locked`), the table lock is quickly aborted.
- If the thread is waiting for free disk space in a write call, the write is aborted with a “disk full” error message.

Warning

Killing a `REPAIR TABLE` or `OPTIMIZE TABLE` operation on a `MyISAM` table results in a table that is corrupted and unusable. Any reads or writes to such a table fail until you optimize or repair it again (without interruption).

13.7.6.5 LOAD INDEX INTO CACHE Statement

```
LOAD INDEX INTO CACHE
  tbl_index_list [, tbl_index_list] ...

tbl_index_list:
  tbl_name
    [PARTITION (partition_list)]
    [{INDEX|KEY} (index_name[, index_name] ...)]
    [IGNORE LEAVES]

partition_list: {
    partition_name[, partition_name] ...
  | ALL
}
```

The `LOAD INDEX INTO CACHE` statement preloads a table index into the key cache to which it has been assigned by an explicit `CACHE INDEX` statement, or into the default key cache otherwise.

`LOAD INDEX INTO CACHE` applies only to `MyISAM` tables, including partitioned `MyISAM` tables. In addition, indexes on partitioned tables can be preloaded for one, several, or all partitions.

The `IGNORE LEAVES` modifier causes only blocks for the nonleaf nodes of the index to be preloaded.

`IGNORE LEAVES` is also supported for partitioned `MyISAM` tables.

The following statement preloads nodes (index blocks) of indexes for the tables `t1` and `t2`:

```
mysql> LOAD INDEX INTO CACHE t1, t2 IGNORE LEAVES;
+---------+--------------+----------+----------+
| Table   | Op           | Msg_type | Msg_text |
+---------+--------------+----------+----------+
| test.t1 | preload_keys | status   | OK       |
| test.t2 | preload_keys | status   | OK       |
+---------+--------------+----------+----------+
```

This statement preloads all index blocks from `t1`. It preloads only blocks for the nonleaf nodes from `t2`.

The syntax of `LOAD INDEX INTO CACHE` enables you to specify that only particular indexes from a table should be preloaded. However, the implementation preloads all the table's indexes into the cache, so there is no reason to specify anything other than the table name.

In MySQL 5.6.11 only, `gtid_next` must be set to `AUTOMATIC` before issuing this statement. (Bug #16062608, Bug #16715809, Bug #69045)

It is possible to preload indexes on specific partitions of partitioned `MyISAM` tables. For example, of the following 2 statements, the first preloads indexes for partition `p0` of a partitioned table `pt`, while the second preloads the indexes for partitions `p1` and `p3` of the same table:

```
LOAD INDEX INTO CACHE pt PARTITION (p0);
LOAD INDEX INTO CACHE pt PARTITION (p1, p3);
```

To preload the indexes for all partitions in table `pt`, you can use either of the following two statements:

```
LOAD INDEX INTO CACHE pt PARTITION (ALL);

LOAD INDEX INTO CACHE pt;
```

The two statements just shown are equivalent, and issuing either one has exactly the same effect. In other words, if you wish to preload indexes for all partitions of a partitioned table, the `PARTITION (ALL)` clause is optional.

When preloading indexes for multiple partitions, the partitions need not be contiguous, and you need not list their names in any particular order.

`LOAD INDEX INTO CACHE ... IGNORE LEAVES` fails unless all indexes in a table have the same block size. To determine index block sizes for a table, use `myisamchk -dv` and check the `Blocksize` column.

13.7.6.6 RESET Statement

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

reset_option: {
    MASTER
  | QUERY CACHE
  | SLAVE
}
```

The `RESET` statement is used to clear the state of various server operations. You must have the `RELOAD` privilege to execute `RESET`.

`RESET` acts as a stronger version of the `FLUSH` statement. See Section 13.7.6.3, “FLUSH Statement”.

The `RESET` statement causes an implicit commit. See Section 13.3.3, “Statements That Cause an Implicit Commit”.

In MySQL 5.6.11 only, `gtid_next` must be set to `AUTOMATIC` before issuing this statement. (Bug #16062608, Bug #16715809, Bug #69045)

The following list describes the permitted `RESET` statement *`reset_option`* values:

- `RESET MASTER`

  Deletes all binary logs listed in the index file, resets the binary log index file to be empty, and creates a new binary log file.

- `RESET QUERY CACHE`

  Removes all query results from the query cache.

- `RESET SLAVE`

  Makes the replica forget its replication position in the source binary logs. Also resets the relay log by deleting any existing relay log files and beginning a new one.

13.8 Utility Statements

13.8.1 DESCRIBE Statement

The `DESCRIBE` and `EXPLAIN` statements are synonyms, used either to obtain information about table structure or query execution plans. For more information, see Section 13.7.5.6, “SHOW COLUMNS Statement”, and Section 13.8.2, “EXPLAIN Statement”.

13.8.2 EXPLAIN Statement

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

{EXPLAIN | DESCRIBE | DESC}
    [explain_type]
    explainable_stmt

explain_type: {
    EXTENDED
  | PARTITIONS
  | FORMAT = format_name
}

format_name: {
    TRADITIONAL
  | JSON
}

explainable_stmt: {
    SELECT statement
  | DELETE statement
  | INSERT statement
  | REPLACE statement
  | UPDATE statement
}
```

The `DESCRIBE` and `EXPLAIN` statements are synonyms. In practice, the `DESCRIBE` keyword is more often used to obtain information about table structure, whereas `EXPLAIN` is used to obtain a query execution plan (that is, an explanation of how MySQL would execute a query).

The following discussion uses the `DESCRIBE` and `EXPLAIN` keywords in accordance with those uses, but the MySQL parser treats them as completely synonymous.

- Obtaining Table Structure Information
- Obtaining Execution Plan Information

Obtaining Table Structure Information

`DESCRIBE` provides information about the columns in a table:

```
mysql> DESCRIBE City;
+------------+----------+------+-----+---------+----------------+
| Field      | Type     | Null | Key | Default | Extra          |
+------------+----------+------+-----+---------+----------------+
| Id         | int(11)  | NO   | PRI | NULL    | auto_increment |
| Name       | char(35) | NO   |     |         |                |
| Country    | char(3)  | NO   | UNI |         |                |
| District   | char(20) | YES  | MUL |         |                |
| Population | int(11)  | NO   |     | 0       |                |
+------------+----------+------+-----+---------+----------------+
```

`DESCRIBE` is a shortcut for `SHOW COLUMNS`. These statements also display information for views. The description for `SHOW COLUMNS` provides more information about the output columns. See Section 13.7.5.6, “SHOW COLUMNS Statement”.

By default, `DESCRIBE` displays information about all columns in the table. *`col_name`*, if given, is the name of a column in the table. In this case, the statement displays information only for the named column. *`wild`*, if given, is a pattern string. It can contain the SQL `%` and `_` wildcard characters. In this case, the statement displays output only for the columns with names matching the string. There is no need to enclose the string within quotation marks unless it contains spaces or other special characters.

The `DESCRIBE` statement is provided for compatibility with Oracle.

The `SHOW CREATE TABLE`, `SHOW TABLE STATUS`, and `SHOW INDEX` statements also provide information about tables. See Section 13.7.5, “SHOW Statements”.

Obtaining Execution Plan Information

The `EXPLAIN` statement provides information about how MySQL executes statements:

- `EXPLAIN` works with `SELECT`, `DELETE`, `INSERT`, `REPLACE`, and `UPDATE` statements.
- When `EXPLAIN` is used with an explainable statement, MySQL displays information from the optimizer about the statement execution plan. That is, MySQL explains how it would process the statement, including information about how tables are joined and in which order. For information about using `EXPLAIN` to obtain execution plan information, see Section 8.8.2, “EXPLAIN Output Format”.
- `EXPLAIN EXTENDED` produces additional execution plan information that can be displayed using `SHOW WARNINGS`. See Section 8.8.3, “Extended EXPLAIN Output Format”.
- `EXPLAIN PARTITIONS` is useful for examining queries involving partitioned tables. See Section 19.3.5, “Obtaining Information About Partitions”.
- The `FORMAT` option can be used to select the output format. `TRADITIONAL` presents the output in tabular format. This is the default if no `FORMAT` option is present. `JSON` format displays the information in JSON format. With `FORMAT = JSON`, the output includes extended and partition information.

`EXPLAIN` requires the same privileges required to execute the explained statement. Additionally, `EXPLAIN` also requires the `SHOW VIEW` privilege for any explained view.

With the help of `EXPLAIN`, you can see where you should add indexes to tables so that the statement executes faster by using indexes to find rows. You can also use `EXPLAIN` to check whether the optimizer joins the tables in an optimal order. To give a hint to the optimizer to use a join order corresponding to the order in which the tables are named in a `SELECT` statement, begin the statement with `SELECT STRAIGHT_JOIN` rather than just `SELECT`. (See Section 13.2.9, “SELECT Statement”.)

If you have a problem with indexes not being used when you believe that they should be, run `ANALYZE TABLE` to update table statistics, such as cardinality of keys, that can affect the choices the optimizer makes. See Section 13.7.2.1, “ANALYZE TABLE Statement”.

Note

MySQL Workbench has a Visual Explain capability that provides a visual representation of `EXPLAIN` output. See Tutorial: Using Explain to Improve Query Performance.

13.8.3 HELP Statement

```
HELP 'search_string'
```

The `HELP` statement returns online information from the MySQL Reference Manual. Its proper operation requires that the help tables in the `mysql` database be initialized with help topic information (see Section 5.1.14, “Server-Side Help Support”).

The `HELP` statement searches the help tables for the given search string and displays the result of the search. The search string is not case-sensitive.

The search string can contain the wildcard characters `%` and `_`. These have the same meaning as for pattern-matching operations performed with the `LIKE` operator. For example, `HELP 'rep%'` returns a list of topics that begin with `rep`.

The HELP statement understands several types of search strings:

- At the most general level, use `contents` to retrieve a list of the top-level help categories:

  ```
  HELP 'contents'
  ```

- For a list of topics in a given help category, such as `Data Types`, use the category name:

  ```
  HELP 'data types'
  ```

- For help on a specific help topic, such as the `ASCII()` function or the `CREATE TABLE` statement, use the associated keyword or keywords:

  ```
  HELP 'ascii'
  HELP 'create table'
  ```

In other words, the search string matches a category, many topics, or a single topic. You cannot necessarily tell in advance whether a given search string returns a list of items or the help information for a single help topic. However, you can tell what kind of response `HELP` returned by examining the number of rows and columns in the result set.

The following descriptions indicate the forms that the result set can take. Output for the example statements is shown using the familiar “tabular” or “vertical” format that you see when using the `mysql` client.

Note

The `mysql` itself reformats `HELP` result sets in a different way.

- Empty result set

  No match could be found for the search string.

- Result set containing a single row with three columns

  This means that the search string yielded a hit for the help topic. The result has three columns:

  - `name`: The topic name.
  - `description`: Descriptive help text for the topic.

- `example`: Usage example or examples. This column might be blank.

Example: `HELP 'replace'`

Yields:

```
name: REPLACE
description: Syntax:
REPLACE(str,from_str,to_str)

Returns the string str with all occurrences of the string from_str
replaced by the string to_str. REPLACE() performs a case-sensitive
match when searching for from_str.
example: mysql> SELECT REPLACE('www.mysql.com', 'w', 'Ww');
        -> 'WwWwWw.mysql.com'
```

- Result set containing multiple rows with two columns

  This means that the search string matched many help topics. The result set indicates the help topic names:

  - `name`: The help topic name.
  - `is_it_category`: `Y` if the name represents a help category, `N` if it does not. If it does not, the `name` value when specified as the argument to the `HELP` statement should yield a single-row result set containing a description for the named item.

  Example: `HELP 'status'`

  Yields:

```
+----------------------+----------------+
| name                 | is_it_category |
+----------------------+----------------+
| SHOW                 | N              |
| SHOW ENGINE          | N              |
| SHOW MASTER STATUS   | N              |
| SHOW PROCEDURE STATUS | N              |
| SHOW SLAVE STATUS    | N              |
| SHOW STATUS          | N              |
| SHOW TABLE STATUS    | N              |
+----------------------+----------------+
```

- Result set containing multiple rows with three columns

  This means the search string matches a category. The result set contains category entries:

  - `source_category_name`: The help category name.
  - `name`: The category or topic name
  - `is_it_category`: `Y` if the name represents a help category, `N` if it does not. If it does not, the `name` value when specified as the argument to the `HELP` statement should yield a single-row result set containing a description for the named item.

  Example: `HELP 'functions'`

  Yields:

```
+---------------------+-------------------------+----------------+
| source_category_name | name                    | is_it_category |
+---------------------+-------------------------+----------------+
| Functions           | CREATE FUNCTION         | N              |
| Functions           | DROP FUNCTION           | N              |
| Functions           | Bit Functions           | Y              |
| Functions           | Comparison operators    | Y              |
```

```
| Functions            | Control flow functions  | Y              |
| Functions            | Date and Time Functions | Y              |
| Functions            | Encryption Functions    | Y              |
| Functions            | Information Functions   | Y              |
| Functions            | Logical operators       | Y              |
| Functions            | Miscellaneous Functions | Y              |
| Functions            | Numeric Functions       | Y              |
| Functions            | String Functions        | Y              |
+----------------------+-------------------------+----------------+
```

13.8.4 USE Statement

```
USE db_name
```

The USE statement tells MySQL to use the named database as the default (current) database for subsequent statements. This statement requires some privilege for the database or some object within it.

The named database remains the default until the end of the session or another USE statement is issued:

```
USE db1;
SELECT COUNT(*) FROM mytable;   # selects from db1.mytable
USE db2;
SELECT COUNT(*) FROM mytable;   # selects from db2.mytable
```

The database name must be specified on a single line. Newlines in database names are not supported.

Making a particular database the default by means of the USE statement does not preclude accessing tables in other databases. The following example accesses the `author` table from the `db1` database and the `editor` table from the `db2` database:

```
USE db1;
SELECT author_name,editor_name FROM author,db2.editor
  WHERE author.editor_id = db2.editor.editor_id;
```

Chapter 14 The InnoDB Storage Engine

Table of Contents

14.1 Introduction to InnoDB 1820
14.1.1 Benefits of Using InnoDB Tables 1822
14.1.2 Best Practices for InnoDB Tables 1823
14.1.3 Checking InnoDB Availability 1824
14.1.4 Testing and Benchmarking with InnoDB 1824
14.1.5 Turning Off InnoDB 1825
14.2 InnoDB and the ACID Model 1825
14.3 InnoDB Multi-Versioning 1827
14.4 InnoDB Architecture 1828
14.5 InnoDB In-Memory Structures 1829
14.5.1 Buffer Pool 1829
14.5.2 Change Buffer 1834
14.5.3 Adaptive Hash Index 1837
14.5.4 Log Buffer 1838
14.6 InnoDB On-Disk Structures 1838
14.6.1 Tables 1838
14.6.2 Indexes 1857
14.6.3 Tablespaces 1863
14.6.4 InnoDB Data Dictionary 1869
14.6.5 Doublewrite Buffer 1869
14.6.6 Redo Log 1870
14.6.7 Undo Logs 1871
14.7 InnoDB Locking and Transaction Model 1872
14.7.1 InnoDB Locking 1872
14.7.2 InnoDB Transaction Model 1876
14.7.3 Locks Set by Different SQL Statements in InnoDB 1883
14.7.4 Phantom Rows 1886
14.7.5 Deadlocks in InnoDB 1887
14.8 InnoDB Configuration 1890
14.8.1 InnoDB Startup Configuration 1890
14.8.2 Configuring InnoDB for Read-Only Operation 1895
14.8.3 InnoDB Buffer Pool Configuration 1896
14.8.4 Configuring the Memory Allocator for InnoDB 1902
14.8.5 Configuring Thread Concurrency for InnoDB 1903
14.8.6 Configuring the Number of Background InnoDB I/O Threads 1904
14.8.7 Using Asynchronous I/O on Linux 1904
14.8.8 Configuring InnoDB I/O Capacity 1905
14.8.9 Configuring Spin Lock Polling 1906
14.8.10 Purge Configuration 1907
14.8.11 Configuring Optimizer Statistics for InnoDB 1908
14.8.12 Configuring the Merge Threshold for Index Pages 1919
14.9 InnoDB Table Compression 1921
14.9.1 Overview of Table Compression 1921
14.9.2 Enabling Compression for a Table 1922
14.9.3 Tuning Compression for InnoDB Tables 1923
14.9.4 Monitoring InnoDB Table Compression at Runtime 1926
14.9.5 How Compression Works for InnoDB Tables 1927
14.9.6 Compression for OLTP Workloads 1930
14.9.7 SQL Compression Syntax Warnings and Errors 1931
14.10 InnoDB File-Format Management 1933
14.10.1 Enabling File Formats 1934
14.10.2 Verifying File Format Compatibility 1934

14.10.3 Identifying the File Format in Use 1937
14.10.4 Modifying the File Format 1938
14.11 InnoDB Row Formats 1938
14.12 InnoDB Disk I/O and File Space Management 1943
14.12.1 InnoDB Disk I/O 1943
14.12.2 File Space Management 1944
14.12.3 InnoDB Checkpoints 1945
14.12.4 Defragmenting a Table 1945
14.12.5 Reclaiming Disk Space with TRUNCATE TABLE 1946
14.13 InnoDB and Online DDL 1946
14.13.1 Online DDL Operations 1947
14.13.2 Online DDL Performance and Concurrency 1955
14.13.3 Online DDL Space Requirements 1958
14.13.4 Simplifying DDL Statements with Online DDL 1959
14.13.5 Online DDL Failure Conditions 1959
14.13.6 Online DDL Limitations 1960
14.14 InnoDB Startup Options and System Variables 1961
14.15 InnoDB INFORMATION_SCHEMA Tables 2028
14.15.1 InnoDB INFORMATION_SCHEMA Tables about Compression 2028
14.15.2 InnoDB INFORMATION_SCHEMA Transaction and Locking Information 2030
14.15.3 InnoDB INFORMATION_SCHEMA System Tables 2035
14.15.4 InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables 2040
14.15.5 InnoDB INFORMATION_SCHEMA Buffer Pool Tables 2043
14.15.6 InnoDB INFORMATION_SCHEMA Metrics Table 2047
14.16 InnoDB Integration with MySQL Performance Schema 2055
14.16.1 Monitoring InnoDB Mutex Waits Using Performance Schema 2057
14.17 InnoDB Monitors 2060
14.17.1 InnoDB Monitor Types 2060
14.17.2 Enabling InnoDB Monitors 2061
14.17.3 InnoDB Standard Monitor and Lock Monitor Output 2064
14.17.4 InnoDB Tablespace Monitor Output 2068
14.17.5 InnoDB Table Monitor Output 2071
14.18 InnoDB Backup and Recovery 2074
14.18.1 InnoDB Backup 2074
14.18.2 InnoDB Recovery 2075
14.19 InnoDB and MySQL Replication 2077
14.20 InnoDB memcached Plugin 2079
14.20.1 Benefits of the InnoDB memcached Plugin 2079
14.20.2 InnoDB memcached Architecture 2080
14.20.3 Setting Up the InnoDB memcached Plugin 2084
14.20.4 Security Considerations for the InnoDB memcached Plugin 2089
14.20.5 Writing Applications for the InnoDB memcached Plugin 2091
14.20.6 The InnoDB memcached Plugin and Replication 2103
14.20.7 InnoDB memcached Plugin Internals 2106
14.20.8 Troubleshooting the InnoDB memcached Plugin 2110
14.21 InnoDB Troubleshooting 2112
14.21.1 Troubleshooting InnoDB I/O Problems 2113
14.21.2 Forcing InnoDB Recovery 2113
14.21.3 Troubleshooting InnoDB Data Dictionary Operations 2115
14.21.4 InnoDB Error Handling 2118
14.22 InnoDB Limits 2119
14.23 InnoDB Restrictions and Limitations 2120

14.1 Introduction to InnoDB

`InnoDB` is a general-purpose storage engine that balances high reliability and high performance. In MySQL 5.6, `InnoDB` is the default MySQL storage engine. Unless you have configured a different

default storage engine, issuing a `CREATE TABLE` statement without an `ENGINE=` clause creates an `InnoDB` table.

`InnoDB` includes all the features that were part of the InnoDB Plugin for MySQL 5.1, plus new features specific to MySQL 5.5 and higher.

Note

The `mysql` and `INFORMATION_SCHEMA` databases that implement some of the MySQL internals still use `MyISAM`. In particular, you cannot switch the grant tables to use `InnoDB`.

Key Advantages of InnoDB

- Its DML operations follow the ACID model, with transactions featuring commit, rollback, and crash-recovery capabilities to protect user data. See Section 14.2, “InnoDB and the ACID Model” for more information.
- Row-level locking and Oracle-style consistent reads increase multi-user concurrency and performance. See Section 14.7, “InnoDB Locking and Transaction Model” for more information.
- `InnoDB` tables arrange your data on disk to optimize queries based on primary keys. Each `InnoDB` table has a primary key index called the clustered index that organizes the data to minimize I/O for primary key lookups. See Section 14.6.2.1, “Clustered and Secondary Indexes” for more information.
- To maintain data integrity, `InnoDB` supports `FOREIGN KEY` constraints. With foreign keys, inserts, updates, and deletes are checked to ensure they do not result in inconsistencies across different tables. See Section 13.1.17.5, “FOREIGN KEY Constraints” for more information.

Table 14.1 InnoDB Storage Engine Features

Feature	Support
B-tree indexes	Yes
Backup/point-in-time recovery (Implemented in the server, rather than in the storage engine.)	Yes
Cluster database support	No
Clustered indexes	Yes
Compressed data	Yes
Data caches	Yes
Encrypted data	Yes (Implemented in the server via encryption functions; In MySQL 5.7 and later, data-at-rest tablespace encryption is supported.)
Foreign key support	Yes
Full-text search indexes	Yes (InnoDB support for FULLTEXT indexes is available in MySQL 5.6 and later.)
Geospatial data type support	Yes
Geospatial indexing support	Yes (InnoDB support for geospatial indexing is available in MySQL 5.7 and later.)
Hash indexes	No (InnoDB utilizes hash indexes internally for its Adaptive Hash Index feature.)
Index caches	Yes

Feature	Support
Locking granularity	Row
MVCC	Yes
Replication support (Implemented in the server, rather than in the storage engine.)	Yes
Storage limits	64TB
T-tree indexes	No
Transactions	Yes
Update statistics for data dictionary	Yes

To compare the features of `InnoDB` with other storage engines provided with MySQL, see the *Storage Engine Features* table in Chapter 15, *Alternative Storage Engines*.

InnoDB Enhancements and New Features

For information about `InnoDB` enhancements and new features, refer to:

- The `InnoDB` enhancements list in Section 1.3, “What Is New in MySQL 5.6”.
- The Release Notes.

Additional InnoDB Information and Resources

- For `InnoDB`-related terms and definitions, see the MySQL Glossary.
- For a forum dedicated to the `InnoDB` storage engine, see MySQL Forums::InnoDB.
- `InnoDB` is published under the same GNU GPL License Version 2 (of June 1991) as MySQL. For more information on MySQL licensing, see http://www.mysql.com/company/legal/licensing/.

14.1.1 Benefits of Using InnoDB Tables

You may find `InnoDB` tables beneficial for the following reasons:

- If your server unexpectedly exits because of a hardware or software issue, regardless of what was happening in the database at the time, you don't need to do anything special after restarting the database. `InnoDB` crash recovery automatically finalizes any changes that were committed before the time of the crash, and undoes any changes that were in process but not committed. Just restart and continue where you left off.
- The `InnoDB` storage engine maintains its own buffer pool that caches table and index data in main memory as data is accessed. Frequently used data is processed directly from memory. This cache applies to many types of information and speeds up processing. On dedicated database servers, up to 80% of physical memory is often assigned to the buffer pool.
- If you split up related data into different tables, you can set up foreign keys that enforce referential integrity. Update or delete data, and the related data in other tables is updated or deleted automatically. Try to insert data into a secondary table without corresponding data in the primary table, and the bad data gets kicked out automatically.
- If data becomes corrupted on disk or in memory, a checksum mechanism alerts you to the bogus data before you use it.
- When you design your database with appropriate primary key columns for each table, operations involving those columns are automatically optimized. It is very fast to reference the primary key columns in `WHERE` clauses, `ORDER BY` clauses, `GROUP BY` clauses, and join operations.

- Inserts, updates, and deletes are optimized by an automatic mechanism called change buffering. `InnoDB` not only allows concurrent read and write access to the same table, it caches changed data to streamline disk I/O.
- Performance benefits are not limited to giant tables with long-running queries. When the same rows are accessed over and over from a table, a feature called the Adaptive Hash Index takes over to make these lookups even faster, as if they came out of a hash table.
- You can compress tables and associated indexes.
- You can create and drop indexes with much less impact on performance and availability.
- Truncating a file-per-table tablespace is very fast, and can free up disk space for the operating system to reuse, rather than freeing up space within the system tablespace that only `InnoDB` can reuse.
- The storage layout for table data is more efficient for `BLOB` and long text fields, with the DYNAMIC row format.
- You can monitor the internal workings of the storage engine by querying INFORMATION_SCHEMA tables.
- You can monitor the performance details of the storage engine by querying Performance Schema tables.
- You can freely mix `InnoDB` tables with tables from other MySQL storage engines, even within the same statement. For example, you can use a join operation to combine data from `InnoDB` and `MEMORY` tables in a single query.
- `InnoDB` has been designed for CPU efficiency and maximum performance when processing large data volumes.
- `InnoDB` tables can handle large quantities of data, even on operating systems where file size is limited to 2GB.

For `InnoDB`-specific tuning techniques you can apply in your application code, see Section 8.5, “Optimizing for InnoDB Tables”.

14.1.2 Best Practices for InnoDB Tables

This section describes best practices when using `InnoDB` tables.

- Specifying a primary key for every table using the most frequently queried column or columns, or an auto-increment value if there is no obvious primary key.
- Using joins wherever data is pulled from multiple tables based on identical ID values from those tables. For fast join performance, define foreign keys on the join columns, and declare those columns with the same data type in each table. Adding foreign keys ensures that referenced columns are indexed, which can improve performance. Foreign keys also propagate deletes or updates to all affected tables, and prevent insertion of data in a child table if the corresponding IDs are not present in the parent table.
- Turning off autocommit. Committing hundreds of times a second puts a cap on performance (limited by the write speed of your storage device).
- Grouping sets of related DML operations into transactions, by bracketing them with `START TRANSACTION` and `COMMIT` statements. While you don't want to commit too often, you also don't want to issue huge batches of `INSERT`, `UPDATE`, or `DELETE` statements that run for hours without committing.
- Not using `LOCK TABLES` statements. `InnoDB` can handle multiple sessions all reading and writing to the same table at once, without sacrificing reliability or high performance. To get exclusive write

access to a set of rows, use the `SELECT ... FOR UPDATE` syntax to lock just the rows you intend to update.

- Enabling the `innodb_file_per_table` option to put the data and indexes for individual tables into separate files, instead of the system tablespace. This setting is required to use other features, such as table compression. It is enabled by default.
- Evaluating whether your data and access patterns benefit from the `InnoDB` table compression feature. You can compress `InnoDB` tables without sacrificing read/write capability.
- Running your server with the option `--sql_mode=NO_ENGINE_SUBSTITUTION` to prevent tables being created with a different storage engine if there is an issue with the engine specified in the `ENGINE=` clause of `CREATE TABLE`.

14.1.3 Checking InnoDB Availability

To determine whether your server supports `InnoDB`:

- Issue the `SHOW ENGINES` statement to view the available MySQL storage engines. Look for `DEFAULT` in the `InnoDB` line.

```
mysql> SHOW ENGINES;
```

Alternatively, query the `INFORMATION_SCHEMA.ENGINES` table.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.ENGINES;
```

(Now that `InnoDB` is the default MySQL storage engine, only very specialized environments might not support it.)

- If `InnoDB` is not present, you have a `mysqld` binary that was compiled without `InnoDB` support and you need to get a different one.
- If `InnoDB` is present but disabled, go back through your startup options and configuration file and get rid of any `--skip-innodb` option.

14.1.4 Testing and Benchmarking with InnoDB

If `InnoDB` is not your default storage engine, you can determine if your database server or applications work correctly with `InnoDB` by restarting the server with `--default-storage-engine=InnoDB` defined on the command line or with `default-storage-engine=innodb` defined in the `[mysqld]` section of your MySQL server option file.

Since changing the default storage engine only affects new tables as they are created, run all your application installation and setup steps to confirm that everything installs properly. Then exercise all the application features to make sure all the data loading, editing, and querying features work. If a table relies on a feature that is specific to another storage engine, you receive an error; add the `ENGINE=`*`other_engine_name`* clause to the `CREATE TABLE` statement to avoid the error.

If you did not make a deliberate decision about the storage engine, and you want to preview how certain tables work when created using `InnoDB`, issue the command `ALTER TABLE table_name ENGINE=InnoDB;` for each table. Or, to run test queries and other statements without disturbing the original table, make a copy:

```
CREATE TABLE InnoDB_Table (...) ENGINE=InnoDB AS SELECT * FROM other_engine_table;
```

To assess performance with a full application under a realistic workload, install the latest MySQL server and run benchmarks.

Test the full application lifecycle, from installation, through heavy usage, and server restart. Kill the server process while the database is busy to simulate a power failure, and verify that the data is recovered successfully when you restart the server.

Test any replication configurations, especially if you use different MySQL versions and options on the source server and replicas.

14.1.5 Turning Off InnoDB

Oracle recommends `InnoDB` as the preferred storage engine for typical database applications, from single-user wikis and blogs running on a local system, to high-end applications pushing the limits of performance. In MySQL 5.6, `InnoDB` is the default storage engine for new tables.

If you do not want to use `InnoDB` tables:

- Start the server with the `--innodb=OFF` or `--skip-innodb` option to disable the `InnoDB` storage engine.

Note

As of MySQL 5.6.21, the `--skip-innodb` option still works but it is deprecated and returns a warning when used. Expect it to be removed in a future MySQL release. This also applies to its synonyms (`--innodb=OFF`, `--disable-innodb`, and so forth).

- Because the default storage engine is `InnoDB`, the server cannot start unless you also use `--default-storage-engine` and `--default-tmp-storage-engine` to set the default to some other engine for both permanent and `TEMPORARY` tables.
- To prevent the server from crashing when the `InnoDB`-related `information_schema` tables are queried, also disable the plugins associated with those tables. Specify in the `[mysqld]` section of the MySQL configuration file:

```
loose-innodb-trx=0
loose-innodb-locks=0
loose-innodb-lock-waits=0
loose-innodb-cmp=0
loose-innodb-cmp-per-index=0
loose-innodb-cmp-per-index-reset=0
loose-innodb-cmp-reset=0
loose-innodb-cmpmem=0
loose-innodb-cmpmem-reset=0
loose-innodb-buffer-page=0
loose-innodb-buffer-page-lru=0
loose-innodb-buffer-pool-stats=0
loose-innodb-metrics=0
loose-innodb-ft-default-stopword=0
loose-innodb-ft-inserted=0
loose-innodb-ft-deleted=0
loose-innodb-ft-being-deleted=0
loose-innodb-ft-config=0
loose-innodb-ft-index-cache=0
loose-innodb-ft-index-table=0
loose-innodb-sys-tables=0
loose-innodb-sys-tablestats=0
loose-innodb-sys-indexes=0
loose-innodb-sys-columns=0
loose-innodb-sys-fields=0
loose-innodb-sys-foreign=0
loose-innodb-sys-foreign-cols=0
```

14.2 InnoDB and the ACID Model

The ACID model is a set of database design principles that emphasize aspects of reliability that are important for business data and mission-critical applications. MySQL includes components such as the `InnoDB` storage engine that adhere closely to the ACID model, so that data is not corrupted and results are not distorted by exceptional conditions such as software crashes and hardware malfunctions. When you rely on ACID-compliant features, you do not need to reinvent the wheel of consistency checking and crash recovery mechanisms. In cases where you have additional software

safeguards, ultra-reliable hardware, or an application that can tolerate a small amount of data loss or inconsistency, you can adjust MySQL settings to trade some of the ACID reliability for greater performance or throughput.

The following sections discuss how MySQL features, in particular the `InnoDB` storage engine, interact with the categories of the ACID model:

- **A**: atomicity.
- **C**: consistency.
- **I:**: isolation.
- **D**: durability.

Atomicity

The **atomicity** aspect of the ACID model mainly involves `InnoDB` transactions. Related MySQL features include:

- Autocommit setting.
- `COMMIT` statement.
- `ROLLBACK` statement.
- Operational data from the `INFORMATION_SCHEMA` tables.

Consistency

The **consistency** aspect of the ACID model mainly involves internal `InnoDB` processing to protect data from crashes. Related MySQL features include:

- `InnoDB` doublewrite buffer.
- `InnoDB` crash recovery.

Isolation

The **isolation** aspect of the ACID model mainly involves `InnoDB` transactions, in particular the isolation level that applies to each transaction. Related MySQL features include:

- Autocommit setting.
- `SET ISOLATION LEVEL` statement.
- The low-level details of `InnoDB` locking. During performance tuning, you see these details through `INFORMATION_SCHEMA` tables.

Durability

The **durability** aspect of the ACID model involves MySQL software features interacting with your particular hardware configuration. Because of the many possibilities depending on the capabilities of your CPU, network, and storage devices, this aspect is the most complicated to provide concrete guidelines for. (And those guidelines might take the form of buy “new hardware”.) Related MySQL features include:

- `InnoDB` doublewrite buffer, turned on and off by the `innodb_doublewrite` configuration option.
- Configuration option `innodb_flush_log_at_trx_commit`.
- Configuration option `sync_binlog`.
- Configuration option `innodb_file_per_table`.

- Write buffer in a storage device, such as a disk drive, SSD, or RAID array.
- Battery-backed cache in a storage device.
- The operating system used to run MySQL, in particular its support for the `fsync()` system call.
- Uninterruptible power supply (UPS) protecting the electrical power to all computer servers and storage devices that run MySQL servers and store MySQL data.
- Your backup strategy, such as frequency and types of backups, and backup retention periods.
- For distributed or hosted data applications, the particular characteristics of the data centers where the hardware for the MySQL servers is located, and network connections between the data centers.

14.3 InnoDB Multi-Versioning

`InnoDB` is a multi-versioned storage engine: it keeps information about old versions of changed rows, to support transactional features such as concurrency and rollback. This information is stored in the tablespace in a data structure called a rollback segment (after an analogous data structure in Oracle). `InnoDB` uses the information in the rollback segment to perform the undo operations needed in a transaction rollback. It also uses the information to build earlier versions of a row for a consistent read.

Internally, `InnoDB` adds three fields to each row stored in the database. A 6-byte `DB_TRX_ID` field indicates the transaction identifier for the last transaction that inserted or updated the row. Also, a deletion is treated internally as an update where a special bit in the row is set to mark it as deleted. Each row also contains a 7-byte `DB_ROLL_PTR` field called the roll pointer. The roll pointer points to an undo log record written to the rollback segment. If the row was updated, the undo log record contains the information necessary to rebuild the content of the row before it was updated. A 6-byte `DB_ROW_ID` field contains a row ID that increases monotonically as new rows are inserted. If `InnoDB` generates a clustered index automatically, the index contains row ID values. Otherwise, the `DB_ROW_ID` column does not appear in any index.

Undo logs in the rollback segment are divided into insert and update undo logs. Insert undo logs are needed only in transaction rollback and can be discarded as soon as the transaction commits. Update undo logs are used also in consistent reads, but they can be discarded only after there is no transaction present for which `InnoDB` has assigned a snapshot that in a consistent read could need the information in the update undo log to build an earlier version of a database row.

Commit your transactions regularly, including those transactions that issue only consistent reads. Otherwise, `InnoDB` cannot discard data from the update undo logs, and the rollback segment may grow too big, filling up your tablespace.

The physical size of an undo log record in the rollback segment is typically smaller than the corresponding inserted or updated row. You can use this information to calculate the space needed for your rollback segment.

In the `InnoDB` multi-versioning scheme, a row is not physically removed from the database immediately when you delete it with an SQL statement. `InnoDB` only physically removes the corresponding row and its index records when it discards the update undo log record written for the deletion. This removal operation is called a purge, and it is quite fast, usually taking the same order of time as the SQL statement that did the deletion.

If you insert and delete rows in smallish batches at about the same rate in the table, the purge thread can start to lag behind and the table can grow bigger and bigger because of all the “dead” rows, making everything disk-bound and very slow. In such a case, throttle new row operations, and allocate more resources to the purge thread by tuning the `innodb_max_purge_lag` system variable. See Section 14.14, “InnoDB Startup Options and System Variables” for more information.

Multi-Versioning and Secondary Indexes

`InnoDB` multiversion concurrency control (MVCC) treats secondary indexes differently than clustered indexes. Records in a clustered index are updated in-place, and their hidden system columns point

undo log entries from which earlier versions of records can be reconstructed. Unlike clustered index records, secondary index records do not contain hidden system columns nor are they updated in-place.

When a secondary index column is updated, old secondary index records are delete-marked, new records are inserted, and delete-marked records are eventually purged. When a secondary index record is delete-marked or the secondary index page is updated by a newer transaction, `InnoDB` looks up the database record in the clustered index. In the clustered index, the record's `DB_TRX_ID` is checked, and the correct version of the record is retrieved from the undo log if the record was modified after the reading transaction was initiated.

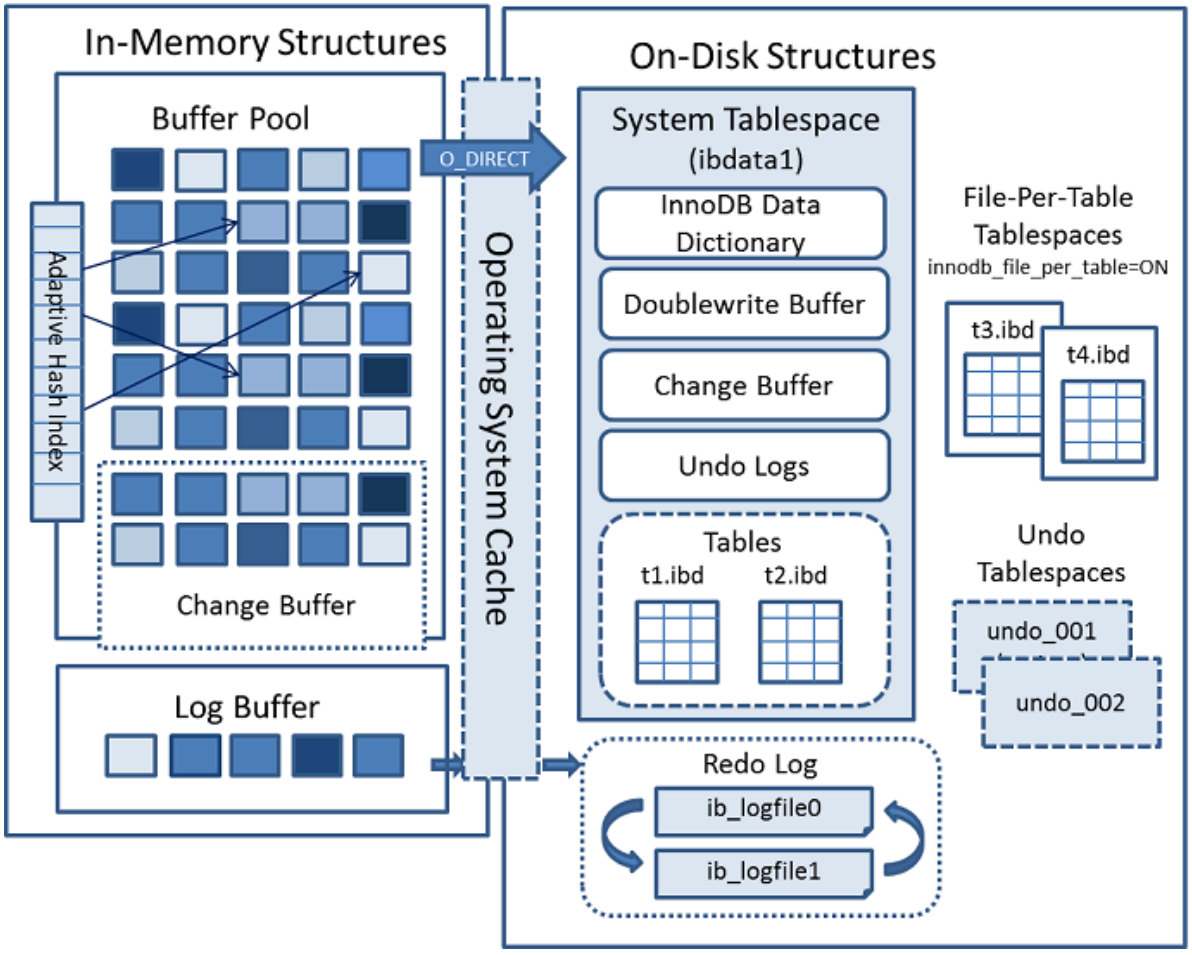
If a secondary index record is marked for deletion or the secondary index page is updated by a newer transaction, the covering index technique is not used. Instead of returning values from the index structure, `InnoDB` looks up the record in the clustered index.

However, if the index condition pushdown (ICP) optimization is enabled, and parts of the `WHERE` condition can be evaluated using only fields from the index, the MySQL server still pushes this part of the `WHERE` condition down to the storage engine where it is evaluated using the index. If no matching records are found, the clustered index lookup is avoided. If matching records are found, even among delete-marked records, `InnoDB` looks up the record in the clustered index.

14.4 InnoDB Architecture

The following diagram shows in-memory and on-disk structures that comprise the `InnoDB` storage engine architecture. For information about each structure, see Section 14.5, “InnoDB In-Memory Structures”, and Section 14.6, “InnoDB On-Disk Structures”.

Figure 14.1 InnoDB Architecture



14.5 InnoDB In-Memory Structures

This section describes `InnoDB` in-memory structures and related topics.

14.5.1 Buffer Pool

The buffer pool is an area in main memory where `InnoDB` caches table and index data as it is accessed. The buffer pool permits frequently used data to be processed directly from memory, which speeds up processing. On dedicated servers, up to 80% of physical memory is often assigned to the buffer pool.

For efficiency of high-volume read operations, the buffer pool is divided into pages that can potentially hold multiple rows. For efficiency of cache management, the buffer pool is implemented as a linked list of pages; data that is rarely used is aged out of the cache using a variation of the LRU algorithm.

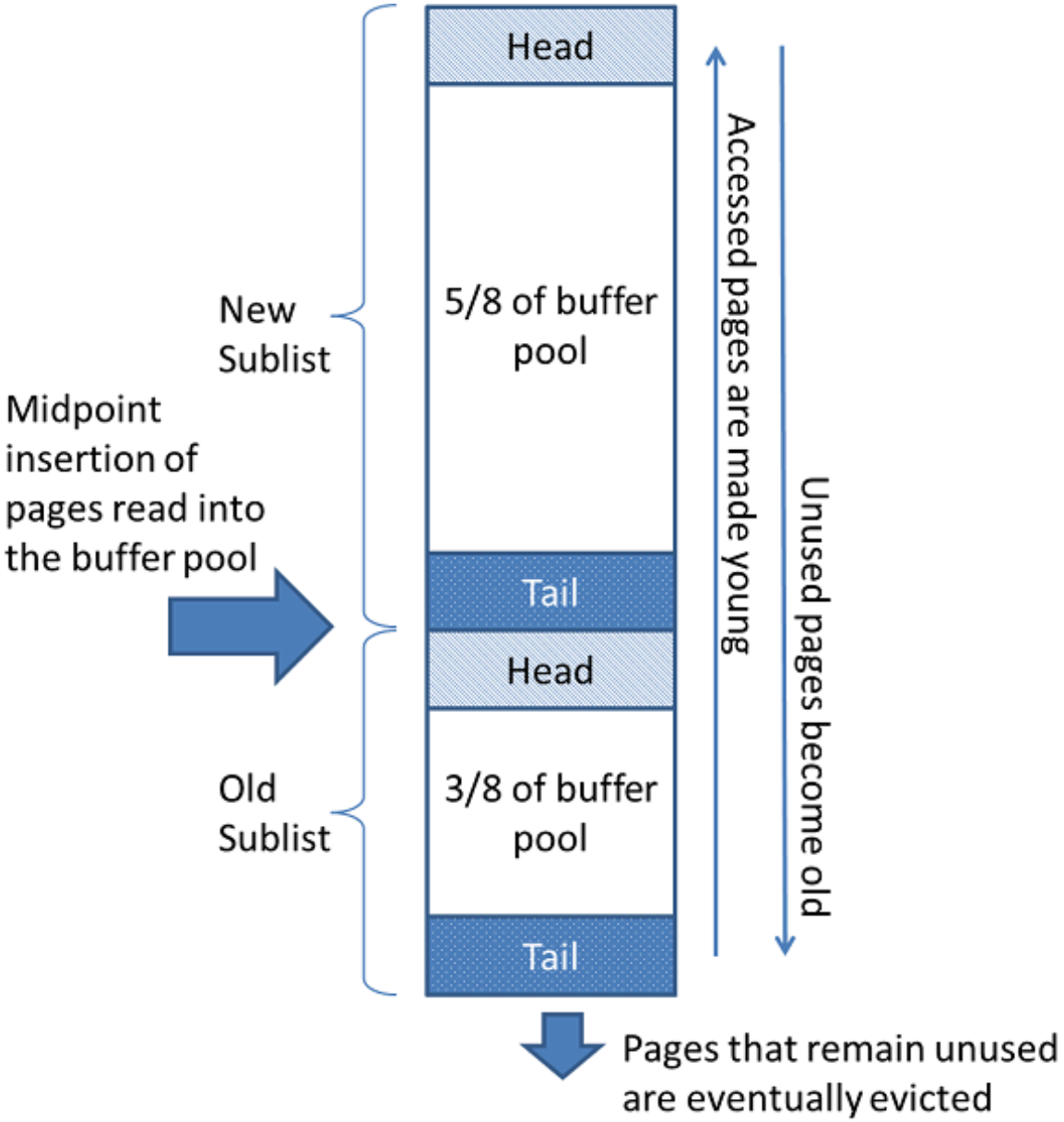
Knowing how to take advantage of the buffer pool to keep frequently accessed data in memory is an important aspect of MySQL tuning.

Buffer Pool LRU Algorithm

The buffer pool is managed as a list using a variation of the least recently used (LRU) algorithm. When room is needed to add a new page to the buffer pool, the least recently used page is evicted and a new page is added to the middle of the list. This midpoint insertion strategy treats the list as two sublists:

- At the head, a sublist of new (“young”) pages that were accessed recently
- At the tail, a sublist of old pages that were accessed less recently

Figure 14.2 Buffer Pool List



The algorithm keeps frequently used pages in the new sublist. The old sublist contains less frequently used pages; these pages are candidates for eviction.

By default, the algorithm operates as follows:

- 3/8 of the buffer pool is devoted to the old sublist.
- The midpoint of the list is the boundary where the tail of the new sublist meets the head of the old sublist.
- When `InnoDB` reads a page into the buffer pool, it initially inserts it at the midpoint (the head of the old sublist). A page can be read because it is required for a user-initiated operation such as an SQL query, or as part of a read-ahead operation performed automatically by `InnoDB`.
- Accessing a page in the old sublist makes it “young”, moving it to the head of the new sublist. If the page was read because it was required by a user-initiated operation, the first access occurs immediately and the page is made young. If the page was read due to a read-ahead operation, the first access does not occur immediately and might not occur at all before the page is evicted.

- As the database operates, pages in the buffer pool that are not accessed “age” by moving toward the tail of the list. Pages in both the new and old sublists age as other pages are made new. Pages in the old sublist also age as pages are inserted at the midpoint. Eventually, a page that remains unused reaches the tail of the old sublist and is evicted.

By default, pages read by queries are immediately moved into the new sublist, meaning they stay in the buffer pool longer. A table scan, performed for a `mysqldump` operation or a `SELECT` statement with no `WHERE` clause, for example, can bring a large amount of data into the buffer pool and evict an equivalent amount of older data, even if the new data is never used again. Similarly, pages that are loaded by the read-ahead background thread and accessed only once are moved to the head of the new list. These situations can push frequently used pages to the old sublist where they become subject to eviction. For information about optimizing this behavior, see Section 14.8.3.2, “Making the Buffer Pool Scan Resistant”, and Section 14.8.3.3, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”.

`InnoDB` Standard Monitor output contains several fields in the `BUFFER POOL AND MEMORY` section regarding operation of the buffer pool LRU algorithm. For details, see Monitoring the Buffer Pool Using the InnoDB Standard Monitor.

Buffer Pool Configuration

You can configure the various aspects of the buffer pool to improve performance.

- Ideally, you set the size of the buffer pool to as large a value as practical, leaving enough memory for other processes on the server to run without excessive paging. The larger the buffer pool, the more `InnoDB` acts like an in-memory database, reading data from disk once and then accessing the data from memory during subsequent reads. Buffer pool size is configured using the `innodb_buffer_pool_size` configuration option.
- On 64-bit systems with sufficient memory, you can split the buffer pool into multiple parts to minimize contention for memory structures among concurrent operations. For details, see Section 14.8.3.1, “Configuring Multiple Buffer Pool Instances”.
- You can keep frequently accessed data in memory regardless of sudden spikes of activity from operations that would bring large amounts of infrequently accessed data into the buffer pool. For details, see Section 14.8.3.2, “Making the Buffer Pool Scan Resistant”.
- You can control how and when to perform read-ahead requests to prefetch pages into the buffer pool asynchronously in anticipation that the pages may be needed soon. For details, see Section 14.8.3.3, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”.
- You can control when background flushing occurs and whether or not the rate of flushing is dynamically adjusted based on workload. For details, see Section 14.8.3.4, “Configuring Buffer Pool Flushing”.
- You can configure how `InnoDB` preserves the current buffer pool state to avoid a lengthy warmup period after a server restart. For details, see Section 14.8.3.5, “Saving and Restoring the Buffer Pool State”.

Monitoring the Buffer Pool Using the InnoDB Standard Monitor

`InnoDB` Standard Monitor output, which can be accessed using `SHOW ENGINE INNODB STATUS`, provides metrics regarding operation of the buffer pool. Buffer pool metrics are located in the `BUFFER POOL AND MEMORY` section of `InnoDB` Standard Monitor output and appear similar to the following:

```
----------------------
BUFFER POOL AND MEMORY
----------------------
Total memory allocated 2197815296; in additional pool allocated 0
Dictionary memory allocated 155455
Buffer pool size   131071
Free buffers       92158
```

```
Database pages     38770
Old database pages 14271
Modified db pages  619
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 4, not young 0
0.00 youngs/s, 0.00 non-youngs/s
Pages read 322, created 38448, written 42083
0.00 reads/s, 222.30 creates/s, 159.47 writes/s
Buffer pool hit rate 1000 / 1000, young-making rate 0 / 1000 not 0 / 1000
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead
0.00/s
LRU len: 38770, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0]
```

The following table describes buffer pool metrics reported by the `InnoDB` Standard Monitor.

Note

Per second averages provided in `InnoDB` Standard Monitor output are based on the elapsed time since `InnoDB` Standard Monitor output was last printed.

Table 14.2 InnoDB Buffer Pool Metrics

Name	Description
Total memory allocated	The total memory allocated for the buffer pool in bytes.
additional pool allocated	The total memory allocated for the additional pool in bytes.
Dictionary memory allocated	The total memory allocated for the `InnoDB` data dictionary in bytes.
Buffer pool size	The total size in pages allocated to the buffer pool.
Free buffers	The total size in pages of the buffer pool free list.
Database pages	The total size in pages of the buffer pool LRU list.
Old database pages	The total size in pages of the buffer pool old LRU sublist.
Modified db pages	The current number of pages modified in the buffer pool.
Pending reads	The number of buffer pool pages waiting to be read into the buffer pool.
Pending writes LRU	The number of old dirty pages within the buffer pool to be written from the bottom of the LRU list.
Pending writes flush list	The number of buffer pool pages to be flushed during checkpointing.
Pending writes single page	The number of pending independent page writes within the buffer pool.
Pages made young	The total number of pages made young in the buffer pool LRU list (moved to the head of sublist of “new” pages).
Pages made not young	The total number of pages not made young in the buffer pool LRU list (pages that have remained in the “old” sublist without being made young).
youngs/s	The per second average of accesses to old pages in the buffer pool LRU list that have resulted in making pages young. See the notes that follow this table for more information.
non-youngs/s	The per second average of accesses to old pages in the buffer pool LRU list that have resulted in not making pages young. See the notes that follow this table for more information.
Pages read	The total number of pages read from the buffer pool.
Pages created	The total number of pages created within the buffer pool.

Name	Description
Pages written	The total number of pages written from the buffer pool.
reads/s	The per second average number of buffer pool page reads per second.
creates/s	The per second average number of buffer pool pages created per second.
writes/s	The per second average number of buffer pool page writes per second.
Buffer pool hit rate	The buffer pool page hit rate for pages read from the buffer pool memory vs from disk storage.
young-making rate	The average hit rate at which page accesses have resulted in making pages young. See the notes that follow this table for more information.
not (young-making rate)	The average hit rate at which page accesses have not resulted in making pages young. See the notes that follow this table for more information.
Pages read ahead	The per second average of read ahead operations.
Pages evicted without access	The per second average of the pages evicted without being accessed from the buffer pool.
Random read ahead	The per second average of random read ahead operations.
LRU len	The total size in pages of the buffer pool LRU list.
unzip_LRU len	The total size in pages of the buffer pool unzip_LRU list.
I/O sum	The total number of buffer pool LRU list pages accessed, for the last 50 seconds.
I/O cur	The total number of buffer pool LRU list pages accessed.
I/O unzip sum	The total number of buffer pool unzip_LRU list pages accessed.
I/O unzip cur	The total number of buffer pool unzip_LRU list pages accessed.

Notes:

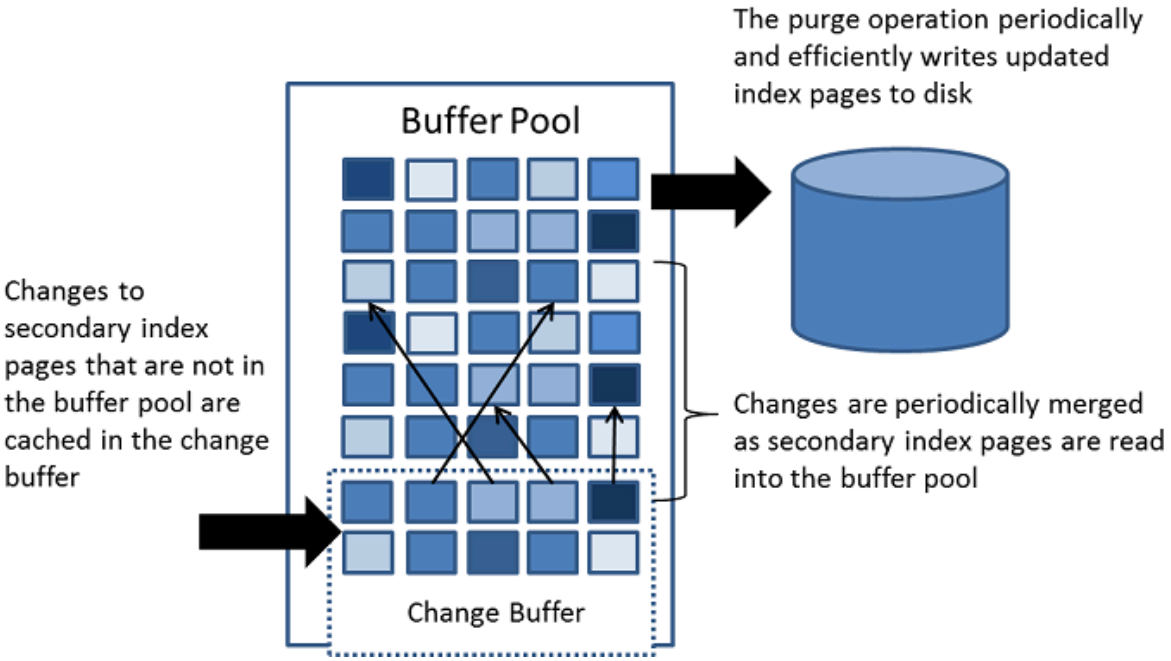
- The `youngs/s` metric is applicable only to old pages. It is based on the number of accesses to pages and not the number of pages. There can be multiple accesses to a given page, all of which are counted. If you see very low `youngs/s` values when there are no large scans occurring, you might need to reduce the delay time or increase the percentage of the buffer pool used for the old sublist. Increasing the percentage makes the old sublist larger, so pages in that sublist take longer to move to the tail, which increases the likelihood that those pages are accessed again and made young.
- The `non-youngs/s` metric is applicable only to old pages. It is based on the number of accesses to pages and not the number of pages. There can be multiple accesses to a given page, all of which are counted. If you do not see a higher `non-youngs/s` value when performing large table scans (and a higher `youngs/s` value), increase the delay value.
- The `young-making` rate accounts for accesses to all buffer pool pages, not just accesses to pages in the old sublist. The `young-making` rate and `not` rate do not normally add up to the overall buffer pool hit rate. Page hits in the old sublist cause pages to move to the new sublist, but page hits in the new sublist cause pages to move to the head of the list only if they are a certain distance from the head.
- `not (young-making rate)` is the average hit rate at which page accesses have not resulted in making pages young due to the delay defined by `innodb_old_blocks_time` not being met, or due to page hits in the new sublist that did not result in pages being moved to the head. This rate accounts for accesses to all buffer pool pages, not just accesses to pages in the old sublist.

Buffer pool server status variables and the `INNODB_BUFFER_POOL_STATS` table provide many of the same buffer pool metrics found in `InnoDB` Standard Monitor output. For more information, see Example 14.10, “Querying the INNODB_BUFFER_POOL_STATS Table”.

14.5.2 Change Buffer

The change buffer is a special data structure that caches changes to secondary index pages when those pages are not in the buffer pool. The buffered changes, which may result from `INSERT`, `UPDATE`, or `DELETE` operations (DML), are merged later when the pages are loaded into the buffer pool by other read operations.

Figure 14.3 Change Buffer



Unlike clustered indexes, secondary indexes are usually nonunique, and inserts into secondary indexes happen in a relatively random order. Similarly, deletes and updates may affect secondary index pages that are not adjacently located in an index tree. Merging cached changes at a later time, when affected pages are read into the buffer pool by other operations, avoids substantial random access I/O that would be required to read secondary index pages into the buffer pool from disk.

Periodically, the purge operation that runs when the system is mostly idle, or during a slow shutdown, writes the updated index pages to disk. The purge operation can write disk blocks for a series of index values more efficiently than if each value were written to disk immediately.

Change buffer merging may take several hours when there are many affected rows and numerous secondary indexes to update. During this time, disk I/O is increased, which can cause a significant slowdown for disk-bound queries. Change buffer merging may also continue to occur after a transaction is committed, and even after a server shutdown and restart (see Section 14.21.2, “Forcing InnoDB Recovery” for more information).

In memory, the change buffer occupies part of the buffer pool. On disk, the change buffer is part of the system tablespace, where index changes are buffered when the database server is shut down.

The type of data cached in the change buffer is governed by the `innodb_change_buffering` variable. For more information, see Configuring Change Buffering. You can also configure the maximum change buffer size. For more information, see Configuring the Change Buffer Maximum Size.

Change buffering is not supported for a secondary index if the index contains a descending index column or if the primary key includes a descending index column.

For answers to frequently asked questions about the change buffer, see Section A.16, “MySQL 5.6 FAQ: InnoDB Change Buffer”.

Configuring Change Buffering

When `INSERT`, `UPDATE`, and `DELETE` operations are performed on a table, the values of indexed columns (particularly the values of secondary keys) are often in an unsorted order, requiring substantial I/O to bring secondary indexes up to date. The change buffer caches changes to secondary index entries when the relevant page is not in the buffer pool, thus avoiding expensive I/O operations by not immediately reading in the page from disk. The buffered changes are merged when the page is loaded into the buffer pool, and the updated page is later flushed to disk. The `InnoDB` main thread merges buffered changes when the server is nearly idle, and during a slow shutdown.

Because it can result in fewer disk reads and writes, the change buffer feature is most valuable for workloads that are I/O-bound, for example applications with a high volume of DML operations such as bulk inserts.

However, the change buffer occupies a part of the buffer pool, reducing the memory available to cache data pages. If the working set almost fits in the buffer pool, or if your tables have relatively few secondary indexes, it may be useful to disable change buffering. If the working data set fits entirely within the buffer pool, change buffering does not impose extra overhead, because it only applies to pages that are not in the buffer pool.

You can control the extent to which `InnoDB` performs change buffering using the `innodb_change_buffering` configuration parameter. You can enable or disable buffering for inserts, delete operations (when index records are initially marked for deletion) and purge operations (when index records are physically deleted). An update operation is a combination of an insert and a delete. The default `innodb_change_buffering` value is `all`.

Permitted `innodb_change_buffering` values include:

- **`all`**

  The default value: buffer inserts, delete-marking operations, and purges.

- **`none`**

  Do not buffer any operations.

- **`inserts`**

  Buffer insert operations.

- **`deletes`**

  Buffer delete-marking operations.

- **`changes`**

  Buffer both inserts and delete-marking operations.

- **`purges`**

  Buffer physical deletion operations that happen in the background.

You can set the `innodb_change_buffering` parameter in the MySQL option file (`my.cnf` or `my.ini`) or change it dynamically with the `SET GLOBAL` statement, which requires privileges sufficient to set global system variables. See Section 5.1.8.1, “System Variable Privileges”. Changing the setting affects the buffering of new operations; the merging of existing buffered entries is not affected.

Configuring the Change Buffer Maximum Size

The `innodb_change_buffer_max_size` variable permits configuring the maximum size of the change buffer as a percentage of the total size of the buffer pool. By default, `innodb_change_buffer_max_size` is set to 25. The maximum setting is 50.

Consider increasing `innodb_change_buffer_max_size` on a MySQL server with heavy insert, update, and delete activity, where change buffer merging does not keep pace with new change buffer entries, causing the change buffer to reach its maximum size limit.

Consider decreasing `innodb_change_buffer_max_size` on a MySQL server with static data used for reporting, or if the change buffer consumes too much of the memory space shared with the buffer pool, causing pages to age out of the buffer pool sooner than desired.

Test different settings with a representative workload to determine an optimal configuration. The `innodb_change_buffer_max_size` setting is dynamic, which permits modifying the setting without restarting the server.

Monitoring the Change Buffer

The following options are available for change buffer monitoring:

- `InnoDB` Standard Monitor output includes change buffer status information. To view monitor data, issue the `SHOW ENGINE INNODB STATUS` statement.

```
mysql> SHOW ENGINE INNODB STATUS\G
```

  Change buffer status information is located under the `INSERT BUFFER AND ADAPTIVE HASH INDEX` heading and appears similar to the following:

```
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf: size 1, free list len 0, seg size 2, 0 merges
merged operations:
 insert 0, delete mark 0, delete 0
discarded operations:
 insert 0, delete mark 0, delete 0
Hash table size 4425293, used cells 32, node heap has 1 buffer(s)
13577.57 hash searches/s, 202.47 non-hash searches/s
```

  For more information, see Section 14.17.3, “InnoDB Standard Monitor and Lock Monitor Output”.

- The `INFORMATION_SCHEMA.INNODB_METRICS` table provides most of the data points found in `InnoDB` Standard Monitor output, plus other data points. To view change buffer metrics and a description of each, issue the following query:

```
mysql> SELECT NAME, COMMENT FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME LIKE '%ibuf%'\G
```

  For `INNODB_METRICS` table usage information, see Section 14.15.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

- The `INFORMATION_SCHEMA.INNODB_BUFFER_PAGE` table provides metadata about each page in the buffer pool, including change buffer index and change buffer bitmap pages. Change buffer pages are identified by `PAGE_TYPE`. `IBUF_INDEX` is the page type for change buffer index pages, and `IBUF_BITMAP` is the page type for change buffer bitmap pages.

Warning

Querying the `INNODB_BUFFER_PAGE` table can introduce significant performance overhead. To avoid impacting performance, reproduce the issue you want to investigate on a test instance and run your queries on the test instance.

For example, you can query the `INNODB_BUFFER_PAGE` table to determine the approximate number of `IBUF_INDEX` and `IBUF_BITMAP` pages as a percentage of total buffer pool pages.

```
mysql> SELECT (SELECT COUNT(*) FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
       WHERE PAGE_TYPE LIKE 'IBUF%') AS change_buffer_pages,
       (SELECT COUNT(*) FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE) AS total_pages,
       (SELECT ((change_buffer_pages/total_pages)*100))
       AS change_buffer_page_percentage;
+---------------------+-------------+-----------------------------+
| change_buffer_pages | total_pages | change_buffer_page_percentage |
+---------------------+-------------+-----------------------------+
|                  25 |        8192 |                      0.3052 |
+---------------------+-------------+-----------------------------+
```

For information about other data provided by the `INNODB_BUFFER_PAGE` table, see Section 21.30.1, “The INFORMATION_SCHEMA INNODB_BUFFER_PAGE Table”. For related usage information, see Section 14.15.5, “InnoDB INFORMATION_SCHEMA Buffer Pool Tables”.

- Performance Schema provides change buffer mutex wait instrumentation for advanced performance monitoring. To view change buffer instrumentation, issue the following query:

```
mysql> SELECT * FROM performance_schema.setup_instruments
       WHERE NAME LIKE '%wait/synch/mutex/innodb/ibuf%';
+-----------------------------------------------------+---------+-------+
| NAME                                                | ENABLED | TIMED |
+-----------------------------------------------------+---------+-------+
| wait/synch/mutex/innodb/ibuf_bitmap_mutex           | YES     | YES   |
| wait/synch/mutex/innodb/ibuf_mutex                  | YES     | YES   |
| wait/synch/mutex/innodb/ibuf_pessimistic_insert_mutex | YES     | YES   |
+-----------------------------------------------------+---------+-------+
```

For information about monitoring `InnoDB` mutex waits, see Section 14.16.1, “Monitoring InnoDB Mutex Waits Using Performance Schema”.

14.5.3 Adaptive Hash Index

The adaptive hash index feature enables `InnoDB` to perform more like an in-memory database on systems with appropriate combinations of workload and sufficient memory for the buffer pool without sacrificing transactional features or reliability. The adaptive hash index feature is enabled by the `innodb_adaptive_hash_index` variable, or turned off at server startup by `--skip-innodb-adaptive-hash-index`.

Based on the observed pattern of searches, a hash index is built using a prefix of the index key. The prefix can be any length, and it may be that only some values in the B-tree appear in the hash index. Hash indexes are built on demand for the pages of the index that are accessed often.

If a table fits almost entirely in main memory, a hash index can speed up queries by enabling direct lookup of any element, turning the index value into a sort of pointer. `InnoDB` has a mechanism that monitors index searches. If `InnoDB` notices that queries could benefit from building a hash index, it does so automatically.

With some workloads, the speedup from hash index lookups greatly outweighs the extra work to monitor index lookups and maintain the hash index structure. Access to the adaptive hash index can sometimes become a source of contention under heavy workloads, such as multiple concurrent joins. Queries with `LIKE` operators and `%` wildcards also tend not to benefit. For workloads that do not benefit from the adaptive hash index feature, turning it off reduces unnecessary performance overhead. Because it is difficult to predict in advance whether the adaptive hash index feature is appropriate for a particular system and workload, consider running benchmarks with it enabled and disabled. Architectural changes in MySQL 5.6 make it more suitable to disable the adaptive hash index feature than in earlier releases.

You can monitor adaptive hash index use and contention in the `SEMAPHORES` section of `SHOW ENGINE INNODB STATUS` output. If there are numerous threads waiting on an RW-latch created in `btr0sea.c`, it might be useful to disable the adaptive hash index feature.

For information about the performance characteristics of hash indexes, see Section 8.3.8, “Comparison of B-Tree and Hash Indexes”.

14.5.4 Log Buffer

The log buffer is the memory area that holds data to be written to the log files on disk. Log buffer size is defined by the `innodb_log_buffer_size` variable. The default size is 16MB. The contents of the log buffer are periodically flushed to disk. A large log buffer enables large transactions to run without the need to write redo log data to disk before the transactions commit. Thus, if you have transactions that update, insert, or delete many rows, increasing the size of the log buffer saves disk I/O.

The `innodb_flush_log_at_trx_commit` variable controls how the contents of the log buffer are written and flushed to disk. The `innodb_flush_log_at_timeout` variable controls log flushing frequency.

For related information, see Memory Configuration, and Section 8.5.4, “Optimizing InnoDB Redo Logging”.

14.6 InnoDB On-Disk Structures

This section describes `InnoDB` on-disk structures and related topics.

14.6.1 Tables

This section covers topics related to `InnoDB` tables.

14.6.1.1 Creating InnoDB Tables

To create an `InnoDB` table, use the `CREATE TABLE` statement.

```
CREATE TABLE t1 (a INT, b CHAR (20), PRIMARY KEY (a)) ENGINE=InnoDB;
```

You do not need to specify the `ENGINE=InnoDB` clause if `InnoDB` is defined as the default storage engine, which it is by default. To check the default storage engine, issue the following statement:

```
mysql> SELECT @@default_storage_engine;
+--------------------------+
| @@default_storage_engine |
+--------------------------+
| InnoDB                   |
+--------------------------+
```

You might still use `ENGINE=InnoDB` clause if you plan to use `mysqldump` or replication to replay the `CREATE TABLE` statement on a server where the default storage engine is not `InnoDB`.

An `InnoDB` table and its indexes can be created in the system tablespace or in a file-per-table tablespace. When `innodb_file_per_table` is enabled, which is the default as of MySQL 5.6.6, an `InnoDB` table is implicitly created in an individual file-per-table tablespace. Conversely, when `innodb_file_per_table` is disabled, an `InnoDB` table is implicitly created in the `InnoDB` system tablespace.

When you create an `InnoDB` table, MySQL creates a .frm file in the database directory under the MySQL data directory. For more information about `.frm` files, see InnoDB Tables and .frm Files. For a table created in a file-per-table tablespace, MySQL also creates an .ibd tablespace file in the database directory, by default. A table created in the `InnoDB` system tablespace is created in an existing ibdata file, which resides in the MySQL data directory.

Internally, `InnoDB` adds an entry for each table to the `InnoDB` data dictionary. The entry includes the database name. For example, if table `t1` is created in the `test` database, the data dictionary entry

for the database name is `'test/t1'`. This means you can create a table of the same name (`t1`) in a different database, and the table names do not collide inside `InnoDB`.

InnoDB Tables and .frm Files

MySQL stores data dictionary information for tables in .frm files in database directories. Unlike other MySQL storage engines, `InnoDB` also encodes information about the table in its own internal data dictionary inside the system tablespace. When MySQL drops a table or a database, it deletes one or more `.frm` files as well as the corresponding entries inside the `InnoDB` data dictionary. You cannot move `InnoDB` tables between databases simply by moving the `.frm` files. For information about moving `InnoDB` tables, see Section 14.6.1.4, “Moving or Copying InnoDB Tables”.

InnoDB Tables and Row Formats

The default row format of an `InnoDB` table is `Compact`. Although this row format is fine for basic experimentation, consider using the `Dynamic` or `Compressed` row format to take advantage of `InnoDB` features such as table compression and efficient off-page storage of long column values. Using these row formats requires that `innodb_file_per_table` is enabled (the default as of MySQL 5.6.6) and that `innodb_file_format` is set to Barracuda:

```
SET GLOBAL innodb_file_per_table=1;
SET GLOBAL innodb_file_format=barracuda;
CREATE TABLE t3 (a INT, b CHAR (20), PRIMARY KEY (a)) ROW_FORMAT=DYNAMIC;
CREATE TABLE t4 (a INT, b CHAR (20), PRIMARY KEY (a)) ROW_FORMAT=COMPRESSED;
```

For more information about `InnoDB` row formats, see Section 14.11, “InnoDB Row Formats”. For how to determine the row format of an `InnoDB` table and the physical characteristics of `InnoDB` row formats, see Section 14.11, “InnoDB Row Formats”.

InnoDB Tables and Primary Keys

Always define a primary key for an `InnoDB` table, specifying the column or columns that:

- Are referenced by the most important queries.
- Are never left blank.
- Never have duplicate values.
- Rarely if ever change value once inserted.

For example, in a table containing information about people, you would not create a primary key on `(firstname, lastname)` because more than one person can have the same name, some people have blank last names, and sometimes people change their names. With so many constraints, often there is not an obvious set of columns to use as a primary key, so you create a new column with a numeric ID to serve as all or part of the primary key. You can declare an auto-increment column so that ascending values are filled in automatically as rows are inserted:

```
# The value of ID can act like a pointer between related items in different tables.
CREATE TABLE t5 (id INT AUTO_INCREMENT, b CHAR (20), PRIMARY KEY (id));

# The primary key can consist of more than one column. Any autoinc column must come first.
CREATE TABLE t6 (id INT AUTO_INCREMENT, a INT, b CHAR (20), PRIMARY KEY (id,a));
```

Although the table works correctly without defining a primary key, the primary key is involved with many aspects of performance and is a crucial design aspect for any large or frequently used table. It is recommended that you always specify a primary key in the `CREATE TABLE` statement. If you create the table, load data, and then run `ALTER TABLE` to add a primary key later, that operation is much slower than defining the primary key when creating the table.

Viewing InnoDB Table Properties

To view the properties of an `InnoDB` table, issue a `SHOW TABLE STATUS` statement:

```
mysql> SHOW TABLE STATUS FROM test LIKE 't%' \G;
*************************** 1. row ***************************
           Name: t1
         Engine: InnoDB
        Version: 10
     Row_format: Compact
           Rows: 0
 Avg_row_length: 0
    Data_length: 16384
Max_data_length: 0
   Index_length: 0
      Data_free: 0
 Auto_increment: NULL
    Create_time: 2015-03-16 16:26:52
    Update_time: NULL
     Check_time: NULL
      Collation: latin1_swedish_ci
       Checksum: NULL
 Create_options:
        Comment:
1 row in set (0.00 sec)
```

For information about `SHOW TABLE STATUS` output, see Section 13.7.5.37, “SHOW TABLE STATUS Statement”.

`InnoDB` table properties may also be queried using the `InnoDB` Information Schema system tables:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_TABLES WHERE NAME='test/t1' \G
*************************** 1. row ***************************
     TABLE_ID: 42
         NAME: test/t1
         FLAG: 1
       N_COLS: 5
        SPACE: 24
  FILE_FORMAT: Antelope
   ROW_FORMAT: Compact
ZIP_PAGE_SIZE: 0
1 row in set (0.02 sec)
```

For more information, see Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”.

14.6.1.2 Creating Tables Externally

There are different reasons for creating `InnoDB` tables externally; that is, creating tables outside of the data directory. Those reasons might include space management, I/O optimization, or placing tables on a storage device with particular performance or capacity characteristics, for example.

You can create an `InnoDB` table in an external directory by specifying a `DATA DIRECTORY` clause in the `CREATE TABLE` statement.

```
CREATE TABLE t1 (c1 INT PRIMARY KEY) DATA DIRECTORY = '/external/directory';
```

The `DATA DIRECTORY` clause is supported for tables created in file-per-table tablespaces. Tables are implicitly created in file-per-table tablespaces when the `innodb_file_per_table` variable is enabled, which it is by default.

```
mysql> SELECT @@innodb_file_per_table;
+-------------------------+
| @@innodb_file_per_table |
+-------------------------+
|                       1 |
+-------------------------+
```

For more information about file-per-table tablespaces, see Section 14.6.3.2, “File-Per-Table Tablespaces”.

Be sure of the directory location you choose, as the `DATA DIRECTORY` clause cannot be used with `ALTER TABLE` to change the location later.

When you specify a `DATA DIRECTORY` clause in a `CREATE TABLE` statement, the table's data file (*`table_name`*`.ibd`) is created in a schema directory under the specified directory, and an `.isl` file (*`table_name`*`.isl`) that contains the data file path is created in the schema directory under the MySQL data directory. An `.isl` file is similar in function to a symbolic link. (Actual symbolic links are not supported for use with `InnoDB` data files.)

The following example demonstrates creating a table in an external directory using the `DATA DIRECTORY` clause. It is assumed that the `innodb_file_per_table` variable is enabled.

```
mysql> USE test;
Database changed

mysql> CREATE TABLE t1 (c1 INT PRIMARY KEY) DATA DIRECTORY = '/external/directory';

# MySQL creates the table's data file in a schema directory
# under the external directory

shell> cd /external/directory/test
shell> ls
t1.ibd

# An .isl file that contains the data file path is created
# in the schema directory under the MySQL data directory

shell> cd /path/to/mysql/data/test
shell> ls
db.opt  t1.frm  t1.isl
```

Usage Notes:

- MySQL initially holds the tablespace data file open, preventing you from dismounting the device, but might eventually close the file if the server is busy. Be careful not to accidentally dismount an external device while MySQL is running, or start MySQL while the device is disconnected. Attempting to access a table when the associated data file is missing causes a serious error that requires a server restart.

  A server restart might fail if the data file is not found at the expected path. In this case, manually remove the `.isl` file from the schema directory. After restarting, drop the table to remove the `.frm` file and the information about the table from the data dictionary.

- Before placing a table on an NFS-mounted volume, review potential issues outlined in Using NFS with MySQL.

- If using an LVM snapshot, file copy, or other file-based mechanism to back up the table's data file, always use the `FLUSH TABLES ... FOR EXPORT` statement first to ensure that all changes buffered in memory are flushed to disk before the backup occurs.

- Using the `DATA DIRECTORY` clause to create a table in an external directory is an alternative to using symbolic links, which `InnoDB` does not support.

- The `DATA DIRECTORY` clause is not supported in a replication environment where the source and replica reside on the same host. The `DATA DIRECTORY` clause requires a full directory path. Replicating the path in this case would cause the source and replica to create the table in same location.

14.6.1.3 Importing InnoDB Tables

This section describes how to import tables using the *Transportable Tablespaces* feature, which permits importing tables that reside in file-per-table tablespaces. There are many reasons why you might want to import tables:

- To run reports on a non-production MySQL server instance to avoid placing extra load on a production server.

- To copy data to a new replica server.

- To restore a table from a backed-up tablespace file.
- As a faster way of moving data than importing a dump file, which requires reinserting data and rebuilding indexes.
- To move a data to a server with storage media that is better suited to your storage requirements. For example, you might move busy tables to an SSD device, or move large tables to a high-capacity HDD device.

The *Transportable Tablespaces* feature is described under the following topics in this section:

- Prerequisites
- Importing a Tablespace
- Limitations
- Usage Notes
- Internals

Prerequisites

- The `innodb_file_per_table` variable must be enabled, which it is by default.
- The page size of the tablespace must match the page size of the destination MySQL server instance. `InnoDB` page size is defined by the `innodb_page_size` variable, which is configured when initializing a MySQL server instance.
- If the table is in a foreign key relationship, `foreign_key_checks` must be disabled before executing `DISCARD TABLESPACE`. Also, you should export all foreign key related tables at the same logical point in time, as the `ALTER TABLE ... IMPORT TABLESPACE` feature does not enforce foreign key constraints on imported data. To do so, stop updating the related tables, commit all transactions, acquire shared locks on the tables, and perform the export operations.
- When importing a table from another MySQL server instance, both MySQL server instances must have General Availability (GA) status and must be the same version. Otherwise, the table must be created on the same MySQL server instance into which it is being imported.
- If the table was created outside of the data directory by specifying the `DATA DIRECTORY` clause in the `CREATE TABLE` statement, the table that you replace on the destination instance must be defined with the same `DATA DIRECTORY` clause. A schema mismatch error is reported if the clauses do not match. To determine if the source table was defined with a `DATA DIRECTORY` clause, use `SHOW CREATE TABLE` to view the table definition. For information about using the `DATA DIRECTORY` clause, see Section 14.6.1.2, “Creating Tables Externally”.

Importing a Tablespace

This example demonstrates how to import a table that resides in a file-per-table tablespace.

1. On the destination instance, create a table with the same definition as the table you intend to import. (You can obtain the table definition using `SHOW CREATE TABLE` syntax.) If the table definition does not match, a schema mismatch error is reported when you attempt the import operation.

```
mysql> USE test;
mysql> CREATE TABLE t1 (c1 INT) ENGINE=INNODB;
```

2. On the destination instance, discard the tablespace of the table that you just created. (Before importing, you must discard the tablespace of the receiving table.)

```
mysql> ALTER TABLE t1 DISCARD TABLESPACE;
```

3. On the source instance, run `FLUSH TABLES ... FOR EXPORT` to quiesce the table you intend to import. When a table is quiesced, only read-only transactions are permitted on the table.

```
mysql> USE test;
mysql> FLUSH TABLES t1 FOR EXPORT;
```

`FLUSH TABLES ... FOR EXPORT` ensures that changes to the named table are flushed to disk so that a binary table copy can be made while the server is running. When `FLUSH TABLES ... FOR EXPORT` is run, `InnoDB` generates a `.cfg` metadata file in the schema directory of the table. The `.cfg` file contains metadata that is used for schema verification during the import operation.

4. Copy the `.ibd` file and `.cfg` metadata file from the source instance to the destination instance. For example:

```
shell> scp /path/to/datadir/test/t1.{ibd,cfg} destination-server:/path/to/datadir/test
```

The `.ibd` file and `.cfg` file must be copied before releasing the shared locks, as described in the next step.

5. On the source instance, use `UNLOCK TABLES` to release the locks acquired by the `FLUSH TABLES ... FOR EXPORT` statement:

```
mysql> USE test;
mysql> UNLOCK TABLES;
```

6. On the destination instance, import the tablespace:

```
mysql> USE test;
mysql> ALTER TABLE t1 IMPORT TABLESPACE;
```

Limitations

- The *Transportable Tablespaces* feature is only supported for tables that reside in file-per-table tablespaces. It is not supported for the tables that reside in the system tablespace. Tables in the system tablespace cannot be quiesced.
- `DISCARD TABLESPACE` is not supported for partitioned tables, which means that importing partitioned tables is not supported. If you run `ALTER TABLE ... DISCARD TABLESPACE` on a partitioned table, the following error is returned: `ERROR 1031 (HY000): Table storage engine for 'part' doesn't have this option`.
- `FLUSH TABLES ... FOR EXPORT` is not supported on tables with a `FULLTEXT` index, as full-text search auxiliary tables cannot be flushed. After importing a table with a `FULLTEXT` index, run `OPTIMIZE TABLE` to rebuild the `FULLTEXT` indexes. Alternatively, drop `FULLTEXT` indexes before the export operation and recreate the indexes after importing the table on the destination instance.

Usage Notes

- A `.cfg` metadata file is not required to import a table. However, metadata checks are not performed when importing without a `.cfg` file, and a warning similar to the following is issued:

```
Message: InnoDB: IO Read error: (2, No such file or directory) Error opening '.\
test\t.cfg', will attempt to import without schema verification
1 row in set (0.00 sec)
```

Importing without a `.cfg` metadata file should only be considered if no schema mismatches are expected. The ability to import without a `.cfg` file could be useful in crash recovery scenarios where metadata is not accessible.

- On Windows, `InnoDB` stores database, tablespace, and table names internally in lowercase. To avoid import problems on case-sensitive operating systems such as Linux and Unix, create all databases, tablespaces, and tables using lowercase names. A convenient way to accomplish this is to add `lower_case_table_names=1` to the `[mysqld]` section of your `my.cnf` or `my.ini` file before creating databases, tablespaces, or tables:

```
[mysqld]
lower_case_table_names=1
```

Internals

The following information describes internals and messages written to the error log during a table import procedure.

When `ALTER TABLE ... DISCARD TABLESPACE` is run on the destination instance:

- The table is locked in X mode.
- The tablespace is detached from the table.

When `FLUSH TABLES ... FOR EXPORT` is run on the source instance:

- The table being flushed for export is locked in shared mode.
- The purge coordinator thread is stopped.
- Dirty pages are synchronized to disk.
- Table metadata is written to the binary `.cfg` file.

Expected error log messages for this operation:

```
[Note] InnoDB: Sync to disk of '"test"."t1"' started.
[Note] InnoDB: Stopping purge
[Note] InnoDB: Writing table metadata to './test/t1.cfg'
[Note] InnoDB: Table '"test"."t1"' flushed to disk
```

When `UNLOCK TABLES` is run on the source instance:

- The binary `.cfg` file is deleted.
- The shared lock on the table being imported is released and the purge coordinator thread is restarted.

Expected error log messages for this operation:

```
[Note] InnoDB: Deleting the meta-data file './test/t1.cfg'
[Note] InnoDB: Resuming purge
```

When `ALTER TABLE ... IMPORT TABLESPACE` is run on the destination instance, the import algorithm performs the following operations for the tablespace being imported:

- Each tablespace page is checked for corruption.
- The space ID and log sequence numbers (LSNs) on each page are updated.
- Flags are validated and LSN updated for the header page.
- Btree pages are updated.
- The page state is set to dirty so that it is written to disk.

Expected error log messages for this operation:

```
[Note] InnoDB: Importing tablespace for table 'test/t1' that was exported
from host 'host_name'
[Note] InnoDB: Phase I - Update all pages
[Note] InnoDB: Sync to disk
[Note] InnoDB: Sync to disk - done!
[Note] InnoDB: Phase III - Flush changes to disk
[Note] InnoDB: Phase IV - Flush complete
```

Note

You may also receive a warning that a tablespace is discarded (if you discarded the tablespace for the destination table) and a message stating that statistics could not be calculated due to a missing `.ibd` file:

```
[Warning] InnoDB: Table "test"."t1" tablespace is set as discarded.
7f34d9a37700 InnoDB: cannot calculate statistics for table
"test"."t1" because the .ibd file is missing. For help, please refer to
http://dev.mysql.com/doc/refman/5.6/en/innodb-troubleshooting.html
```

14.6.1.4 Moving or Copying InnoDB Tables

This section describes techniques for moving or copying some or all `InnoDB` tables to a different server or instance. For example, you might move an entire MySQL instance to a larger, faster server; you might clone an entire MySQL instance to a new replica server; you might copy individual tables to another instance to develop and test an application, or to a data warehouse server to produce reports.

On Windows, `InnoDB` always stores database and table names internally in lowercase. To move databases in a binary format from Unix to Windows or from Windows to Unix, create all databases and tables using lowercase names. A convenient way to accomplish this is to add the following line to the `[mysqld]` section of your `my.cnf` or `my.ini` file before creating any databases or tables:

```
[mysqld]
lower_case_table_names=1
```

Techniques for moving or copying `InnoDB` tables include:

- Importing Tables
- MySQL Enterprise Backup
- Copying Data Files (Cold Backup Method)
- Restoring from a Logical Backup

Importing Tables

A table that resides in a file-per-table tablespace can be imported from another MySQL server instance or from a backup using the *Transportable Tablespace* feature. See Section 14.6.1.3, “Importing InnoDB Tables”.

MySQL Enterprise Backup

The MySQL Enterprise Backup product lets you back up a running MySQL database with minimal disruption to operations while producing a consistent snapshot of the database. When MySQL Enterprise Backup is copying tables, reads and writes can continue. In addition, MySQL Enterprise Backup can create compressed backup files, and back up subsets of tables. In conjunction with the MySQL binary log, you can perform point-in-time recovery. MySQL Enterprise Backup is included as part of the MySQL Enterprise subscription.

For more details about MySQL Enterprise Backup, see Section 24.2, “MySQL Enterprise Backup Overview”.

Copying Data Files (Cold Backup Method)

You can move an `InnoDB` database simply by copying all the relevant files listed under "Cold Backups" in Section 14.18.1, “InnoDB Backup”.

`InnoDB` data and log files are binary-compatible on all platforms having the same floating-point number format. If the floating-point formats differ but you have not used `FLOAT` or `DOUBLE` data types in your tables, then the procedure is the same: simply copy the relevant files.

When you move or copy `.ibd` files, the database directory name must be the same on the source and destination systems. The table definition stored in the `InnoDB` shared tablespace includes the database name. The transaction IDs and log sequence numbers stored in the tablespace files also differ between databases.

To move an `.ibd` file and the associated table from one database to another, use a `RENAME TABLE` statement:

```
RENAME TABLE db1.tbl_name TO db2.tbl_name;
```

If you have a “clean” backup of an `.ibd` file, you can restore it to the MySQL installation from which it originated as follows:

1. The table must not have been dropped or truncated since you copied the `.ibd` file, because doing so changes the table ID stored inside the tablespace.
2. Issue this `ALTER TABLE` statement to delete the current `.ibd` file:

   ```
   ALTER TABLE tbl_name DISCARD TABLESPACE;
   ```

3. Copy the backup `.ibd` file to the proper database directory.
4. Issue this `ALTER TABLE` statement to tell `InnoDB` to use the new `.ibd` file for the table:

   ```
   ALTER TABLE tbl_name IMPORT TABLESPACE;
   ```

Note

The `ALTER TABLE ... IMPORT TABLESPACE` feature does not enforce foreign key constraints on imported data.

In this context, a “clean” `.ibd` file backup is one for which the following requirements are satisfied:

- There are no uncommitted modifications by transactions in the `.ibd` file.
- There are no unmerged insert buffer entries in the `.ibd` file.
- Purge has removed all delete-marked index records from the `.ibd` file.
- `mysqld` has flushed all modified pages of the `.ibd` file from the buffer pool to the file.

You can make a clean backup `.ibd` file using the following method:

1. Stop all activity from the `mysqld` server and commit all transactions.
2. Wait until `SHOW ENGINE INNODB STATUS` shows that there are no active transactions in the database, and the main thread status of `InnoDB` is `Waiting for server activity`. Then you can make a copy of the `.ibd` file.

Another method for making a clean copy of an `.ibd` file is to use the MySQL Enterprise Backup product:

1. Use MySQL Enterprise Backup to back up the `InnoDB` installation.
2. Start a second `mysqld` server on the backup and let it clean up the `.ibd` files in the backup.

Restoring from a Logical Backup

You can use a utility such as `mysqldump` to perform a logical backup, which produces a set of SQL statements that can be executed to reproduce the original database object definitions and table data for transfer to another SQL server. Using this method, it does not matter whether the formats differ or if your tables contain floating-point data.

To improve the performance of this method, disable `autocommit` when importing data. Perform a commit only after importing an entire table or segment of a table.

14.6.1.5 Converting Tables from MyISAM to InnoDB

If you have `MyISAM` tables that you want to convert to `InnoDB` for better reliability and scalability, review the following guidelines and tips before converting.

- Adjusting Memory Usage for MyISAM and InnoDB
- Handling Too-Long Or Too-Short Transactions
- Handling Deadlocks
- Planning the Storage Layout
- Converting an Existing Table
- Cloning the Structure of a Table
- Transferring Existing Data
- Storage Requirements
- Defining a PRIMARY KEY for Each Table
- Application Performance Considerations
- Understanding Files Associated with InnoDB Tables

Adjusting Memory Usage for MyISAM and InnoDB

As you transition away from `MyISAM` tables, lower the value of the `key_buffer_size` configuration option to free memory no longer needed for caching results. Increase the value of the `innodb_buffer_pool_size` configuration option, which performs a similar role of allocating cache memory for `InnoDB` tables. The `InnoDB` buffer pool caches both table data and index data, speeding up lookups for queries and keeping query results in memory for reuse. For guidance regarding buffer pool size configuration, see Section 8.12.4.1, “How MySQL Uses Memory”.

On a busy server, run benchmarks with the query cache turned off. The `InnoDB` buffer pool provides similar benefits, so the query cache might be tying up memory unnecessarily. For information about the query cache, see Section 8.10.3, “The MySQL Query Cache”.

Handling Too-Long Or Too-Short Transactions

Because `MyISAM` tables do not support transactions, you might not have paid much attention to the `autocommit` configuration option and the `COMMIT` and `ROLLBACK` statements. These keywords are important to allow multiple sessions to read and write `InnoDB` tables concurrently, providing substantial scalability benefits in write-heavy workloads.

While a transaction is open, the system keeps a snapshot of the data as seen at the beginning of the transaction, which can cause substantial overhead if the system inserts, updates, and deletes millions of rows while a stray transaction keeps running. Thus, take care to avoid transactions that run for too long:

- If you are using a `mysql` session for interactive experiments, always `COMMIT` (to finalize the changes) or `ROLLBACK` (to undo the changes) when finished. Close down interactive sessions rather than leave them open for long periods, to avoid keeping transactions open for long periods by accident.
- Make sure that any error handlers in your application also `ROLLBACK` incomplete changes or `COMMIT` completed changes.
- `ROLLBACK` is a relatively expensive operation, because `INSERT`, `UPDATE`, and `DELETE` operations are written to `InnoDB` tables prior to the `COMMIT`, with the expectation that most changes are committed successfully and rollbacks are rare. When experimenting with large volumes of data, avoid making changes to large numbers of rows and then rolling back those changes.
- When loading large volumes of data with a sequence of `INSERT` statements, periodically `COMMIT` the results to avoid having transactions that last for hours. In typical load operations for data warehousing, if something goes wrong, you truncate the table (using `TRUNCATE TABLE`) and start over from the beginning rather than doing a `ROLLBACK`.

The preceding tips save memory and disk space that can be wasted during too-long transactions. When transactions are shorter than they should be, the problem is excessive I/O. With each `COMMIT`, MySQL makes sure each change is safely recorded to disk, which involves some I/O.

- For most operations on `InnoDB` tables, you should use the setting `autocommit=0`. From an efficiency perspective, this avoids unnecessary I/O when you issue large numbers of consecutive `INSERT`, `UPDATE`, or `DELETE` statements. From a safety perspective, this allows you to issue a `ROLLBACK` statement to recover lost or garbled data if you make a mistake on the `mysql` command line, or in an exception handler in your application.
- The time when `autocommit=1` is suitable for `InnoDB` tables is when running a sequence of queries for generating reports or analyzing statistics. In this situation, there is no I/O penalty related to `COMMIT` or `ROLLBACK`, and `InnoDB` can automatically optimize the read-only workload.
- If you make a series of related changes, finalize all the changes at once with a single `COMMIT` at the end. For example, if you insert related pieces of information into several tables, do a single `COMMIT` after making all the changes. Or if you run many consecutive `INSERT` statements, do a single `COMMIT` after all the data is loaded; if you are doing millions of `INSERT` statements, perhaps split up the huge transaction by issuing a `COMMIT` every ten thousand or hundred thousand records, so the transaction does not grow too large.
- Remember that even a `SELECT` statement opens a transaction, so after running some report or debugging queries in an interactive `mysql` session, either issue a `COMMIT` or close the `mysql` session.

Handling Deadlocks

You might see warning messages referring to “deadlocks” in the MySQL error log, or the output of `SHOW ENGINE INNODB STATUS`. Despite the scary-sounding name, a deadlock is not a serious issue for `InnoDB` tables, and often does not require any corrective action. When two transactions start modifying multiple tables, accessing the tables in a different order, they can reach a state where each transaction is waiting for the other and neither can proceed. MySQL immediately detects this condition and cancels (rolls back) the “smaller” transaction, allowing the other to proceed.

Your applications do need error-handling logic to restart a transaction that is forcibly cancelled like this. When you re-issue the same SQL statements as before, the original timing issue no longer applies. Either the other transaction has already finished and yours can proceed, or the other transaction is still in progress and your transaction waits until it finishes.

If deadlock warnings occur constantly, you might review the application code to reorder the SQL operations in a consistent way, or to shorten the transactions. You can test with the `innodb_print_all_deadlocks` option enabled to see all deadlock warnings in the MySQL error log, rather than only the last warning in the `SHOW ENGINE INNODB STATUS` output.

For more information, see Section 14.7.5, “Deadlocks in InnoDB”.

Planning the Storage Layout

To get the best performance from `InnoDB` tables, you can adjust a number of parameters related to storage layout.

When you convert `MyISAM` tables that are large, frequently accessed, and hold vital data, investigate and consider the `innodb_file_per_table`, `innodb_file_format`, and `innodb_page_size` configuration options, and the `ROW_FORMAT` and `KEY_BLOCK_SIZE` clauses of the `CREATE TABLE` statement.

During your initial experiments, the most important setting is `innodb_file_per_table`. When this setting is enabled, which is the default as of MySQL 5.6.6, new `InnoDB` tables are implicitly created in file-per-table tablespaces. In contrast with the `InnoDB` system tablespace, file-per-table tablespaces allow disk space to be reclaimed by the operating system when a table is truncated or dropped. File-per-table tablespaces also support the Barracuda file format and associated features such as table

compression, efficient off-page storage for long variable-length columns, and large index prefixes. For more information, see Section 14.6.3.2, “File-Per-Table Tablespaces”.

Converting an Existing Table

To convert a non-`InnoDB` table to use `InnoDB` use `ALTER TABLE`:

```
ALTER TABLE table_name ENGINE=InnoDB;
```

Warning

Do *not* convert MySQL system tables in the `mysql` database from `MyISAM` to `InnoDB` tables. This is an unsupported operation. If you do this, MySQL does not restart until you restore the old system tables from a backup or regenerate them by reinitializing the data directory (see Section 2.10.1, “Initializing the Data Directory”).

Cloning the Structure of a Table

You might make an `InnoDB` table that is a clone of a MyISAM table, rather than using `ALTER TABLE` to perform conversion, to test the old and new table side-by-side before switching.

Create an empty `InnoDB` table with identical column and index definitions. Use `SHOW CREATE TABLE table_name\G` to see the full `CREATE TABLE` statement to use. Change the `ENGINE` clause to `ENGINE=INNODB`.

Transferring Existing Data

To transfer a large volume of data into an empty `InnoDB` table created as shown in the previous section, insert the rows with `INSERT INTO innodb_table SELECT * FROM myisam_table ORDER BY primary_key_columns`.

You can also create the indexes for the `InnoDB` table after inserting the data. Historically, creating new secondary indexes was a slow operation for InnoDB, but now you can create the indexes after the data is loaded with relatively little overhead from the index creation step.

If you have `UNIQUE` constraints on secondary keys, you can speed up a table import by turning off the uniqueness checks temporarily during the import operation:

```
SET unique_checks=0;
... import operation ...
SET unique_checks=1;
```

For big tables, this saves disk I/O because `InnoDB` can use its change buffer to write secondary index records as a batch. Be certain that the data contains no duplicate keys. `unique_checks` permits but does not require storage engines to ignore duplicate keys.

For better control over the insertion process, you can insert big tables in pieces:

```
INSERT INTO newtable SELECT * FROM oldtable
   WHERE yourkey > something AND yourkey <= somethingelse;
```

After all records are inserted, you can rename the tables.

During the conversion of big tables, increase the size of the `InnoDB` buffer pool to reduce disk I/O, to a maximum of 80% of physical memory. You can also increase the size of `InnoDB` log files.

Storage Requirements

If you intend to make several temporary copies of your data in `InnoDB` tables during the conversion process, it is recommended that you create the tables in file-per-table tablespaces so that you can reclaim the disk space when you drop the tables. When the `innodb_file_per_table` configuration option is enabled (the default), newly created `InnoDB` tables are implicitly created in file-per-table tablespaces.

Whether you convert the MyISAM table directly or create a cloned InnoDB table, make sure that you have sufficient disk space to hold both the old and new tables during the process. **InnoDB tables require more disk space than MyISAM tables.** If an ALTER TABLE operation runs out of space, it starts a rollback, and that can take hours if it is disk-bound. For inserts, InnoDB uses the insert buffer to merge secondary index records to indexes in batches. That saves a lot of disk I/O. For rollback, no such mechanism is used, and the rollback can take 30 times longer than the insertion.

In the case of a runaway rollback, if you do not have valuable data in your database, it may be advisable to kill the database process rather than wait for millions of disk I/O operations to complete. For the complete procedure, see Section 14.21.2, “Forcing InnoDB Recovery”.

Defining a PRIMARY KEY for Each Table

The PRIMARY KEY clause is a critical factor affecting the performance of MySQL queries and the space usage for tables and indexes. The primary key uniquely identifies a row in a table. Every row in the table must have a primary key value, and no two rows can have the same primary key value.

These are guidelines for the primary key, followed by more detailed explanations.

- Declare a PRIMARY KEY for each table. Typically, it is the most important column that you refer to in WHERE clauses when looking up a single row.
- Declare the PRIMARY KEY clause in the original CREATE TABLE statement, rather than adding it later through an ALTER TABLE statement.
- Choose the column and its data type carefully. Prefer numeric columns over character or string ones.
- Consider using an auto-increment column if there is not another stable, unique, non-null, numeric column to use.
- An auto-increment column is also a good choice if there is any doubt whether the value of the primary key column could ever change. Changing the value of a primary key column is an expensive operation, possibly involving rearranging data within the table and within each secondary index.

Consider adding a primary key to any table that does not already have one. Use the smallest practical numeric type based on the maximum projected size of the table. This can make each row slightly more compact, which can yield substantial space savings for large tables. The space savings are multiplied if the table has any secondary indexes, because the primary key value is repeated in each secondary index entry. In addition to reducing data size on disk, a small primary key also lets more data fit into the buffer pool, speeding up all kinds of operations and improving concurrency.

If the table already has a primary key on some longer column, such as a VARCHAR, consider adding a new unsigned AUTO_INCREMENT column and switching the primary key to that, even if that column is not referenced in queries. This design change can produce substantial space savings in the secondary indexes. You can designate the former primary key columns as UNIQUE NOT NULL to enforce the same constraints as the PRIMARY KEY clause, that is, to prevent duplicate or null values across all those columns.

If you spread related information across multiple tables, typically each table uses the same column for its primary key. For example, a personnel database might have several tables, each with a primary key of employee number. A sales database might have some tables with a primary key of customer number, and other tables with a primary key of order number. Because lookups using the primary key are very fast, you can construct efficient join queries for such tables.

If you leave the PRIMARY KEY clause out entirely, MySQL creates an invisible one for you. It is a 6-byte value that might be longer than you need, thus wasting space. Because it is hidden, you cannot refer to it in queries.

Application Performance Considerations

The reliability and scalability features of InnoDB require more disk storage than equivalent MyISAM tables. You might change the column and index definitions slightly, for better space utilization, reduced

I/O and memory consumption when processing result sets, and better query optimization plans making efficient use of index lookups.

If you do set up a numeric ID column for the primary key, use that value to cross-reference with related values in any other tables, particularly for join queries. For example, rather than accepting a country name as input and doing queries searching for the same name, do one lookup to determine the country ID, then do other queries (or a single join query) to look up relevant information across several tables. Rather than storing a customer or catalog item number as a string of digits, potentially using up several bytes, convert it to a numeric ID for storing and querying. A 4-byte unsigned `INT` column can index over 4 billion items (with the US meaning of billion: 1000 million). For the ranges of the different integer types, see Section 11.1.2, “Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT”.

Understanding Files Associated with InnoDB Tables

`InnoDB` files require more care and planning than `MyISAM` files do.

- You must not delete the ibdata files that represent the `InnoDB` system tablespace.
- Methods of moving or copying `InnoDB` tables to a different server are described in Section 14.6.1.4, “Moving or Copying InnoDB Tables”.

14.6.1.6 AUTO_INCREMENT Handling in InnoDB

`InnoDB` provides a configurable locking mechanism that can significantly improve scalability and performance of SQL statements that add rows to tables with `AUTO_INCREMENT` columns. To use the `AUTO_INCREMENT` mechanism with an `InnoDB` table, an `AUTO_INCREMENT` column must be defined as part of an index such that it is possible to perform the equivalent of an indexed `SELECT MAX(ai_col)` lookup on the table to obtain the maximum column value. Typically, this is achieved by making the column the first column of some table index.

This section describes the behavior of `AUTO_INCREMENT` lock modes, usage implications for different `AUTO_INCREMENT` lock mode settings, and how `InnoDB` initializes the `AUTO_INCREMENT` counter.

- InnoDB AUTO_INCREMENT Lock Modes
- InnoDB AUTO_INCREMENT Lock Mode Usage Implications
- InnoDB AUTO_INCREMENT Counter Initialization
- Notes

InnoDB AUTO_INCREMENT Lock Modes

This section describes the behavior of `AUTO_INCREMENT` lock modes used to generate auto-increment values, and how each lock mode affects replication. Auto-increment lock modes are configured at startup using the `innodb_autoinc_lock_mode` configuration parameter.

The following terms are used in describing `innodb_autoinc_lock_mode` settings:

- “`INSERT`-like” statements

  All statements that generate new rows in a table, including `INSERT`, `INSERT ... SELECT`, `REPLACE`, `REPLACE ... SELECT`, and `LOAD DATA`. Includes “simple-inserts”, “bulk-inserts”, and “mixed-mode” inserts.

- “Simple inserts”

  Statements for which the number of rows to be inserted can be determined in advance (when the statement is initially processed). This includes single-row and multiple-row `INSERT` and `REPLACE` statements that do not have a nested subquery, but not `INSERT ... ON DUPLICATE KEY UPDATE`.

- “Bulk inserts”

  Statements for which the number of rows to be inserted (and the number of required auto-increment values) is not known in advance. This includes `INSERT ... SELECT`, `REPLACE ... SELECT`, and `LOAD DATA` statements, but not plain `INSERT`. `InnoDB` assigns new values for the `AUTO_INCREMENT` column one at a time as each row is processed.

- “Mixed-mode inserts”

  These are “simple insert” statements that specify the auto-increment value for some (but not all) of the new rows. An example follows, where `c1` is an `AUTO_INCREMENT` column of table `t1`:

  ```
  INSERT INTO t1 (c1,c2) VALUES (1,'a'), (NULL,'b'), (5,'c'), (NULL,'d');
  ```

  Another type of “mixed-mode insert” is `INSERT ... ON DUPLICATE KEY UPDATE`, which in the worst case is in effect an `INSERT` followed by a `UPDATE`, where the allocated value for the `AUTO_INCREMENT` column may or may not be used during the update phase.

There are three possible settings for the `innodb_autoinc_lock_mode` configuration parameter. The settings are 0, 1, or 2, for “traditional”, “consecutive”, or “interleaved” lock mode, respectively.

- `innodb_autoinc_lock_mode = 0` (“traditional” lock mode)

  The traditional lock mode provides the same behavior that existed before the `innodb_autoinc_lock_mode` configuration parameter was introduced in MySQL 5.1. The traditional lock mode option is provided for backward compatibility, performance testing, and working around issues with “mixed-mode inserts”, due to possible differences in semantics.

  In this lock mode, all “INSERT-like” statements obtain a special table-level `AUTO-INC` lock for inserts into tables with `AUTO_INCREMENT` columns. This lock is normally held to the end of the statement (not to the end of the transaction) to ensure that auto-increment values are assigned in a predictable and repeatable order for a given sequence of `INSERT` statements, and to ensure that auto-increment values assigned by any given statement are consecutive.

  In the case of statement-based replication, this means that when an SQL statement is replicated on a replica server, the same values are used for the auto-increment column as on the source server. The result of execution of multiple `INSERT` statements is deterministic, and the replica reproduces the same data as on the source. If auto-increment values generated by multiple `INSERT` statements were interleaved, the result of two concurrent `INSERT` statements would be nondeterministic, and could not reliably be propagated to a replica server using statement-based replication.

  To make this clear, consider an example that uses this table:

  ```
  CREATE TABLE t1 (
    c1 INT(11) NOT NULL AUTO_INCREMENT,
    c2 VARCHAR(10) DEFAULT NULL,
    PRIMARY KEY (c1)
  ) ENGINE=InnoDB;
  ```

  Suppose that there are two transactions running, each inserting rows into a table with an `AUTO_INCREMENT` column. One transaction is using an `INSERT ... SELECT` statement that inserts 1000 rows, and another is using a simple `INSERT` statement that inserts one row:

  ```
  Tx1: INSERT INTO t1 (c2) SELECT 1000 rows from another table ...
  Tx2: INSERT INTO t1 (c2) VALUES ('xxx');
  ```

  `InnoDB` cannot tell in advance how many rows are retrieved from the `SELECT` in the `INSERT` statement in Tx1, and it assigns the auto-increment values one at a time as the statement proceeds. With a table-level lock, held to the end of the statement, only one `INSERT` statement referring to table `t1` can execute at a time, and the generation of auto-increment numbers by different statements is not interleaved. The auto-increment value generated by the Tx1 `INSERT ... SELECT` statement is consecutive, and the (single) auto-increment value used by the `INSERT`

statement in Tx2 is either be smaller or larger than all those used for Tx1, depending on which statement executes first.

As long as the SQL statements execute in the same order when replayed from the binary log (when using statement-based replication, or in recovery scenarios), the results are the same as they were when Tx1 and Tx2 first ran. Thus, table-level locks held until the end of a statement make `INSERT` statements using auto-increment safe for use with statement-based replication. However, those table-level locks limit concurrency and scalability when multiple transactions are executing insert statements at the same time.

In the preceding example, if there were no table-level lock, the value of the auto-increment column used for the `INSERT` in Tx2 depends on precisely when the statement executes. If the `INSERT` of Tx2 executes while the `INSERT` of Tx1 is running (rather than before it starts or after it completes), the specific auto-increment values assigned by the two `INSERT` statements are nondeterministic, and may vary from run to run.

Under the consecutive lock mode, `InnoDB` can avoid using table-level `AUTO-INC` locks for "simple insert" statements where the number of rows is known in advance, and still preserve deterministic execution and safety for statement-based replication.

If you are not using the binary log to replay SQL statements as part of recovery or replication, the interleaved lock mode can be used to eliminate all use of table-level `AUTO-INC` locks for even greater concurrency and performance, at the cost of permitting gaps in auto-increment numbers assigned by a statement and potentially having the numbers assigned by concurrently executing statements interleaved.

- `innodb_autoinc_lock_mode = 1` ("consecutive" lock mode)

  This is the default lock mode. In this mode, "bulk inserts" use the special `AUTO-INC` table-level lock and hold it until the end of the statement. This applies to all `INSERT ... SELECT`, `REPLACE ... SELECT`, and `LOAD DATA` statements. Only one statement holding the `AUTO-INC` lock can execute at a time. If the source table of the bulk insert operation is different from the target table, the `AUTO-INC` lock on the target table is taken after a shared lock is taken on the first row selected from the source table. If the source and target of the bulk insert operation are the same table, the `AUTO-INC` lock is taken after shared locks are taken on all selected rows.

  "Simple inserts" (for which the number of rows to be inserted is known in advance) avoid table-level `AUTO-INC` locks by obtaining the required number of auto-increment values under the control of a mutex (a light-weight lock) that is only held for the duration of the allocation process, *not* until the statement completes. No table-level `AUTO-INC` lock is used unless an `AUTO-INC` lock is held by another transaction. If another transaction holds an `AUTO-INC` lock, a "simple insert" waits for the `AUTO-INC` lock, as if it were a "bulk insert".

  This lock mode ensures that, in the presence of `INSERT` statements where the number of rows is not known in advance (and where auto-increment numbers are assigned as the statement progresses), all auto-increment values assigned by any "`INSERT`-like" statement are consecutive, and operations are safe for statement-based replication.

  Simply put, this lock mode significantly improves scalability while being safe for use with statement-based replication. Further, as with "traditional" lock mode, auto-increment numbers assigned by any given statement are *consecutive*. There is *no change* in semantics compared to "traditional" mode for any statement that uses auto-increment, with one important exception.

  The exception is for "mixed-mode inserts", where the user provides explicit values for an `AUTO_INCREMENT` column for some, but not all, rows in a multiple-row "simple insert". For such inserts, `InnoDB` allocates more auto-increment values than the number of rows to be inserted. However, all values automatically assigned are consecutively generated (and thus higher than) the auto-increment value generated by the most recently executed previous statement. "Excess" numbers are lost.

- `innodb_autoinc_lock_mode = 2` (“interleaved” lock mode)

  In this lock mode, no “`INSERT`-like” statements use the table-level `AUTO-INC` lock, and multiple statements can execute at the same time. This is the fastest and most scalable lock mode, but it is *not safe* when using statement-based replication or recovery scenarios when SQL statements are replayed from the binary log.

  In this lock mode, auto-increment values are guaranteed to be unique and monotonically increasing across all concurrently executing “`INSERT`-like” statements. However, because multiple statements can be generating numbers at the same time (that is, allocation of numbers is *interleaved* across statements), the values generated for the rows inserted by any given statement may not be consecutive.

  If the only statements executing are “simple inserts” where the number of rows to be inserted is known ahead of time, there are no gaps in the numbers generated for a single statement, except for “mixed-mode inserts”. However, when “bulk inserts” are executed, there may be gaps in the auto-increment values assigned by any given statement.

InnoDB AUTO_INCREMENT Lock Mode Usage Implications

- Using auto-increment with replication

  If you are using statement-based replication, set `innodb_autoinc_lock_mode` to 0 or 1 and use the same value on the source and its replicas. Auto-increment values are not ensured to be the same on the replicas as on the source if you use `innodb_autoinc_lock_mode` = 2 (“interleaved”) or configurations where the source and replicas do not use the same lock mode.

  If you are using row-based or mixed-format replication, all of the auto-increment lock modes are safe, since row-based replication is not sensitive to the order of execution of the SQL statements (and the mixed format uses row-based replication for any statements that are unsafe for statement-based replication).

- “Lost” auto-increment values and sequence gaps

  In all lock modes (0, 1, and 2), if a transaction that generated auto-increment values rolls back, those auto-increment values are “lost”. Once a value is generated for an auto-increment column, it cannot be rolled back, whether or not the “`INSERT`-like” statement is completed, and whether or not the containing transaction is rolled back. Such lost values are not reused. Thus, there may be gaps in the values stored in an `AUTO_INCREMENT` column of a table.

- Specifying NULL or 0 for the `AUTO_INCREMENT` column

  In all lock modes (0, 1, and 2), if a user specifies NULL or 0 for the `AUTO_INCREMENT` column in an `INSERT`, `InnoDB` treats the row as if the value was not specified and generates a new value for it.

- Assigning a negative value to the `AUTO_INCREMENT` column

  In all lock modes (0, 1, and 2), the behavior of the auto-increment mechanism is not defined if you assign a negative value to the `AUTO_INCREMENT` column.

- If the `AUTO_INCREMENT` value becomes larger than the maximum integer for the specified integer type

  In all lock modes (0, 1, and 2), the behavior of the auto-increment mechanism is not defined if the value becomes larger than the maximum integer that can be stored in the specified integer type.

- Gaps in auto-increment values for “bulk inserts”

  With `innodb_autoinc_lock_mode` set to 0 (“traditional”) or 1 (“consecutive”), the auto-increment values generated by any given statement are consecutive, without gaps, because the table-level `AUTO-INC` lock is held until the end of the statement, and only one such statement can execute at a time.

With `innodb_autoinc_lock_mode` set to 2 (“interleaved”), there may be gaps in the auto-increment values generated by “bulk inserts,” but only if there are concurrently executing “`INSERT`-like” statements.

For lock modes 1 or 2, gaps may occur between successive statements because for bulk inserts the exact number of auto-increment values required by each statement may not be known and overestimation is possible.

- Auto-increment values assigned by “mixed-mode inserts”

  Consider a “mixed-mode insert,” where a “simple insert” specifies the auto-increment value for some (but not all) resulting rows. Such a statement behaves differently in lock modes 0, 1, and 2. For example, assume `c1` is an `AUTO_INCREMENT` column of table `t1`, and that the most recent automatically generated sequence number is 100.

```
mysql> CREATE TABLE t1 (
    -> c1 INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    -> c2 CHAR(1)
    -> ) ENGINE = INNODB;
```

  Now, consider the following “mixed-mode insert” statement:

```
mysql> INSERT INTO t1 (c1,c2) VALUES (1,'a'), (NULL,'b'), (5,'c'), (NULL,'d');
```

  With `innodb_autoinc_lock_mode` set to 0 (“traditional”), the four new rows are:

```
mysql> SELECT c1, c2 FROM t1 ORDER BY c2;
+-----+------+
| c1  | c2   |
+-----+------+
|   1 | a    |
| 101 | b    |
|   5 | c    |
| 102 | d    |
+-----+------+
```

  The next available auto-increment value is 103 because the auto-increment values are allocated one at a time, not all at once at the beginning of statement execution. This result is true whether or not there are concurrently executing “`INSERT`-like” statements (of any type).

  With `innodb_autoinc_lock_mode` set to 1 (“consecutive”), the four new rows are also:

```
mysql> SELECT c1, c2 FROM t1 ORDER BY c2;
+-----+------+
| c1  | c2   |
+-----+------+
|   1 | a    |
| 101 | b    |
|   5 | c    |
| 102 | d    |
+-----+------+
```

  However, in this case, the next available auto-increment value is 105, not 103 because four auto-increment values are allocated at the time the statement is processed, but only two are used. This result is true whether or not there are concurrently executing “`INSERT`-like” statements (of any type).

  With `innodb_autoinc_lock_mode` set to mode 2 (“interleaved”), the four new rows are:

```
mysql> SELECT c1, c2 FROM t1 ORDER BY c2;
+-----+------+
| c1  | c2   |
+-----+------+
|   1 | a    |
|   x | b    |
|   5 | c    |
|   y | d    |
```

```
+-----+------+
```

The values of *x* and *y* are unique and larger than any previously generated rows. However, the specific values of *x* and *y* depend on the number of auto-increment values generated by concurrently executing statements.

Finally, consider the following statement, issued when the most-recently generated sequence number is 100:

```
mysql> INSERT INTO t1 (c1,c2) VALUES (1,'a'), (NULL,'b'), (101,'c'), (NULL,'d');
```

With any `innodb_autoinc_lock_mode` setting, this statement generates a duplicate-key error 23000 (`Can't write; duplicate key in table`) because 101 is allocated for the row `(NULL, 'b')` and insertion of the row `(101, 'c')` fails.

- Modifying `AUTO_INCREMENT` column values in the middle of a sequence of `INSERT` statements

  In all lock modes (0, 1, and 2), modifying an `AUTO_INCREMENT` column value in the middle of a sequence of `INSERT` statements could lead to “Duplicate entry” errors. For example, if you perform an `UPDATE` operation that changes an `AUTO_INCREMENT` column value to a value larger than the current maximum auto-increment value, subsequent `INSERT` operations that do not specify an unused auto-increment value could encounter “Duplicate entry” errors. This behavior is demonstrated in the following example.

```
mysql> CREATE TABLE t1 (
    -> c1 INT NOT NULL AUTO_INCREMENT,
    -> PRIMARY KEY (c1)
    ->  ) ENGINE = InnoDB;

mysql> INSERT INTO t1 VALUES(0), (0), (3);

mysql> SELECT c1 FROM t1;
+----+
| c1 |
+----+
|  1 |
|  2 |
|  3 |
+----+

mysql> UPDATE t1 SET c1 = 4 WHERE c1 = 1;

mysql> SELECT c1 FROM t1;
+----+
| c1 |
+----+
|  2 |
|  3 |
|  4 |
+----+

mysql> INSERT INTO t1 VALUES(0);
ERROR 1062 (23000): Duplicate entry '4' for key 'PRIMARY'
```

InnoDB AUTO_INCREMENT Counter Initialization

This section describes how `InnoDB` initializes `AUTO_INCREMENT` counters.

If you specify an `AUTO_INCREMENT` column for an `InnoDB` table, the table handle in the `InnoDB` data dictionary contains a special counter called the auto-increment counter that is used in assigning new values for the column. This counter is stored only in main memory, not on disk.

To initialize an auto-increment counter after a server restart, `InnoDB` executes the equivalent of the following statement on the first insert into a table containing an `AUTO_INCREMENT` column.

```
SELECT MAX(ai_col) FROM table_name FOR UPDATE;
```

`InnoDB` increments the value retrieved by the statement and assigns it to the column and to the auto-increment counter for the table. By default, the value is incremented by 1. This default can be overridden by the `auto_increment_increment` configuration setting.

If the table is empty, `InnoDB` uses the value `1`. This default can be overridden by the `auto_increment_offset` configuration setting.

If a `SHOW TABLE STATUS` statement examines the table before the auto-increment counter is initialized, `InnoDB` initializes but does not increment the value. The value is stored for use by later inserts. This initialization uses a normal exclusive-locking read on the table and the lock lasts to the end of the transaction. `InnoDB` follows the same procedure for initializing the auto-increment counter for a newly created table.

After the auto-increment counter has been initialized, if you do not explicitly specify a value for an `AUTO_INCREMENT` column, `InnoDB` increments the counter and assigns the new value to the column. If you insert a row that explicitly specifies the column value, and the value is greater than the current counter value, the counter is set to the specified column value.

`InnoDB` uses the in-memory auto-increment counter as long as the server runs. When the server is stopped and restarted, `InnoDB` reinitializes the counter for each table for the first `INSERT` to the table, as described earlier.

A server restart also cancels the effect of the `AUTO_INCREMENT = N` table option in `CREATE TABLE` and `ALTER TABLE` statements, which you can use with `InnoDB` tables to set the initial counter value or alter the current counter value.

Notes

- When an `AUTO_INCREMENT` integer column runs out of values, a subsequent `INSERT` operation returns a duplicate-key error. This is general MySQL behavior.
- When you restart the MySQL server, `InnoDB` may reuse an old value that was generated for an `AUTO_INCREMENT` column but never stored (that is, a value that was generated during an old transaction that was rolled back).

14.6.2 Indexes

This section covers topics related to `InnoDB` indexes.

14.6.2.1 Clustered and Secondary Indexes

Every `InnoDB` table has a special index called the clustered index where the data for the rows is stored. Typically, the clustered index is synonymous with the primary key. To get the best performance from queries, inserts, and other database operations, you must understand how `InnoDB` uses the clustered index to optimize the most common lookup and DML operations for each table.

- When you define a `PRIMARY KEY` on your table, `InnoDB` uses it as the clustered index. Define a primary key for each table that you create. If there is no logical unique and non-null column or set of columns, add a new auto-increment column, whose values are filled in automatically.
- If you do not define a `PRIMARY KEY` for your table, MySQL locates the first `UNIQUE` index where all the key columns are `NOT NULL` and `InnoDB` uses it as the clustered index.
- If the table has no `PRIMARY KEY` or suitable `UNIQUE` index, `InnoDB` internally generates a hidden clustered index named `GEN_CLUST_INDEX` on a synthetic column containing row ID values. The rows are ordered by the ID that `InnoDB` assigns to the rows in such a table. The row ID is a 6-byte field that increases monotonically as new rows are inserted. Thus, the rows ordered by the row ID are physically in insertion order.

How the Clustered Index Speeds Up Queries

Accessing a row through the clustered index is fast because the index search leads directly to the page with all the row data. If a table is large, the clustered index architecture often saves a disk I/O operation

when compared to storage organizations that store row data using a different page from the index record.

How Secondary Indexes Relate to the Clustered Index

All indexes other than the clustered index are known as secondary indexes. In `InnoDB`, each record in a secondary index contains the primary key columns for the row, as well as the columns specified for the secondary index. `InnoDB` uses this primary key value to search for the row in the clustered index.

If the primary key is long, the secondary indexes use more space, so it is advantageous to have a short primary key.

For guidelines to take advantage of `InnoDB` clustered and secondary indexes, see Section 8.3, “Optimization and Indexes”.

14.6.2.2 The Physical Structure of an InnoDB Index

All `InnoDB` indexes are B-trees where the index records are stored in the leaf pages of the tree. The default size of an index page is 16KB.

When new records are inserted into an `InnoDB` clustered index, `InnoDB` tries to leave 1/16 of the page free for future insertions and updates of the index records. If index records are inserted in a sequential order (ascending or descending), the resulting index pages are about 15/16 full. If records are inserted in a random order, the pages are from 1/2 to 15/16 full. If the fill factor of an index page drops below 1/2, `InnoDB` tries to contract the index tree to free the page.

You can define the page size for all `InnoDB` tablespaces in a MySQL instance by setting the `innodb_page_size` configuration option prior to initializing the MySQL instance. Once the page size for an instance is defined, you cannot change it without reinitializing the instance. Supported sizes are 16KB, 8KB, and 4KB.

A MySQL instance using a particular `InnoDB` page size cannot use data files or log files from an instance that uses a different page size.

14.6.2.3 InnoDB FULLTEXT Indexes

`FULLTEXT` indexes are created on text-based columns (`CHAR`, `VARCHAR`, or `TEXT` columns) to help speed up queries and DML operations on data contained within those columns, omitting any words that are defined as stopwords.

A `FULLTEXT` index is defined as part of a `CREATE TABLE` statement or added to an existing table using `ALTER TABLE` or `CREATE INDEX`.

Full-text search is performed using `MATCH() ... AGAINST` syntax. For usage information, see Section 12.10, “Full-Text Search Functions”.

`InnoDB` `FULLTEXT` indexes are described under the following topics in this section:

- InnoDB Full-Text Index Design
- InnoDB Full-Text Index Tables
- InnoDB Full-Text Index Cache
- InnoDB Full-Text Index Document ID and FTS_DOC_ID Column
- InnoDB Full-Text Index Deletion Handling
- InnoDB Full-Text Index Transaction Handling
- Monitoring InnoDB Full-Text Indexes

InnoDB Full-Text Index Design

InnoDB FULLTEXT indexes have an inverted index design. Inverted indexes store a list of words, and for each word, a list of documents that the word appears in. To support proximity search, position information for each word is also stored, as a byte offset.

InnoDB Full-Text Index Tables

When creating an InnoDB FULLTEXT index, a set of index tables is created, as shown in the following example:

```
mysql> CREATE TABLE opening_lines (
       id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
       opening_line TEXT(500),
       author VARCHAR(200),
       title VARCHAR(200),
       FULLTEXT idx (opening_line)
       ) ENGINE=InnoDB;

mysql> SELECT table_id, name, space from INFORMATION_SCHEMA.INNODB_SYS_TABLES
       WHERE name LIKE 'test/%';
+----------+--------------------------------------------------------+-------+
| table_id | name                                                   | space |
+----------+--------------------------------------------------------+-------+
|      333 | test/FTS_0000000000000147_00000000000001c9_INDEX_1 |   289 |
|      334 | test/FTS_0000000000000147_00000000000001c9_INDEX_2 |   290 |
|      335 | test/FTS_0000000000000147_00000000000001c9_INDEX_3 |   291 |
|      336 | test/FTS_0000000000000147_00000000000001c9_INDEX_4 |   292 |
|      337 | test/FTS_0000000000000147_00000000000001c9_INDEX_5 |   293 |
|      338 | test/FTS_0000000000000147_00000000000001c9_INDEX_6 |   294 |
|      330 | test/FTS_0000000000000147_BEING_DELETED            |   286 |
|      331 | test/FTS_0000000000000147_BEING_DELETED_CACHE      |   287 |
|      332 | test/FTS_0000000000000147_CONFIG                   |   288 |
|      328 | test/FTS_0000000000000147_DELETED                  |   284 |
|      329 | test/FTS_0000000000000147_DELETED_CACHE            |   285 |
|      327 | test/opening_lines                                 |   283 |
+----------+--------------------------------------------------------+-------+
```

The first six tables represent the inverted index and are referred to as auxiliary index tables. When incoming documents are tokenized, the individual words (also referred to as “tokens”) are inserted into the index tables along with position information and the associated Document ID (DOC_ID). The words are fully sorted and partitioned among the six index tables based on the character set sort weight of the word's first character.

The inverted index is partitioned into six auxiliary index tables to support parallel index creation. By default, two threads tokenize, sort, and insert words and associated data into the index tables. The number of threads is configurable using the innodb_ft_sort_pll_degree option. Consider increasing the number of threads when creating FULLTEXT indexes on large tables.

Auxiliary index table names are prefixed with FTS_ and postfixed with INDEX_*. Each index table is associated with the indexed table by a hex value in the index table name that matches the table_id of the indexed table. For example, the table_id of the test/opening_lines table is 327, for which the hex value is 0x147. As shown in the preceding example, the “147” hex value appears in the names of index tables that are associated with the test/opening_lines table.

A hex value representing the index_id of the FULLTEXT index also appears in auxiliary index table names. For example, in the auxiliary table name test/FTS_0000000000000147_00000000000001c9_INDEX_1, the hex value 1c9 has a decimal value of 457. The index defined on the opening_lines table (idx) can be identified by querying the INFORMATION_SCHEMA.INNODB_SYS_INDEXES table for this value (457).

```
mysql> SELECT index_id, name, table_id, space from INFORMATION_SCHEMA.INNODB_SYS_INDEXES
       WHERE index_id=457;
+----------+------+----------+-------+
| index_id | name | table_id | space |
+----------+------+----------+-------+
|      457 | idx  |      327 |   283 |
```

```
+----------+------+----------+-------+
```

Index tables are stored in their own tablespace when `innodb_file_per_table` is enabled. If `innodb_file_per_table` is disabled, index tables are stored in the `InnoDB` system tablespace (space 0).

Note

Due to a bug introduced in MySQL 5.6.5, index tables are created in the `InnoDB` system tablespace (space 0) when `innodb_file_per_table` is enabled. The bug is fixed in MySQL 5.6.20 and MySQL 5.7.5 (Bug#18635485).

The other index tables shown in the preceding example are referred to as common index tables and are used for deletion handling and storing the internal state of `FULLTEXT` indexes. Unlike the inverted index tables, which are created for each full-text index, this set of tables is common to all full-text indexes created on a particular table.

Common auxiliary tables are retained even if full-text indexes are dropped. When a full-text index is dropped, the `FTS_DOC_ID` column that was created for the index is retained, as removing the `FTS_DOC_ID` column would require rebuilding the table. Common axillary tables are required to manage the `FTS_DOC_ID` column.

- `FTS_*_DELETED` and `FTS_*_DELETED_CACHE`

  Contain the document IDs (DOC_ID) for documents that are deleted but whose data is not yet removed from the full-text index. The `FTS_*_DELETED_CACHE` is the in-memory version of the `FTS_*_DELETED` table.

- `FTS_*_BEING_DELETED` and `FTS_*_BEING_DELETED_CACHE`

  Contain the document IDs (DOC_ID) for documents that are deleted and whose data is currently in the process of being removed from the full-text index. The `FTS_*_BEING_DELETED_CACHE` table is the in-memory version of the `FTS_*_BEING_DELETED` table.

- `FTS_*_CONFIG`

  Stores information about the internal state of the `FULLTEXT` index. Most importantly, it stores the `FTS_SYNCED_DOC_ID`, which identifies documents that have been parsed and flushed to disk. In case of crash recovery, `FTS_SYNCED_DOC_ID` values are used to identify documents that have not been flushed to disk so that the documents can be re-parsed and added back to the `FULLTEXT` index cache. To view the data in this table, query the `INFORMATION_SCHEMA.INNODB_FT_CONFIG` table.

InnoDB Full-Text Index Cache

When a document is inserted, it is tokenized, and the individual words and associated data are inserted into the `FULLTEXT` index. This process, even for small documents, could result in numerous small insertions into the auxiliary index tables, making concurrent access to these tables a point of contention. To avoid this problem, `InnoDB` uses a `FULLTEXT` index cache to temporarily cache index table insertions for recently inserted rows. This in-memory cache structure holds insertions until the cache is full and then batch flushes them to disk (to the auxiliary index tables). You can query the `INFORMATION_SCHEMA.INNODB_FT_INDEX_CACHE` table to view tokenized data for recently inserted rows.

The caching and batch flushing behavior avoids frequent updates to auxiliary index tables, which could result in concurrent access issues during busy insert and update times. The batching technique also avoids multiple insertions for the same word, and minimizes duplicate entries. Instead of flushing each word individually, insertions for the same word are merged and flushed to disk as a single entry, improving insertion efficiency while keeping auxiliary index tables as small as possible.

The `innodb_ft_cache_size` variable is used to configure the full-text index cache size (on a per-table basis), which affects how often the full-text index cache is flushed. You can

also define a global full-text index cache size limit for all tables in a given instance using the `innodb_ft_total_cache_size` option.

The full-text index cache stores the same information as auxiliary index tables. However, the full-text index cache only caches tokenized data for recently inserted rows. The data that is already flushed to disk (to the full-text auxiliary tables) is not brought back into the full-text index cache when queried. The data in auxiliary index tables is queried directly, and results from the auxiliary index tables are merged with results from the full-text index cache before being returned.

InnoDB Full-Text Index Document ID and FTS_DOC_ID Column

`InnoDB` uses a unique document identifier referred to as a Document ID (`DOC_ID`) to map words in the full-text index to document records where the word appears. The mapping requires an `FTS_DOC_ID` column on the indexed table. If an `FTS_DOC_ID` column is not defined, `InnoDB` automatically adds a hidden `FTS_DOC_ID` column when the full-text index is created. The following example demonstrates this behavior.

The following table definition does not include an `FTS_DOC_ID` column:

```
mysql> CREATE TABLE opening_lines (
       id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
       opening_line TEXT(500),
       author VARCHAR(200),
       title VARCHAR(200)
       ) ENGINE=InnoDB;
```

When you create a full-text index on the table using `CREATE FULLTEXT INDEX` syntax, a warning is returned which reports that `InnoDB` is rebuilding the table to add the `FTS_DOC_ID` column.

```
mysql> CREATE FULLTEXT INDEX idx ON opening_lines(opening_line);
Query OK, 0 rows affected, 1 warning (0.19 sec)
Records: 0  Duplicates: 0  Warnings: 1

mysql> SHOW WARNINGS;
+---------+------+-----------------------------------------------------+
| Level   | Code | Message                                             |
+---------+------+-----------------------------------------------------+
| Warning |  124 | InnoDB rebuilding table to add column FTS_DOC_ID |
+---------+------+-----------------------------------------------------+
```

The same warning is returned when using `ALTER TABLE` to add a full-text index to a table that does not have an `FTS_DOC_ID` column. If you create a full-text index at `CREATE TABLE` time and do not specify an `FTS_DOC_ID` column, `InnoDB` adds a hidden `FTS_DOC_ID` column, without warning.

Defining an `FTS_DOC_ID` column at `CREATE TABLE` time is less expensive than creating a full-text index on a table that is already loaded with data. If an `FTS_DOC_ID` column is defined on a table prior to loading data, the table and its indexes do not have to be rebuilt to add the new column. If you are not concerned with `CREATE FULLTEXT INDEX` performance, leave out the `FTS_DOC_ID` column to have `InnoDB` create it for you. `InnoDB` creates a hidden `FTS_DOC_ID` column along with a unique index (`FTS_DOC_ID_INDEX`) on the `FTS_DOC_ID` column. If you want to create your own `FTS_DOC_ID` column, the column must be defined as `BIGINT UNSIGNED NOT NULL` and named `FTS_DOC_ID` (all uppercase), as in the following example:

Note

The `FTS_DOC_ID` column does not need to be defined as an `AUTO_INCREMENT` column but `AUTO_INCREMENT` could make loading data easier.

```
mysql> CREATE TABLE opening_lines (
       FTS_DOC_ID BIGINT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
       opening_line TEXT(500),
       author VARCHAR(200),
       title VARCHAR(200)
       ) ENGINE=InnoDB;
```

If you choose to define the FTS_DOC_ID column yourself, you are responsible for managing the column to avoid empty or duplicate values. FTS_DOC_ID values cannot be reused, which means FTS_DOC_ID values must be ever increasing.

Optionally, you can create the required unique FTS_DOC_ID_INDEX (all uppercase) on the FTS_DOC_ID column.

```
mysql> CREATE UNIQUE INDEX FTS_DOC_ID_INDEX on opening_lines(FTS_DOC_ID);
```

If you do not create the FTS_DOC_ID_INDEX, InnoDB creates it automatically.

Before MySQL 5.6.31, the permitted gap between the largest used FTS_DOC_ID value and new FTS_DOC_ID value is 10000. In MySQL 5.6.31 and later, the permitted gap is 65535.

To avoid rebuilding the table, the FTS_DOC_ID column is retained when dropping a full-text index.

InnoDB Full-Text Index Deletion Handling

Deleting a record that has a full-text index column could result in numerous small deletions in the auxiliary index tables, making concurrent access to these tables a point of contention. To avoid this problem, the Document ID (DOC_ID) of a deleted document is logged in a special FTS_*_DELETED table whenever a record is deleted from an indexed table, and the indexed record remains in the full-text index. Before returning query results, information in the FTS_*_DELETED table is used to filter out deleted Document IDs. The benefit of this design is that deletions are fast and inexpensive. The drawback is that the size of the index is not immediately reduced after deleting records. To remove full-text index entries for deleted records, run OPTIMIZE TABLE on the indexed table with innodb_optimize_fulltext_only=ON to rebuild the full-text index. For more information, see Optimizing InnoDB Full-Text Indexes.

InnoDB Full-Text Index Transaction Handling

InnoDB FULLTEXT indexes have special transaction handling characteristics due its caching and batch processing behavior. Specifically, updates and insertions on a FULLTEXT index are processed at transaction commit time, which means that a FULLTEXT search can only see committed data. The following example demonstrates this behavior. The FULLTEXT search only returns a result after the inserted lines are committed.

```
mysql> CREATE TABLE opening_lines (
       id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
       opening_line TEXT(500),
       author VARCHAR(200),
       title VARCHAR(200),
       FULLTEXT idx (opening_line)
       ) ENGINE=InnoDB;

mysql> BEGIN;

mysql> INSERT INTO opening_lines(opening_line,author,title) VALUES
       ('Call me Ishmael.','Herman Melville','Moby-Dick'),
       ('A screaming comes across the sky.','Thomas Pynchon','Gravity\'s Rainbow'),
       ('I am an invisible man.','Ralph Ellison','Invisible Man'),
       ('Where now? Who now? When now?','Samuel Beckett','The Unnamable'),
       ('It was love at first sight.','Joseph Heller','Catch-22'),
       ('All this happened, more or less.','Kurt Vonnegut','Slaughterhouse-Five'),
       ('Mrs. Dalloway said she would buy the flowers herself.','Virginia Woolf','Mrs. Dalloway'),
       ('It was a pleasure to burn.','Ray Bradbury','Fahrenheit 451');

mysql> SELECT COUNT(*) FROM opening_lines WHERE MATCH(opening_line) AGAINST('Ishmael');
+----------+
| COUNT(*) |
+----------+
|        0 |
+----------+

mysql> COMMIT;

mysql> SELECT COUNT(*) FROM opening_lines WHERE MATCH(opening_line) AGAINST('Ishmael');
```

```
+----------+
| COUNT(*) |
+----------+
|        1 |
+----------+
```

Monitoring InnoDB Full-Text Indexes

You can monitor and examine the special text-processing aspects of `InnoDB` `FULLTEXT` indexes by querying the following `INFORMATION_SCHEMA` tables:

- `INNODB_FT_CONFIG`
- `INNODB_FT_INDEX_TABLE`
- `INNODB_FT_INDEX_CACHE`
- `INNODB_FT_DEFAULT_STOPWORD`
- `INNODB_FT_DELETED`
- `INNODB_FT_BEING_DELETED`

You can also view basic information for `FULLTEXT` indexes and tables by querying `INNODB_SYS_INDEXES` and `INNODB_SYS_TABLES`.

For more information, see Section 14.15.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”.

14.6.3 Tablespaces

This section covers topics related to `InnoDB` tablespaces.

14.6.3.1 The System Tablespace

The system tablespace is the storage area for the `InnoDB` data dictionary, the doublewrite buffer, the change buffer, and undo logs. It may also contain table and index data if tables are created in the system tablespace rather than file-per-table tablespaces.

The system tablespace can have one or more data files. By default, a single system tablespace data file, named `ibdata1`, is created in the data directory. The size and number of system tablespace data files is defined by the `innodb_data_file_path` startup option. For configuration information, see System Tablespace Data File Configuration.

Additional information about the system tablespace is provided under the following topics in the section:

- Resizing the System Tablespace
- Using Raw Disk Partitions for the System Tablespace

Resizing the System Tablespace

This section describes how to increase or decrease the size of the system tablespace.

Increasing the Size of the System Tablespace

The easiest way to increase the size of the system tablespace is to configure it to be auto-extending. To do so, specify the `autoextend` attribute for the last data file in the `innodb_data_file_path` setting, and restart the server. For example:

```
innodb_data_file_path=ibdata1:10M:autoextend
```

When the `autoextend` attribute is specified, the data file automatically increases in size by 8MB increments as space is required. The `innodb_autoextend_increment` variable controls the increment size.

You can also increase system tablespace size by adding another data file. To do so:

1. Stop the MySQL server.
2. If the last data file in the `innodb_data_file_path` setting is defined with the `autoextend` attribute, remove it, and modify the size attribute to reflect the current data file size. To determine the appropriate data file size to specify, check your file system for the file size, and round that value down to the closest MB value, where a MB is equal to 1024 x 1024.
3. Append a new data file to the `innodb_data_file_path` setting, optionally specifying the `autoextend` attribute. The `autoextend` attribute can be specified only for the last data file in the `innodb_data_file_path` setting.
4. Start the MySQL server.

For example, this tablespace has one auto-extending data file:

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:10M:autoextend
```

Suppose that the data file has grown to 988MB over time. This is the `innodb_data_file_path` setting after modifying the size attribute to reflect the current data file size, and after specifying a new 50MB auto-extending data file:

```
innodb_data_home_dir =
innodb_data_file_path = /ibdata/ibdata1:988M;/disk2/ibdata2:50M:autoextend
```

When adding a new data file, do not specify an existing file name. `InnoDB` creates and initializes the new data file when you start the server.

Note

You cannot increase the size of an existing system tablespace data file by changing its size attribute. For example, changing the `innodb_data_file_path` setting from `ibdata1:10M:autoextend` to `ibdata1:12M:autoextend` produces the following error when starting the server:

```
[ERROR] [MY-012263] [InnoDB] The Auto-extending innodb_system
data file './ibdata1' is of a different size 640 pages (rounded down to MB) than
specified in the .cnf file: initial 768 pages, max 0 (relevant if non-zero) pages!
```

The error indicates that the existing data file size (expressed in `InnoDB` pages) is different from the size specified in the configuration file. If you encounter this error, restore the previous `innodb_data_file_path` setting, and refer to the system tablespace resizing instructions.

`InnoDB` page size is defined by the `innodb_page_size` variable. The default is 16384 bytes.

Decreasing the Size of the InnoDB System Tablespace

You cannot remove a data file from the system tablespace. To decrease the system tablespace size, use this procedure:

1. Use `mysqldump` to dump all of your `InnoDB` tables, including `InnoDB` tables located in the `mysql` schema. Identify `InnoDB` tables in the `mysql` schema using the following query:

```
mysql> SELECT TABLE_NAME from INFORMATION_SCHEMA.TABLES WHERE TABLE_SCHEMA='mysql' and ENGINE='InnoDB';
+----------------------+
| table_name           |
+----------------------+
| innodb_index_stats   |
| innodb_table_stats   |
| slave_master_info    |
| slave_relay_log_info |
```

```
| slave_worker_info    |
+----------------------+
5 rows in set (0.00 sec)
```

2. Stop the server.
3. Remove all of the existing tablespace files (`*.ibd`), including the `ibdata` and `ib_log` files. Do not forget to remove `*.ibd` files for tables located in the `mysql` schema.
4. Remove any `.frm` files for `InnoDB` tables.
5. Configure the data files for the new system tablespace. See System Tablespace Data File Configuration.
6. Restart the server.
7. Import the dump files.

Note

If your databases only use the `InnoDB` engine, it may be simpler to dump **all** databases, stop the server, remove all databases and `InnoDB` log files, restart the server, and import the dump files.

To avoid large system tablespaces, consider using file-per-table tablespaces for your data. File-per-table tablespaces are the default tablespace type and are used implicitly when creating an `InnoDB` table. Unlike the system tablespace, disk space is returned to the operating system after truncating or dropping a table created in a file-per-table tablespace. For more information, see Section 14.6.3.2, “File-Per-Table Tablespaces”.

Using Raw Disk Partitions for the System Tablespace

You can use raw disk partitions as data files in the `InnoDB` system tablespace. This technique enables nonbuffered I/O on Windows and on some Linux and Unix systems without file system overhead. Perform tests with and without raw partitions to verify whether this change actually improves performance on your system.

When you use a raw disk partition, ensure that the user ID that runs the MySQL server has read and write privileges for that partition. For example, if you run the server as the `mysql` user, the partition must be readable and writeable by `mysql`. If you run the server with the `--memlock` option, the server must be run as `root`, so the partition must be readable and writeable by `root`.

The procedures described below involve option file modification. For additional information, see Section 4.2.2.2, “Using Option Files”.

Allocating a Raw Disk Partition on Linux and Unix Systems

1. When you create a new data file, specify the keyword `newraw` immediately after the data file size for the `innodb_data_file_path` option. The partition must be at least as large as the size that you specify. Note that 1MB in `InnoDB` is 1024 × 1024 bytes, whereas 1MB in disk specifications usually means 1,000,000 bytes.

   ```
   [mysqld]
   innodb_data_home_dir=
   innodb_data_file_path=/dev/hdd1:3Gnewraw;/dev/hdd2:2Gnewraw
   ```

2. Restart the server. `InnoDB` notices the `newraw` keyword and initializes the new partition. However, do not create or change any `InnoDB` tables yet. Otherwise, when you next restart the server, `InnoDB` reinitializes the partition and your changes are lost. (As a safety measure `InnoDB` prevents users from modifying data when any partition with `newraw` is specified.)
3. After `InnoDB` has initialized the new partition, stop the server, change `newraw` in the data file specification to `raw`:

```
[mysqld]
innodb_data_home_dir=
innodb_data_file_path=/dev/hdd1:3Graw;/dev/hdd2:2Graw
```

4. Restart the server. InnoDB now permits changes to be made.

Allocating a Raw Disk Partition on Windows

On Windows systems, the same steps and accompanying guidelines described for Linux and Unix systems apply except that the innodb_data_file_path setting differs slightly on Windows.

1. When you create a new data file, specify the keyword newraw immediately after the data file size for the innodb_data_file_path option:

   ```
   [mysqld]
   innodb_data_home_dir=
   innodb_data_file_path=//./D::10Gnewraw
   ```

   The //./ corresponds to the Windows syntax of \\.\ for accessing physical drives. In the example above, D: is the drive letter of the partition.

2. Restart the server. InnoDB notices the newraw keyword and initializes the new partition.

3. After InnoDB has initialized the new partition, stop the server, change newraw in the data file specification to raw:

   ```
   [mysqld]
   innodb_data_home_dir=
   innodb_data_file_path=//./D::10Graw
   ```

4. Restart the server. InnoDB now permits changes to be made.

14.6.3.2 File-Per-Table Tablespaces

A file-per-table tablespace contains data and indexes for a single InnoDB table, and is stored on the file system in its own data file.

File-per-table tablespace characteristics are described under the following topics in this section:

- File-Per-Table Tablespace Configuration
- File-Per-Table Tablespace Data Files
- File-Per-Table Tablespace Advantages
- File-Per-Table Tablespace Disadvantages

File-Per-Table Tablespace Configuration

InnoDB creates tables in file-per-table tablespaces by default. This behavior is controlled by the innodb_file_per_table variable. Disabling innodb_file_per_table causes InnoDB to create tables in the system tablespace.

An innodb_file_per_table setting can be specified in an option file or configured at runtime using a SET GLOBAL statement. Changing the setting at runtime requires privileges sufficient to set global system variables. See Section 5.1.8.1, “System Variable Privileges”.

Option file:

```
[mysqld]
innodb_file_per_table=ON
```

Using SET GLOBAL at runtime:

```
mysql> SET GLOBAL innodb_file_per_table=ON;
```

`innodb_file_per_table` is enabled by default in MySQL 5.6 and higher. You might consider disabling it if backward compatibility with earlier versions of MySQL is a concern.

Warning

Disabling `innodb_file_per_table` prevents table-copying `ALTER TABLE` operations from implicitly moving a table that resides in the system tablespace to a file-per-table tablespace. A table-copying `ALTER TABLE` operation recreates the table using the current `innodb_file_per_table` setting. This behavior does not apply when adding or dropping secondary indexes, nor does it apply to `ALTER TABLE` operations that use the `INPLACE` algorithm.

File-Per-Table Tablespace Data Files

A file-per-table tablespace is created in an `.idb` data file in a schema directory under the MySQL data directory. The `.ibd` file is named for the table (*`table_name`*`.ibd`). For example, the data file for table `test.t1` is created in the `test` directory under the MySQL data directory:

```
mysql> USE test;

mysql> CREATE TABLE t1 (
   id INT PRIMARY KEY AUTO_INCREMENT,
   name VARCHAR(100)
 ) ENGINE = InnoDB;

shell> cd /path/to/mysql/data/test
shell> ls
t1.ibd
```

You can use the `DATA DIRECTORY` clause of the `CREATE TABLE` statement to implicitly create a file-per-table tablespace data file outside of the data directory. For more information, see Section 14.6.1.2, “Creating Tables Externally”.

File-Per-Table Tablespace Advantages

File-per-table tablespaces have the following advantages over the shared system tablespace.

- Disk space is returned to the operating system after truncating or dropping a table created in a file-per-table tablespace. Truncating or dropping a table stored in the system tablespace creates free space within the system tablespace, which can only be used for `InnoDB` data. In other words, a system tablespace does not shrink in size after a table is truncated or dropped.
- A table-copying `ALTER TABLE` operation on a table that resides in the system tablespace can increase the amount of disk space occupied by the tablespace. Such operations may require as much additional space as the data in the table plus indexes. This space is not released back to the operating system as it is for file-per-table tablespaces.
- `TRUNCATE TABLE` performance is better when executed on tables that reside in file-per-table tablespaces.
- File-per-table tablespace data files can be created on separate storage devices for I/O optimization, space management, or backup purposes. See Section 14.6.1.2, “Creating Tables Externally”.
- You can import table that resides in a file-per-table tablespace from another MySQL instance. See Section 14.6.1.3, “Importing InnoDB Tables”.
- Tables created in file-per-table tablespaces use the Barracuda file format. See Section 14.10, “InnoDB File-Format Management”. The Barracuda file format enables features associated with `DYNAMIC` and `COMPRESSED` row formats. See Section 14.11, “InnoDB Row Formats”.
- Tables stored in individual tablespace data files can save time and improve chances for a successful recovery when data corruption occurs, when backups or binary logs are unavailable, or when the MySQL server instance cannot be restarted.

- You can backup or restore tables created in file-per-table tablespaces quickly using MySQL Enterprise Backup, without interrupting the use of other `InnoDB` tables. This is beneficial for tables on varying backup schedules or that require backup less frequently. See Making a Partial Backup for details.
- File-per-table tablespaces permit monitoring table size on the file system by monitoring the size of the tablespace data file.
- Common Linux file systems do not permit concurrent writes to a single file such as a system tablespace data file when `innodb_flush_method` is set to `O_DIRECT`. As a result, there are possible performance improvements when using file-per-table tablespaces in conjunction with this setting.
- Tables in the shared system tablespace, which contains other structures such as the `InnoDB` data dictionary and undo logs, are limited in size by the 64TB tablespace size limit. By comparison, each file-per-table tablespace has a 64TB size limit, which provides plenty of room for individual tables to grow in size.

File-Per-Table Tablespace Disadvantages

File-per-table tablespaces have the following disadvantages compared to the shared system tablespace.

- With file-per-table tablespaces, each table may have unused space that can only be utilized by rows of the same table, which can lead to wasted space if not properly managed.
- `fsync` operations are performed on multiple file-per-table data files instead of a shared system tablespace data file. Because `fsync` operations are per file, write operations for multiple tables cannot be combined, which can result in a higher total number of `fsync` operations.
- `mysqld` must keep an open file handle for each file-per-table tablespace, which may impact performance if you have numerous tables in file-per-table tablespaces.
- More file descriptors are required when each table has its own data file.
- There is potential for more fragmentation, which can impede `DROP TABLE` and table scan performance. However, if fragmentation is managed, file-per-table tablespaces can improve performance for these operations.
- The buffer pool is scanned when dropping a table that resides in a file-per-table tablespace, which can take several seconds for large buffer pools. The scan is performed with a broad internal lock, which may delay other operations.
- The `innodb_autoextend_increment` variable, which defines the increment size for extending the size of an auto-extending system tablespace file when it becomes full, does not apply to file-per-table tablespace files, which are auto-extending regardless of the `innodb_autoextend_increment` setting. Initial file-per-table tablespace extensions are by small amounts, after which extensions occur in increments of 4MB.

14.6.3.3 Undo Tablespaces

Undo tablespaces contain undo logs, which are collections of undo log records that contain information about how to undo the latest change by a transaction to a clustered index record. Undo logs exist within undo log segments, which are contained within rollback segments. The `innodb_rollback_segments` variable defines the number of rollback segments allocated to each undo tablespace.

Undo logs can be stored in one or more undo tablespaces instead of the system tablespace. This layout differs from the default configuration in which undo logs reside in the system tablespace. The I/O patterns for undo logs make undo tablespaces good candidates for SSD storage, while keeping the system tablespace on hard disk storage.

The number of undo tablespaces used by InnoDB is controlled by the innodb_undo_tablespaces configuration option. This option can only be configured when initializing the MySQL instance. It cannot be changed afterward.

Undo tablespaces and individual segments inside those tablespaces cannot be dropped.

Configuring Undo Tablespaces

To configure undo tablespaces for a MySQL instance, perform the following steps. It is assumed that you are performing the procedure on a test instance prior to deploying the configuration to a production system.

Important

The number of undo tablespaces can only be configured when initializing a MySQL instance and is fixed for the life of the instance.

1. Specify a directory location for undo tablespaces using the innodb_undo_directory configuration option. If a directory location is not specified, undo tablespaces are created in the data directory.

2. Define the number of rollback segments using the innodb_rollback_segments configuration option. Start with a relatively low value and increase it incrementally over time to examine the effect on performance. The default setting for innodb_rollback_segments is 128, which is also the maximum value.

   One rollback segment is always assigned to the system tablespace. Therefore, to allocate rollback segments to undo tablespaces, set innodb_rollback_segments to a value greater than 1. For example, if you have two undo tablespaces, set innodb_rollback_segments to 3 to assign one rollback segment to each of the two undo tablespaces. Rollback segments are distributed among undo tablespaces in a circular fashion.

   When you configure separate undo tablespaces, the rollback segment in the system tablespace is rendered inactive.

3. Define the number of undo tablespaces using the innodb_undo_tablespaces option. The specified number of undo tablespaces is fixed for the life of the MySQL instance, so if you are uncertain about an optimal value, estimate on the high side.

4. Create a new MySQL test instance using the option values you have chosen.

5. Use a realistic workload on your test instance with data volume similar to your production servers to test the configuration.

6. Benchmark the performance of I/O intensive workloads.

7. Periodically increase the value of innodb_rollback_segments and rerun performance tests until there are no further improvements in I/O performance.

14.6.4 InnoDB Data Dictionary

The InnoDB data dictionary is comprised of internal system tables that contain metadata used to keep track of objects such as tables, indexes, and table columns. The metadata is physically located in the InnoDB system tablespace. For historical reasons, data dictionary metadata overlaps to some degree with information stored in InnoDB table metadata files (.frm files).

14.6.5 Doublewrite Buffer

The doublewrite buffer is a storage area where InnoDB writes pages flushed from the buffer pool before writing the pages to their proper positions in the InnoDB data files. If there is an operating

system, storage subsystem, or unexpected `mysqld` process exit in the middle of a page write, `InnoDB` can find a good copy of the page from the doublewrite buffer during crash recovery.

Although data is written twice, the doublewrite buffer does not require twice as much I/O overhead or twice as many I/O operations. Data is written to the doublewrite buffer in a large sequential chunk, with a single `fsync()` call to the operating system (except in the case that `innodb_flush_method` is set to `O_DIRECT_NO_FSYNC`).

The doublewrite buffer is enabled by default. To disable the doublewrite buffer, set `innodb_doublewrite` to 0.

14.6.6 Redo Log

The redo log is a disk-based data structure used during crash recovery to correct data written by incomplete transactions. During normal operations, the redo log encodes requests to change table data that result from SQL statements or low-level API calls. Modifications that did not finish updating the data files before an unexpected shutdown are replayed automatically during initialization, and before the connections are accepted. For information about the role of the redo log in crash recovery, see Section 14.18.2, “InnoDB Recovery”.

By default, the redo log is physically represented on disk by two files named `ib_logfile0` and `ib_logfile1`. MySQL writes to the redo log files in a circular fashion. Data in the redo log is encoded in terms of records affected; this data is collectively referred to as redo. The passage of data through the redo log is represented by an ever-increasing LSN value.

For related information, see Redo Log File Configuration, and Section 8.5.4, “Optimizing InnoDB Redo Logging”.

Changing the Number or Size of Redo Log Files

To change the number or size of `InnoDB` redo log files in MySQL 5.6.7 or earlier, perform the following steps:

1. If `innodb_fast_shutdown` is set to 2, set `innodb_fast_shutdown` to 1:

   ```
   mysql> SET GLOBAL innodb_fast_shutdown = 1;
   ```

2. After ensuring that `innodb_fast_shutdown` is not set to 2, stop the MySQL server and make sure that it shuts down without errors (to ensure that there is no information for outstanding transactions in the log).

3. Copy the old log files into a safe place in case something went wrong during the shutdown and you need them to recover the tablespace.

4. Delete the old log files from the log file directory.

5. Edit `my.cnf` to change the log file configuration.

6. Start the MySQL server again. `mysqld` sees that no `InnoDB` log files exist at startup and creates new ones.

The `innodb_fast_shutdown` setting is no longer relevant when changing the number or the size of `InnoDB` log files. Additionally, you are no longer required to remove old log files, although you may still want to copy the old log files to a safe place, as a backup. To change the number or size of `InnoDB` log files, perform the following steps:

1. Stop the MySQL server and make sure that it shuts down without errors.

2. Edit `my.cnf` to change the log file configuration. To change the log file size, configure `innodb_log_file_size`. To increase the number of log files, configure `innodb_log_files_in_group`.

3. Start the MySQL server again.

If InnoDB detects that the innodb_log_file_size differs from the redo log file size, it writes a log checkpoint, closes and removes the old log files, creates new log files at the requested size, and opens the new log files.

Group Commit for Redo Log Flushing

InnoDB, like any other ACID-compliant database engine, flushes the redo log of a transaction before it is committed. InnoDB uses group commit functionality to group multiple such flush requests together to avoid one flush for each commit. With group commit, InnoDB issues a single write to the log file to perform the commit action for multiple user transactions that commit at about the same time, significantly improving throughput.

For more information about performance of COMMIT and other transactional operations, see Section 8.5.2, “Optimizing InnoDB Transaction Management”.

14.6.7 Undo Logs

An undo log is a collection of undo log records associated with a single read-write transaction. An undo log record contains information about how to undo the latest change by a transaction to a clustered index record. If another transaction needs to see the original data as part of a consistent read operation, the unmodified data is retrieved from undo log records. Undo logs exist within undo log segments, which are contained within rollback segments. By default, rollback segments are physically part of the system tablespace, but they can also reside in undo tablespaces. For more information, see Section 14.6.3.3, “Undo Tablespaces”.

InnoDB supports 128 rollback segments. The innodb_rollback_segments variable defines the number of rollback segments used by InnoDB.

The number of transactions that a rollback segment supports depends on the number of undo slots in the rollback segment and the number of undo logs required by each transaction.

The number of undo slots in a rollback segment differs according to InnoDB page size.

InnoDB Page Size	Number of Undo Slots in a Rollback Segment (InnoDB Page Size / 16)
4096 (4KB)	256
8192 (8KB)	512
16384 (16KB)	1024

A transaction is assigned up to two undo logs, one for each of the following operation types:

1. INSERT operations
2. UPDATE and DELETE operations

Undo logs are assigned as needed. For example, a transaction that performs INSERT, UPDATE, and DELETE operations is assigned two undo logs. A transaction that performs only INSERT operations is assigned a single undo log. Undo logs are assigned to a transaction from a rollback segment that is also assigned to the transaction.

An undo log assigned to a transaction remains tied to the transaction for its duration. For example, an undo log assigned to a transaction for an INSERT operation is used for all INSERT operations performed by that transaction.

Given the factors described above, the following formulas can be used to estimate the number of concurrent read-write transactions that InnoDB is capable of supporting.

Note

A transaction can encounter a concurrent transaction limit error before reaching the number of concurrent read-write transactions that InnoDB is capable of supporting. This occurs when the rollback segment assigned to a transaction runs out of undo slots. In such cases, try rerunning the transaction.

- If each transaction performs either an INSERT **or** an UPDATE or DELETE operation, the number of concurrent read-write transactions that InnoDB is capable of supporting is:

```
(innodb_page_size / 16) * innodb_rollback_segments
```

- If each transaction performs an INSERT **and** an UPDATE or DELETE operation, the number of concurrent read-write transactions that InnoDB is capable of supporting is:

```
(innodb_page_size / 16 / 2) * innodb_rollback_segments
```

14.7 InnoDB Locking and Transaction Model

To implement a large-scale, busy, or highly reliable database application, to port substantial code from a different database system, or to tune MySQL performance, it is important to understand InnoDB locking and the InnoDB transaction model.

This section discusses several topics related to InnoDB locking and the InnoDB transaction model with which you should be familiar.

- Section 14.7.1, “InnoDB Locking” describes lock types used by InnoDB.
- Section 14.7.2, “InnoDB Transaction Model” describes transaction isolation levels and the locking strategies used by each. It also discusses the use of autocommit, consistent non-locking reads, and locking reads.
- Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB” discusses specific types of locks set in InnoDB for various statements.
- Section 14.7.4, “Phantom Rows” describes how InnoDB uses next-key locking to avoid phantom rows.
- Section 14.7.5, “Deadlocks in InnoDB” provides a deadlock example, discusses deadlock detection, and provides tips for minimizing and handling deadlocks in InnoDB.

14.7.1 InnoDB Locking

This section describes lock types used by InnoDB.

- Shared and Exclusive Locks
- Intention Locks
- Record Locks
- Gap Locks
- Next-Key Locks
- Insert Intention Locks
- AUTO-INC Locks

Shared and Exclusive Locks

InnoDB implements standard row-level locking where there are two types of locks, shared (S) locks and exclusive (X) locks.

- A shared (S) lock permits the transaction that holds the lock to read a row.
- An exclusive (X) lock permits the transaction that holds the lock to update or delete a row.

If transaction T1 holds a shared (S) lock on row r, then requests from some distinct transaction T2 for a lock on row r are handled as follows:

- A request by T2 for an S lock can be granted immediately. As a result, both T1 and T2 hold an S lock on r.
- A request by T2 for an X lock cannot be granted immediately.

If a transaction T1 holds an exclusive (X) lock on row r, a request from some distinct transaction T2 for a lock of either type on r cannot be granted immediately. Instead, transaction T2 has to wait for transaction T1 to release its lock on row r.

Intention Locks

InnoDB supports *multiple granularity locking* which permits coexistence of row locks and table locks. For example, a statement such as LOCK TABLES ... WRITE takes an exclusive lock (an X lock) on the specified table. To make locking at multiple granularity levels practical, InnoDB uses intention locks. Intention locks are table-level locks that indicate which type of lock (shared or exclusive) a transaction requires later for a row in a table. There are two types of intention locks:

- An intention shared lock (IS) indicates that a transaction intends to set a *shared* lock on individual rows in a table.
- An intention exclusive lock (IX) indicates that a transaction intends to set an exclusive lock on individual rows in a table.

For example, SELECT ... LOCK IN SHARE MODE sets an IS lock, and SELECT ... FOR UPDATE sets an IX lock.

The intention locking protocol is as follows:

- Before a transaction can acquire a shared lock on a row in a table, it must first acquire an IS lock or stronger on the table.
- Before a transaction can acquire an exclusive lock on a row in a table, it must first acquire an IX lock on the table.

Table-level lock type compatibility is summarized in the following matrix.

	X	**IX**	**S**	**IS**
X	Conflict	Conflict	Conflict	Conflict
IX	Conflict	Compatible	Conflict	Compatible
S	Conflict	Conflict	Compatible	Compatible
IS	Conflict	Compatible	Compatible	Compatible

A lock is granted to a requesting transaction if it is compatible with existing locks, but not if it conflicts with existing locks. A transaction waits until the conflicting existing lock is released. If a lock request conflicts with an existing lock and cannot be granted because it would cause deadlock, an error occurs.

Intention locks do not block anything except full table requests (for example, LOCK TABLES ... WRITE). The main purpose of intention locks is to show that someone is locking a row, or going to lock a row in the table.

Transaction data for an intention lock appears similar to the following in SHOW ENGINE INNODB STATUS and InnoDB monitor output:

```
TABLE LOCK table `test`.`t` trx id 10080 lock mode IX
```

Record Locks

A record lock is a lock on an index record. For example, `SELECT c1 FROM t WHERE c1 = 10 FOR UPDATE;` prevents any other transaction from inserting, updating, or deleting rows where the value of `t.c1` is `10`.

Record locks always lock index records, even if a table is defined with no indexes. For such cases, `InnoDB` creates a hidden clustered index and uses this index for record locking. See Section 14.6.2.1, "Clustered and Secondary Indexes".

Transaction data for a record lock appears similar to the following in `SHOW ENGINE INNODB STATUS` and InnoDB monitor output:

```
RECORD LOCKS space id 58 page no 3 n bits 72 index `PRIMARY` of table `test`.`t`
trx id 10078 lock_mode X locks rec but not gap
Record lock, heap no 2 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 8000000a; asc     ;;
 1: len 6; hex 00000000274f; asc     'O;;
 2: len 7; hex b60000019d0110; asc        ;;
```

Gap Locks

A gap lock is a lock on a gap between index records, or a lock on the gap before the first or after the last index record. For example, `SELECT c1 FROM t WHERE c1 BETWEEN 10 and 20 FOR UPDATE;` prevents other transactions from inserting a value of `15` into column `t.c1`, whether or not there was already any such value in the column, because the gaps between all existing values in the range are locked.

A gap might span a single index value, multiple index values, or even be empty.

Gap locks are part of the tradeoff between performance and concurrency, and are used in some transaction isolation levels and not others.

Gap locking is not needed for statements that lock rows using a unique index to search for a unique row. (This does not include the case that the search condition includes only some columns of a multiple-column unique index; in that case, gap locking does occur.) For example, if the `id` column has a unique index, the following statement uses only an index-record lock for the row having `id` value 100 and it does not matter whether other sessions insert rows in the preceding gap:

```
SELECT * FROM child WHERE id = 100;
```

If `id` is not indexed or has a nonunique index, the statement does lock the preceding gap.

It is also worth noting here that conflicting locks can be held on a gap by different transactions. For example, transaction A can hold a shared gap lock (gap S-lock) on a gap while transaction B holds an exclusive gap lock (gap X-lock) on the same gap. The reason conflicting gap locks are allowed is that if a record is purged from an index, the gap locks held on the record by different transactions must be merged.

Gap locks in `InnoDB` are "purely inhibitive", which means that their only purpose is to prevent other transactions from inserting to the gap. Gap locks can co-exist. A gap lock taken by one transaction does not prevent another transaction from taking a gap lock on the same gap. There is no difference between shared and exclusive gap locks. They do not conflict with each other, and they perform the same function.

Gap locking can be disabled explicitly. This occurs if you change the transaction isolation level to `READ COMMITTED` or enable the `innodb_locks_unsafe_for_binlog` system variable (which is now deprecated). Under these circumstances, gap locking is disabled for searches and index scans and is used only for foreign-key constraint checking and duplicate-key checking.

There are also other effects of using the `READ COMMITTED` isolation level or enabling `innodb_locks_unsafe_for_binlog`. Record locks for nonmatching rows are released after MySQL has evaluated the `WHERE` condition. For `UPDATE` statements, `InnoDB` does a "semi-consistent"

read, such that it returns the latest committed version to MySQL so that MySQL can determine whether the row matches the `WHERE` condition of the `UPDATE`.

Next-Key Locks

A next-key lock is a combination of a record lock on the index record and a gap lock on the gap before the index record.

`InnoDB` performs row-level locking in such a way that when it searches or scans a table index, it sets shared or exclusive locks on the index records it encounters. Thus, the row-level locks are actually index-record locks. A next-key lock on an index record also affects the “gap” before that index record. That is, a next-key lock is an index-record lock plus a gap lock on the gap preceding the index record. If one session has a shared or exclusive lock on record `R` in an index, another session cannot insert a new index record in the gap immediately before `R` in the index order.

Suppose that an index contains the values 10, 11, 13, and 20. The possible next-key locks for this index cover the following intervals, where a round bracket denotes exclusion of the interval endpoint and a square bracket denotes inclusion of the endpoint:

```
(negative infinity, 10]
(10, 11]
(11, 13]
(13, 20]
(20, positive infinity)
```

For the last interval, the next-key lock locks the gap above the largest value in the index and the “supremum” pseudo-record having a value higher than any value actually in the index. The supremum is not a real index record, so, in effect, this next-key lock locks only the gap following the largest index value.

By default, `InnoDB` operates in `REPEATABLE READ` transaction isolation level and with the `innodb_locks_unsafe_for_binlog` system variable disabled. In this case, `InnoDB` uses next-key locks for searches and index scans, which prevents phantom rows (see Section 14.7.4, “Phantom Rows”).

Transaction data for a next-key lock appears similar to the following in `SHOW ENGINE INNODB STATUS` and InnoDB monitor output:

```
RECORD LOCKS space id 58 page no 3 n bits 72 index `PRIMARY` of table `test`.`t`
trx id 10080 lock_mode X
Record lock, heap no 1 PHYSICAL RECORD: n_fields 1; compact format; info bits 0
 0: len 8; hex 73757072656d756d; asc supremum;;

Record lock, heap no 2 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 8000000a; asc     ;;
 1: len 6; hex 00000000274f; asc     'O;;
 2: len 7; hex b60000019d0110; asc        ;;
```

Insert Intention Locks

An insert intention lock is a type of gap lock set by `INSERT` operations prior to row insertion. This lock signals the intent to insert in such a way that multiple transactions inserting into the same index gap need not wait for each other if they are not inserting at the same position within the gap. Suppose that there are index records with values of 4 and 7. Separate transactions that attempt to insert values of 5 and 6, respectively, each lock the gap between 4 and 7 with insert intention locks prior to obtaining the exclusive lock on the inserted row, but do not block each other because the rows are nonconflicting.

The following example demonstrates a transaction taking an insert intention lock prior to obtaining an exclusive lock on the inserted record. The example involves two clients, A and B.

Client A creates a table containing two index records (90 and 102) and then starts a transaction that places an exclusive lock on index records with an ID greater than 100. The exclusive lock includes a gap lock before record 102:

```
mysql> CREATE TABLE child (id int(11) NOT NULL, PRIMARY KEY(id)) ENGINE=InnoDB;
mysql> INSERT INTO child (id) values (90),(102);

mysql> START TRANSACTION;
mysql> SELECT * FROM child WHERE id > 100 FOR UPDATE;
+-----+
| id  |
+-----+
| 102 |
+-----+
```

Client B begins a transaction to insert a record into the gap. The transaction takes an insert intention lock while it waits to obtain an exclusive lock.

```
mysql> START TRANSACTION;
mysql> INSERT INTO child (id) VALUES (101);
```

Transaction data for an insert intention lock appears similar to the following in `SHOW ENGINE INNODB STATUS` and InnoDB monitor output:

```
RECORD LOCKS space id 31 page no 3 n bits 72 index `PRIMARY` of table `test`.`child`
trx id 8731 lock_mode X locks gap before rec insert intention waiting
Record lock, heap no 3 PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 80000066; asc    f;;
 1: len 6; hex 000000002215; asc     " ;;
 2: len 7; hex 9000000172011c; asc     r  ;;...
```

AUTO-INC Locks

An `AUTO-INC` lock is a special table-level lock taken by transactions inserting into tables with `AUTO_INCREMENT` columns. In the simplest case, if one transaction is inserting values into the table, any other transactions must wait to do their own inserts into that table, so that rows inserted by the first transaction receive consecutive primary key values.

The `innodb_autoinc_lock_mode` configuration option controls the algorithm used for auto-increment locking. It allows you to choose how to trade off between predictable sequences of auto-increment values and maximum concurrency for insert operations.

For more information, see Section 14.6.1.6, “AUTO_INCREMENT Handling in InnoDB”.

14.7.2 InnoDB Transaction Model

In the `InnoDB` transaction model, the goal is to combine the best properties of a multi-versioning database with traditional two-phase locking. `InnoDB` performs locking at the row level and runs queries as nonlocking consistent reads by default, in the style of Oracle. The lock information in `InnoDB` is stored space-efficiently so that lock escalation is not needed. Typically, several users are permitted to lock every row in `InnoDB` tables, or any random subset of the rows, without causing `InnoDB` memory exhaustion.

14.7.2.1 Transaction Isolation Levels

Transaction isolation is one of the foundations of database processing. Isolation is the I in the acronym ACID; the isolation level is the setting that fine-tunes the balance between performance and reliability, consistency, and reproducibility of results when multiple transactions are making changes and performing queries at the same time.

`InnoDB` offers all four transaction isolation levels described by the SQL:1992 standard: `READ UNCOMMITTED`, `READ COMMITTED`, `REPEATABLE READ`, and `SERIALIZABLE`. The default isolation level for `InnoDB` is `REPEATABLE READ`.

A user can change the isolation level for a single session or for all subsequent connections with the `SET TRANSACTION` statement. To set the server's default isolation level for all connections, use the `--transaction-isolation` option on the command line or in an option file. For detailed information about isolation levels and level-setting syntax, see Section 13.3.6, “SET TRANSACTION Statement”.

`InnoDB` supports each of the transaction isolation levels described here using different locking strategies. You can enforce a high degree of consistency with the default `REPEATABLE READ` level, for operations on crucial data where ACID compliance is important. Or you can relax the consistency rules with `READ COMMITTED` or even `READ UNCOMMITTED`, in situations such as bulk reporting where precise consistency and repeatable results are less important than minimizing the amount of overhead for locking. `SERIALIZABLE` enforces even stricter rules than `REPEATABLE READ`, and is used mainly in specialized situations, such as with XA transactions and for troubleshooting issues with concurrency and deadlocks.

The following list describes how MySQL supports the different transaction levels. The list goes from the most commonly used level to the least used.

- `REPEATABLE READ`

  This is the default isolation level for `InnoDB`. Consistent reads within the same transaction read the snapshot established by the first read. This means that if you issue several plain (nonlocking) `SELECT` statements within the same transaction, these `SELECT` statements are consistent also with respect to each other. See Section 14.7.2.3, “Consistent Nonlocking Reads”.

  For locking reads (`SELECT` with `FOR UPDATE` or `LOCK IN SHARE MODE`), `UPDATE`, and `DELETE` statements, locking depends on whether the statement uses a unique index with a unique search condition, or a range-type search condition.

  - For a unique index with a unique search condition, `InnoDB` locks only the index record found, not the gap before it.
  - For other search conditions, `InnoDB` locks the index range scanned, using gap locks or next-key locks to block insertions by other sessions into the gaps covered by the range. For information about gap locks and next-key locks, see Section 14.7.1, “InnoDB Locking”.

- `READ COMMITTED`

  Each consistent read, even within the same transaction, sets and reads its own fresh snapshot. For information about consistent reads, see Section 14.7.2.3, “Consistent Nonlocking Reads”.

  For locking reads (`SELECT` with `FOR UPDATE` or `LOCK IN SHARE MODE`), `UPDATE` statements, and `DELETE` statements, `InnoDB` locks only index records, not the gaps before them, and thus permits the free insertion of new records next to locked records. Gap locking is only used for foreign-key constraint checking and duplicate-key checking.

  Because gap locking is disabled, phantom problems may occur, as other sessions can insert new rows into the gaps. For information about phantoms, see Section 14.7.4, “Phantom Rows”.

  Only row-based binary logging is supported with the `READ COMMITTED` isolation level. If you use `READ COMMITTED` with `binlog_format=MIXED`, the server automatically uses row-based logging.

  Using `READ COMMITTED` has additional effects:

  - For `UPDATE` or `DELETE` statements, `InnoDB` holds locks only for rows that it updates or deletes. Record locks for nonmatching rows are released after MySQL has evaluated the `WHERE` condition. This greatly reduces the probability of deadlocks, but they can still happen.
  - For `UPDATE` statements, if a row is already locked, `InnoDB` performs a “semi-consistent” read, returning the latest committed version to MySQL so that MySQL can determine whether the row matches the `WHERE` condition of the `UPDATE`. If the row matches (must be updated), MySQL reads the row again and this time `InnoDB` either locks it or waits for a lock on it.

  Consider the following example, beginning with this table:

```
CREATE TABLE t (a INT NOT NULL, b INT) ENGINE = InnoDB;
INSERT INTO t VALUES (1,2),(2,3),(3,2),(4,3),(5,2);
```

```
COMMIT;
```

In this case, the table has no indexes, so searches and index scans use the hidden clustered index for record locking (see Section 14.6.2.1, “Clustered and Secondary Indexes”) rather than indexed columns.

Suppose that one session performs an `UPDATE` using these statements:

```
# Session A
START TRANSACTION;
UPDATE t SET b = 5 WHERE b = 3;
```

Suppose also that a second session performs an `UPDATE` by executing this statement following those of the first session:

```
# Session B
UPDATE t SET b = 4 WHERE b = 2;
```

As `InnoDB` executes each `UPDATE`, it first acquires an exclusive lock for each row that it reads, and then determines whether to modify it. If `InnoDB` does not modify the row, it releases the lock. Otherwise, `InnoDB` retains the lock until the end of the transaction. This affects transaction processing as follows.

When using the default `REPEATABLE READ` isolation level, the first `UPDATE` acquires an x-lock on each row that it reads and does not release any of them:

```
x-lock(1,2); retain x-lock
x-lock(2,3); update(2,3) to (2,5); retain x-lock
x-lock(3,2); retain x-lock
x-lock(4,3); update(4,3) to (4,5); retain x-lock
x-lock(5,2); retain x-lock
```

The second `UPDATE` blocks as soon as it tries to acquire any locks (because first update has retained locks on all rows), and does not proceed until the first `UPDATE` commits or rolls back:

```
x-lock(1,2); block and wait for first UPDATE to commit or roll back
```

If `READ COMMITTED` is used instead, the first `UPDATE` acquires an x-lock on each row that it reads and releases those for rows that it does not modify:

```
x-lock(1,2); unlock(1,2)
x-lock(2,3); update(2,3) to (2,5); retain x-lock
x-lock(3,2); unlock(3,2)
x-lock(4,3); update(4,3) to (4,5); retain x-lock
x-lock(5,2); unlock(5,2)
```

For the second `UPDATE`, `InnoDB` does a “semi-consistent” read, returning the latest committed version of each row that it reads to MySQL so that MySQL can determine whether the row matches the `WHERE` condition of the `UPDATE`:

```
x-lock(1,2); update(1,2) to (1,4); retain x-lock
x-lock(2,3); unlock(2,3)
x-lock(3,2); update(3,2) to (3,4); retain x-lock
x-lock(4,3); unlock(4,3)
x-lock(5,2); update(5,2) to (5,4); retain x-lock
```

However, if the `WHERE` condition includes an indexed column, and `InnoDB` uses the index, only the indexed column is considered when taking and retaining record locks. In the following example, the first `UPDATE` takes and retains an x-lock on each row where b = 2. The second `UPDATE` blocks when it tries to acquire x-locks on the same records, as it also uses the index defined on column b.

```
CREATE TABLE t (a INT NOT NULL, b INT, c INT, INDEX (b)) ENGINE = InnoDB;
INSERT INTO t VALUES (1,2,3),(2,2,4);
COMMIT;

# Session A
```

```
START TRANSACTION;
UPDATE t SET b = 3 WHERE b = 2 AND c = 3;

# Session B
UPDATE t SET b = 4 WHERE b = 2 AND c = 4;
```

The effects of using the `READ COMMITTED` isolation level are the same as enabling the deprecated `innodb_locks_unsafe_for_binlog` configuration option, with these exceptions:

- Enabling `innodb_locks_unsafe_for_binlog` is a global setting and affects all sessions, whereas the isolation level can be set globally for all sessions, or individually per session.
- `innodb_locks_unsafe_for_binlog` can be set only at server startup, whereas the isolation level can be set at startup or changed at runtime.

`READ COMMITTED` therefore offers finer and more flexible control than `innodb_locks_unsafe_for_binlog`.

- `READ UNCOMMITTED`

  `SELECT` statements are performed in a nonlocking fashion, but a possible earlier version of a row might be used. Thus, using this isolation level, such reads are not consistent. This is also called a dirty read. Otherwise, this isolation level works like `READ COMMITTED`.

- `SERIALIZABLE`

  This level is like `REPEATABLE READ`, but `InnoDB` implicitly converts all plain `SELECT` statements to `SELECT ... LOCK IN SHARE MODE` if `autocommit` is disabled. If `autocommit` is enabled, the `SELECT` is its own transaction. It therefore is known to be read only and can be serialized if performed as a consistent (nonlocking) read and need not block for other transactions. (To force a plain `SELECT` to block if other transactions have modified the selected rows, disable `autocommit`.)

14.7.2.2 autocommit, Commit, and Rollback

In `InnoDB`, all user activity occurs inside a transaction. If `autocommit` mode is enabled, each SQL statement forms a single transaction on its own. By default, MySQL starts the session for each new connection with `autocommit` enabled, so MySQL does a commit after each SQL statement if that statement did not return an error. If a statement returns an error, the commit or rollback behavior depends on the error. See Section 14.21.4, “InnoDB Error Handling”.

A session that has `autocommit` enabled can perform a multiple-statement transaction by starting it with an explicit `START TRANSACTION` or `BEGIN` statement and ending it with a `COMMIT` or `ROLLBACK` statement. See Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”.

If `autocommit` mode is disabled within a session with `SET autocommit = 0`, the session always has a transaction open. A `COMMIT` or `ROLLBACK` statement ends the current transaction and a new one starts.

If a session that has `autocommit` disabled ends without explicitly committing the final transaction, MySQL rolls back that transaction.

Some statements implicitly end a transaction, as if you had done a `COMMIT` before executing the statement. For details, see Section 13.3.3, “Statements That Cause an Implicit Commit”.

A `COMMIT` means that the changes made in the current transaction are made permanent and become visible to other sessions. A `ROLLBACK` statement, on the other hand, cancels all modifications made by the current transaction. Both `COMMIT` and `ROLLBACK` release all `InnoDB` locks that were set during the current transaction.

Grouping DML Operations with Transactions

By default, connection to the MySQL server begins with autocommit mode enabled, which automatically commits every SQL statement as you execute it. This mode of operation might be

unfamiliar if you have experience with other database systems, where it is standard practice to issue a sequence of DML statements and commit them or roll them back all together.

To use multiple-statement transactions, switch autocommit off with the SQL statement `SET autocommit = 0` and end each transaction with `COMMIT` or `ROLLBACK` as appropriate. To leave autocommit on, begin each transaction with `START TRANSACTION` and end it with `COMMIT` or `ROLLBACK`. The following example shows two transactions. The first is committed; the second is rolled back.

```
shell> mysql test
```

```
mysql> CREATE TABLE customer (a INT, b CHAR (20), INDEX (a));
Query OK, 0 rows affected (0.00 sec)
mysql> -- Do a transaction with autocommit turned on.
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO customer VALUES (10, 'Heikki');
Query OK, 1 row affected (0.00 sec)
mysql> COMMIT;
Query OK, 0 rows affected (0.00 sec)
mysql> -- Do another transaction with autocommit turned off.
mysql> SET autocommit=0;
Query OK, 0 rows affected (0.00 sec)
mysql> INSERT INTO customer VALUES (15, 'John');
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO customer VALUES (20, 'Paul');
Query OK, 1 row affected (0.00 sec)
mysql> DELETE FROM customer WHERE b = 'Heikki';
Query OK, 1 row affected (0.00 sec)
mysql> -- Now we undo those last 2 inserts and the delete.
mysql> ROLLBACK;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT * FROM customer;
+------+--------+
| a    | b      |
+------+--------+
|   10 | Heikki |
+------+--------+
1 row in set (0.00 sec)
mysql>
```

Transactions in Client-Side Languages

In APIs such as PHP, Perl DBI, JDBC, ODBC, or the standard C call interface of MySQL, you can send transaction control statements such as `COMMIT` to the MySQL server as strings just like any other SQL statements such as `SELECT` or `INSERT`. Some APIs also offer separate special transaction commit and rollback functions or methods.

14.7.2.3 Consistent Nonlocking Reads

A consistent read means that `InnoDB` uses multi-versioning to present to a query a snapshot of the database at a point in time. The query sees the changes made by transactions that committed before that point of time, and no changes made by later or uncommitted transactions. The exception to this rule is that the query sees the changes made by earlier statements within the same transaction. This exception causes the following anomaly: If you update some rows in a table, a `SELECT` sees the latest version of the updated rows, but it might also see older versions of any rows. If other sessions simultaneously update the same table, the anomaly means that you might see the table in a state that never existed in the database.

If the transaction isolation level is `REPEATABLE READ` (the default level), all consistent reads within the same transaction read the snapshot established by the first such read in that transaction. You can get a fresher snapshot for your queries by committing the current transaction and after that issuing new queries.

With `READ COMMITTED` isolation level, each consistent read within a transaction sets and reads its own fresh snapshot.

Consistent read is the default mode in which `InnoDB` processes `SELECT` statements in `READ COMMITTED` and `REPEATABLE READ` isolation levels. A consistent read does not set any locks on the tables it accesses, and therefore other sessions are free to modify those tables at the same time a consistent read is being performed on the table.

Suppose that you are running in the default `REPEATABLE READ` isolation level. When you issue a consistent read (that is, an ordinary `SELECT` statement), `InnoDB` gives your transaction a timepoint according to which your query sees the database. If another transaction deletes a row and commits after your timepoint was assigned, you do not see the row as having been deleted. Inserts and updates are treated similarly.

Note

The snapshot of the database state applies to `SELECT` statements within a transaction, not necessarily to DML statements. If you insert or modify some rows and then commit that transaction, a `DELETE` or `UPDATE` statement issued from another concurrent `REPEATABLE READ` transaction could affect those just-committed rows, even though the session could not query them. If a transaction does update or delete rows committed by a different transaction, those changes do become visible to the current transaction. For example, you might encounter a situation like the following:

```
SELECT COUNT(c1) FROM t1 WHERE c1 = 'xyz';
-- Returns 0: no rows match.
DELETE FROM t1 WHERE c1 = 'xyz';
-- Deletes several rows recently committed by other transaction.

SELECT COUNT(c2) FROM t1 WHERE c2 = 'abc';
-- Returns 0: no rows match.
UPDATE t1 SET c2 = 'cba' WHERE c2 = 'abc';
-- Affects 10 rows: another txn just committed 10 rows with 'abc' values.
SELECT COUNT(c2) FROM t1 WHERE c2 = 'cba';
-- Returns 10: this txn can now see the rows it just updated.
```

You can advance your timepoint by committing your transaction and then doing another `SELECT` or `START TRANSACTION WITH CONSISTENT SNAPSHOT`.

This is called *multi-versioned concurrency control*.

In the following example, session A sees the row inserted by B only when B has committed the insert and A has committed as well, so that the timepoint is advanced past the commit of B.

```
             Session A              Session B

           SET autocommit=0;      SET autocommit=0;
time
|          SELECT * FROM t;
|          empty set
|                                  INSERT INTO t VALUES (1, 2);
|
v          SELECT * FROM t;
           empty set
                                  COMMIT;

           SELECT * FROM t;
           empty set

           COMMIT;

           SELECT * FROM t;
           --------------------
           |    1    |    2    |
           --------------------
```

If you want to see the “freshest” state of the database, use either the `READ COMMITTED` isolation level or a locking read:

```
SELECT * FROM t LOCK IN SHARE MODE;
```

With `READ COMMITTED` isolation level, each consistent read within a transaction sets and reads its own fresh snapshot. With `LOCK IN SHARE MODE`, a locking read occurs instead: A `SELECT` blocks until the transaction containing the freshest rows ends (see Section 14.7.2.4, “Locking Reads”).

Consistent read does not work over certain DDL statements:

- Consistent read does not work over `DROP TABLE`, because MySQL cannot use a table that has been dropped and `InnoDB` destroys the table.
- Consistent read does not work over `ALTER TABLE` operations that make a temporary copy of the original table and delete the original table when the temporary copy is built. When you reissue a consistent read within a transaction, rows in the new table are not visible because those rows did not exist when the transaction's snapshot was taken. In this case, the transaction returns an error: `ER_TABLE_DEF_CHANGED`, “Table definition has changed, please retry transaction”.

The type of read varies for select operations in clauses like `INSERT INTO ... SELECT`, `UPDATE ... (SELECT)`, and `CREATE TABLE ... SELECT` that do not specify `FOR UPDATE` or `LOCK IN SHARE MODE`:

- By default, `InnoDB` uses stronger locks in those statements and the `SELECT` part acts like `READ COMMITTED`, where each consistent read, even within the same transaction, sets and reads its own fresh snapshot.
- To perform a nonlocking read in such cases, enable the `innodb_locks_unsafe_for_binlog` option and set the isolation level of the transaction to `READ UNCOMMITTED`, `READ COMMITTED`, or `REPEATABLE READ` to avoid setting locks on rows read from the selected table.

14.7.2.4 Locking Reads

If you query data and then insert or update related data within the same transaction, the regular `SELECT` statement does not give enough protection. Other transactions can update or delete the same rows you just queried. `InnoDB` supports two types of locking reads that offer extra safety:

- `SELECT ... LOCK IN SHARE MODE`

  Sets a shared mode lock on any rows that are read. Other sessions can read the rows, but cannot modify them until your transaction commits. If any of these rows were changed by another transaction that has not yet committed, your query waits until that transaction ends and then uses the latest values.

- `SELECT ... FOR UPDATE`

  For index records the search encounters, locks the rows and any associated index entries, the same as if you issued an `UPDATE` statement for those rows. Other transactions are blocked from updating those rows, from doing `SELECT ... LOCK IN SHARE MODE`, or from reading the data in certain transaction isolation levels. Consistent reads ignore any locks set on the records that exist in the read view. (Old versions of a record cannot be locked; they are reconstructed by applying undo logs on an in-memory copy of the record.)

These clauses are primarily useful when dealing with tree-structured or graph-structured data, either in a single table or split across multiple tables. You traverse edges or tree branches from one place to another, while reserving the right to come back and change any of these “pointer” values.

All locks set by `LOCK IN SHARE MODE` and `FOR UPDATE` queries are released when the transaction is committed or rolled back.

Note

Locking reads are only possible when autocommit is disabled (either by beginning transaction with `START TRANSACTION` or by setting `autocommit` to 0.

A locking read clause in an outer statement does not lock the rows of a table in a nested subquery unless a locking read clause is also specified in the subquery. For example, the following statement does not lock rows in table `t2`.

```
SELECT * FROM t1 WHERE c1 = (SELECT c1 FROM t2) FOR UPDATE;
```

To lock rows in table `t2`, add a locking read clause to the subquery:

```
SELECT * FROM t1 WHERE c1 = (SELECT c1 FROM t2 FOR UPDATE) FOR UPDATE;
```

Locking Read Examples

Suppose that you want to insert a new row into a table `child`, and make sure that the child row has a parent row in table `parent`. Your application code can ensure referential integrity throughout this sequence of operations.

First, use a consistent read to query the table `PARENT` and verify that the parent row exists. Can you safely insert the child row to table `CHILD`? No, because some other session could delete the parent row in the moment between your `SELECT` and your `INSERT`, without you being aware of it.

To avoid this potential issue, perform the `SELECT` using `LOCK IN SHARE MODE`:

```
SELECT * FROM parent WHERE NAME = 'Jones' LOCK IN SHARE MODE;
```

After the `LOCK IN SHARE MODE` query returns the parent `'Jones'`, you can safely add the child record to the `CHILD` table and commit the transaction. Any transaction that tries to acquire an exclusive lock in the applicable row in the `PARENT` table waits until you are finished, that is, until the data in all tables is in a consistent state.

For another example, consider an integer counter field in a table `CHILD_CODES`, used to assign a unique identifier to each child added to table `CHILD`. Do not use either consistent read or a shared mode read to read the present value of the counter, because two users of the database could see the same value for the counter, and a duplicate-key error occurs if two transactions attempt to add rows with the same identifier to the `CHILD` table.

Here, `LOCK IN SHARE MODE` is not a good solution because if two users read the counter at the same time, at least one of them ends up in deadlock when it attempts to update the counter.

To implement reading and incrementing the counter, first perform a locking read of the counter using `FOR UPDATE`, and then increment the counter. For example:

```
SELECT counter_field FROM child_codes FOR UPDATE;
UPDATE child_codes SET counter_field = counter_field + 1;
```

A `SELECT ... FOR UPDATE` reads the latest available data, setting exclusive locks on each row it reads. Thus, it sets the same locks a searched SQL `UPDATE` would set on the rows.

The preceding description is merely an example of how `SELECT ... FOR UPDATE` works. In MySQL, the specific task of generating a unique identifier actually can be accomplished using only a single access to the table:

```
UPDATE child_codes SET counter_field = LAST_INSERT_ID(counter_field + 1);
SELECT LAST_INSERT_ID();
```

The `SELECT` statement merely retrieves the identifier information (specific to the current connection). It does not access any table.

14.7.3 Locks Set by Different SQL Statements in InnoDB

A locking read, an `UPDATE`, or a `DELETE` generally set record locks on every index record that is scanned in the processing of the SQL statement. It does not matter whether there are `WHERE` conditions in the statement that would exclude the row. `InnoDB` does not remember the exact `WHERE` condition, but only knows which index ranges were scanned. The locks are normally next-key locks that also block inserts into the “gap” immediately before the record. However, gap locking can be disabled explicitly, which causes next-key locking not to be used. For more information, see

Section 14.7.1, “InnoDB Locking”. The transaction isolation level also can affect which locks are set; see Section 14.7.2.1, “Transaction Isolation Levels”.

If a secondary index is used in a search and index record locks to be set are exclusive, `InnoDB` also retrieves the corresponding clustered index records and sets locks on them.

If you have no indexes suitable for your statement and MySQL must scan the entire table to process the statement, every row of the table becomes locked, which in turn blocks all inserts by other users to the table. It is important to create good indexes so that your queries do not unnecessarily scan many rows.

`InnoDB` sets specific types of locks as follows.

- `SELECT ... FROM` is a consistent read, reading a snapshot of the database and setting no locks unless the transaction isolation level is set to `SERIALIZABLE`. For `SERIALIZABLE` level, the search sets shared next-key locks on the index records it encounters. However, only an index record lock is required for statements that lock rows using a unique index to search for a unique row.
- For locking reads (`SELECT` with `FOR UPDATE` or `LOCK IN SHARE MODE`), `UPDATE`, and `DELETE` statements, the locks that are taken depend on whether the statement uses a unique index with a unique search condition, or a range-type search condition.
  - For a unique index with a unique search condition, `InnoDB` locks only the index record found, not the gap before it.
  - For other search conditions, and for non-unique indexes, `InnoDB` locks the index range scanned, using gap locks or next-key locks to block insertions by other sessions into the gaps covered by the range. For information about gap locks and next-key locks, see Section 14.7.1, “InnoDB Locking”.
- For index records the search encounters, `SELECT ... FOR UPDATE` blocks other sessions from doing `SELECT ... LOCK IN SHARE MODE` or from reading in certain transaction isolation levels. Consistent reads ignore any locks set on the records that exist in the read view.
- `UPDATE ... WHERE ...` sets an exclusive next-key lock on every record the search encounters. However, only an index record lock is required for statements that lock rows using a unique index to search for a unique row.
- When `UPDATE` modifies a clustered index record, implicit locks are taken on affected secondary index records. The `UPDATE` operation also takes shared locks on affected secondary index records when performing duplicate check scans prior to inserting new secondary index records, and when inserting new secondary index records.
- `DELETE FROM ... WHERE ...` sets an exclusive next-key lock on every record the search encounters. However, only an index record lock is required for statements that lock rows using a unique index to search for a unique row.
- `INSERT` sets an exclusive lock on the inserted row. This lock is an index-record lock, not a next-key lock (that is, there is no gap lock) and does not prevent other sessions from inserting into the gap before the inserted row.

  Prior to inserting the row, a type of gap lock called an insert intention gap lock is set. This lock signals the intent to insert in such a way that multiple transactions inserting into the same index gap need not wait for each other if they are not inserting at the same position within the gap. Suppose that there are index records with values of 4 and 7. Separate transactions that attempt to insert values of 5 and 6 each lock the gap between 4 and 7 with insert intention locks prior to obtaining the exclusive lock on the inserted row, but do not block each other because the rows are nonconflicting.

  If a duplicate-key error occurs, a shared lock on the duplicate index record is set. This use of a shared lock can result in deadlock should there be multiple sessions trying to insert the same row if another session already has an exclusive lock. This can occur if another session deletes the row. Suppose that an `InnoDB` table `t1` has the following structure:

```
CREATE TABLE t1 (i INT, PRIMARY KEY (i)) ENGINE = InnoDB;
```

Now suppose that three sessions perform the following operations in order:

Session 1:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 2:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 3:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 1:

```
ROLLBACK;
```

The first operation by session 1 acquires an exclusive lock for the row. The operations by sessions 2 and 3 both result in a duplicate-key error and they both request a shared lock for the row. When session 1 rolls back, it releases its exclusive lock on the row and the queued shared lock requests for sessions 2 and 3 are granted. At this point, sessions 2 and 3 deadlock: Neither can acquire an exclusive lock for the row because of the shared lock held by the other.

A similar situation occurs if the table already contains a row with key value 1 and three sessions perform the following operations in order:

Session 1:

```
START TRANSACTION;
DELETE FROM t1 WHERE i = 1;
```

Session 2:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 3:

```
START TRANSACTION;
INSERT INTO t1 VALUES(1);
```

Session 1:

```
COMMIT;
```

The first operation by session 1 acquires an exclusive lock for the row. The operations by sessions 2 and 3 both result in a duplicate-key error and they both request a shared lock for the row. When session 1 commits, it releases its exclusive lock on the row and the queued shared lock requests for sessions 2 and 3 are granted. At this point, sessions 2 and 3 deadlock: Neither can acquire an exclusive lock for the row because of the shared lock held by the other.

- `INSERT ... ON DUPLICATE KEY UPDATE` differs from a simple `INSERT` in that an exclusive lock rather than a shared lock is placed on the row to be updated when a duplicate-key error occurs. An exclusive index-record lock is taken for a duplicate primary key value. An exclusive next-key lock is taken for a duplicate unique key value.
- `REPLACE` is done like an `INSERT` if there is no collision on a unique key. Otherwise, an exclusive next-key lock is placed on the row to be replaced.

- `INSERT INTO T SELECT ... FROM S WHERE ...` sets an exclusive index record lock (without a gap lock) on each row inserted into `T`. If the transaction isolation level is `READ COMMITTED`, or `innodb_locks_unsafe_for_binlog` is enabled and the transaction isolation level is not `SERIALIZABLE`, `InnoDB` does the search on `S` as a consistent read (no locks). Otherwise, `InnoDB` sets shared next-key locks on rows from `S`. `InnoDB` has to set locks in the latter case: During roll-forward recovery using a statement-based binary log, every SQL statement must be executed in exactly the same way it was done originally.

  `CREATE TABLE ... SELECT ...` performs the `SELECT` with shared next-key locks or as a consistent read, as for `INSERT ... SELECT`.

  When a `SELECT` is used in the constructs `REPLACE INTO t SELECT ... FROM s WHERE ...` or `UPDATE t ... WHERE col IN (SELECT ... FROM s ...)`, `InnoDB` sets shared next-key locks on rows from table `s`.

- `InnoDB` sets an exclusive lock on the end of the index associated with the `AUTO_INCREMENT` column while initializing a previously specified `AUTO_INCREMENT` column on a table.

  With `innodb_autoinc_lock_mode=0`, `InnoDB` uses a special `AUTO-INC` table lock mode where the lock is obtained and held to the end of the current SQL statement (not to the end of the entire transaction) while accessing the auto-increment counter. Other clients cannot insert into the table while the `AUTO-INC` table lock is held. The same behavior occurs for "bulk inserts" with `innodb_autoinc_lock_mode=1`. Table-level `AUTO-INC` locks are not used with `innodb_autoinc_lock_mode=2`. For more information, See Section 14.6.1.6, "AUTO_INCREMENT Handling in InnoDB".

  `InnoDB` fetches the value of a previously initialized `AUTO_INCREMENT` column without setting any locks.

- If a `FOREIGN KEY` constraint is defined on a table, any insert, update, or delete that requires the constraint condition to be checked sets shared record-level locks on the records that it looks at to check the constraint. `InnoDB` also sets these locks in the case where the constraint fails.

- `LOCK TABLES` sets table locks, but it is the higher MySQL layer above the `InnoDB` layer that sets these locks. `InnoDB` is aware of table locks if `innodb_table_locks = 1` (the default) and `autocommit = 0`, and the MySQL layer above `InnoDB` knows about row-level locks.

  Otherwise, `InnoDB`'s automatic deadlock detection cannot detect deadlocks where such table locks are involved. Also, because in this case the higher MySQL layer does not know about row-level locks, it is possible to get a table lock on a table where another session currently has row-level locks. However, this does not endanger transaction integrity, as discussed in Section 14.7.5.2, "Deadlock Detection".

- `LOCK TABLES` acquires two locks on each table if `innodb_table_locks=1` (the default). In addition to a table lock on the MySQL layer, it also acquires an `InnoDB` table lock. Versions of MySQL before 4.1.2 did not acquire `InnoDB` table locks; the old behavior can be selected by setting `innodb_table_locks=0`. If no `InnoDB` table lock is acquired, `LOCK TABLES` completes even if some records of the tables are being locked by other transactions.

  In MySQL 5.6, `innodb_table_locks=0` has no effect for tables locked explicitly with `LOCK TABLES ... WRITE`. It does have an effect for tables locked for read or write by `LOCK TABLES ... WRITE` implicitly (for example, through triggers) or by `LOCK TABLES ... READ`.

- All `InnoDB` locks held by a transaction are released when the transaction is committed or aborted. Thus, it does not make much sense to invoke `LOCK TABLES` on `InnoDB` tables in `autocommit=1` mode because the acquired `InnoDB` table locks would be released immediately.

- You cannot lock additional tables in the middle of a transaction because `LOCK TABLES` performs an implicit `COMMIT` and `UNLOCK TABLES`.

14.7.4 Phantom Rows

The so-called *phantom* problem occurs within a transaction when the same query produces different sets of rows at different times. For example, if a `SELECT` is executed twice, but returns a row the second time that was not returned the first time, the row is a “phantom” row.

Suppose that there is an index on the `id` column of the `child` table and that you want to read and lock all rows from the table having an identifier value larger than 100, with the intention of updating some column in the selected rows later:

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

The query scans the index starting from the first record where `id` is bigger than 100. Let the table contain rows having `id` values of 90 and 102. If the locks set on the index records in the scanned range do not lock out inserts made in the gaps (in this case, the gap between 90 and 102), another session can insert a new row into the table with an `id` of 101. If you were to execute the same `SELECT` within the same transaction, you would see a new row with an `id` of 101 (a “phantom”) in the result set returned by the query. If we regard a set of rows as a data item, the new phantom child would violate the isolation principle of transactions that a transaction should be able to run so that the data it has read does not change during the transaction.

To prevent phantoms, `InnoDB` uses an algorithm called *next-key locking* that combines index-row locking with gap locking. `InnoDB` performs row-level locking in such a way that when it searches or scans a table index, it sets shared or exclusive locks on the index records it encounters. Thus, the row-level locks are actually index-record locks. In addition, a next-key lock on an index record also affects the “gap” before that index record. That is, a next-key lock is an index-record lock plus a gap lock on the gap preceding the index record. If one session has a shared or exclusive lock on record `R` in an index, another session cannot insert a new index record in the gap immediately before `R` in the index order.

When `InnoDB` scans an index, it can also lock the gap after the last record in the index. Just that happens in the preceding example: To prevent any insert into the table where `id` would be bigger than 100, the locks set by `InnoDB` include a lock on the gap following `id` value 102.

You can use next-key locking to implement a uniqueness check in your application: If you read your data in share mode and do not see a duplicate for a row you are going to insert, then you can safely insert your row and know that the next-key lock set on the successor of your row during the read prevents anyone meanwhile inserting a duplicate for your row. Thus, the next-key locking enables you to “lock” the nonexistence of something in your table.

Gap locking can be disabled as discussed in Section 14.7.1, “InnoDB Locking”. This may cause phantom problems because other sessions can insert new rows into the gaps when gap locking is disabled.

14.7.5 Deadlocks in InnoDB

A deadlock is a situation where different transactions are unable to proceed because each holds a lock that the other needs. Because both transactions are waiting for a resource to become available, neither ever release the locks it holds.

A deadlock can occur when transactions lock rows in multiple tables (through statements such as `UPDATE` or `SELECT ... FOR UPDATE`), but in the opposite order. A deadlock can also occur when such statements lock ranges of index records and gaps, with each transaction acquiring some locks but not others due to a timing issue. For a deadlock example, see Section 14.7.5.1, “An InnoDB Deadlock Example”.

To reduce the possibility of deadlocks, use transactions rather than `LOCK TABLES` statements; keep transactions that insert or update data small enough that they do not stay open for long periods of time; when different transactions update multiple tables or large ranges of rows, use the same order of operations (such as `SELECT ... FOR UPDATE`) in each transaction; create indexes on the columns used in `SELECT ... FOR UPDATE` and `UPDATE ... WHERE` statements. The possibility of deadlocks is not affected by the isolation level, because the isolation level changes the behavior

of read operations, while deadlocks occur because of write operations. For more information about avoiding and recovering from deadlock conditions, see Section 14.7.5.3, “How to Minimize and Handle Deadlocks”.

If a deadlock does occur, `InnoDB` detects the condition and rolls back one of the transactions (the victim). Thus, even if your application logic is correct, you must still handle the case where a transaction must be retried. To see the last deadlock in an `InnoDB` user transaction, use the `SHOW ENGINE INNODB STATUS` command. If frequent deadlocks highlight a problem with transaction structure or application error handling, run with the `innodb_print_all_deadlocks` setting enabled to print information about all deadlocks to the `mysqld` error log. For more information about how deadlocks are automatically detected and handled, see Section 14.7.5.2, “Deadlock Detection”.

14.7.5.1 An InnoDB Deadlock Example

The following example illustrates how an error can occur when a lock request would cause a deadlock. The example involves two clients, A and B.

First, client A creates a table containing one row, and then begins a transaction. Within the transaction, A obtains an `S` lock on the row by selecting it in share mode:

```
mysql> CREATE TABLE t (i INT) ENGINE = InnoDB;
Query OK, 0 rows affected (1.07 sec)

mysql> INSERT INTO t (i) VALUES(1);
Query OK, 1 row affected (0.09 sec)

mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM t WHERE i = 1 LOCK IN SHARE MODE;
+------+
| i    |
+------+
|    1 |
+------+
```

Next, client B begins a transaction and attempts to delete the row from the table:

```
mysql> START TRANSACTION;
Query OK, 0 rows affected (0.00 sec)

mysql> DELETE FROM t WHERE i = 1;
```

The delete operation requires an `X` lock. The lock cannot be granted because it is incompatible with the `S` lock that client A holds, so the request goes on the queue of lock requests for the row and client B blocks.

Finally, client A also attempts to delete the row from the table:

```
mysql> DELETE FROM t WHERE i = 1;
ERROR 1213 (40001): Deadlock found when trying to get lock;
try restarting transaction
```

Deadlock occurs here because client A needs an `X` lock to delete the row. However, that lock request cannot be granted because client B already has a request for an `X` lock and is waiting for client A to release its `S` lock. Nor can the `S` lock held by A be upgraded to an `X` lock because of the prior request by B for an `X` lock. As a result, `InnoDB` generates an error for one of the clients and releases its locks. The client returns this error:

```
ERROR 1213 (40001): Deadlock found when trying to get lock;
try restarting transaction
```

At that point, the lock request for the other client can be granted and it deletes the row from the table.

14.7.5.2 Deadlock Detection

InnoDB automatically detects transaction deadlocks and rolls back a transaction or transactions to break the deadlock. InnoDB tries to pick small transactions to roll back, where the size of a transaction is determined by the number of rows inserted, updated, or deleted.

InnoDB is aware of table locks if innodb_table_locks = 1 (the default) and autocommit = 0, and the MySQL layer above it knows about row-level locks. Otherwise, InnoDB cannot detect deadlocks where a table lock set by a MySQL LOCK TABLES statement or a lock set by a storage engine other than InnoDB is involved. Resolve these situations by setting the value of the innodb_lock_wait_timeout system variable.

If the LATEST DETECTED DEADLOCK section of InnoDB Monitor output includes a message stating, "TOO DEEP OR LONG SEARCH IN THE LOCK TABLE WAITS-FOR GRAPH, WE WILL ROLL BACK FOLLOWING TRANSACTION," this indicates that the number of transactions on the wait-for list has reached a limit of 200. A wait-for list that exceeds 200 transactions is treated as a deadlock and the transaction attempting to check the wait-for list is rolled back. The same error may also occur if the locking thread must look at more than 1,000,000 locks owned by transactions on the wait-for list.

For techniques to organize database operations to avoid deadlocks, see Section 14.7.5, "Deadlocks in InnoDB".

14.7.5.3 How to Minimize and Handle Deadlocks

This section builds on the conceptual information about deadlocks in Section 14.7.5.2, "Deadlock Detection". It explains how to organize database operations to minimize deadlocks and the subsequent error handling required in applications.

Deadlocks are a classic problem in transactional databases, but they are not dangerous unless they are so frequent that you cannot run certain transactions at all. Normally, you must write your applications so that they are always prepared to re-issue a transaction if it gets rolled back because of a deadlock.

InnoDB uses automatic row-level locking. You can get deadlocks even in the case of transactions that just insert or delete a single row. That is because these operations are not really "atomic"; they automatically set locks on the (possibly several) index records of the row inserted or deleted.

You can cope with deadlocks and reduce the likelihood of their occurrence with the following techniques:

- At any time, issue the SHOW ENGINE INNODB STATUS command to determine the cause of the most recent deadlock. That can help you to tune your application to avoid deadlocks.
- If frequent deadlock warnings cause concern, collect more extensive debugging information by enabling the innodb_print_all_deadlocks configuration option. Information about each deadlock, not just the latest one, is recorded in the MySQL error log. Disable this option when you are finished debugging.
- Always be prepared to re-issue a transaction if it fails due to deadlock. Deadlocks are not dangerous. Just try again.
- Keep transactions small and short in duration to make them less prone to collision.
- Commit transactions immediately after making a set of related changes to make them less prone to collision. In particular, do not leave an interactive mysql session open for a long time with an uncommitted transaction.
- If you use locking reads (SELECT ... FOR UPDATE or SELECT ... LOCK IN SHARE MODE), try using a lower isolation level such as READ COMMITTED.
- When modifying multiple tables within a transaction, or different sets of rows in the same table, do those operations in a consistent order each time. Then transactions form well-defined queues and do not deadlock. For example, organize database operations into functions within your application, or

call stored routines, rather than coding multiple similar sequences of `INSERT`, `UPDATE`, and `DELETE` statements in different places.

- Add well-chosen indexes to your tables. Then your queries need to scan fewer index records and consequently set fewer locks. Use `EXPLAIN SELECT` to determine which indexes the MySQL server regards as the most appropriate for your queries.
- Use less locking. If you can afford to permit a `SELECT` to return data from an old snapshot, do not add the clause `FOR UPDATE` or `LOCK IN SHARE MODE` to it. Using the `READ COMMITTED` isolation level is good here, because each consistent read within the same transaction reads from its own fresh snapshot.
- If nothing else helps, serialize your transactions with table-level locks. The correct way to use `LOCK TABLES` with transactional tables, such as `InnoDB` tables, is to begin a transaction with `SET autocommit = 0` (not `START TRANSACTION`) followed by `LOCK TABLES`, and to not call `UNLOCK TABLES` until you commit the transaction explicitly. For example, if you need to write to table `t1` and read from table `t2`, you can do this:

```
SET autocommit=0;
LOCK TABLES t1 WRITE, t2 READ, ...;
... do something with tables t1 and t2 here ...
COMMIT;
UNLOCK TABLES;
```

  Table-level locks prevent concurrent updates to the table, avoiding deadlocks at the expense of less responsiveness for a busy system.
- Another way to serialize transactions is to create an auxiliary “semaphore” table that contains just a single row. Have each transaction update that row before accessing other tables. In that way, all transactions happen in a serial fashion. Note that the `InnoDB` instant deadlock detection algorithm also works in this case, because the serializing lock is a row-level lock. With MySQL table-level locks, the timeout method must be used to resolve deadlocks.

14.8 InnoDB Configuration

This section provides configuration information and procedures for `InnoDB` initialization, startup, and various components and features of the `InnoDB` storage engine. For information about optimizing database operations for `InnoDB` tables, see Section 8.5, “Optimizing for InnoDB Tables”.

14.8.1 InnoDB Startup Configuration

The first decisions to make about `InnoDB` configuration involve the configuration of data files, log files, page size, and memory buffers. It is recommended that you define data file, log file, and page size configuration before creating the `InnoDB` instance. Modifying data file or log file configuration after the `InnoDB` instance is created may involve a non-trivial procedure, and page size can only be defined when the `InnoDB` instance is first initialized.

In addition to these topics, this section provides information about specifying `InnoDB` options in a configuration file, viewing `InnoDB` initialization information, and important storage considerations.

- Specifying Options in a MySQL Configuration File
- Viewing InnoDB Initialization Information
- Important Storage Considerations
- System Tablespace Data File Configuration
- Redo Log File Configuration
- Undo Tablespace Configuration
- Page Size Configuration

- Memory Configuration

Specifying Options in a MySQL Configuration File

Because MySQL uses data file, log file, and page size configuration settings to initialize the `InnoDB` instance, it is recommended that you define these settings in a configuration file that MySQL reads at startup, prior to initializing `InnoDB` for the first time. `InnoDB` is initialized when the MySQL server is started, and the first initialization of `InnoDB` normally occurs the first time you start the MySQL server.

You can place `InnoDB` options in the `[mysqld]` group of any option file that your server reads when it starts. The locations of MySQL option files are described in Section 4.2.2.2, “Using Option Files”.

To make sure that `mysqld` reads options only from a specific file, use the `--defaults-file` option as the first option on the command line when starting the server:

```
mysqld --defaults-file=path_to_configuration_file
```

Viewing InnoDB Initialization Information

To view `InnoDB` initialization information during startup, start `mysqld` from a command prompt. When `mysqld` is started from a command prompt, initialization information is printed to the console.

For example, on Windows, if `mysqld` is located in `C:\Program Files\MySQL\MySQL Server 5.6\bin`, start the MySQL server like this:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.6\bin\mysqld" --console
```

On Unix-like systems, `mysqld` is located in the `bin` directory of your MySQL installation:

```
shell> bin/mysqld --user=mysql &
```

If you do not send server output to the console, check the error log after startup to see the initialization information `InnoDB` printed during the startup process.

For information about starting MySQL using other methods, see Section 2.10.5, “Starting and Stopping MySQL Automatically”.

Note

`InnoDB` does not open all user tables and associated data files at startup. However, `InnoDB` does check for the existence of tablespace files (`*.ibd` files) that are referenced in the data dictionary. If a tablespace file is not found, `InnoDB` logs an error and continues the startup sequence. Tablespace files that are referenced in the redo log may be opened during crash recovery for redo application.

Important Storage Considerations

Review the following storage-related considerations before proceeding with your startup configuration.

- In some cases, database performance improves if the data is not all placed on the same physical disk. Putting log files on a different disk from data is very often beneficial for performance. For example, you can place system tablespace data files and log files on different disks. You can also use raw disk partitions (raw devices) for `InnoDB` data files, which may speed up I/O. See Using Raw Disk Partitions for the System Tablespace.

- `InnoDB` is a transaction-safe (ACID compliant) storage engine for MySQL that has commit, rollback, and crash-recovery capabilities to protect user data. **However, it cannot do so** if the underlying operating system or hardware does not work as advertised. Many operating systems or disk subsystems may delay or reorder write operations to improve performance. On some operating systems, the very `fsync()` system call that should wait until all unwritten data for a file has been flushed might actually return before the data has been flushed to stable storage. Because of this, an operating system crash or a power outage may destroy recently committed data, or in the worst

case, even corrupt the database because of write operations having been reordered. If data integrity is important to you, perform some “pull-the-plug” tests before using anything in production. On macOS, `InnoDB` uses a special `fcntl()` file flush method. Under Linux, it is advisable to **disable the write-back cache**.

On ATA/SATA disk drives, a command such `hdparm -W0 /dev/hda` may work to disable the write-back cache. **Beware that some drives or disk controllers may be unable to disable the write-back cache.**

- With regard to `InnoDB` recovery capabilities that protect user data, `InnoDB` uses a file flush technique involving a structure called the doublewrite buffer, which is enabled by default (`innodb_doublewrite=ON`). The doublewrite buffer adds safety to recovery following an unexpected exit or power outage, and improves performance on most varieties of Unix by reducing the need for `fsync()` operations. It is recommended that the `innodb_doublewrite` option remains enabled if you are concerned with data integrity or possible failures. For additional information about the doublewrite buffer, see Section 14.12.1, “InnoDB Disk I/O”.
- Before using NFS with `InnoDB`, review potential issues outlined in Using NFS with MySQL.
- Running MySQL server on a 4K sector hard drive on Windows is not supported with `innodb_flush_method=async_unbuffered`, which is the default setting. The workaround is to use `innodb_flush_method=normal`.

System Tablespace Data File Configuration

The `innodb_data_file_path` startup option defines the name, size, and attributes of `InnoDB` system tablespace data files. If you do not configure this option prior to initializing the MySQL server, the default behavior is to create a single auto-extending data file, slightly larger than 12MB, named `ibdata1`:

```
mysql> SHOW VARIABLES LIKE 'innodb_data_file_path';
+-----------------------+------------------------+
| Variable_name         | Value                  |
+-----------------------+------------------------+
| innodb_data_file_path | ibdata1:12M:autoextend |
+-----------------------+------------------------+
```

The full data file specification syntax includes the file name, file size, `autoextend` attribute, and `max` attribute:

```
file_name:file_size[:autoextend[:max:max_file_size]]
```

File sizes are specified in kilobytes, megabytes, or gigabytes by appending `K`, `M` or `G` to the size value. If specifying the data file size in kilobytes, do so in multiples of 1024. Otherwise, kilobyte values are rounded to nearest megabyte (MB) boundary. The sum of file sizes must be, at a minimum, slightly larger than 12MB.

You can specify more than one data file using a semicolon-separated list. For example:

```
[mysqld]
innodb_data_file_path=ibdata1:50M;ibdata2:50M:autoextend
```

The `autoextend` and `max` attributes can be used only for the data file that is specified last.

When the `autoextend` attribute is specified, the data file automatically increases in size by 64MB increments as space is required. The `innodb_autoextend_increment` variable controls the increment size.

To specify a maximum size for an auto-extending data file, use the `max` attribute following the `autoextend` attribute. Use the `max` attribute only in cases where constraining disk usage is of critical importance. The following configuration permits `ibdata1` to grow to a limit of 500MB:

```
[mysqld]
innodb_data_file_path=ibdata1:12M:autoextend:max:500M
```

If your disk becomes full, you can add a data file on another disk. For instructions, see Resizing the System Tablespace.

The size limit for individual files is determined by your operating system. You can set the file size to more than 4GB on operating systems that support large files. You can also use raw disk partitions as data files. See Using Raw Disk Partitions for the System Tablespace.

`InnoDB` is not aware of the file system maximum file size, so be cautious on file systems where the maximum file size is a small value such as 2GB.

System tablespace files are created in the data directory by default (`datadir`). To specify an alternate location, use the `innodb_data_home_dir` option. For example, to create a system tablespace data file in a directory named `myibdata`, use this configuration:

```
[mysqld]
innodb_data_home_dir = /myibdata/
innodb_data_file_path=ibdata1:50M:autoextend
```

A trailing slash is required when specifying a value for `innodb_data_home_dir`. `InnoDB` does not create directories, so ensure that the specified directory exists before you start the server. Also, ensure sure that the MySQL server has the proper access rights to create files in the directory.

`InnoDB` forms the directory path for each data file by textually concatenating the value of `innodb_data_home_dir` to the data file name. If `innodb_data_home_dir` is not defined, the default value is "./", which is the data directory. (The MySQL server changes its current working directory to the data directory when it begins executing.)

If you specify `innodb_data_home_dir` as an empty string, you can specify absolute paths for data files listed in the `innodb_data_file_path` value. The following configuration is equivalent to the preceding one:

```
[mysqld]
innodb_data_home_dir =
innodb_data_file_path=/myibdata/ibdata1:50M:autoextend
```

Redo Log File Configuration

By default, `InnoDB` creates two 48MB redo log files in the data directory named `ib_logfile0` and `ib_logfile1`.

The following options can be used to modify the default configuration:

- `innodb_log_group_home_dir` defines directory path to the `InnoDB` log files (the redo logs). If this option is not configured, `InnoDB` log files are created in the MySQL data directory (`datadir`).

  You might use this option to place `InnoDB` log files in a different physical storage location than `InnoDB` data files to avoid potential I/O resource conflicts. For example:

  ```
  [mysqld]
  innodb_log_group_home_dir = /dr3/iblogs
  ```

  Note

  `InnoDB` does not create directories, so make sure that the log directory exists before you start the server. Use the Unix or DOS `mkdir` command to create any necessary directories.

  Make sure that the MySQL server has the proper access rights to create files in the log directory. More generally, the server must have access rights in any directory where it needs to create log files.

- `innodb_log_files_in_group` defines the number of log files in the log group. The default and recommended value is 2.

- `innodb_log_file_size` defines the size in bytes of each log file in the log group. The combined size of log files (`innodb_log_file_size` * `innodb_log_files_in_group`) cannot exceed a maximum value that is slightly less than 512GB. A pair of 255 GB log files, for example, approaches the limit but does not exceed it. The default log file size is 48MB. Generally, the combined size of the log files should be large enough that the server can smooth out peaks and troughs in workload activity, which often means that there is enough redo log space to handle more than an hour of write activity. The larger the value, the less checkpoint flush activity is needed in the buffer pool, saving disk I/O. For additional information, see Section 8.5.4, “Optimizing InnoDB Redo Logging”.

Undo Tablespace Configuration

By default, `InnoDB` undo logs are part of the system tablespace. However, you can choose to store `InnoDB` undo logs in one or more separate undo tablespaces, typically on a different storage device.

The `innodb_undo_directory` configuration option defines the path where `InnoDB` creates separate tablespaces for the undo logs. This option is typically used in conjunction with the `innodb_rollback_segments` and `innodb_undo_tablespaces` options, which determine the disk layout of the undo logs outside the system tablespace.

For more information, see Section 14.6.3.3, “Undo Tablespaces”.

Page Size Configuration

The `innodb_page_size` option specifies the page size for all `InnoDB` tablespaces in a MySQL instance. This value is set when the instance is created and remains constant afterward. Valid values are 16KB (the default), 8KB, and 4KB. Alternatively, you can specify page size in bytes (16384, 8192, 4096).

The default page size of 16KB is appropriate for a wide range of workloads, particularly for queries involving table scans and DML operations involving bulk updates. Smaller page sizes might be more efficient for OLTP workloads involving many small writes, where contention can be an issue when a single page contains many rows. Smaller pages might also be efficient with SSD storage devices, which typically use small block sizes. Keeping the `InnoDB` page size close to the storage device block size minimizes the amount of unchanged data that is rewritten to disk.

Memory Configuration

MySQL allocates memory to various caches and buffers to improve performance of database operations. When allocating memory for `InnoDB`, always consider memory required by the operating system, memory allocated to other applications, and memory allocated for other MySQL buffers and caches. For example, if you use `MyISAM` tables, consider the amount of memory allocated for the key buffer (`key_buffer_size`). For an overview of MySQL buffers and caches, see Section 8.12.4.1, “How MySQL Uses Memory”.

Buffers specific to `InnoDB` are configured using the following parameters:

- `innodb_buffer_pool_size` defines size of the buffer pool, which is the memory area that holds cached data for `InnoDB` tables, indexes, and other auxiliary buffers. The size of the buffer pool is important for system performance, and it is typically recommended that `innodb_buffer_pool_size` is configured to 50 to 75 percent of system memory. The default buffer pool size is 128MB. For additional guidance, see Section 8.12.4.1, “How MySQL Uses Memory”. For information about how to configure `InnoDB` buffer pool size, see Configuring InnoDB Buffer Pool Size. Buffer pool size can be configured at startup.

  On systems with a large amount of memory, you can improve concurrency by dividing the buffer pool into multiple buffer pool instances. The number of buffer pool instances is controlled by the by `innodb_buffer_pool_instances` option. By default, `InnoDB` creates one buffer pool instance. The number of buffer pool instances can be configured at startup. For more information, see Section 14.8.3.1, “Configuring Multiple Buffer Pool Instances”.

- `innodb_additional_mem_pool_size` defines size in bytes of a memory pool `InnoDB` uses to store data dictionary information and other internal data structures. The more tables you have in your application, the more memory you allocate here. If `InnoDB` runs out of memory in this pool, it starts to allocate memory from the operating system and writes warning messages to the MySQL error log. The default value is 8MB. `innodb_additional_mem_pool_size` is deprecated in MySQL 5.6.3.

- `innodb_log_buffer_size` defines the size in bytes of the buffer that `InnoDB` uses to write to the log files on disk. The default size is 8MB. A large log buffer enables large transactions to run without a need to write the log to disk before the transactions commit. If you have transactions that update, insert, or delete many rows, you might consider increasing the size of the log buffer to save disk I/O. `innodb_log_buffer_size` can be configured at startup. For related information, see Section 8.5.4, “Optimizing InnoDB Redo Logging”.

Warning

On 32-bit GNU/Linux x86, be careful not to set memory usage too high. `glibc` may permit the process heap to grow over thread stacks, which crashes your server. It is a risk if the memory allocated to the `mysqld` process for global and per-thread buffers and caches is close to or exceeds 2GB.

A formula similar to the following that calculates global and per-thread memory allocation for MySQL can be used to estimate MySQL memory usage. You may need to modify the formula to account for buffers and caches in your MySQL version and configuration. For an overview of MySQL buffers and caches, see Section 8.12.4.1, “How MySQL Uses Memory”.

```
innodb_buffer_pool_size
+ key_buffer_size
+ max_connections*(sort_buffer_size+read_buffer_size+binlog_cache_size)
+ max_connections*2MB
```

Each thread uses a stack (often 2MB, but only 256KB in MySQL binaries provided by Oracle Corporation.) and in the worst case also uses `sort_buffer_size + read_buffer_size` additional memory.

On Linux, if the kernel is enabled for large page support, `InnoDB` can use large pages to allocate memory for its buffer pool and additional memory pool. See Section 8.12.4.2, “Enabling Large Page Support”.

14.8.2 Configuring InnoDB for Read-Only Operation

You can query `InnoDB` tables where the MySQL data directory is on read-only media by enabling the `--innodb-read-only` configuration option at server startup.

How to Enable

To prepare an instance for read-only operation, make sure all the necessary information is flushed to the data files before storing it on the read-only medium. Run the server with change buffering disabled (`innodb_change_buffering=0`) and do a slow shutdown.

To enable read-only mode for an entire MySQL instance, specify the following configuration options at server startup:

- `--innodb-read-only=1`

- If the instance is on read-only media such as a DVD or CD, or the `/var` directory is not writeable by all: `--pid-file=`*`path_on_writeable_media`* and `--event-scheduler=disabled`

Usage Scenarios

This mode of operation is appropriate in situations such as:

- Distributing a MySQL application, or a set of MySQL data, on a read-only storage medium such as a DVD or CD.
- Multiple MySQL instances querying the same data directory simultaneously, typically in a data warehousing configuration. You might use this technique to avoid bottlenecks that can occur with a heavily loaded MySQL instance, or you might use different configuration options for the various instances to tune each one for particular kinds of queries.
- Querying data that has been put into a read-only state for security or data integrity reasons, such as archived backup data.

Note

This feature is mainly intended for flexibility in distribution and deployment, rather than raw performance based on the read-only aspect. See Section 8.5.3, "Optimizing InnoDB Read-Only Transactions" for ways to tune the performance of read-only queries, which do not require making the entire server read-only.

How It Works

When the server is run in read-only mode through the `--innodb-read-only` option, certain `InnoDB` features and components are reduced or turned off entirely:

- No change buffering is done, in particular no merges from the change buffer. To make sure the change buffer is empty when you prepare the instance for read-only operation, disable change buffering (`innodb_change_buffering=0`) and do a slow shutdown first.
- There is no crash recovery phase at startup. The instance must have performed a slow shutdown before being put into the read-only state.
- Because the redo log is not used in read-only operation, you can set `innodb_log_file_size` to the smallest size possible (1 MB) before making the instance read-only.
- All background threads other than I/O read threads are turned off. As a consequence, a read-only instance cannot encounter any deadlocks.
- Information about deadlocks, monitor output, and so on is not written to temporary files. As a consequence, `SHOW ENGINE INNODB STATUS` does not produce any output.
- If the MySQL server is started with `--innodb-read-only` but the data directory is still on writeable media, the root user can still perform DCL operations such as `GRANT` and `REVOKE`.
- Changes to configuration option settings that would normally change the behavior of write operations, have no effect when the server is in read-only mode.
- The MVCC processing to enforce isolation levels is turned off. All queries read the latest version of a record, because update and deletes are not possible.
- The undo log is not used. Disable any settings for the `innodb_undo_tablespaces` and `innodb_undo_directory` configuration options.

14.8.3 InnoDB Buffer Pool Configuration

This section provides configuration and tuning information for the `InnoDB` buffer pool.

14.8.3.1 Configuring Multiple Buffer Pool Instances

For systems with buffer pools in the multi-gigabyte range, dividing the buffer pool into separate instances can improve concurrency, by reducing contention as different threads read and write to cached pages. This feature is typically intended for systems with a buffer pool size in the multi-gigabyte

range. Multiple buffer pool instances are configured using the `innodb_buffer_pool_instances` configuration option, and you might also adjust the `innodb_buffer_pool_size` value.

When the `InnoDB` buffer pool is large, many data requests can be satisfied by retrieving from memory. You might encounter bottlenecks from multiple threads trying to access the buffer pool at once. You can enable multiple buffer pools to minimize this contention. Each page that is stored in or read from the buffer pool is assigned to one of the buffer pools randomly, using a hashing function. Each buffer pool manages its own free lists, flush lists, LRUs, and all other data structures connected to a buffer pool, and is protected by its own buffer pool mutex.

To enable multiple buffer pool instances, set the `innodb_buffer_pool_instances` configuration option to a value greater than 1 (the default) up to 64 (the maximum). This option takes effect only when you set `innodb_buffer_pool_size` to a size of 1GB or more. The total size you specify is divided among all the buffer pools. For best efficiency, specify a combination of `innodb_buffer_pool_instances` and `innodb_buffer_pool_size` so that each buffer pool instance is at least 1GB.

14.8.3.2 Making the Buffer Pool Scan Resistant

Rather than using a strict LRU algorithm, `InnoDB` uses a technique to minimize the amount of data that is brought into the buffer pool and never accessed again. The goal is to make sure that frequently accessed (“hot”) pages remain in the buffer pool, even as read-ahead and full table scans bring in new blocks that might or might not be accessed afterward.

Newly read blocks are inserted into the middle of the LRU list. All newly read pages are inserted at a location that by default is `3/8` from the tail of the LRU list. The pages are moved to the front of the list (the most-recently used end) when they are accessed in the buffer pool for the first time. Thus, pages that are never accessed never make it to the front portion of the LRU list, and “age out” sooner than with a strict LRU approach. This arrangement divides the LRU list into two segments, where the pages downstream of the insertion point are considered “old” and are desirable victims for LRU eviction.

For an explanation of the inner workings of the `InnoDB` buffer pool and specifics about the LRU algorithm, see Section 14.5.1, “Buffer Pool”.

You can control the insertion point in the LRU list and choose whether `InnoDB` applies the same optimization to blocks brought into the buffer pool by table or index scans. The configuration parameter `innodb_old_blocks_pct` controls the percentage of “old” blocks in the LRU list. The default value of `innodb_old_blocks_pct` is `37`, corresponding to the original fixed ratio of 3/8. The value range is `5` (new pages in the buffer pool age out very quickly) to `95` (only 5% of the buffer pool is reserved for hot pages, making the algorithm close to the familiar LRU strategy).

The optimization that keeps the buffer pool from being churned by read-ahead can avoid similar problems due to table or index scans. In these scans, a data page is typically accessed a few times in quick succession and is never touched again. The configuration parameter `innodb_old_blocks_time` specifies the time window (in milliseconds) after the first access to a page during which it can be accessed without being moved to the front (most-recently used end) of the LRU list. The default value of `innodb_old_blocks_time` is `1000`. Increasing this value makes more and more blocks likely to age out faster from the buffer pool.

Both `innodb_old_blocks_pct` and `innodb_old_blocks_time` can be specified in the MySQL option file (`my.cnf` or `my.ini`) or changed at runtime with the `SET GLOBAL` statement. Changing the value at runtime requires privileges sufficient to set global system variables. See Section 5.1.8.1, “System Variable Privileges”.

To help you gauge the effect of setting these parameters, the `SHOW ENGINE INNODB STATUS` command reports buffer pool statistics. For details, see Monitoring the Buffer Pool Using the InnoDB Standard Monitor.

Because the effects of these parameters can vary widely based on your hardware configuration, your data, and the details of your workload, always benchmark to verify the effectiveness before changing these settings in any performance-critical or production environment.

In mixed workloads where most of the activity is OLTP type with periodic batch reporting queries which result in large scans, setting the value of `innodb_old_blocks_time` during the batch runs can help keep the working set of the normal workload in the buffer pool.

When scanning large tables that cannot fit entirely in the buffer pool, setting `innodb_old_blocks_pct` to a small value keeps the data that is only read once from consuming a significant portion of the buffer pool. For example, setting `innodb_old_blocks_pct=5` restricts this data that is only read once to 5% of the buffer pool.

When scanning small tables that do fit into memory, there is less overhead for moving pages around within the buffer pool, so you can leave `innodb_old_blocks_pct` at its default value, or even higher, such as `innodb_old_blocks_pct=50`.

The effect of the `innodb_old_blocks_time` parameter is harder to predict than the `innodb_old_blocks_pct` parameter, is relatively small, and varies more with the workload. To arrive at an optimal value, conduct your own benchmarks if the performance improvement from adjusting `innodb_old_blocks_pct` is not sufficient.

14.8.3.3 Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)

A read-ahead request is an I/O request to prefetch multiple pages in the buffer pool asynchronously, in anticipation that these pages may be needed soon. The requests bring in all the pages in one extent. `InnoDB` uses two read-ahead algorithms to improve I/O performance:

Linear read-ahead is a technique that predicts what pages might be needed soon based on pages in the buffer pool being accessed sequentially. You control when `InnoDB` performs a read-ahead operation by adjusting the number of sequential page accesses required to trigger an asynchronous read request, using the configuration parameter `innodb_read_ahead_threshold`. Before this parameter was added, `InnoDB` would only calculate whether to issue an asynchronous prefetch request for the entire next extent when it read the last page of the current extent.

The configuration parameter `innodb_read_ahead_threshold` controls how sensitive `InnoDB` is in detecting patterns of sequential page access. If the number of pages read sequentially from an extent is greater than or equal to `innodb_read_ahead_threshold`, `InnoDB` initiates an asynchronous read-ahead operation of the entire following extent. `innodb_read_ahead_threshold` can be set to any value from 0-64. The default value is 56. The higher the value, the more strict the access pattern check. For example, if you set the value to 48, `InnoDB` triggers a linear read-ahead request only when 48 pages in the current extent have been accessed sequentially. If the value is 8, `InnoDB` triggers an asynchronous read-ahead even if as few as 8 pages in the extent are accessed sequentially. You can set the value of this parameter in the MySQL configuration file, or change it dynamically with the `SET GLOBAL` statement, which requires privileges sufficient to set global system variables. See Section 5.1.8.1, “System Variable Privileges”.

Random read-ahead is a technique that predicts when pages might be needed soon based on pages already in the buffer pool, regardless of the order in which those pages were read. If 13 consecutive pages from the same extent are found in the buffer pool, `InnoDB` asynchronously issues a request to prefetch the remaining pages of the extent. To enable this feature, set the configuration variable `innodb_random_read_ahead` to `ON`.

The `SHOW ENGINE INNODB STATUS` command displays statistics to help you evaluate the effectiveness of the read-ahead algorithm. Statistics include counter information for the following global status variables:

- `Innodb_buffer_pool_read_ahead`
- `Innodb_buffer_pool_read_ahead_evicted`
- `Innodb_buffer_pool_read_ahead_rnd`

This information can be useful when fine-tuning the `innodb_random_read_ahead` setting.

For more information about I/O performance, see Section 8.5.8, “Optimizing InnoDB Disk I/O” and Section 8.12.2, “Optimizing Disk I/O”.

14.8.3.4 Configuring Buffer Pool Flushing

InnoDB performs certain tasks in the background, including flushing of dirty pages from the buffer pool. Dirty pages are those that have been modified but are not yet written to the data files on disk.

Buffer pool flushing is initiated when the percentage of dirty pages reaches the low water mark value defined by the innodb_max_dirty_pages_pct_lwm variable. The default low water mark is 0, which disables this early flushing behaviour.

The purpose of the innodb_max_dirty_pages_pct_lwm threshold is to control the percentage dirty pages in the buffer pool, and to prevent the amount of dirty pages from reaching the threshold defined by the innodb_max_dirty_pages_pct variable, which has a default value of 75. InnoDB aggressively flushes buffer pool pages if the percentage of dirty pages in the buffer pool reaches the innodb_max_dirty_pages_pct threshold.

Additional variables permit fine-tuning of buffer pool flushing behavior:

- The innodb_flush_neighbors variable defines whether flushing a page from the buffer pool also flushes other dirty pages in the same extent.
  - A setting of 0 disables innodb_flush_neighbors. Dirty pages in the same extent are not flushed.
  - The default setting of 1 flushes contiguous dirty pages in the same extent.
  - A setting of 2 flushes dirty pages in the same extent.

  When table data is stored on a traditional HDD storage device, flushing neighbor pages in one operation reduces I/O overhead (primarily for disk seek operations) compared to flushing individual pages at different times. For table data stored on SSD, seek time is not a significant factor and you can disable this setting to spread out write operations.
- The innodb_lru_scan_depth variable specifies, per buffer pool instance, how far down the buffer pool LRU list the page cleaner thread scans looking for dirty pages to flush. This is a background operation performed by a page cleaner thread once per second.

  A setting smaller than the default is generally suitable for most workloads. A value that is significantly higher than necessary may impact performance. Only consider increasing the value if you have spare I/O capacity under a typical workload. Conversely, if a write-intensive workload saturates your I/O capacity, decrease the value, especially in the case of a large buffer pool.

  When tuning innodb_lru_scan_depth, start with a low value and configure the setting upward with the goal of rarely seeing zero free pages. Also, consider adjusting innodb_lru_scan_depth when changing the number of buffer pool instances, since innodb_lru_scan_depth * innodb_buffer_pool_instances defines the amount of work performed by the page cleaner thread each second.

The innodb_flush_neighbors and innodb_lru_scan_depth variables are primarily intended for write-intensive workloads. With heavy DML activity, flushing can fall behind if it is not aggressive enough, or disk writes can saturate I/O capacity if flushing is too aggressive. The ideal settings depend on your workload, data access patterns, and storage configuration (for example, whether data is stored on HDD or SSD devices).

Adaptive Flushing

InnoDB uses an adaptive flushing algorithm to dynamically adjust the rate of flushing based on the speed of redo log generation and the current rate of flushing. The intent is to smooth overall performance by ensuring that flushing activity keeps pace with the current workload. Automatically

adjusting the flushing rate helps avoid sudden dips in throughput that can occur when bursts of I/O activity due to buffer pool flushing affects the I/O capacity available for ordinary read and write activity.

Sharp checkpoints, which are typically associated with write-intensive workloads that generate a lot of redo entries, can cause a sudden change in throughput, for example. A sharp checkpoint occurs when `InnoDB` wants to reuse a portion of a log file. Before doing so, all dirty pages with redo entries in that portion of the log file must be flushed. If log files become full, a sharp checkpoint occurs, causing a temporary reduction in throughput. This scenario can occur even if `innodb_max_dirty_pages_pct` threshold is not reached.

The adaptive flushing algorithm helps avoid such scenarios by tracking the number of dirty pages in the buffer pool and the rate at which redo log records are being generated. Based on this information, it decides how many dirty pages to flush from the buffer pool each second, which permits it to manage sudden changes in workload.

The `innodb_adaptive_flushing_lwm` variable defines a low water mark for redo log capacity. When that threshold is crossed, adaptive flushing is enabled, even if the `innodb_adaptive_flushing` variable is disabled.

Internal benchmarking has shown that the algorithm not only maintains throughput over time, but can also improve overall throughput significantly. However, adaptive flushing can affect the I/O pattern of a workload significantly and may not be appropriate in all cases. It gives the most benefit when the redo log is in danger of filling up. If adaptive flushing is not appropriate to the characteristics of your workload, you can disable it. Adaptive flushing controlled by the `innodb_adaptive_flushing` variable, which is enabled by default.

`innodb_flushing_avg_loops` defines the number of iterations that `InnoDB` keeps the previously calculated snapshot of the flushing state, controlling how quickly adaptive flushing responds to foreground workload changes. A high `innodb_flushing_avg_loops` value means that `InnoDB` keeps the previously calculated snapshot longer, so adaptive flushing responds more slowly. When setting a high value it is important to ensure that redo log utilization does not reach 75% (the hardcoded limit at which asynchronous flushing starts), and that the `innodb_max_dirty_pages_pct` threshold keeps the number of dirty pages to a level that is appropriate for the workload.

Systems with consistent workloads, a large log file size (`innodb_log_file_size`), and small spikes that do not reach 75% log space utilization should use a high `innodb_flushing_avg_loops` value to keep flushing as smooth as possible. For systems with extreme load spikes or log files that do not provide a lot of space, a smaller value allows flushing to closely track workload changes, and helps to avoid reaching 75% log space utilization.

Be aware that if flushing falls behind, the rate of buffer pool flushing can exceed the I/O capacity available to `InnoDB`, as defined by `innodb_io_capacity` setting. The `innodb_io_capacity_max` value defines an upper limit on I/O capacity in such situations, so that a spike in I/O activity does not consume the entire I/O capacity of the server.

The `innodb_io_capacity` setting is applicable to all buffer pool instances. When dirty pages are flushed, I/O capacity is divided equally among buffer pool instances.

14.8.3.5 Saving and Restoring the Buffer Pool State

To avoid a lengthy warmup period after restarting the server, particularly for instances with large buffer pools, you can save the buffer pool state at server shutdown and restore the buffer pool to the same state at server startup.

After restarting a busy server, there is typically a warmup period with steadily increasing throughput, as disk pages that were in the buffer pool are brought back into memory (as the same data is queried, updated, and so on). The ability to restore the buffer pool to the pre-shutdown state shortens the warmup period by reloading disk pages that were in the buffer pool before the restart rather than waiting for DML operations to access corresponding rows. Also, I/O requests can be performed in large batches, making the overall I/O faster. Page loading happens in the background, and does not delay database startup.

In addition to saving the buffer pool state at shutdown and restoring it at startup, you can save and restore the buffer pool state at any time, while the server is running. For example, you can save the state of the buffer pool after reaching a stable throughput under a steady workload. You could also restore the previous buffer pool state after running reports or maintenance jobs that bring data pages into the buffer pool that are only requited for those operations, or after running some other non-typical workload.

Even though a buffer pool can be many gigabytes in size, the buffer pool data that `InnoDB` saves to disk is tiny by comparison. Only tablespace IDs and page IDs necessary to locate the appropriate pages are saved to disk. This information is derived from the `INNODB_BUFFER_PAGE_LRU INFORMATION_SCHEMA` table. By default, tablespace ID and page ID data is saved in a file named `ib_buffer_pool`, which is saved to the `InnoDB` data directory. The file name and location can be modified using the `innodb_buffer_pool_filename` configuration parameter.

Because data is cached in and aged out of the buffer pool as it is with regular database operations, there is no problem if the disk pages are recently updated, or if a DML operation involves data that has not yet been loaded. The loading mechanism skips requested pages that no longer exist.

The underlying mechanism involves a background thread that is dispatched to perform the dump and load operations.

Disk pages from compressed tables are loaded into the buffer pool in their compressed form. Pages are uncompressed as usual when page contents are accessed during DML operations. Because uncompressing pages is a CPU-intensive process, it is more efficient for concurrency to perform the operation in a connection thread rather than in the single thread that performs the buffer pool restore operation.

Operations related to saving and restoring the buffer pool state are described in the following topics:

- Saving the Buffer Pool State at Shutdown and Restoring it at Startup
- Saving and Restoring the Buffer Pool State Online
- Displaying Buffer Pool Dump Progress
- Displaying Buffer Pool Load Progress
- Aborting a Buffer Pool Load Operation

Saving the Buffer Pool State at Shutdown and Restoring it at Startup

To save the state of the buffer pool at server shutdown, issue the following statement prior to shutting down the server:

```
SET GLOBAL innodb_buffer_pool_dump_at_shutdown=ON;
```

To restore the buffer pool state at server startup, specify the `--innodb-buffer-pool-load-at-startup` option when starting the server:

```
mysqld --innodb-buffer-pool-load-at-startup=ON;
```

Saving and Restoring the Buffer Pool State Online

To save the state of the buffer pool while MySQL server is running, issue the following statement:

```
SET GLOBAL innodb_buffer_pool_dump_now=ON;
```

To restore the buffer pool state while MySQL is running, issue the following statement:

```
SET GLOBAL innodb_buffer_pool_load_now=ON;
```

Displaying Buffer Pool Dump Progress

To display progress when saving the buffer pool state to disk, use one of the following options:

```
SHOW STATUS LIKE 'Innodb_buffer_pool_dump_status';
```

or:

```
SELECT variable_value FROM information_schema.global_status WHERE
variable_name = 'INNODB_BUFFER_POOL_DUMP_STATUS';
```

If the operation has not yet started, “not started” is returned. If the operation is complete, the completion time is printed (e.g. Finished at 110505 12:18:02). If the operation is in progress, status information is provided (e.g. Dumping buffer pool 5/7, page 237/2873).

Displaying Buffer Pool Load Progress

To display progress when loading the buffer pool, use one of the following options:

```
SHOW STATUS LIKE 'Innodb_buffer_pool_load_status';
```

or:

```
SELECT variable_value FROM information_schema.global_status WHERE
variable_name = 'INNODB_BUFFER_POOL_LOAD_STATUS';
```

If the operation has not yet started, “not started” is returned. If the operation is complete, the completion time is printed (e.g. Finished at 110505 12:23:24). If the operation is in progress, status information is provided (e.g. Loaded 123/22301 pages).

Aborting a Buffer Pool Load Operation

To abort a buffer pool load operation, issue the following statement:

```
SET GLOBAL innodb_buffer_pool_load_abort=ON;
```

14.8.4 Configuring the Memory Allocator for InnoDB

When `InnoDB` was developed, the memory allocators supplied with operating systems and run-time libraries were often lacking in performance and scalability. At that time, there were no memory allocator libraries tuned for multi-core CPUs. Therefore, `InnoDB` implemented its own memory allocator in the `mem` subsystem. This allocator is guarded by a single mutex, which may become a bottleneck. `InnoDB` also implements a wrapper interface around the system allocator (`malloc` and `free`) that is likewise guarded by a single mutex.

Today, as multi-core systems have become more widely available, and as operating systems have matured, significant improvements have been made in the memory allocators provided with operating systems. These new memory allocators perform better and are more scalable than they were in the past. Most workloads, especially those where memory is frequently allocated and released (such as multi-table joins), benefit from using a more highly tuned memory allocator as opposed to the internal, `InnoDB`-specific memory allocator.

You can control whether `InnoDB` uses its own memory allocator or an allocator of the operating system, by setting the value of the system configuration parameter `innodb_use_sys_malloc` in the MySQL option file (`my.cnf` or `my.ini`). If set to `ON` or `1` (the default), `InnoDB` uses the `malloc` and `free` functions of the underlying system rather than manage memory pools itself. This parameter is not dynamic, and takes effect only when the system is started. To continue to use the `InnoDB` memory allocator, set `innodb_use_sys_malloc` to `0`.

When the `InnoDB` memory allocator is disabled, `InnoDB` ignores the value of the parameter `innodb_additional_mem_pool_size`. The `InnoDB` memory allocator uses an additional memory pool for satisfying allocation requests without having to fall back to the system memory allocator. When the `InnoDB` memory allocator is disabled, all such allocation requests are fulfilled by the system memory allocator.

On Unix-like systems that use dynamic linking, replacing the memory allocator may be as easy as making the environment variable `LD_PRELOAD` or `LD_LIBRARY_PATH` point to the dynamic library

that implements the allocator. On other systems, some relinking may be necessary. Please refer to the documentation of the memory allocator library of your choice.

Since `InnoDB` cannot track all memory use when the system memory allocator is used (`innodb_use_sys_malloc` is `ON`), the section "BUFFER POOL AND MEMORY" in the output of the `SHOW ENGINE INNODB STATUS` command only includes the buffer pool statistics in the "Total memory allocated". Any memory allocated using the `mem` subsystem or using `ut_malloc` is excluded.

Note

`innodb_use_sys_malloc` and `innodb_additional_mem_pool_size` were deprecated in MySQL 5.6.3 and are removed in MySQL 5.7.4.

For more information about the performance implications of `InnoDB` memory usage, see Section 8.10, "Buffering and Caching".

14.8.5 Configuring Thread Concurrency for InnoDB

`InnoDB` uses operating system threads to process requests from user transactions. (Transactions may issue many requests to `InnoDB` before they commit or roll back.) On modern operating systems and servers with multi-core processors, where context switching is efficient, most workloads run well without any limit on the number of concurrent threads. Scalability improvements in MySQL 5.5 and up reduce the need to limit the number of concurrently executing threads inside `InnoDB`.

In situations where it is helpful to minimize context switching between threads, `InnoDB` can use a number of techniques to limit the number of concurrently executing operating system threads (and thus the number of requests that are processed at any one time). When `InnoDB` receives a new request from a user session, if the number of threads concurrently executing is at a pre-defined limit, the new request sleeps for a short time before it tries again. A request that cannot be rescheduled after the sleep is put in a first-in/first-out queue and eventually is processed. Threads waiting for locks are not counted in the number of concurrently executing threads.

You can limit the number of concurrent threads by setting the configuration parameter `innodb_thread_concurrency`. Once the number of executing threads reaches this limit, additional threads sleep for a number of microseconds, set by the configuration parameter `innodb_thread_sleep_delay`, before being placed into the queue.

Previously, it required experimentation to find the optimal value for `innodb_thread_sleep_delay`, and the optimal value could change depending on the workload. In MySQL 5.6.3 and higher, you can set the configuration option `innodb_adaptive_max_sleep_delay` to the highest value you would allow for `innodb_thread_sleep_delay`, and `InnoDB` automatically adjusts `innodb_thread_sleep_delay` up or down depending on the current thread-scheduling activity. This dynamic adjustment helps the thread scheduling mechanism to work smoothly during times when the system is lightly loaded and when it is operating near full capacity.

The default value for `innodb_thread_concurrency` and the implied default limit on the number of concurrent threads has been changed in various releases of MySQL and `InnoDB`. The default value of `innodb_thread_concurrency` is `0`, so that by default there is no limit on the number of concurrently executing threads.

`InnoDB` causes threads to sleep only when the number of concurrent threads is limited. When there is no limit on the number of threads, all contend equally to be scheduled. That is, if `innodb_thread_concurrency` is `0`, the value of `innodb_thread_sleep_delay` is ignored.

When there is a limit on the number of threads (when `innodb_thread_concurrency` is > 0), `InnoDB` reduces context switching overhead by permitting multiple requests made during the execution of a *single SQL statement* to enter `InnoDB` without observing the limit set by `innodb_thread_concurrency`. Since an SQL statement (such as a join) may comprise multiple row operations within `InnoDB`, `InnoDB` assigns a specified number of "tickets" that allow a thread to be scheduled repeatedly with minimal overhead.

When a new SQL statement starts, a thread has no tickets, and it must observe `innodb_thread_concurrency`. Once the thread is entitled to enter `InnoDB`, it is assigned a number of tickets that it can use for subsequently entering `InnoDB` to perform row operations. If the tickets run out, the thread is evicted, and `innodb_thread_concurrency` is observed again which may place the thread back into the first-in/first-out queue of waiting threads. When the thread is once again entitled to enter `InnoDB`, tickets are assigned again. The number of tickets assigned is specified by the global option `innodb_concurrency_tickets`, which is 5000 by default. A thread that is waiting for a lock is given one ticket once the lock becomes available.

The correct values of these variables depend on your environment and workload. Try a range of different values to determine what value works for your applications. Before limiting the number of concurrently executing threads, review configuration options that may improve the performance of `InnoDB` on multi-core and multi-processor computers, such as `innodb_adaptive_hash_index`.

For general performance information about MySQL thread handling, see Section 5.1.11.1, “Connection Interfaces”.

14.8.6 Configuring the Number of Background InnoDB I/O Threads

`InnoDB` uses background threads to service various types of I/O requests. You can configure the number of background threads that service read and write I/O on data pages using the `innodb_read_io_threads` and `innodb_write_io_threads` configuration parameters. These parameters signify the number of background threads used for read and write requests, respectively. They are effective on all supported platforms. You can set values for these parameters in the MySQL option file (`my.cnf` or `my.ini`); you cannot change values dynamically. The default value for these parameters is `4` and permissible values range from `1-64`.

The purpose of these configuration options to make `InnoDB` more scalable on high end systems. Each background thread can handle up to 256 pending I/O requests. A major source of background I/O is read-ahead requests. `InnoDB` tries to balance the load of incoming requests in such way that most background threads share work equally. `InnoDB` also attempts to allocate read requests from the same extent to the same thread, to increase the chances of coalescing the requests. If you have a high end I/O subsystem and you see more than 64 × `innodb_read_io_threads` pending read requests in `SHOW ENGINE INNODB STATUS` output, you might improve performance by increasing the value of `innodb_read_io_threads`.

On Linux systems, `InnoDB` uses the asynchronous I/O subsystem by default to perform read-ahead and write requests for data file pages, which changes the way that `InnoDB` background threads service these types of I/O requests. For more information, see Section 14.8.7, “Using Asynchronous I/O on Linux”.

For more information about `InnoDB` I/O performance, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

14.8.7 Using Asynchronous I/O on Linux

`InnoDB` uses the asynchronous I/O subsystem (native AIO) on Linux to perform read-ahead and write requests for data file pages. This behavior is controlled by the `innodb_use_native_aio` configuration option, which applies to Linux systems only and is enabled by default. On other Unix-like systems, `InnoDB` uses synchronous I/O only. Historically, `InnoDB` only used asynchronous I/O on Windows systems. Using the asynchronous I/O subsystem on Linux requires the `libaio` library.

With synchronous I/O, query threads queue I/O requests, and `InnoDB` background threads retrieve the queued requests one at a time, issuing a synchronous I/O call for each. When an I/O request is completed and the I/O call returns, the `InnoDB` background thread that is handling the request calls an I/O completion routine and returns to process the next request. The number of requests that can be processed in parallel is *n*, where *n* is the number of `InnoDB` background threads. The number of `InnoDB` background threads is controlled by `innodb_read_io_threads` and `innodb_write_io_threads`. See Section 14.8.6, “Configuring the Number of Background InnoDB I/O Threads”.

With native AIO, query threads dispatch I/O requests directly to the operating system, thereby removing the limit imposed by the number of background threads. `InnoDB` background threads wait for I/O events to signal completed requests. When a request is completed, a background thread calls an I/O completion routine and resumes waiting for I/O events.

The advantage of native AIO is scalability for heavily I/O-bound systems that typically show many pending reads/writes in `SHOW ENGINE INNODB STATUS\G` output. The increase in parallel processing when using native AIO means that the type of I/O scheduler or properties of the disk array controller have a greater influence on I/O performance.

A potential disadvantage of native AIO for heavily I/O-bound systems is lack of control over the number of I/O write requests dispatched to the operating system at once. Too many I/O write requests dispatched to the operating system for parallel processing could, in some cases, result in I/O read starvation, depending on the amount of I/O activity and system capabilities.

If a problem with the asynchronous I/O subsystem in the OS prevents `InnoDB` from starting, you can start the server with `innodb_use_native_aio=0`. This option may also be disabled automatically during startup if `InnoDB` detects a potential problem such as a combination of `tmpdir` location, `tmpfs` file system, and Linux kernel that does not support asynchronous I/O on `tmpfs`.

14.8.8 Configuring InnoDB I/O Capacity

The `InnoDB` master thread and other threads perform various tasks in the background, most of which are I/O related, such as flushing dirty pages from the buffer pool and writing changes from the change buffer to the appropriate secondary indexes. `InnoDB` attempts to perform these tasks in a way that does not adversely affect the normal working of the server. It tries to estimate the available I/O bandwidth and tune its activities to take advantage of available capacity.

The `innodb_io_capacity` variable defines the overall I/O capacity available to `InnoDB`. It should be set to approximately the number of I/O operations that the system can perform per second (IOPS). When `innodb_io_capacity` is set, `InnoDB` estimates the I/O bandwidth available for background tasks based on the set value.

You can set `innodb_io_capacity` to a value of 100 or greater. The default value is `200`. Typically, values around 100 are appropriate for consumer-level storage devices, such as hard drives up to 7200 RPMs. Faster hard drives, RAID configurations, and solid state drives (SSDs) benefit from higher values.

Ideally, keep the setting as low as practical, but not so low that background activities fall behind. If the value is too high, data is removed from the buffer pool and change buffer too quickly for caching to provide a significant benefit. For busy systems capable of higher I/O rates, you can set a higher value to help the server handle the background maintenance work associated with a high rate of row changes. Generally, you can increase the value as a function of the number of drives used for `InnoDB` I/O. For example, you can increase the value on systems that use multiple disks or SSDs.

The default setting of 200 is generally sufficient for a lower-end SSD. For a higher-end, bus-attached SSD, consider a higher setting such as 1000, for example. For systems with individual 5400 RPM or 7200 RPM drives, you might lower the value to 100, which represents an estimated proportion of the I/O operations per second (IOPS) available to older-generation disk drives that can perform about 100 IOPS.

Although you can specify a high value such as a million, in practice such large values have little benefit. Generally, a value higher than 20000 is not recommended unless you are certain that lower values are insufficient for your workload.

Consider write workload when tuning `innodb_io_capacity`. Systems with large write workloads are likely to benefit from a higher setting. A lower setting may be sufficient for systems with a small write workload.

The `innodb_io_capacity` setting is not a per buffer pool instance setting. Available I/O capacity is distributed equally among buffer pool instances for flushing activities.

You can set the innodb_io_capacity value in the MySQL option file (my.cnf or my.ini) or modify it at runtime using a SET GLOBAL statement, which requires privileges sufficient to set global system variables. See Section 5.1.8.1, “System Variable Privileges”.

Configuring an I/O Capacity Maximum

If flushing activity falls behind, InnoDB can flush more aggressively, at a higher rate of I/O operations per second (IOPS) than defined by the innodb_io_capacity variable. The innodb_io_capacity_max variable defines a maximum number of IOPS performed by InnoDB background tasks in such situations.

If you specify an innodb_io_capacity setting at startup but do not specify a value for innodb_io_capacity_max, innodb_io_capacity_max defaults to twice the value of innodb_io_capacity, with a minimum value of 2000.

When configuring innodb_io_capacity_max, twice the innodb_io_capacity is often a good starting point. The default value of 2000 is intended for workloads that use an SSD or more than one regular disk drive. A setting of 2000 is likely too high for workloads that do not use SSDs or multiple disk drives, and could allow too much flushing. For a single regular disk drive, a setting between 200 and 400 is recommended. For a high-end, bus-attached SSD, consider a higher setting such as 2500. As with the innodb_io_capacity setting, keep the setting as low as practical, but not so low that InnoDB cannot sufficiently extend rate of IOPS beyond the innodb_io_capacity setting.

Consider write workload when tuning innodb_io_capacity_max. Systems with large write workloads may benefit from a higher setting. A lower setting may be sufficient for systems with a small write workload.

innodb_io_capacity_max cannot be set to a value lower than the innodb_io_capacity value.

Setting innodb_io_capacity_max to DEFAULT using a SET statement (SET GLOBAL innodb_io_capacity_max=DEFAULT) sets innodb_io_capacity_max to the maximum value.

The innodb_io_capacity_max limit applies to all buffer pool instances. It is not a per buffer pool instance setting.

14.8.9 Configuring Spin Lock Polling

InnoDB mutexes and rw-locks are typically reserved for short intervals. On a multi-core system, it can be more efficient for a thread to continuously check if it can acquire a mutex or rw-lock for a period of time before it sleeps. If the mutex or rw-lock becomes available during this period, the thread can continue immediately, in the same time slice. However, too-frequent polling of a shared object such as a mutex or rw-lock by multiple threads can cause “cache ping pong”, which results in processors invalidating portions of each other's cache. InnoDB minimizes this issue by forcing a random delay between polls to desychronize polling activity. The random delay is implemented as a spin-wait loop.

The duration of a spin-wait loop is determined by the number of PAUSE instructions that occur in the loop. That number is generated by randomly selecting an integer ranging from 0 up to but not including the innodb_spin_wait_delay value, and multiplying that value by 50. For example, an integer is randomly selected from the following range for an innodb_spin_wait_delay setting of 6:

```
{0,1,2,3,4,5}
```

The selected integer is multiplied by 50, resulting in one of six possible PAUSE instruction values:

```
{0,50,100,150,200,250}
```

For that set of values, 250 is the maximum number of PAUSE instructions that can occur in a spin-wait loop. An innodb_spin_wait_delay setting of 5 results in a set of five possible values {0,50,100,150,200}, where 200 is the maximum number of PAUSE instructions, and so on. In this way, the innodb_spin_wait_delay setting controls the maximum delay between spin lock polls.

The duration of the delay loop depends on the C compiler and the target processor. In the 100MHz Pentium era, an `innodb_spin_wait_delay` unit was calibrated to be equivalent to one microsecond. That time equivalence did not hold, but PAUSE instruction duration has remained fairly constant in terms of processor cycles relative to other CPU instructions on most processor architectures.

On a system where all processor cores share a fast cache memory, you might reduce the maximum delay or disable the busy loop altogether by setting `innodb_spin_wait_delay=0`. On a system with multiple processor chips, the effect of cache invalidation can be more significant and you might increase the maximum delay.

The `innodb_spin_wait_delay` variable is dynamic. It can be specified in a MySQL option file or modified at runtime using a `SET GLOBAL` statement. Runtime modification requires privileges sufficient to set global system variables. See Section 5.1.8.1, “System Variable Privileges”.

14.8.10 Purge Configuration

`InnoDB` does not physically remove a row from the database immediately when you delete it with an SQL statement. A row and its index records are only physically removed when `InnoDB` discards the undo log record written for the deletion. This removal operation, which only occurs after the row is no longer required for multi-version concurrency control (MVCC) or rollback, is called a purge.

Purge runs on a periodic schedule. It parses and processes undo log pages from the history list, which is a list of undo log pages for committed transactions that is maintained by the `InnoDB` transaction system. Purge frees the undo log pages from the history list after processing them.

Configuring Purge Threads

Purge operations are performed in the background by one or more purge threads. The number of purge threads is controlled by the `innodb_purge_threads` variable. The default value is 1. If DML action is concentrated on a single table or a few tables, keep the setting low so that the threads do not contend with each other for access to the busy tables. If DML operations are spread across many tables, increase the setting. The maximum number of purge threads is 32.

The `innodb_purge_threads` setting is the maximum number of purge threads permitted. The purge system automatically adjusts the number of purge threads as necessary.

Configuring Purge Batch Size

The `innodb_purge_batch_size` variable defines the number of undo log pages that purge parses and processes in one batch from the history list. The default value is 300. In a multithreaded purge configuration, the coordinator purge thread divides `innodb_purge_batch_size` by `innodb_purge_threads` and assigns that number of pages to each purge thread.

The purge system also frees the undo log pages that are no longer required. It does so every 128 iterations through the undo logs. In addition to defining the number of undo log pages parsed and processed in a batch, the `innodb_purge_batch_size` variable defines the number of undo log pages that purge frees every 128 iterations through the undo logs.

The `innodb_purge_batch_size` variable is intended for advanced performance tuning and experimentation. Most users need not change `innodb_purge_batch_size` from its default value.

Configuring the Maximum Purge Lag

The `innodb_max_purge_lag` variable defines the desired maximum purge lag. When the purge lag exceeds the `innodb_max_purge_lag` threshold, a delay is imposed on `INSERT`, `UPDATE`, and `DELETE` operations to allow time for purge operations to catch up. The default value is 0, which means there is no maximum purge lag and no delay.

The `InnoDB` transaction system maintains a list of transactions that have index records delete-marked by `UPDATE` or `DELETE` operations. The length of the list is the purge lag. The purge lag delay is calculated by the following formula, which results in a minimum delay of 5000 microseconds:

```
(purge lag/innodb_max_purge_lag - 0.5) * 10000
```

The delay is calculated at the beginning of a purge batch.

A typical `innodb_max_purge_lag` setting for a problematic workload might be 1000000 (1 million), assuming that transactions are small, only 100 bytes in size, and it is permissible to have 100MB of unpurged table rows.

The purge lag is presented as the `History list length` value in the `TRANSACTIONS` section of `SHOW ENGINE INNODB STATUS` output.

```
mysql> SHOW ENGINE INNODB STATUS;
...
------------
TRANSACTIONS
------------
Trx id counter 0 290328385
Purge done for trx's n:o < 0 290315608 undo n:o < 0 17
History list length 20
```

The `History list length` is typically a low value, usually less than a few thousand, but a write-heavy workload or long running transactions can cause it to increase, even for transactions that are read only. The reason that a long running transaction can cause the `History list length` to increase is that under a consistent read transaction isolation level such as `REPEATABLE READ`, a transaction must return the same result as when the read view for that transaction was created. Consequently, the `InnoDB` multi-version concurrency control (MVCC) system must keep a copy of the data in the undo log until all transactions that depend on that data have completed. The following are examples of long running transactions that could cause the `History list length` to increase:

- A `mysqldump` operation that uses the `--single-transaction` option while there is a significant amount of concurrent DML.
- Running a `SELECT` query after disabling `autocommit`, and forgetting to issue an explicit `COMMIT` or `ROLLBACK`.

To prevent excessive delays in extreme situations where the purge lag becomes huge, you can limit the delay by setting the `innodb_max_purge_lag_delay` variable. The `innodb_max_purge_lag_delay` variable specifies the maximum delay in microseconds for the delay imposed when the `innodb_max_purge_lag` threshold is exceeded. The specified `innodb_max_purge_lag_delay` value is an upper limit on the delay period calculated by the `innodb_max_purge_lag` formula.

14.8.11 Configuring Optimizer Statistics for InnoDB

This section describes how to configure persistent and non-persistent optimizer statistics for `InnoDB` tables. Persistent optimizer statistics were introduced in MySQL 5.6.2 and were made the default in MySQL 5.6.6 (`innodb_stats_persistent=ON`).

Persistent optimizer statistics are persisted across server restarts, allowing for greater plan stability and more consistent query performance. Persistent optimizer statistics also provide control and flexibility with these additional benefits:

- You can use the `innodb_stats_auto_recalc` configuration option to control whether statistics are updated automatically after substantial changes to a table.
- You can use the `STATS_PERSISTENT`, `STATS_AUTO_RECALC`, and `STATS_SAMPLE_PAGES` clauses with `CREATE TABLE` and `ALTER TABLE` statements to configure optimizer statistics for individual tables.
- You can query optimizer statistics data in the `mysql.innodb_table_stats` and `mysql.innodb_index_stats` tables.
- You can view the `last_update` column of the `mysql.innodb_table_stats` and `mysql.innodb_index_stats` tables to see when statistics were last updated.

- You can manually modify the `mysql.innodb_table_stats` and `mysql.innodb_index_stats` tables to force a specific query optimization plan or to test alternative plans without modifying the database.

Non-persistent optimizer statistics are cleared on each server restart and after some other operations, and recomputed on the next table access. As a result, different estimates could be produced when recomputing statistics, leading to different choices in execution plans and variations in query performance.

This section also provides information about estimating `ANALYZE TABLE` complexity, which may be useful when attempting to achieve a balance between accurate statistics and `ANALYZE TABLE` execution time.

14.8.11.1 Configuring Persistent Optimizer Statistics Parameters

The persistent optimizer statistics feature improves plan stability by storing statistics to disk and making them persistent across server restarts so that the optimizer is more likely to make consistent choices each time for a given query.

Optimizer statistics are persisted to disk when `innodb_stats_persistent=ON` or when individual tables are defined with `STATS_PERSISTENT=1`. `innodb_stats_persistent` is enabled by default as of MySQL 5.6.6.

Formerly, optimizer statistics were cleared when restarting the server and after some other types of operations, and recomputed on the next table access. Consequently, different estimates could be produced when recalculating statistics leading to different choices in query execution plans and variation in query performance.

Persistent statistics are stored in the `mysql.innodb_table_stats` and `mysql.innodb_index_stats` tables. See InnoDB Persistent Statistics Tables.

If you prefer not to persist optimizer statistics to disk, see Section 14.8.11.2, “Configuring Non-Persistent Optimizer Statistics Parameters”

Configuring Automatic Statistics Calculation for Persistent Optimizer Statistics

The `innodb_stats_auto_recalc` variable, which is enabled by default, controls whether statistics are calculated automatically when a table undergoes changes to more than 10% of its rows. You can also configure automatic statistics recalculation for individual tables by specifying the `STATS_AUTO_RECALC` clause when creating or altering a table.

Because of the asynchronous nature of automatic statistics recalculation, which occurs in the background, statistics may not be recalculated instantly after running a DML operation that affects more than 10% of a table, even when `innodb_stats_auto_recalc` is enabled. Statistics recalculation can be delayed by few seconds in some cases. If up-to-date statistics are required immediately, run `ANALYZE TABLE` to initiate a synchronous (foreground) recalculation of statistics.

If `innodb_stats_auto_recalc` is disabled, you can ensure the accuracy of optimizer statistics by executing the `ANALYZE TABLE` statement after making substantial changes to indexed columns. You might also consider adding `ANALYZE TABLE` to setup scripts that you run after loading data, and running `ANALYZE TABLE` on a schedule at times of low activity.

When an index is added to an existing table, or when a column is added or dropped, index statistics are calculated and added to the `innodb_index_stats` table regardless of the value of `innodb_stats_auto_recalc`.

Configuring Optimizer Statistics Parameters for Individual Tables

`innodb_stats_persistent`, `innodb_stats_auto_recalc`, and `innodb_stats_persistent_sample_pages` are global variables. To override these system-wide settings and configure optimizer statistics parameters for individual tables, you can define

STATS_PERSISTENT, STATS_AUTO_RECALC, and STATS_SAMPLE_PAGES clauses in CREATE TABLE or ALTER TABLE statements.

- STATS_PERSISTENT specifies whether to enable persistent statistics for an InnoDB table. The value DEFAULT causes the persistent statistics setting for the table to be determined by the innodb_stats_persistent setting. A value of 1 enables persistent statistics for the table, while a value of 0 disables the feature. After enabling persistent statistics for an individual table, use ANALYZE TABLE to calculate statistics after table data is loaded.
- STATS_AUTO_RECALC specifies whether to automatically recalculate persistent statistics. The value DEFAULT causes the persistent statistics setting for the table to be determined by the innodb_stats_auto_recalc setting. A value of 1 causes statistics to be recalculated when 10% of table data has changed. A value 0 prevents automatic recalculation for the table. When using a value of 0, use ANALYZE TABLE to recalculate statistics after making substantial changes to the table.
- STATS_SAMPLE_PAGES specifies the number of index pages to sample when cardinality and other statistics are calculated for an indexed column, by an ANALYZE TABLE operation, for example.

All three clauses are specified in the following CREATE TABLE example:

```
CREATE TABLE `t1` (
`id` int(8) NOT NULL auto_increment,
`data` varchar(255),
`date` datetime,
PRIMARY KEY  (`id`),
INDEX `DATE_IX` (`date`)
) ENGINE=InnoDB,
  STATS_PERSISTENT=1,
  STATS_AUTO_RECALC=1,
  STATS_SAMPLE_PAGES=25;
```

Configuring the Number of Sampled Pages for InnoDB Optimizer Statistics

The optimizer uses estimated statistics about key distributions to choose the indexes for an execution plan, based on the relative selectivity of the index. Operations such as ANALYZE TABLE cause InnoDB to sample random pages from each index on a table to estimate the cardinality of the index. This sampling technique is known as a random dive.

The innodb_stats_persistent_sample_pages controls the number of sampled pages. You can adjust the setting at runtime to manage the quality of statistics estimates used by the optimizer. The default value is 20. Consider modifying the setting when encountering the following issues:

1. *Statistics are not accurate enough and the optimizer chooses suboptimal plans*, as shown in EXPLAIN output. You can check the accuracy of statistics by comparing the actual cardinality of an index (determined by running SELECT DISTINCT on the index columns) with the estimates in the mysql.innodb_index_stats table.

   If it is determined that statistics are not accurate enough, the value of innodb_stats_persistent_sample_pages should be increased until the statistics estimates are sufficiently accurate. Increasing innodb_stats_persistent_sample_pages too much, however, could cause ANALYZE TABLE to run slowly.

2. *ANALYZE TABLE is too slow*. In this case innodb_stats_persistent_sample_pages should be decreased until ANALYZE TABLE execution time is acceptable. Decreasing the value too much, however, could lead to the first problem of inaccurate statistics and suboptimal query execution plans.

   If a balance cannot be achieved between accurate statistics and ANALYZE TABLE execution time, consider decreasing the number of indexed columns in the table or limiting the number of partitions to reduce ANALYZE TABLE complexity. The number of columns in the table's primary key is also important to consider, as primary key columns are appended to each nonunique index.

For related information, see Section 14.8.11.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”.

Including Delete-marked Records in Persistent Statistics Calculations

By default, `InnoDB` reads uncommitted data when calculating statistics. In the case of an uncommitted transaction that deletes rows from a table, delete-marked records are excluded when calculating row estimates and index statistics, which can lead to non-optimal execution plans for other transactions that are operating on the table concurrently using a transaction isolation level other than `READ UNCOMMITTED`. To avoid this scenario, `innodb_stats_include_delete_marked` can be enabled to ensure that delete-marked records are included when calculating persistent optimizer statistics.

When `innodb_stats_include_delete_marked` is enabled, `ANALYZE TABLE` considers delete-marked records when recalculating statistics.

`innodb_stats_include_delete_marked` is a global setting that affects all `InnoDB` tables, and it is only applicable to persistent optimizer statistics.

`innodb_stats_include_delete_marked` was introduced in MySQL 5.6.34.

InnoDB Persistent Statistics Tables

The persistent statistics feature relies on the internally managed tables in the `mysql` database, named `innodb_table_stats` and `innodb_index_stats`. These tables are set up automatically in all install, upgrade, and build-from-source procedures.

Table 14.3 Columns of innodb_table_stats

Column name	Description
`database_name`	Database name
`table_name`	Table name, partition name, or subpartition name
`last_update`	A timestamp indicating the last time that `InnoDB` updated this row
`n_rows`	The number of rows in the table
`clustered_index_size`	The size of the primary index, in pages
`sum_of_other_index_sizes`	The total size of other (non-primary) indexes, in pages

Table 14.4 Columns of innodb_index_stats

Column name	Description
`database_name`	Database name
`table_name`	Table name, partition name, or subpartition name
`index_name`	Index name
`last_update`	A timestamp indicating the last time that `InnoDB` updated this row
`stat_name`	The name of the statistic, whose value is reported in the `stat_value` column
`stat_value`	The value of the statistic that is named in `stat_name` column
`sample_size`	The number of pages sampled for the estimate provided in the `stat_value` column
`stat_description`	Description of the statistic that is named in the `stat_name` column

The `innodb_table_stats` and `innodb_index_stats` tables include a `last_update` column that shows when index statistics were last updated:

```
mysql> SELECT * FROM innodb_table_stats \G
```

```
*************************** 1. row ***************************
           database_name: sakila
              table_name: actor
             last_update: 2014-05-28 16:16:44
                 n_rows: 200
    clustered_index_size: 1
sum_of_other_index_sizes: 1
...
```

```
mysql> SELECT * FROM innodb_index_stats \G
*************************** 1. row ***************************
   database_name: sakila
      table_name: actor
      index_name: PRIMARY
     last_update: 2014-05-28 16:16:44
       stat_name: n_diff_pfx01
      stat_value: 200
     sample_size: 1
     ...
```

The `innodb_table_stats` and `innodb_index_stats` tables can be updated manually, which makes it possible to force a specific query optimization plan or test alternative plans without modifying the database. If you manually update statistics, use the `FLUSH TABLE` *`tbl_name`* statement to load the updated statistics.

Persistent statistics are considered local information, because they relate to the server instance. The `innodb_table_stats` and `innodb_index_stats` tables are therefore not replicated when automatic statistics recalculation takes place. If you run `ANALYZE TABLE` to initiate a synchronous recalculation of statistics, this statement is replicated (unless you suppressed logging for it), and recalculation takes place on the replicas.

InnoDB Persistent Statistics Tables Example

The `innodb_table_stats` table contains one row for each table. The following example demonstrates the type of data collected.

Table `t1` contains a primary index (columns `a`, `b`) secondary index (columns `c`, `d`), and unique index (columns `e`, `f`):

```
CREATE TABLE t1 (
a INT, b INT, c INT, d INT, e INT, f INT,
PRIMARY KEY (a, b), KEY i1 (c, d), UNIQUE KEY i2uniq (e, f)
) ENGINE=INNODB;
```

After inserting five rows of sample data, table `t1` appears as follows:

```
mysql> SELECT * FROM t1;
+---+---+------+------+------+------+
| a | b | c    | d    | e    | f    |
+---+---+------+------+------+------+
| 1 | 1 |   10 |   11 |  100 |  101 |
| 1 | 2 |   10 |   11 |  200 |  102 |
| 1 | 3 |   10 |   11 |  100 |  103 |
| 1 | 4 |   10 |   12 |  200 |  104 |
| 1 | 5 |   10 |   12 |  100 |  105 |
+---+---+------+------+------+------+
```

To immediately update statistics, run `ANALYZE TABLE` (if `innodb_stats_auto_recalc` is enabled, statistics are updated automatically within a few seconds assuming that the 10% threshold for changed table rows is reached):

```
mysql> ANALYZE TABLE t1;
+---------+---------+----------+----------+
| Table   | Op      | Msg_type | Msg_text |
+---------+---------+----------+----------+
| test.t1 | analyze | status   | OK       |
+---------+---------+----------+----------+
```

Table statistics for table `t1` show the last time `InnoDB` updated the table statistics (`2014-03-14 14:36:34`), the number of rows in the table (`5`), the clustered index size (`1` page), and the combined size of the other indexes (`2` pages).

```
mysql> SELECT * FROM mysql.innodb_table_stats WHERE table_name like 't1'\G
*************************** 1. row ***************************
           database_name: test
              table_name: t1
             last_update: 2014-03-14 14:36:34
                 n_rows: 5
    clustered_index_size: 1
sum_of_other_index_sizes: 2
```

The `innodb_index_stats` table contains multiple rows for each index. Each row in the `innodb_index_stats` table provides data related to a particular index statistic which is named in the `stat_name` column and described in the `stat_description` column. For example:

```
mysql> SELECT index_name, stat_name, stat_value, stat_description
       FROM mysql.innodb_index_stats WHERE table_name like 't1';
+------------+--------------+------------+---------------------------------+
| index_name | stat_name    | stat_value | stat_description                |
+------------+--------------+------------+---------------------------------+
| PRIMARY    | n_diff_pfx01 |          1 | a                               |
| PRIMARY    | n_diff_pfx02 |          5 | a,b                             |
| PRIMARY    | n_leaf_pages |          1 | Number of leaf pages in the index |
| PRIMARY    | size         |          1 | Number of pages in the index    |
| i1         | n_diff_pfx01 |          1 | c                               |
| i1         | n_diff_pfx02 |          2 | c,d                             |
| i1         | n_diff_pfx03 |          2 | c,d,a                           |
| i1         | n_diff_pfx04 |          5 | c,d,a,b                         |
| i1         | n_leaf_pages |          1 | Number of leaf pages in the index |
| i1         | size         |          1 | Number of pages in the index    |
| i2uniq     | n_diff_pfx01 |          2 | e                               |
| i2uniq     | n_diff_pfx02 |          5 | e,f                             |
| i2uniq     | n_leaf_pages |          1 | Number of leaf pages in the index |
| i2uniq     | size         |          1 | Number of pages in the index    |
+------------+--------------+------------+---------------------------------+
```

The `stat_name` column shows the following types of statistics:

- `size`: Where `stat_name=size`, the `stat_value` column displays the total number of pages in the index.
- `n_leaf_pages`: Where `stat_name=n_leaf_pages`, the `stat_value` column displays the number of leaf pages in the index.
- `n_diff_pfx`*`NN`*: Where `stat_name=n_diff_pfx01`, the `stat_value` column displays the number of distinct values in the first column of the index. Where `stat_name=n_diff_pfx02`, the `stat_value` column displays the number of distinct values in the first two columns of the index, and so on. Where `stat_name=n_diff_pfx`*`NN`*, the `stat_description` column shows a comma separated list of the index columns that are counted.

To further illustrate the `n_diff_pfx`*`NN`* statistic, which provides cardinality data, consider once again the `t1` table example that was introduced previously. As shown below, the `t1` table is created with a primary index (columns `a`, `b`), a secondary index (columns `c`, `d`), and a unique index (columns `e`, `f`):

```
CREATE TABLE t1 (
  a INT, b INT, c INT, d INT, e INT, f INT,
  PRIMARY KEY (a, b), KEY i1 (c, d), UNIQUE KEY i2uniq (e, f)
) ENGINE=INNODB;
```

After inserting five rows of sample data, table `t1` appears as follows:

```
mysql> SELECT * FROM t1;
+---+---+------+------+------+------+
| a | b | c    | d    | e    | f    |
+---+---+------+------+------+------+
```

```
| 1 | 1 |   10 |   11 |  100 |  101 |
| 1 | 2 |   10 |   11 |  200 |  102 |
| 1 | 3 |   10 |   11 |  100 |  103 |
| 1 | 4 |   10 |   12 |  200 |  104 |
| 1 | 5 |   10 |   12 |  100 |  105 |
+---+---+------+------+------+------+
```

When you query the `index_name`, `stat_name`, `stat_value`, and `stat_description`, where `stat_name LIKE 'n_diff%'`, the following result set is returned:

```
mysql> SELECT index_name, stat_name, stat_value, stat_description
       FROM mysql.innodb_index_stats
       WHERE table_name like 't1' AND stat_name LIKE 'n_diff%';
+------------+-------------+------------+------------------+
| index_name | stat_name   | stat_value | stat_description |
+------------+-------------+------------+------------------+
| PRIMARY    | n_diff_pfx01 |          1 | a                |
| PRIMARY    | n_diff_pfx02 |          5 | a,b              |
| i1         | n_diff_pfx01 |          1 | c                |
| i1         | n_diff_pfx02 |          2 | c,d              |
| i1         | n_diff_pfx03 |          2 | c,d,a            |
| i1         | n_diff_pfx04 |          5 | c,d,a,b          |
| i2uniq     | n_diff_pfx01 |          2 | e                |
| i2uniq     | n_diff_pfx02 |          5 | e,f              |
+------------+-------------+------------+------------------+
```

For the `PRIMARY` index, there are two `n_diff%` rows. The number of rows is equal to the number of columns in the index.

Note

For nonunique indexes, `InnoDB` appends the columns of the primary key.

- Where `index_name=PRIMARY` and `stat_name=n_diff_pfx01`, the `stat_value` is `1`, which indicates that there is a single distinct value in the first column of the index (column `a`). The number of distinct values in column `a` is confirmed by viewing the data in column `a` in table `t1`, in which there is a single distinct value (`1`). The counted column (`a`) is shown in the `stat_description` column of the result set.
- Where `index_name=PRIMARY` and `stat_name=n_diff_pfx02`, the `stat_value` is `5`, which indicates that there are five distinct values in the two columns of the index (`a,b`). The number of distinct values in columns `a` and `b` is confirmed by viewing the data in columns `a` and `b` in table `t1`, in which there are five distinct values: (`1,1`), (`1,2`), (`1,3`), (`1,4`) and (`1,5`). The counted columns (`a,b`) are shown in the `stat_description` column of the result set.

For the secondary index (`i1`), there are four `n_diff%` rows. Only two columns are defined for the secondary index (`c,d`) but there are four `n_diff%` rows for the secondary index because `InnoDB` suffixes all nonunique indexes with the primary key. As a result, there are four `n_diff%` rows instead of two to account for the both the secondary index columns (`c,d`) and the primary key columns (`a,b`).

- Where `index_name=i1` and `stat_name=n_diff_pfx01`, the `stat_value` is `1`, which indicates that there is a single distinct value in the first column of the index (column `c`). The number of distinct values in column `c` is confirmed by viewing the data in column `c` in table `t1`, in which there is a single distinct value: (`10`). The counted column (`c`) is shown in the `stat_description` column of the result set.
- Where `index_name=i1` and `stat_name=n_diff_pfx02`, the `stat_value` is `2`, which indicates that there are two distinct values in the first two columns of the index (`c,d`). The number of distinct values in columns `c` an `d` is confirmed by viewing the data in columns `c` and `d` in table `t1`, in which there are two distinct values: (`10,11`) and (`10,12`). The counted columns (`c,d`) are shown in the `stat_description` column of the result set.
- Where `index_name=i1` and `stat_name=n_diff_pfx03`, the `stat_value` is `2`, which indicates that there are two distinct values in the first three columns of the index (`c,d,a`). The number of

distinct values in columns `c`, `d`, and `a` is confirmed by viewing the data in column `c`, `d`, and `a` in table `t1`, in which there are two distinct values: (`10,11,1`) and (`10,12,1`). The counted columns (`c,d,a`) are shown in the `stat_description` column of the result set.

- Where `index_name=i1` and `stat_name=n_diff_pfx04`, the `stat_value` is `5`, which indicates that there are five distinct values in the four columns of the index (`c,d,a,b`). The number of distinct values in columns `c`, `d`, `a` and `b` is confirmed by viewing the data in columns `c`, `d`, `a`, and `b` in table `t1`, in which there are five distinct values: (`10,11,1,1`), (`10,11,1,2`), (`10,11,1,3`), (`10,12,1,4`), and (`10,12,1,5`). The counted columns (`c,d,a,b`) are shown in the `stat_description` column of the result set.

For the unique index (`i2uniq`), there are two `n_diff%` rows.

- Where `index_name=i2uniq` and `stat_name=n_diff_pfx01`, the `stat_value` is `2`, which indicates that there are two distinct values in the first column of the index (column `e`). The number of distinct values in column `e` is confirmed by viewing the data in column `e` in table `t1`, in which there are two distinct values: (`100`) and (`200`). The counted column (`e`) is shown in the `stat_description` column of the result set.

- Where `index_name=i2uniq` and `stat_name=n_diff_pfx02`, the `stat_value` is `5`, which indicates that there are five distinct values in the two columns of the index (`e,f`). The number of distinct values in columns `e` and `f` is confirmed by viewing the data in columns `e` and `f` in table `t1`, in which there are five distinct values: (`100,101`), (`200,102`), (`100,103`), (`200,104`), and (`100,105`). The counted columns (`e,f`) are shown in the `stat_description` column of the result set.

Retrieving Index Size Using the innodb_index_stats Table

You can retrieve the index size for tables, partitions, or subpartitions can using the `innodb_index_stats` table. In the following example, index sizes are retrieved for table `t1`. For a definition of table `t1` and corresponding index statistics, see InnoDB Persistent Statistics Tables Example.

```
mysql> SELECT SUM(stat_value) pages, index_name,
       SUM(stat_value)*@@innodb_page_size size
       FROM mysql.innodb_index_stats WHERE table_name='t1'
       AND stat_name = 'size' GROUP BY index_name;
+-------+------------+-------+
| pages | index_name | size  |
+-------+------------+-------+
|     1 | PRIMARY    | 16384 |
|     1 | i1         | 16384 |
|     1 | i2uniq     | 16384 |
+-------+------------+-------+
```

For partitions or subpartitions, you can use the same query with a modified `WHERE` clause to retrieve index sizes. For example, the following query retrieves index sizes for partitions of table `t1`:

```
mysql> SELECT SUM(stat_value) pages, index_name,
       SUM(stat_value)*@@innodb_page_size size
       FROM mysql.innodb_index_stats WHERE table_name like 't1#P%'
       AND stat_name = 'size' GROUP BY index_name;
```

14.8.11.2 Configuring Non-Persistent Optimizer Statistics Parameters

This section describes how to configure non-persistent optimizer statistics. Optimizer statistics are not persisted to disk when `innodb_stats_persistent=OFF` or when individual tables are created or altered with `STATS_PERSISTENT=0`. Instead, statistics are stored in memory, and are lost when the server is shut down. Statistics are also updated periodically by certain operations and under certain conditions.

As of MySQL 5.6.6, optimizer statistics are persisted to disk by default, enabled by the `innodb_stats_persistent` configuration option. For information about persistent optimizer statistics, see Section 14.8.11.1, “Configuring Persistent Optimizer Statistics Parameters”.

Optimizer Statistics Updates

Non-persistent optimizer statistics are updated when:

- Running `ANALYZE TABLE`.
- Running `SHOW TABLE STATUS`, `SHOW INDEX`, or querying the `INFORMATION_SCHEMA.TABLES` or `INFORMATION_SCHEMA.STATISTICS` tables with the `innodb_stats_on_metadata` option enabled.

  The default setting for `innodb_stats_on_metadata` was changed to `OFF` when persistent optimizer statistics were enabled by default in MySQL 5.6.6. Enabling `innodb_stats_on_metadata` may reduce access speed for schemas that have a large number of tables or indexes, and reduce stability of execution plans for queries that involve `InnoDB` tables. `innodb_stats_on_metadata` is configured globally using a `SET` statement.

  ```
  SET GLOBAL innodb_stats_on_metadata=ON
  ```

  Note

  `innodb_stats_on_metadata` only applies when optimizer statistics are configured to be non-persistent (when `innodb_stats_persistent` is disabled).

- Starting a `mysql` client with the `--auto-rehash` option enabled, which is the default. The `auto-rehash` option causes all `InnoDB` tables to be opened, and the open table operations cause statistics to be recalculated.

  To improve the start up time of the `mysql` client and to updating statistics, you can turn off `auto-rehash` using the `--disable-auto-rehash` option. The `auto-rehash` feature enables automatic name completion of database, table, and column names for interactive users.

- A table is first opened.
- `InnoDB` detects that 1 / 16 of table has been modified since the last time statistics were updated.

Configuring the Number of Sampled Pages

The MySQL query optimizer uses estimated statistics about key distributions to choose the indexes for an execution plan, based on the relative selectivity of the index. When `InnoDB` updates optimizer statistics, it samples random pages from each index on a table to estimate the cardinality of the index. (This technique is known as random dives.)

To give you control over the quality of the statistics estimate (and thus better information for the query optimizer), you can change the number of sampled pages using the parameter `innodb_stats_transient_sample_pages`. The default number of sampled pages is 8, which could be insufficient to produce an accurate estimate, leading to poor index choices by the query optimizer. This technique is especially important for large tables and tables used in joins. Unnecessary full table scans for such tables can be a substantial performance issue. See Section 8.2.1.19, “Avoiding Full Table Scans” for tips on tuning such queries. `innodb_stats_transient_sample_pages` is a global parameter that can be set at runtime.

The value of `innodb_stats_transient_sample_pages` affects the index sampling for all `InnoDB` tables and indexes when `innodb_stats_persistent=0`. Be aware of the following potentially significant impacts when you change the index sample size:

- Small values like 1 or 2 can result in inaccurate estimates of cardinality.
- Increasing the `innodb_stats_transient_sample_pages` value might require more disk reads. Values much larger than 8 (say, 100), can cause a significant slowdown in the time it takes to open a table or execute `SHOW TABLE STATUS`.

- The optimizer might choose very different query plans based on different estimates of index selectivity.

Whatever value of `innodb_stats_transient_sample_pages` works best for a system, set the option and leave it at that value. Choose a value that results in reasonably accurate estimates for all tables in your database without requiring excessive I/O. Because the statistics are automatically recalculated at various times other than on execution of `ANALYZE TABLE`, it does not make sense to increase the index sample size, run `ANALYZE TABLE`, then decrease sample size again.

Smaller tables generally require fewer index samples than larger tables. If your database has many large tables, consider using a higher value for `innodb_stats_transient_sample_pages` than if you have mostly smaller tables.

14.8.11.3 Estimating ANALYZE TABLE Complexity for InnoDB Tables

`ANALYZE TABLE` complexity for `InnoDB` tables is dependent on:

- The number of pages sampled, as defined by `innodb_stats_persistent_sample_pages`.
- The number of indexed columns in a table
- The number of partitions. If a table has no partitions, the number of partitions is considered to be 1.

Using these parameters, an approximate formula for estimating `ANALYZE TABLE` complexity would be:

The value of `innodb_stats_persistent_sample_pages` * number of indexed columns in a table * the number of partitions

Typically, the greater the resulting value, the greater the execution time for `ANALYZE TABLE`.

Note

`innodb_stats_persistent_sample_pages` defines the number of pages sampled at a global level. To set the number of pages sampled for an individual table, use the `STATS_SAMPLE_PAGES` option with `CREATE TABLE` or `ALTER TABLE`. For more information, see Section 14.8.11.1, “Configuring Persistent Optimizer Statistics Parameters”.

If `innodb_stats_persistent=OFF`, the number of pages sampled is defined by `innodb_stats_transient_sample_pages`. See Section 14.8.11.2, “Configuring Non-Persistent Optimizer Statistics Parameters” for additional information.

For a more in-depth approach to estimating `ANALYZE TABLE` complexity, consider the following example.

In Big O notation, `ANALYZE TABLE` complexity is described as:

```
O(n_sample
  * (n_cols_in_uniq_i
     + n_cols_in_non_uniq_i
     + n_cols_in_pk * (1 + n_non_uniq_i))
  * n_part)
```

where:

- `n_sample` is the number of pages sampled (defined by `innodb_stats_persistent_sample_pages`)
- `n_cols_in_uniq_i` is total number of all columns in all unique indexes (not counting the primary key columns)
- `n_cols_in_non_uniq_i` is the total number of all columns in all nonunique indexes

- `n_cols_in_pk` is the number of columns in the primary key (if a primary key is not defined, `InnoDB` creates a single column primary key internally)
- `n_non_uniq_i` is the number of nonunique indexes in the table
- `n_part` is the number of partitions. If no partitions are defined, the table is considered to be a single partition.

Now, consider the following table (table `t`), which has a primary key (2 columns), a unique index (2 columns), and two nonunique indexes (two columns each):

```
CREATE TABLE t (
  a INT,
  b INT,
  c INT,
  d INT,
  e INT,
  f INT,
  g INT,
  h INT,
  PRIMARY KEY (a, b),
  UNIQUE KEY i1uniq (c, d),
  KEY i2nonuniq (e, f),
  KEY i3nonuniq (g, h)
);
```

For the column and index data required by the algorithm described above, query the `mysql.innodb_index_stats` persistent index statistics table for table `t`. The `n_diff_pfx%` statistics show the columns that are counted for each index. For example, columns `a` and `b` are counted for the primary key index. For the nonunique indexes, the primary key columns (a,b) are counted in addition to the user defined columns.

Note

For additional information about the `InnoDB` persistent statistics tables, see Section 14.8.11.1, “Configuring Persistent Optimizer Statistics Parameters”

```
mysql> SELECT index_name, stat_name, stat_description
       FROM mysql.innodb_index_stats WHERE
       database_name='test' AND
       table_name='t' AND
       stat_name like 'n_diff_pfx%';
  +------------+--------------+------------------+
  | index_name | stat_name    | stat_description |
  +------------+--------------+------------------+
  | PRIMARY    | n_diff_pfx01 | a                |
  | PRIMARY    | n_diff_pfx02 | a,b              |
  | i1uniq     | n_diff_pfx01 | c                |
  | i1uniq     | n_diff_pfx02 | c,d              |
  | i2nonuniq  | n_diff_pfx01 | e                |
  | i2nonuniq  | n_diff_pfx02 | e,f              |
  | i2nonuniq  | n_diff_pfx03 | e,f,a            |
  | i2nonuniq  | n_diff_pfx04 | e,f,a,b          |
  | i3nonuniq  | n_diff_pfx01 | g                |
  | i3nonuniq  | n_diff_pfx02 | g,h              |
  | i3nonuniq  | n_diff_pfx03 | g,h,a            |
  | i3nonuniq  | n_diff_pfx04 | g,h,a,b          |
  +------------+--------------+------------------+
```

Based on the index statistics data shown above and the table definition, the following values can be determined:

- `n_cols_in_uniq_i`, the total number of all columns in all unique indexes not counting the primary key columns, is 2 (`c` and `d`)
- `n_cols_in_non_uniq_i`, the total number of all columns in all nonunique indexes, is 4 (`e`, `f`, `g` and `h`)

- `n_cols_in_pk`, the number of columns in the primary key, is 2 (`a` and `b`)
- `n_non_uniq_i`, the number of nonunique indexes in the table, is 2 (`i2nonuniq` and `i3nonuniq`))
- `n_part`, the number of partitions, is 1.

You can now calculate `innodb_stats_persistent_sample_pages` * (2 + 4 + 2 * (1 + 2)) * 1 to determine the number of leaf pages that are scanned. With `innodb_stats_persistent_sample_pages` set to the default value of `20`, and with a default page size of 16 `KiB` (`innodb_page_size`=16384), you can then estimate that 20 * 12 * 16384 `bytes` are read for table `t`, or about 4 `MiB`.

Note

All 4 `MiB` may not be read from disk, as some leaf pages may already be cached in the buffer pool.

14.8.12 Configuring the Merge Threshold for Index Pages

Staring in MySQL 5.7.6, you can configure the `MERGE_THRESHOLD` value for index pages. If the "page-full" percentage for an index page falls below the `MERGE_THRESHOLD` value when a row is deleted or when a row is shortened by an `UPDATE` operation, `InnoDB` attempts to merge the index page with a neighboring index page. The default `MERGE_THRESHOLD` value is 50, which is the previously hardcoded value. The minimum `MERGE_THRESHOLD` value is 1 and the maximum value is 50.

When the "page-full" percentage for an index page falls below 50%, which is the default `MERGE_THRESHOLD` setting, `InnoDB` attempts to merge the index page with a neighboring page. If both pages are close to 50% full, a page split can occur soon after the pages are merged. If this merge-split behavior occurs frequently, it can have an adverse affect on performance. To avoid frequent merge-splits, you can lower the `MERGE_THRESHOLD` value so that `InnoDB` attempts page merges at a lower "page-full" percentage. Merging pages at a lower page-full percentage leaves more room in index pages and helps reduce merge-split behavior.

The `MERGE_THRESHOLD` for index pages can be defined for a table or for individual indexes. A `MERGE_THRESHOLD` value defined for an individual index takes priority over a `MERGE_THRESHOLD` value defined for the table. If undefined, the `MERGE_THRESHOLD` value defaults to 50.

Setting MERGE_THRESHOLD for a Table

You can set the `MERGE_THRESHOLD` value for a table using the *`table_option`* `COMMENT` clause of the `CREATE TABLE` statement. For example:

```
CREATE TABLE t1 (
   id INT,
  KEY id_index (id)
) COMMENT='MERGE_THRESHOLD=45';
```

You can also set the `MERGE_THRESHOLD` value for an existing table using the *`table_option`* `COMMENT` clause with `ALTER TABLE`:

```
CREATE TABLE t1 (
   id INT,
  KEY id_index (id)
);

ALTER TABLE t1 COMMENT='MERGE_THRESHOLD=40';
```

Setting MERGE_THRESHOLD for Individual Indexes

To set the `MERGE_THRESHOLD` value for an individual index, you can use the *`index_option`* `COMMENT` clause with `CREATE TABLE`, `ALTER TABLE`, or `CREATE INDEX`, as shown in the following examples:

- Setting MERGE_THRESHOLD for an individual index using CREATE TABLE:

```
CREATE TABLE t1 (
   id INT,
  KEY id_index (id) COMMENT 'MERGE_THRESHOLD=40'
);
```

- Setting MERGE_THRESHOLD for an individual index using ALTER TABLE:

```
CREATE TABLE t1 (
   id INT,
  KEY id_index (id)
);

ALTER TABLE t1 DROP KEY id_index;
ALTER TABLE t1 ADD KEY id_index (id) COMMENT 'MERGE_THRESHOLD=40';
```

- Setting MERGE_THRESHOLD for an individual index using CREATE INDEX:

```
CREATE TABLE t1 (id INT);
CREATE INDEX id_index ON t1 (id) COMMENT 'MERGE_THRESHOLD=40';
```

Note

You cannot modify the MERGE_THRESHOLD value at the index level for GEN_CLUST_INDEX, which is the clustered index created by InnoDB when an InnoDB table is created without a primary key or unique key index. You can only modify the MERGE_THRESHOLD value for GEN_CLUST_INDEX by setting MERGE_THRESHOLD for the table.

Querying the MERGE_THRESHOLD Value for an Index

The current MERGE_THRESHOLD value for an index can be obtained by querying the INNODB_SYS_INDEXES table. For example:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_INDEXES WHERE NAME='id_index' \G
*************************** 1. row ***************************
       INDEX_ID: 91
           NAME: id_index
       TABLE_ID: 68
           TYPE: 0
       N_FIELDS: 1
        PAGE_NO: 4
          SPACE: 57
MERGE_THRESHOLD: 40
```

You can use SHOW CREATE TABLE to view the MERGE_THRESHOLD value for a table, if explicitly defined using the *table_option* COMMENT clause:

```
mysql> SHOW CREATE TABLE t2 \G
*************************** 1. row ***************************
       Table: t2
Create Table: CREATE TABLE `t2` (
  `id` int(11) DEFAULT NULL,
  KEY `id_index` (`id`) COMMENT 'MERGE_THRESHOLD=40'
) ENGINE=InnoDB DEFAULT CHARSET=latin1
```

Note

A MERGE_THRESHOLD value defined at the index level takes priority over a MERGE_THRESHOLD value defined for the table. If undefined, MERGE_THRESHOLD defaults to 50% (MERGE_THRESHOLD=50, which is the previously hardcoded value.

Likewise, you can use SHOW INDEX to view the MERGE_THRESHOLD value for an index, if explicitly defined using the *index_option* COMMENT clause:

```
mysql> SHOW INDEX FROM t2 \G
*************************** 1. row ***************************
        Table: t2
   Non_unique: 1
     Key_name: id_index
 Seq_in_index: 1
  Column_name: id
    Collation: A
  Cardinality: 0
     Sub_part: NULL
       Packed: NULL
         Null: YES
   Index_type: BTREE
      Comment:
Index_comment: MERGE_THRESHOLD=40
```

Measuring the Effect of MERGE_THRESHOLD Settings

The `INNODB_METRICS` table provides two counters that can be used to measure the effect of a `MERGE_THRESHOLD` setting on index page merges.

```
mysql> SELECT NAME, COMMENT FROM INFORMATION_SCHEMA.INNODB_METRICS
       WHERE NAME like '%index_page_merge%';
+-----------------------------+---------------------------------------+
| NAME                        | COMMENT                               |
+-----------------------------+---------------------------------------+
| index_page_merge_attempts   | Number of index page merge attempts   |
| index_page_merge_successful | Number of successful index page merges |
+-----------------------------+---------------------------------------+
```

When lowering the `MERGE_THRESHOLD` value, the objectives are:

- A smaller number of page merge attempts and successful page merges
- A similar number of page merge attempts and successful page merges

A `MERGE_THRESHOLD` setting that is too small could result in large data files due to an excessive amount of empty page space.

For information about using `INNODB_METRICS` counters, see Section 14.15.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

14.9 InnoDB Table Compression

By using the SQL syntax and MySQL configuration options for compression, you can create tables where the data is stored in compressed form. Compression can help to improve both raw performance and scalability. The compression means less data is transferred between disk and memory, and takes up less space on disk and in memory. The benefits are amplified for tables with secondary indexes, because index data is compressed also. Compression can be especially important for SSD storage devices, because they tend to have lower capacity than HDD devices.

14.9.1 Overview of Table Compression

Because processors and cache memories have increased in speed more than disk storage devices, many workloads are disk-bound. Data compression enables smaller database size, reduced I/O, and improved throughput, at the small cost of increased CPU utilization. Compression is especially valuable for read-intensive applications, on systems with enough RAM to keep frequently used data in memory.

An InnoDB table created with `ROW_FORMAT=COMPRESSED` can use a smaller page size on disk than the usual 16KB default. Smaller pages require less I/O to read from and write to disk, which is especially valuable for SSD devices.

The page size is specified through the `KEY_BLOCK_SIZE` parameter. The different page size means the table must be in its own `.ibd` file rather than in the system tablespace, which requires enabling

the innodb_file_per_table option. The level of compression is the same regardless of the KEY_BLOCK_SIZE value. As you specify smaller values for KEY_BLOCK_SIZE, you get the I/O benefits of increasingly smaller pages. But if you specify a value that is too small, there is additional overhead to reorganize the pages when data values cannot be compressed enough to fit multiple rows in each page. There is a hard limit on how small KEY_BLOCK_SIZE can be for a table, based on the lengths of the key columns for each of its indexes. Specify a value that is too small, and the CREATE TABLE or ALTER TABLE statement fails.

In the buffer pool, the compressed data is held in small pages, with a page size based on the KEY_BLOCK_SIZE value. For extracting or updating the column values, MySQL also creates a 16KB page in the buffer pool with the uncompressed data. Within the buffer pool, any updates to the uncompressed page are also re-written back to the equivalent compressed page. You might need to size your buffer pool to accommodate the additional data of both compressed and uncompressed pages, although the uncompressed pages are evicted from the buffer pool when space is needed, and then uncompressed again on the next access.

14.9.2 Enabling Compression for a Table

Before creating a compressed table, make sure the innodb_file_per_table configuration option is enabled, and innodb_file_format is set to Barracuda. You can set these parameters in the MySQL configuration file my.cnf or my.ini, or with the SET statement without shutting down the MySQL server.

To enable compression for a table, you use the clauses ROW_FORMAT=COMPRESSED, KEY_BLOCK_SIZE, or both in a CREATE TABLE or ALTER TABLE statement.

To create a compressed table, you might use statements like these:

```
SET GLOBAL innodb_file_per_table=1;
SET GLOBAL innodb_file_format=Barracuda;
CREATE TABLE t1
 (c1 INT PRIMARY KEY)
 ROW_FORMAT=COMPRESSED
 KEY_BLOCK_SIZE=8;
```

- If you specify ROW_FORMAT=COMPRESSED, you can omit KEY_BLOCK_SIZE; the default page size value is used, which is half the innodb_page_size value.
- If you specify KEY_BLOCK_SIZE, you can omit ROW_FORMAT=COMPRESSED; compression is enabled automatically.
- To determine the best value for KEY_BLOCK_SIZE, typically you create several copies of the same table with different values for this clause, then measure the size of the resulting .ibd files and see how well each performs with a realistic workload.
- The KEY_BLOCK_SIZE value is treated as a hint; a different size could be used by InnoDB if necessary. A value of 0 represents the default compressed page size, which is half of the innodb_page_size value. The KEY_BLOCK_SIZE can only be less than or equal to the innodb_page_size value. If you specify a value greater than the innodb_page_size value, the specified value is ignored, a warning is issued, and KEY_BLOCK_SIZE is set to half of the innodb_page_size value. If innodb_strict_mode=ON, specifying an invalid KEY_BLOCK_SIZE value returns an error.
- For additional performance-related configuration options, see Section 14.9.3, “Tuning Compression for InnoDB Tables”.

The default uncompressed size of InnoDB data pages is 16KB. Depending on the combination of option values, MySQL uses a page size of 1KB, 2KB, 4KB, 8KB, or 16KB for the .ibd file of the table. The actual compression algorithm is not affected by the KEY_BLOCK_SIZE value; the value determines how large each compressed chunk is, which in turn affects how many rows can be packed into each compressed page.

Setting KEY_BLOCK_SIZE equal to the InnoDB page size does not typically result in much compression. For example, setting KEY_BLOCK_SIZE=16 typically would not result in much compression, since the normal InnoDB page size is 16KB. This setting may still be useful for tables with many long BLOB, VARCHAR or TEXT columns, because such values often do compress well, and might therefore require fewer overflow pages as described in Section 14.9.5, "How Compression Works for InnoDB Tables".

All indexes of a table (including the clustered index) are compressed using the same page size, as specified in the CREATE TABLE or ALTER TABLE statement. Table attributes such as ROW_FORMAT and KEY_BLOCK_SIZE are not part of the CREATE INDEX syntax for InnoDB tables, and are ignored if they are specified (although you see them in the output of the SHOW CREATE TABLE statement).

Restrictions on Compressed Tables

Because MySQL versions prior to 5.1 cannot process compressed tables, using compression requires specifying the configuration parameter innodb_file_format=Barracuda, to avoid accidentally introducing compatibility issues.

Table compression is also not available for the InnoDB system tablespace. The system tablespace (space 0, the ibdata* files) can contain user data, but it also contains internal system information, and therefore is never compressed. Thus, compression applies only to tables (and indexes) stored in their own tablespaces, that is, created with the innodb_file_per_table option enabled.

Compression applies to an entire table and all its associated indexes, not to individual rows, despite the clause name ROW_FORMAT.

14.9.3 Tuning Compression for InnoDB Tables

Most often, the internal optimizations described in InnoDB Data Storage and Compression ensure that the system runs well with compressed data. However, because the efficiency of compression depends on the nature of your data, you can make decisions that affect the performance of compressed tables:

- Which tables to compress.
- What compressed page size to use.
- Whether to adjust the size of the buffer pool based on run-time performance characteristics, such as the amount of time the system spends compressing and uncompressing data. Whether the workload is more like a data warehouse (primarily queries) or an OLTP system (mix of queries and DML).
- If the system performs DML operations on compressed tables, and the way the data is distributed leads to expensive compression failures at runtime, you might adjust additional advanced configuration options.

Use the guidelines in this section to help make those architectural and configuration choices. When you are ready to conduct long-term testing and put compressed tables into production, see Section 14.9.4, "Monitoring InnoDB Table Compression at Runtime" for ways to verify the effectiveness of those choices under real-world conditions.

When to Use Compression

In general, compression works best on tables that include a reasonable number of character string columns and where the data is read far more often than it is written. Because there are no guaranteed ways to predict whether or not compression benefits a particular situation, always test with a specific workload and data set running on a representative configuration. Consider the following factors when deciding which tables to compress.

Data Characteristics and Compression

A key determinant of the efficiency of compression in reducing the size of data files is the nature of the data itself. Recall that compression works by identifying repeated strings of bytes in a block of

data. Completely randomized data is the worst case. Typical data often has repeated values, and so compresses effectively. Character strings often compress well, whether defined in CHAR, VARCHAR, TEXT or BLOB columns. On the other hand, tables containing mostly binary data (integers or floating point numbers) or data that is previously compressed (for example JPEG or PNG images) may not generally compress well, significantly or at all.

You choose whether to turn on compression for each InnoDB table. A table and all of its indexes use the same (compressed) page size. It might be that the primary key (clustered) index, which contains the data for all columns of a table, compresses more effectively than the secondary indexes. For those cases where there are long rows, the use of compression might result in long column values being stored “off-page”, as discussed in DYNAMIC Row Format. Those overflow pages may compress well. Given these considerations, for many applications, some tables compress more effectively than others, and you might find that your workload performs best only with a subset of tables compressed.

To determine whether or not to compress a particular table, conduct experiments. You can get a rough estimate of how efficiently your data can be compressed by using a utility that implements LZ77 compression (such as gzip or WinZip) on a copy of the .ibd file for an uncompressed table. You can expect less compression from a MySQL compressed table than from file-based compression tools, because MySQL compresses data in chunks based on the page size, 16KB by default. In addition to user data, the page format includes some internal system data that is not compressed. File-based compression utilities can examine much larger chunks of data, and so might find more repeated strings in a huge file than MySQL can find in an individual page.

Another way to test compression on a specific table is to copy some data from your uncompressed table to a similar, compressed table (having all the same indexes) and look at the size of the resulting .ibd file. For example:

```
USE test;
SET GLOBAL innodb_file_per_table=1;
SET GLOBAL innodb_file_format=Barracuda;
SET GLOBAL autocommit=0;

-- Create an uncompressed table with a million or two rows.
CREATE TABLE big_table AS SELECT * FROM information_schema.columns;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
INSERT INTO big_table SELECT * FROM big_table;
COMMIT;
ALTER TABLE big_table ADD id int unsigned NOT NULL PRIMARY KEY auto_increment;

SHOW CREATE TABLE big_table\G

select count(id) from big_table;

-- Check how much space is needed for the uncompressed table.
\! ls -l data/test/big_table.ibd

CREATE TABLE key_block_size_4 LIKE big_table;
ALTER TABLE key_block_size_4 key_block_size=4 row_format=compressed;

INSERT INTO key_block_size_4 SELECT * FROM big_table;
commit;

-- Check how much space is needed for a compressed table
-- with particular compression settings.
\! ls -l data/test/key_block_size_4.ibd
```

This experiment produced the following numbers, which of course could vary considerably depending on your table structure and data:

```
-rw-rw----  1 cirrus  staff  310378496 Jan  9 13:44 data/test/big_table.ibd
-rw-rw----  1 cirrus  staff  83886080 Jan  9 15:10 data/test/key_block_size_4.ibd
```

To see whether compression is efficient for your particular workload:

- For simple tests, use a MySQL instance with no other compressed tables and run queries against the `INFORMATION_SCHEMA.INNODB_CMP` table.
- For more elaborate tests involving workloads with multiple compressed tables, run queries against the `INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX` table. Because the statistics in the `INNODB_CMP_PER_INDEX` table are expensive to collect, you must enable the configuration option `innodb_cmp_per_index_enabled` before querying that table, and you might restrict such testing to a development server or a non-critical replica server.
- Run some typical SQL statements against the compressed table you are testing.
- Examine the ratio of successful compression operations to overall compression operations by querying the `INFORMATION_SCHEMA.INNODB_CMP` or `INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX` table, and comparing `COMPRESS_OPS` to `COMPRESS_OPS_OK`.
- If a high percentage of compression operations complete successfully, the table might be a good candidate for compression.
- If you get a high proportion of compression failures, you can adjust `innodb_compression_level`, `innodb_compression_failure_threshold_pct`, and `innodb_compression_pad_pct_max` options as described in Section 14.9.6, “Compression for OLTP Workloads”, and try further tests.

Database Compression versus Application Compression

Decide whether to compress data in your application or in the table; do not use both types of compression for the same data. When you compress the data in the application and store the results in a compressed table, extra space savings are extremely unlikely, and the double compression just wastes CPU cycles.

Compressing in the Database

When enabled, MySQL table compression is automatic and applies to all columns and index values. The columns can still be tested with operators such as `LIKE`, and sort operations can still use indexes even when the index values are compressed. Because indexes are often a significant fraction of the total size of a database, compression could result in significant savings in storage, I/O or processor time. The compression and decompression operations happen on the database server, which likely is a powerful system that is sized to handle the expected load.

Compressing in the Application

If you compress data such as text in your application, before it is inserted into the database, You might save overhead for data that does not compress well by compressing some columns and not others. This approach uses CPU cycles for compression and uncompression on the client machine rather than the database server, which might be appropriate for a distributed application with many clients, or where the client machine has spare CPU cycles.

Hybrid Approach

Of course, it is possible to combine these approaches. For some applications, it may be appropriate to use some compressed tables and some uncompressed tables. It may be best to externally compress some data (and store it in uncompressed tables) and allow MySQL to compress (some of) the other tables in the application. As always, up-front design and real-life testing are valuable in reaching the right decision.

Workload Characteristics and Compression

In addition to choosing which tables to compress (and the page size), the workload is another key determinant of performance. If the application is dominated by reads, rather than updates, fewer pages need to be reorganized and recompressed after the index page runs out of room for the per-page "modification log" that MySQL maintains for compressed data. If the updates predominantly change non-indexed columns or those containing `BLOB`s or large strings that happen to be stored "off-page", the overhead of compression may be acceptable. If the only changes to a table are `INSERT`s that use a monotonically increasing primary key, and there are few secondary indexes, there is little need to reorganize and recompress index pages. Since MySQL can "delete-mark" and delete rows on compressed pages "in place" by modifying uncompressed data, `DELETE` operations on a table are relatively efficient.

For some environments, the time it takes to load data can be as important as run-time retrieval. Especially in data warehouse environments, many tables may be read-only or read-mostly. In those cases, it might or might not be acceptable to pay the price of compression in terms of increased load time, unless the resulting savings in fewer disk reads or in storage cost is significant.

Fundamentally, compression works best when the CPU time is available for compressing and uncompressing data. Thus, if your workload is I/O bound, rather than CPU-bound, you might find that compression can improve overall performance. When you test your application performance with different compression configurations, test on a platform similar to the planned configuration of the production system.

Configuration Characteristics and Compression

Reading and writing database pages from and to disk is the slowest aspect of system performance. Compression attempts to reduce I/O by using CPU time to compress and uncompress data, and is most effective when I/O is a relatively scarce resource compared to processor cycles.

This is often especially the case when running in a multi-user environment with fast, multi-core CPUs. When a page of a compressed table is in memory, MySQL often uses additional memory, typically 16KB, in the buffer pool for an uncompressed copy of the page. The adaptive LRU algorithm attempts to balance the use of memory between compressed and uncompressed pages to take into account whether the workload is running in an I/O-bound or CPU-bound manner. Still, a configuration with more memory dedicated to the buffer pool tends to run better when using compressed tables than a configuration where memory is highly constrained.

Choosing the Compressed Page Size

The optimal setting of the compressed page size depends on the type and distribution of data that the table and its indexes contain. The compressed page size should always be bigger than the maximum record size, or operations may fail as noted in Compression of B-Tree Pages.

Setting the compressed page size too large wastes some space, but the pages do not have to be compressed as often. If the compressed page size is set too small, inserts or updates may require time-consuming recompression, and the B-tree nodes may have to be split more frequently, leading to bigger data files and less efficient indexing.

Typically, you set the compressed page size to 8K or 4K bytes. Given that the maximum row size for an InnoDB table is around 8K, `KEY_BLOCK_SIZE=8` is usually a safe choice.

14.9.4 Monitoring InnoDB Table Compression at Runtime

Overall application performance, CPU and I/O utilization and the size of disk files are good indicators of how effective compression is for your application. This section builds on the performance tuning advice from Section 14.9.3, "Tuning Compression for InnoDB Tables", and shows how to find problems that might not turn up during initial testing.

To dig deeper into performance considerations for compressed tables, you can monitor compression performance at runtime using the Information Schema tables described in Example 14.1, "Using the

Compression Information Schema Tables". These tables reflect the internal use of memory and the rates of compression used overall.

The `INNODB_CMP` table reports information about compression activity for each compressed page size (`KEY_BLOCK_SIZE`) in use. The information in these tables is system-wide: it summarizes the compression statistics across all compressed tables in your database. You can use this data to help decide whether or not to compress a table by examining these tables when no other compressed tables are being accessed. It involves relatively low overhead on the server, so you might query it periodically on a production server to check the overall efficiency of the compression feature.

The `INNODB_CMP_PER_INDEX` table reports information about compression activity for individual tables and indexes. This information is more targeted and more useful for evaluating compression efficiency and diagnosing performance issues one table or index at a time. (Because that each `InnoDB` table is represented as a clustered index, MySQL does not make a big distinction between tables and indexes in this context.) The `INNODB_CMP_PER_INDEX` table does involve substantial overhead, so it is more suitable for development servers, where you can compare the effects of different workloads, data, and compression settings in isolation. To guard against imposing this monitoring overhead by accident, you must enable the `innodb_cmp_per_index_enabled` configuration option before you can query the `INNODB_CMP_PER_INDEX` table.

The key statistics to consider are the number of, and amount of time spent performing, compression and uncompression operations. Since MySQL splits B-tree nodes when they are too full to contain the compressed data following a modification, compare the number of "successful" compression operations with the number of such operations overall. Based on the information in the `INNODB_CMP` and `INNODB_CMP_PER_INDEX` tables and overall application performance and hardware resource utilization, you might make changes in your hardware configuration, adjust the size of the buffer pool, choose a different page size, or select a different set of tables to compress.

If the amount of CPU time required for compressing and uncompressing is high, changing to faster or multi-core CPUs can help improve performance with the same data, application workload and set of compressed tables. Increasing the size of the buffer pool might also help performance, so that more uncompressed pages can stay in memory, reducing the need to uncompress pages that exist in memory only in compressed form.

A large number of compression operations overall (compared to the number of `INSERT`, `UPDATE` and `DELETE` operations in your application and the size of the database) could indicate that some of your compressed tables are being updated too heavily for effective compression. If so, choose a larger page size, or be more selective about which tables you compress.

If the number of "successful" compression operations (`COMPRESS_OPS_OK`) is a high percentage of the total number of compression operations (`COMPRESS_OPS`), then the system is likely performing well. If the ratio is low, then MySQL is reorganizing, recompressing, and splitting B-tree nodes more often than is desirable. In this case, avoid compressing some tables, or increase `KEY_BLOCK_SIZE` for some of the compressed tables. You might turn off compression for tables that cause the number of "compression failures" in your application to be more than 1% or 2% of the total. (Such a failure ratio might be acceptable during a temporary operation such as a data load).

14.9.5 How Compression Works for InnoDB Tables

This section describes some internal implementation details about compression for InnoDB tables. The information presented here may be helpful in tuning for performance, but is not necessary to know for basic use of compression.

Compression Algorithms

Some operating systems implement compression at the file system level. Files are typically divided into fixed-size blocks that are compressed into variable-size blocks, which easily leads into fragmentation. Every time something inside a block is modified, the whole block is recompressed before it is written to disk. These properties make this compression technique unsuitable for use in an update-intensive database system.

MySQL implements compression with the help of the well-known zlib library, which implements the LZ77 compression algorithm. This compression algorithm is mature, robust, and efficient in both CPU utilization and in reduction of data size. The algorithm is “lossless”, so that the original uncompressed data can always be reconstructed from the compressed form. LZ77 compression works by finding sequences of data that are repeated within the data to be compressed. The patterns of values in your data determine how well it compresses, but typical user data often compresses by 50% or more.

Note

Prior to MySQL 5.6.42, `InnoDB` supports the `zlib` library up to version 1.2.3. In MySQL 5.6.42 and later, `InnoDB` supports the `zlib` library up to version 1.2.11.

Unlike compression performed by an application, or compression features of some other database management systems, InnoDB compression applies both to user data and to indexes. In many cases, indexes can constitute 40-50% or more of the total database size, so this difference is significant. When compression is working well for a data set, the size of the InnoDB data files (the `.ibd` files) is 25% to 50% of the uncompressed size or possibly smaller. Depending on the workload, this smaller database can in turn lead to a reduction in I/O, and an increase in throughput, at a modest cost in terms of increased CPU utilization. You can adjust the balance between compression level and CPU overhead by modifying the `innodb_compression_level` configuration option.

InnoDB Data Storage and Compression

All user data in InnoDB tables is stored in pages comprising a B-tree index (the clustered index). In some other database systems, this type of index is called an “index-organized table”. Each row in the index node contains the values of the (user-specified or system-generated) primary key and all the other columns of the table.

Secondary indexes in InnoDB tables are also B-trees, containing pairs of values: the index key and a pointer to a row in the clustered index. The pointer is in fact the value of the primary key of the table, which is used to access the clustered index if columns other than the index key and primary key are required. Secondary index records must always fit on a single B-tree page.

The compression of B-tree nodes (of both clustered and secondary indexes) is handled differently from compression of overflow pages used to store long `VARCHAR`, `BLOB`, or `TEXT` columns, as explained in the following sections.

Compression of B-Tree Pages

Because they are frequently updated, B-tree pages require special treatment. It is important to minimize the number of times B-tree nodes are split, as well as to minimize the need to uncompress and recompress their content.

One technique MySQL uses is to maintain some system information in the B-tree node in uncompressed form, thus facilitating certain in-place updates. For example, this allows rows to be delete-marked and deleted without any compression operation.

In addition, MySQL attempts to avoid unnecessary uncompression and recompression of index pages when they are changed. Within each B-tree page, the system keeps an uncompressed “modification log” to record changes made to the page. Updates and inserts of small records may be written to this modification log without requiring the entire page to be completely reconstructed.

When the space for the modification log runs out, InnoDB uncompresses the page, applies the changes and recompresses the page. If recompression fails (a situation known as a compression failure), the B-tree nodes are split and the process is repeated until the update or insert succeeds.

To avoid frequent compression failures in write-intensive workloads, such as for OLTP applications, MySQL sometimes reserves some empty space (padding) in the page, so that the modification log fills up sooner and the page is recompressed while there is still enough room to avoid splitting it. The amount of padding space left in each page varies as the system keeps track of the frequency

of page splits. On a busy server doing frequent writes to compressed tables, you can adjust the `innodb_compression_failure_threshold_pct`, and `innodb_compression_pad_pct_max` configuration options to fine-tune this mechanism.

Generally, MySQL requires that each B-tree page in an InnoDB table can accommodate at least two records. For compressed tables, this requirement has been relaxed. Leaf pages of B-tree nodes (whether of the primary key or secondary indexes) only need to accommodate one record, but that record must fit, in uncompressed form, in the per-page modification log. If `innodb_strict_mode` is `ON`, MySQL checks the maximum row size during `CREATE TABLE` or `CREATE INDEX`. If the row does not fit, the following error message is issued: `ERROR HY000: Too big row`.

If you create a table when `innodb_strict_mode` is OFF, and a subsequent `INSERT` or `UPDATE` statement attempts to create an index entry that does not fit in the size of the compressed page, the operation fails with `ERROR 42000: Row size too large`. (This error message does not name the index for which the record is too large, or mention the length of the index record or the maximum record size on that particular index page.) To solve this problem, rebuild the table with `ALTER TABLE` and select a larger compressed page size (`KEY_BLOCK_SIZE`), shorten any column prefix indexes, or disable compression entirely with `ROW_FORMAT=DYNAMIC` or `ROW_FORMAT=COMPACT`.

Compressing BLOB, VARCHAR, and TEXT Columns

In an InnoDB table, `BLOB`, `VARCHAR`, and `TEXT` columns that are not part of the primary key may be stored on separately allocated overflow pages. We refer to these columns as off-page columns. Their values are stored on singly-linked lists of overflow pages.

For tables created in `ROW_FORMAT=DYNAMIC` or `ROW_FORMAT=COMPRESSED`, the values of `BLOB`, `TEXT`, or `VARCHAR` columns may be stored fully off-page, depending on their length and the length of the entire row. For columns that are stored off-page, the clustered index record only contains 20-byte pointers to the overflow pages, one per column. Whether any columns are stored off-page depends on the page size and the total size of the row. When the row is too long to fit entirely within the page of the clustered index, MySQL chooses the longest columns for off-page storage until the row fits on the clustered index page. As noted above, if a row does not fit by itself on a compressed page, an error occurs.

Note

For tables created in `ROW_FORMAT=DYNAMIC` or `ROW_FORMAT=COMPRESSED`, `TEXT` and `BLOB` columns that are less than or equal to 40 bytes are always stored in-line.

Tables created in older versions of MySQL use the Antelope file format, which supports only `ROW_FORMAT=REDUNDANT` and `ROW_FORMAT=COMPACT`. In these formats, MySQL stores the first 768 bytes of `BLOB`, `VARCHAR`, and `TEXT` columns in the clustered index record along with the primary key. The 768-byte prefix is followed by a 20-byte pointer to the overflow pages that contain the rest of the column value.

When a table is in `COMPRESSED` format, all data written to overflow pages is compressed “as is”; that is, MySQL applies the zlib compression algorithm to the entire data item. Other than the data, compressed overflow pages contain an uncompressed header and trailer comprising a page checksum and a link to the next overflow page, among other things. Therefore, very significant storage savings can be obtained for longer `BLOB`, `TEXT`, or `VARCHAR` columns if the data is highly compressible, as is often the case with text data. Image data, such as `JPEG`, is typically already compressed and so does not benefit much from being stored in a compressed table; the double compression can waste CPU cycles for little or no space savings.

The overflow pages are of the same size as other pages. A row containing ten columns stored off-page occupies ten overflow pages, even if the total length of the columns is only 8K bytes. In an uncompressed table, ten uncompressed overflow pages occupy 160K bytes. In a compressed table with an 8K page size, they occupy only 80K bytes. Thus, it is often more efficient to use compressed table format for tables with long column values.

Using a 16K compressed page size can reduce storage and I/O costs for `BLOB`, `VARCHAR`, or `TEXT` columns, because such data often compress well, and might therefore require fewer overflow pages, even though the B-tree nodes themselves take as many pages as in the uncompressed form.

Compression and the InnoDB Buffer Pool

In a compressed InnoDB table, every compressed page (whether 1K, 2K, 4K or 8K) corresponds to an uncompressed page of 16K bytes (or a smaller size if `innodb_page_size` is set). To access the data in a page, MySQL reads the compressed page from disk if it is not already in the buffer pool, then uncompresses the page to its original form. This section describes how InnoDB manages the buffer pool with respect to pages of compressed tables.

To minimize I/O and to reduce the need to uncompress a page, at times the buffer pool contains both the compressed and uncompressed form of a database page. To make room for other required database pages, MySQL can evict from the buffer pool an uncompressed page, while leaving the compressed page in memory. Or, if a page has not been accessed in a while, the compressed form of the page might be written to disk, to free space for other data. Thus, at any given time, the buffer pool might contain both the compressed and uncompressed forms of the page, or only the compressed form of the page, or neither.

MySQL keeps track of which pages to keep in memory and which to evict using a least-recently-used (LRU) list, so that hot (frequently accessed) data tends to stay in memory. When compressed tables are accessed, MySQL uses an adaptive LRU algorithm to achieve an appropriate balance of compressed and uncompressed pages in memory. This adaptive algorithm is sensitive to whether the system is running in an I/O-bound or CPU-bound manner. The goal is to avoid spending too much processing time uncompressing pages when the CPU is busy, and to avoid doing excess I/O when the CPU has spare cycles that can be used for uncompressing compressed pages (that may already be in memory). When the system is I/O-bound, the algorithm prefers to evict the uncompressed copy of a page rather than both copies, to make more room for other disk pages to become memory resident. When the system is CPU-bound, MySQL prefers to evict both the compressed and uncompressed page, so that more memory can be used for “hot” pages and reducing the need to uncompress data in memory only in compressed form.

Compression and the InnoDB Redo Log Files

Before a compressed page is written to a data file, MySQL writes a copy of the page to the redo log (if it has been recompressed since the last time it was written to the database). This is done to ensure that redo logs are usable for crash recovery, even in the unlikely case that the `zlib` library is upgraded and that change introduces a compatibility problem with the compressed data. Therefore, some increase in the size of log files, or a need for more frequent checkpoints, can be expected when using compression. The amount of increase in the log file size or checkpoint frequency depends on the number of times compressed pages are modified in a way that requires reorganization and recompression.

Compressed tables use a different file format for the redo log and the per-table tablespaces than in MySQL 5.1 and earlier. The MySQL Enterprise Backup product supports this latest Barracuda file format for compressed InnoDB tables.

14.9.6 Compression for OLTP Workloads

Traditionally, the `InnoDB` compression feature was recommended primarily for read-only or read-mostly workloads, such as in a data warehouse configuration. The rise of SSD storage devices, which are fast but relatively small and expensive, makes compression attractive also for `OLTP` workloads: high-traffic, interactive websites can reduce their storage requirements and their I/O operations per second (IOPS) by using compressed tables with applications that do frequent `INSERT`, `UPDATE`, and `DELETE` operations.

Configuration options introduced in MySQL 5.6 let you adjust the way compression works for a particular MySQL instance, with an emphasis on performance and scalability for write-intensive operations:

- `innodb_compression_level` lets you turn the degree of compression up or down. A higher value lets you fit more data onto a storage device, at the expense of more CPU overhead during compression. A lower value lets you reduce CPU overhead when storage space is not critical, or you expect the data is not especially compressible.
- `innodb_compression_failure_threshold_pct` specifies a cutoff point for compression failures during updates to a compressed table. When this threshold is passed, MySQL begins to leave additional free space within each new compressed page, dynamically adjusting the amount of free space up to the percentage of page size specified by `innodb_compression_pad_pct_max`
- `innodb_compression_pad_pct_max` lets you adjust the maximum amount of space reserved within each page to record changes to compressed rows, without needing to compress the entire page again. The higher the value, the more changes can be recorded without recompressing the page. MySQL uses a variable amount of free space for the pages within each compressed table, only when a designated percentage of compression operations "fail" at runtime, requiring an expensive operation to split the compressed page.
- `innodb_log_compressed_pages` lets you disable writing of images of re-compressed pages to the redo log. Re-compression may occur when changes are made to compressed data. This option is enabled by default to prevent corruption that could occur if a different version of the `zlib` compression algorithm is used during recovery. If you are certain that the `zlib` version is not subject to change, disable `innodb_log_compressed_pages` to reduce redo log generation for workloads that modify compressed data.

Because working with compressed data sometimes involves keeping both compressed and uncompressed versions of a page in memory at the same time, when using compression with an OLTP-style workload, be prepared to increase the value of the `innodb_buffer_pool_size` configuration option.

14.9.7 SQL Compression Syntax Warnings and Errors

Specifying `ROW_FORMAT=COMPRESSED` or `KEY_BLOCK_SIZE` in `CREATE TABLE` or `ALTER TABLE` statements produces the following warnings if the Barracuda file format is not enabled. You can view them with the `SHOW WARNINGS` statement.

Level	Code	Message
Warning	1478	`InnoDB: KEY_BLOCK_SIZE requires innodb_file_per_table.`
Warning	1478	`InnoDB: KEY_BLOCK_SIZE requires innodb_file_format=1`
Warning	1478	`InnoDB: ignoring KEY_BLOCK_SIZE=4.`
Warning	1478	`InnoDB: ROW_FORMAT=COMPRESSED requires innodb_file_per_table.`
Warning	1478	`InnoDB: assuming ROW_FORMAT=COMPACT.`

Notes:

- By default, these messages are only warnings, not errors, and the table is created without compression, as if the options were not specified.
- When `innodb_strict_mode` is enabled, MySQL generates an error, not a warning, for these cases. The table is not created if the current configuration does not permit using compressed tables.

The "non-strict" behavior lets you import a `mysqldump` file into a database that does not support compressed tables, even if the source database contained compressed tables. In that case, MySQL creates the table in `ROW_FORMAT=COMPACT` instead of preventing the operation.

To import the dump file into a new database, and have the tables re-created as they exist in the original database, ensure the server has the proper settings for the configuration parameters `innodb_file_format` and `innodb_file_per_table`.

The attribute `KEY_BLOCK_SIZE` is permitted only when `ROW_FORMAT` is specified as `COMPRESSED` or is omitted. Specifying a `KEY_BLOCK_SIZE` with any other `ROW_FORMAT` generates a warning that you can view with `SHOW WARNINGS`. However, the table is non-compressed; the specified `KEY_BLOCK_SIZE` is ignored).

Level	Code	Message
Warning	1478	`InnoDB: ignoring KEY_BLOCK_SIZE=n unless ROW_FORMAT=COMPRESSED.`

If you are running with `innodb_strict_mode` enabled, the combination of a `KEY_BLOCK_SIZE` with any `ROW_FORMAT` other than `COMPRESSED` generates an error, not a warning, and the table is not created.

Table 14.5, "ROW_FORMAT and KEY_BLOCK_SIZE Options" provides an overview the `ROW_FORMAT` and `KEY_BLOCK_SIZE` options that are used with `CREATE TABLE` or `ALTER TABLE`.

Table 14.5 ROW_FORMAT and KEY_BLOCK_SIZE Options

Option	Usage Notes	Description
`ROW_FORMAT=REDUNDANT`	Storage format used prior to MySQL 5.0.3	Less efficient than `ROW_FORMAT=COMPACT`; for backward compatibility
`ROW_FORMAT=COMPACT`	Default storage format since MySQL 5.0.3	Stores a prefix of 768 bytes of long column values in the clustered index page, with the remaining bytes stored in an overflow page
`ROW_FORMAT=DYNAMIC`	Available only with `innodb_file_format=Barracuda`	Store values within the clustered index page if they fit; if not, stores only a 20-byte pointer to an overflow page (no prefix)
`ROW_FORMAT=COMPRESSED`	Available only with `innodb_file_format=Barracuda`	Compresses the table and indexes using zlib
`KEY_BLOCK_SIZE=n`	Available only with `innodb_file_format=Barracuda`	Specifies compressed page size of 1, 2, 4, 8 or 16 kilobytes; implies `ROW_FORMAT=COMPRESSED`

Table 14.6, "CREATE/ALTER TABLE Warnings and Errors when InnoDB Strict Mode is OFF" summarizes error conditions that occur with certain combinations of configuration parameters and options on the `CREATE TABLE` or `ALTER TABLE` statements, and how the options appear in the output of `SHOW TABLE STATUS`.

When `innodb_strict_mode` is `OFF`, MySQL creates or alters the table, but ignores certain settings as shown below. You can see the warning messages in the MySQL error log. When `innodb_strict_mode` is `ON`, these specified combinations of options generate errors, and the table is not created or altered. To see the full description of the error condition, issue the `SHOW ERRORS` statement: example:

```
mysql> CREATE TABLE x (id INT PRIMARY KEY, c INT)

-> ENGINE=INNODB KEY_BLOCK_SIZE=33333;

ERROR 1005 (HY000): Can't create table 'test.x' (errno: 1478)

mysql> SHOW ERRORS;
+-------+------+-----------------------------------------+
```

```
| Level | Code | Message                                       |
+-------+------+-----------------------------------------------+
| Error | 1478 | InnoDB: invalid KEY_BLOCK_SIZE=33333.         |
| Error | 1005 | Can't create table 'test.x' (errno: 1478) |
+-------+------+-----------------------------------------------+
```

Table 14.6 CREATE/ALTER TABLE Warnings and Errors when InnoDB Strict Mode is OFF

Syntax	Warning or Error Condition	Resulting `ROW_FORMAT`, as shown in `SHOW TABLE STATUS`
`ROW_FORMAT=REDUNDANT`	None	`REDUNDANT`
`ROW_FORMAT=COMPACT`	None	`COMPACT`
`ROW_FORMAT=COMPRESSED` or `ROW_FORMAT=DYNAMIC` or `KEY_BLOCK_SIZE` is specified	Ignored unless both `innodb_file_format=Barracuda` and `innodb_file_per_table` are enabled	`COMPACT`
Invalid `KEY_BLOCK_SIZE` is specified (not 1, 2, 4, 8 or 16)	`KEY_BLOCK_SIZE` is ignored	the requested row format, or `COMPACT` by default
`ROW_FORMAT=COMPRESSED` and valid `KEY_BLOCK_SIZE` are specified	None; `KEY_BLOCK_SIZE` specified is used	`COMPRESSED`
`KEY_BLOCK_SIZE` is specified with `REDUNDANT`, `COMPACT` or `DYNAMIC` row format	`KEY_BLOCK_SIZE` is ignored	`REDUNDANT`, `COMPACT` or `DYNAMIC`
`ROW_FORMAT` is not one of `REDUNDANT`, `COMPACT`, `DYNAMIC` or `COMPRESSED`	Ignored if recognized by the MySQL parser. Otherwise, an error is issued.	`COMPACT` or N/A

When `innodb_strict_mode` is `ON`, MySQL rejects invalid `ROW_FORMAT` or `KEY_BLOCK_SIZE` parameters. For compatibility with earlier versions of MySQL, strict mode is not enabled by default; instead, MySQL issues warnings (not errors) for ignored invalid parameters.

It is not possible to see the chosen `KEY_BLOCK_SIZE` using `SHOW TABLE STATUS`. The statement `SHOW CREATE TABLE` displays the `KEY_BLOCK_SIZE` (even if it was ignored when creating the table). The real compressed page size of the table cannot be displayed by MySQL.

14.10 InnoDB File-Format Management

As `InnoDB` evolves, data file formats that are not compatible with prior versions of `InnoDB` are sometimes required to support new features. To help manage compatibility in upgrade and downgrade situations, and systems that run different versions of MySQL, `InnoDB` uses named file formats. `InnoDB` currently supports two named file formats, Antelope and Barracuda.

- Antelope is the original `InnoDB` file format, which previously did not have a name. It supports COMPACT and REDUNDANT row formats for `InnoDB` tables and is the default file format in MySQL 5.6 to ensure maximum compatibility with earlier MySQL versions that do not support the Barracuda file format.

- Barracuda is the newest file format. It supports all `InnoDB` row formats including the newer COMPRESSED and DYNAMIC row formats. The features associated with COMPRESSED and DYNAMIC row formats include compressed tables, efficient storage of off-page columns, and index key prefixes up to 3072 bytes (`innodb_large_prefix`). See Section 14.11, “InnoDB Row Formats”.

This section discusses enabling `InnoDB` file formats, verifying compatibility of different file formats between MySQL releases, identifying the file format in use, and downgrading the file format.

14.10.1 Enabling File Formats

The `innodb_file_format` configuration option enables an `InnoDB` file format for file-per-table tablespaces.

`Antelope` is the default `innodb_file_format`.

To preclude the use of features supported by the Barracuda file that make your database inaccessible to the built-in `InnoDB` in MySQL 5.1 and prior releases, set `innodb_file_format` to `Antelope`. Alternatively, you can disable `innodb_file_per_table` to have new tables created in the system tablespace. The system tablespace is stored in the original Antelope file format.

You can set the value of `innodb_file_format` on the command line when you start `mysqld`, or in the option file (`my.cnf` on Unix, `my.ini` on Windows). You can also change it dynamically with a `SET GLOBAL` statement.

```
SET GLOBAL innodb_file_format=Barracuda;
```

Although Oracle recommends using the Barracuda format for new tables where practical, in MySQL 5.6 the default file format is Antelope, for maximum compatibility with replication configurations containing earlier MySQL releases.

14.10.2 Verifying File Format Compatibility

InnoDB incorporates several checks to guard against the possible crashes and data corruptions that might occur if you run an old release of the MySQL server on InnoDB data files that use a newer file format. These checks take place when the server is started, and when you first access a table. This section describes these checks, how you can control them, and error and warning conditions that might arise.

Backward Compatibility

You only need to consider backward file format compatibility when using a recent version of InnoDB (MySQL 5.5 and higher with InnoDB) alongside an older version (MySQL 5.1 or earlier, with the built-in InnoDB rather than the InnoDB Plugin). To minimize the chance of compatibility issues, you can standardize on the InnoDB Plugin for all your MySQL 5.1 and earlier database servers.

In general, a newer version of InnoDB may create a table or index that cannot safely be read or written with an older version of InnoDB without risk of crashes, hangs, wrong results or corruptions. InnoDB includes a mechanism to guard against these conditions, and to help preserve compatibility among database files and versions of InnoDB. This mechanism lets you take advantage of some new features of an InnoDB release (such as performance improvements and bug fixes), and still preserve the option of using your database with a prior version of InnoDB, by preventing accidental use of new features that create downward-incompatible disk files.

If a version of InnoDB supports a particular file format (whether or not that format is the default), you can query and update any table that requires that format or an earlier format. Only the creation of new tables using new features is limited based on the particular file format enabled. Conversely, if a tablespace contains a table or index that uses a file format that is not supported, it cannot be accessed at all, even for read access.

The only way to “downgrade” an InnoDB tablespace to the earlier Antelope file format is to copy the data to a new table, in a tablespace that uses the earlier format.

The easiest way to determine the file format of an existing InnoDB tablespace is to examine the properties of the table it contains, using the `SHOW TABLE STATUS` command or querying the table `INFORMATION_SCHEMA.TABLES`. If the `Row_format` of the table is reported as `'Compressed'` or `'Dynamic'`, the tablespace containing the table uses the Barracuda format. Otherwise, it uses the prior InnoDB file format, Antelope.

Internal Details

Every InnoDB file-per-table tablespace (represented by a `*.ibd` file) file is labeled with a file format identifier. The system tablespace (represented by the `ibdata` files) is tagged with the "highest" file format in use in a group of InnoDB database files, and this tag is checked when the files are opened.

Creating a compressed table, or a table with `ROW_FORMAT=DYNAMIC`, updates the file header of the corresponding file-per-table `.ibd` file and the table type in the InnoDB data dictionary with the identifier for the Barracuda file format. From that point forward, the table cannot be used with a version of InnoDB that does not support the Barracuda file format. To protect against anomalous behavior, InnoDB performs a compatibility check when the table is opened. (In many cases, the `ALTER TABLE` statement recreates a table and thus changes its properties. The special case of adding or dropping indexes without rebuilding the table is described in Section 14.13.1, "Online DDL Operations".)

Definition of ib-file set

To avoid confusion, for the purposes of this discussion we define the term "ib-file set" to mean the set of operating system files that InnoDB manages as a unit. The ib-file set includes the following files:

- The system tablespace (one or more `ibdata` files) that contain internal system information (including internal catalogs and undo information) and may include user data and indexes.
- Zero or more single-table tablespaces (also called "file per table" files, named `*.ibd` files).
- InnoDB log files; usually two, `ib_logfile0` and `ib_logfile1`. Used for crash recovery and in backups.

An "ib-file set" does not include the corresponding `.frm` files that contain metadata about InnoDB tables. The `.frm` files are created and managed by MySQL, and can sometimes get out of sync with the internal metadata in InnoDB.

Multiple tables, even from more than one database, can be stored in a single "ib-file set". (In MySQL, a "database" is a logical collection of tables, what other systems refer to as a "schema" or "catalog".)

14.10.2.1 Compatibility Check When InnoDB Is Started

To prevent possible crashes or data corruptions when InnoDB opens an ib-file set, it checks that it can fully support the file formats in use within the ib-file set. If the system is restarted following a crash, or a "fast shutdown" (i.e., `innodb_fast_shutdown` is greater than zero), there may be on-disk data structures (such as redo or undo entries, or doublewrite pages) that are in a "too-new" format for the current software. During the recovery process, serious damage can be done to your data files if these data structures are accessed. The startup check of the file format occurs before any recovery process begins, thereby preventing consistency issues with the new tables or startup problems for the MySQL server.

Beginning with version InnoDB 1.0.1, the system tablespace records an identifier or tag for the "highest" file format used by any table in any of the tablespaces that is part of the ib-file set. Checks against this file format tag are controlled by the configuration parameter `innodb_file_format_check`, which is `ON` by default.

If the file format tag in the system tablespace is newer or higher than the highest version supported by the particular currently executing software and if `innodb_file_format_check` is `ON`, the following error is issued when the server is started:

```
InnoDB: Error: the system tablespace is in a
file format that this version doesn't support
```

You can also set `innodb_file_format` to a file format name. Doing so prevents InnoDB from starting if the current software does not support the file format specified. It also sets the "high water mark" to the value you specify. The ability to set `innodb_file_format_check` is useful (with future releases) if you manually "downgrade" all of the tables in an ib-file set. You can then rely on the file format check at startup if you subsequently use an older version of InnoDB to access the ib-file set.

In some limited circumstances, you might want to start the server and use an ib-file set that is in a new file format that is not supported by the software you are using. If you set the configuration parameter `innodb_file_format_check` to `OFF`, InnoDB opens the database, but issues this warning message in the error log:

```
InnoDB: Warning: the system tablespace is in a
file format that this version doesn't support
```

Note

This is a dangerous setting, as it permits the recovery process to run, possibly corrupting your database if the previous shutdown was an unexpected exit or "fast shutdown". You should only set `innodb_file_format_check` to `OFF` if you are sure that the previous shutdown was done with `innodb_fast_shutdown=0`, so that essentially no recovery process occurs.

The parameter `innodb_file_format_check` affects only what happens when a database is opened, not subsequently. Conversely, the parameter `innodb_file_format` (which enables a specific format) only determines whether or not a new table can be created in the enabled format and has no effect on whether or not a database can be opened.

The file format tag is a "high water mark", and as such it is increased after the server is started, if a table in a "higher" format is created or an existing table is accessed for read or write (assuming its format is supported). If you access an existing table in a format higher than the format the running software supports, the system tablespace tag is not updated, but table-level compatibility checking applies (and an error is issued), as described in Section 14.10.2.2, "Compatibility Check When a Table Is Opened". Any time the high water mark is updated, the value of `innodb_file_format_check` is updated as well, so the command `SELECT @@innodb_file_format_check;` displays the name of the latest file format known to be used by tables in the currently open ib-file set and supported by the currently executing software.

14.10.2.2 Compatibility Check When a Table Is Opened

When a table is first accessed, InnoDB (including some releases prior to InnoDB 1.0) checks that the file format of the tablespace in which the table is stored is fully supported. This check prevents crashes or corruptions that would otherwise occur when tables using a "too new" data structure are encountered.

All tables using any file format supported by a release can be read or written (assuming the user has sufficient privileges). The setting of the system configuration parameter `innodb_file_format` can prevent creating a new table that uses a specific file format, even if the file format is supported by a given release. Such a setting might be used to preserve backward compatibility, but it does not prevent accessing any table that uses a supported format.

Versions of MySQL older than 5.0.21 cannot reliably use database files created by newer versions if a new file format was used when a table was created. To prevent various error conditions or corruptions, InnoDB checks file format compatibility when it opens a file (for example, upon first access to a table). If the currently running version of InnoDB does not support the file format identified by the table type in the InnoDB data dictionary, MySQL reports the following error:

```
ERROR 1146 (42S02): Table 'test.t1' doesn't exist
```

InnoDB also writes a message to the error log:

```
InnoDB: table test/t1: unknown table type 33
```

The table type should be equal to the tablespace flags, which contains the file format version as discussed in Section 14.10.3, "Identifying the File Format in Use".

Versions of InnoDB prior to MySQL 4.1 did not include table format identifiers in the database files, and versions prior to MySQL 5.0.21 did not include a table format compatibility check. Therefore, there is no

way to ensure proper operations if a table in a newer file format is used with versions of InnoDB prior to 5.0.21.

The file format management capability in InnoDB 1.0 and higher (tablespace tagging and run-time checks) allows InnoDB to verify as soon as possible that the running version of software can properly process the tables existing in the database.

If you permit InnoDB to open a database containing files in a format it does not support (by setting the parameter `innodb_file_format_check` to `OFF`), the table-level checking described in this section still applies.

Users are *strongly* urged not to use database files that contain Barracuda file format tables with releases of InnoDB older than the MySQL 5.1 with the InnoDB Plugin. It may be possible to rebuild such tables to use the Antelope format.

14.10.3 Identifying the File Format in Use

If you enable a different file format using the `innodb_file_format` configuration option, the change only applies to newly created tables. Also, when you create a new table, the tablespace containing the table is tagged with the “earliest” or “simplest” file format that is required to support the table's features. For example, if you enable the `Barracuda` file format, and create a new table that does not use the Dynamic or Compressed row format, the new tablespace that contains the table is tagged as using the `Antelope` file format .

It is easy to identify the file format used by a given table. The table uses the `Antelope` file format if the row format reported by `SHOW TABLE STATUS` is either `Compact` or `Redundant`. The table uses the `Barracuda` file format if the row format reported by `SHOW TABLE STATUS` is either `Compressed` or `Dynamic`.

```
mysql> SHOW TABLE STATUS\G
*************************** 1. row ***************************
           Name: t1
         Engine: InnoDB
        Version: 10
     Row_format: Compact
           Rows: 0
 Avg_row_length: 0
    Data_length: 16384
Max_data_length: 0
   Index_length: 16384
      Data_free: 0
 Auto_increment: 1
    Create_time: 2014-11-03 13:32:10
    Update_time: NULL
     Check_time: NULL
      Collation: latin1_swedish_ci
       Checksum: NULL
 Create_options:
        Comment:
```

You can also identify the file format used by a given table or tablespace using `InnoDB INFORMATION_SCHEMA` tables. For example:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_TABLES WHERE NAME='test/t1'\G
*************************** 1. row ***************************
     TABLE_ID: 44
         NAME: test/t1
         FLAG: 1
       N_COLS: 6
        SPACE: 30
  FILE_FORMAT: Antelope
   ROW_FORMAT: Compact
ZIP_PAGE_SIZE: 0

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_TABLESPACES WHERE NAME='test/t1'\G
*************************** 1. row ***************************
```

```
        SPACE: 30
         NAME: test/t1
         FLAG: 0
  FILE_FORMAT: Antelope
   ROW_FORMAT: Compact or Redundant
    PAGE_SIZE: 16384
ZIP_PAGE_SIZE: 0
```

14.10.4 Modifying the File Format

Each InnoDB tablespace file (with a name matching `*.ibd`) is tagged with the file format used to create its table and indexes. The way to modify the file format is to re-create the table and its indexes. The easiest way to recreate a table and its indexes is to use the following command on each table that you want to modify:

```
ALTER TABLE t ROW_FORMAT=format_name;
```

If you are modifying the file format to downgrade to an older MySQL version, there may be incompatibilities in table storage formats that require additional steps. For information about downgrading to a previous MySQL version, see Section 2.12, “Downgrading MySQL”.

14.11 InnoDB Row Formats

The row format of a table determines how its rows are physically stored, which in turn can affect the performance of queries and DML operations. As more rows fit into a single disk page, queries and index lookups can work faster, less cache memory is required in the buffer pool, and less I/O is required to write out updated values.

The data in each table is divided into pages. The pages that make up each table are arranged in a tree data structure called a B-tree index. Table data and secondary indexes both use this type of structure. The B-tree index that represents an entire table is known as the clustered index, which is organized according to the primary key columns. The nodes of a clustered index data structure contain the values of all columns in the row. The nodes of a secondary index structure contain the values of index columns and primary key columns.

Variable-length columns are an exception to the rule that column values are stored in B-tree index nodes. Variable-length columns that are too long to fit on a B-tree page are stored on separately allocated disk pages called overflow pages. Such columns are referred to as off-page columns. The values of off-page columns are stored in singly-linked lists of overflow pages, with each such column having its own list of one or more overflow pages. Depending on column length, all or a prefix of variable-length column values are stored in the B-tree to avoid wasting storage and having to read a separate page.

The `InnoDB` storage engine supports four row formats: `REDUNDANT`, `COMPACT`, `DYNAMIC`, and `COMPRESSED`.

Table 14.7 InnoDB Row Format Overview

Row Format	Compact Storage Characteristics	Enhanced Variable-Length Column Storage	Large Index Key Prefix Support	Compression Support	Supported Tablespace Types	Required File Format
`REDUNDANT`	No	No	No	No	system, file-per-table	Antelope or Barracuda
`COMPACT`	Yes	No	No	No	system, file-per-table	Antelope or Barracuda
`DYNAMIC`	Yes	Yes	Yes	No	file-per-table	Barracuda
`COMPRESSED`	Yes	Yes	Yes	Yes	file-per-table	Barracuda

The topics that follow describe row format storage characteristics and how to define and determine the row format of a table.

- REDUNDANT Row Format
- COMPACT Row Format
- DYNAMIC Row Format
- COMPRESSED Row Format
- Defining the Row Format of a Table
- Determining the Row Format of a Table

REDUNDANT Row Format

The `REDUNDANT` format provides compatibility with older versions of MySQL.

The `REDUNDANT` row format is supported by both `InnoDB` file formats (`Antelope` and `Barracuda`). For more information, see Section 14.10, “InnoDB File-Format Management”.

Tables that use the `REDUNDANT` row format store the first 768 bytes of variable-length column values (`VARCHAR`, `VARBINARY`, and `BLOB` and `TEXT` types) in the index record within the B-tree node, with the remainder stored on overflow pages. Fixed-length columns greater than or equal to 768 bytes are encoded as variable-length columns, which can be stored off-page. For example, a `CHAR(255)` column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with `utf8mb4`.

If the value of a column is 768 bytes or less, an overflow page is not used, and some savings in I/O may result, since the value is stored entirely in the B-tree node. This works well for relatively short `BLOB` column values, but may cause B-tree nodes to fill with data rather than key values, reducing their efficiency. Tables with many `BLOB` columns could cause B-tree nodes to become too full, and contain too few rows, making the entire index less efficient than if rows were shorter or column values were stored off-page.

REDUNDANT Row Format Storage Characteristics

The `REDUNDANT` row format has the following storage characteristics:

- Each index record contains a 6-byte header. The header is used to link together consecutive records, and for row-level locking.
- Records in the clustered index contain fields for all user-defined columns. In addition, there is a 6-byte transaction ID field and a 7-byte roll pointer field.
- If no primary key is defined for a table, each clustered index record also contains a 6-byte row ID field.
- Each secondary index record contains all the primary key columns defined for the clustered index key that are not in the secondary index.
- A record contains a pointer to each field of the record. If the total length of the fields in a record is less than 128 bytes, the pointer is one byte; otherwise, two bytes. The array of pointers is called the record directory. The area where the pointers point is the data part of the record.
- Internally, fixed-length character columns such as `CHAR(10)` are stored in fixed-length format. Trailing spaces are not truncated from `VARCHAR` columns.
- Fixed-length columns greater than or equal to 768 bytes are encoded as variable-length columns, which can be stored off-page. For example, a `CHAR(255)` column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with `utf8mb4`.

- An SQL NULL value reserves one or two bytes in the record directory. An SQL NULL value reserves zero bytes in the data part of the record if stored in a variable-length column. For a fixed-length column, the fixed length of the column is reserved in the data part of the record. Reserving fixed space for NULL values permits columns to be updated in place from NULL to non-NULL values without causing index page fragmentation.

COMPACT Row Format

The COMPACT row format reduces row storage space by about 20% compared to the REDUNDANT row format, at the cost of increasing CPU use for some operations. If your workload is a typical one that is limited by cache hit rates and disk speed, COMPACT format is likely to be faster. If the workload is limited by CPU speed, compact format might be slower.

The COMPACT row format is supported by both InnoDB file formats (Antelope and Barracuda). For more information, see Section 14.10, “InnoDB File-Format Management”.

Tables that use the COMPACT row format store the first 768 bytes of variable-length column values (VARCHAR, VARBINARY, and BLOB and TEXT types) in the index record within the B-tree node, with the remainder stored on overflow pages. Fixed-length columns greater than or equal to 768 bytes are encoded as variable-length columns, which can be stored off-page. For example, a CHAR(255) column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with utf8mb4.

If the value of a column is 768 bytes or less, an overflow page is not used, and some savings in I/O may result, since the value is stored entirely in the B-tree node. This works well for relatively short BLOB column values, but may cause B-tree nodes to fill with data rather than key values, reducing their efficiency. Tables with many BLOB columns could cause B-tree nodes to become too full, and contain too few rows, making the entire index less efficient than if rows were shorter or column values were stored off-page.

COMPACT Row Format Storage Characteristics

The COMPACT row format has the following storage characteristics:

- Each index record contains a 5-byte header that may be preceded by a variable-length header. The header is used to link together consecutive records, and for row-level locking.
- The variable-length part of the record header contains a bit vector for indicating NULL columns. If the number of columns in the index that can be NULL is *N*, the bit vector occupies CEILING(*N*/8) bytes. (For example, if there are anywhere from 9 to 16 columns that can be NULL, the bit vector uses two bytes.) Columns that are NULL do not occupy space other than the bit in this vector. The variable-length part of the header also contains the lengths of variable-length columns. Each length takes one or two bytes, depending on the maximum length of the column. If all columns in the index are NOT NULL and have a fixed length, the record header has no variable-length part.
- For each non-NULL variable-length field, the record header contains the length of the column in one or two bytes. Two bytes are only needed if part of the column is stored externally in overflow pages or the maximum length exceeds 255 bytes and the actual length exceeds 127 bytes. For an externally stored column, the 2-byte length indicates the length of the internally stored part plus the 20-byte pointer to the externally stored part. The internal part is 768 bytes, so the length is 768+20. The 20-byte pointer stores the true length of the column.
- The record header is followed by the data contents of non-NULL columns.
- Records in the clustered index contain fields for all user-defined columns. In addition, there is a 6-byte transaction ID field and a 7-byte roll pointer field.
- If no primary key is defined for a table, each clustered index record also contains a 6-byte row ID field.

- Each secondary index record contains all the primary key columns defined for the clustered index key that are not in the secondary index. If any of the primary key columns are variable length, the record header for each secondary index has a variable-length part to record their lengths, even if the secondary index is defined on fixed-length columns.

- Internally, for nonvariable-length character sets, fixed-length character columns such as `CHAR(10)` are stored in a fixed-length format.

  Trailing spaces are not truncated from `VARCHAR` columns.

- Internally, for variable-length character sets such as `utf8mb3` and `utf8mb4`, `InnoDB` attempts to store `CHAR(N)` in *`N`* bytes by trimming trailing spaces. If the byte length of a `CHAR(N)` column value exceeds *`N`* bytes, trailing spaces are trimmed to a minimum of the column value byte length. The maximum length of a `CHAR(N)` column is the maximum character byte length × *`N`*.

  A minimum of *`N`* bytes is reserved for `CHAR(N)`. Reserving the minimum space *`N`* in many cases enables column updates to be done in place without causing index page fragmentation. By comparison, `CHAR(N)` columns occupy the maximum character byte length × *`N`* when using the `REDUNDANT` row format.

  Fixed-length columns greater than or equal to 768 bytes are encoded as variable-length fields, which can be stored off-page. For example, a `CHAR(255)` column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with `utf8mb4`.

DYNAMIC Row Format

The `DYNAMIC` row format offers the same storage characteristics as the `COMPACT` row format but adds enhanced storage capabilities for long variable-length columns and supports large index key prefixes.

The `DYNAMIC` row format is supported by the `Barracuda` file format. For more information, see Section 14.10, “InnoDB File-Format Management”. To create tables that use the `DYNAMIC` row format, the `innodb_file_format` variable must be set to `Barracuda`, and the `innodb_file_per_table` variable must be enabled.

When a table is created with `ROW_FORMAT=DYNAMIC`, `InnoDB` can store long variable-length column values (for `VARCHAR`, `VARBINARY`, and `BLOB` and `TEXT` types) fully off-page, with the clustered index record containing only a 20-byte pointer to the overflow page. Fixed-length fields greater than or equal to 768 bytes are encoded as variable-length fields. For example, a `CHAR(255)` column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with `utf8mb4`.

Whether columns are stored off-page depends on the page size and the total size of the row. When a row is too long, the longest columns are chosen for off-page storage until the clustered index record fits on the B-tree page. `TEXT` and `BLOB` columns that are less than or equal to 40 bytes are stored in line.

The `DYNAMIC` row format maintains the efficiency of storing the entire row in the index node if it fits (as do the `COMPACT` and `REDUNDANT` formats), but the `DYNAMIC` row format avoids the problem of filling B-tree nodes with a large number of data bytes of long columns. The `DYNAMIC` row format is based on the idea that if a portion of a long data value is stored off-page, it is usually most efficient to store the entire value off-page. With `DYNAMIC` format, shorter columns are likely to remain in the B-tree node, minimizing the number of overflow pages required for a given row.

The `DYNAMIC` row format supports index key prefixes up to 3072 bytes. This feature is controlled by the `innodb_large_prefix` variable, which is disabled by default. See the `innodb_large_prefix` variable description for more information.

DYNAMIC Row Format Storage Characteristics

The `DYNAMIC` row format is a variation of the `COMPACT` row format. For storage characteristics, see COMPACT Row Format Storage Characteristics.

COMPRESSED Row Format

The COMPRESSED row format offers the same the storage characteristics and capabilities as the DYNAMIC row format but adds support for table and index data compression.

The COMPRESSED row format is supported by the Barracuda file format. For more information, see Section 14.10, “InnoDB File-Format Management”. To create tables that use the COMPRESSED row format, the innodb_file_format variable must be set to Barracuda, and the innodb_file_per_table variable must be enabled.

The COMPRESSED row format uses similar internal details for off-page storage as the DYNAMIC row format, with additional storage and performance considerations from the table and index data being compressed and using smaller page sizes. With the COMPRESSED row format, the KEY_BLOCK_SIZE option controls how much column data is stored in the clustered index, and how much is placed on overflow pages. For more information about the COMPRESSED row format, see InnoDB Table Compression.

The COMPRESSED row format supports index key prefixes up to 3072 bytes. This feature is controlled by the innodb_large_prefix variable, which is disabled by default. See the innodb_large_prefix variable description for more information.

Compressed Row Format Storage Characteristics

The COMPRESSED row format is a variation of the COMPACT row format. For storage characteristics, see COMPACT Row Format Storage Characteristics.

Defining the Row Format of a Table

The default row format for InnoDB tables is COMPACT.

The row format of a table can be defined explicitly using the ROW_FORMAT table option in a CREATE TABLE or ALTER TABLE statement. For example:

```
CREATE TABLE t1 (c1 INT) ROW_FORMAT=COMPACT;
```

ROW_FORMAT options include REDUNDANT, COMPACT, DYNAMIC, and COMPRESSED.

To create tables that use the DYNAMIC or COMPRESSED row format, the innodb_file_format variable must be set to Barracuda, and the innodb_file_per_table variable must be enabled. Otherwise, if innodb_strict_mode is not enabled, InnoDB tables are created with the default COMPACT row format.

Oracle recommends enabling innodb_strict_mode when using the ROW_FORMAT table option in CREATE TABLE and ALTER TABLE statements.

Determining the Row Format of a Table

To determine the row format of a table, use SHOW TABLE STATUS:

```
mysql> SHOW TABLE STATUS IN test1\G
*************************** 1. row ***************************
           Name: t1
         Engine: InnoDB
        Version: 10
     Row_format: Compact
           Rows: 0
 Avg_row_length: 0
    Data_length: 16384
Max_data_length: 0
   Index_length: 16384
      Data_free: 0
 Auto_increment: 1
    Create_time: 2014-10-31 16:02:01
```

```
    Update_time: NULL
     Check_time: NULL
      Collation: latin1_swedish_ci
       Checksum: NULL
 Create_options:
        Comment:
```

Alternatively, query the `INFORMATION_SCHEMA.INNODB_SYS_TABLES` table:

```
mysql> SELECT NAME, ROW_FORMAT FROM INFORMATION_SCHEMA.INNODB_SYS_TABLES WHERE NAME='test1/t1';
+----------+------------+
| NAME     | ROW_FORMAT |
+----------+------------+
| test1/t1 | Dynamic    |
+----------+------------+
```

14.12 InnoDB Disk I/O and File Space Management

As a DBA, you must manage disk I/O to keep the I/O subsystem from becoming saturated, and manage disk space to avoid filling up storage devices. The ACID design model requires a certain amount of I/O that might seem redundant, but helps to ensure data reliability. Within these constraints, `InnoDB` tries to optimize the database work and the organization of disk files to minimize the amount of disk I/O. Sometimes, I/O is postponed until the database is not busy, or until everything needs to be brought to a consistent state, such as during a database restart after a fast shutdown.

This section discusses the main considerations for I/O and disk space with the default kind of MySQL tables (also known as `InnoDB` tables):

- Controlling the amount of background I/O used to improve query performance.
- Enabling or disabling features that provide extra durability at the expense of additional I/O.
- Organizing tables into many small files, a few larger files, or a combination of both.
- Balancing the size of redo log files against the I/O activity that occurs when the log files become full.
- How to reorganize a table for optimal query performance.

14.12.1 InnoDB Disk I/O

`InnoDB` uses asynchronous disk I/O where possible, by creating a number of threads to handle I/O operations, while permitting other database operations to proceed while the I/O is still in progress. On Linux and Windows platforms, `InnoDB` uses the available OS and library functions to perform “native” asynchronous I/O. On other platforms, `InnoDB` still uses I/O threads, but the threads may actually wait for I/O requests to complete; this technique is known as “simulated” asynchronous I/O.

Read-Ahead

If `InnoDB` can determine there is a high probability that data might be needed soon, it performs read-ahead operations to bring that data into the buffer pool so that it is available in memory. Making a few large read requests for contiguous data can be more efficient than making several small, spread-out requests. There are two read-ahead heuristics in `InnoDB`:

- In sequential read-ahead, if `InnoDB` notices that the access pattern to a segment in the tablespace is sequential, it posts in advance a batch of reads of database pages to the I/O system.
- In random read-ahead, if `InnoDB` notices that some area in a tablespace seems to be in the process of being fully read into the buffer pool, it posts the remaining reads to the I/O system.

For information about configuring read-ahead heuristics, see Section 14.8.3.3, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”.

Doublewrite Buffer

InnoDB uses a novel file flush technique involving a structure called the doublewrite buffer, which is enabled by default (innodb_doublewrite=ON). It adds safety to recovery following an unexpected exit or power outage, and improves performance on most varieties of Unix by reducing the need for fsync() operations.

Before writing pages to a data file, InnoDB first writes them to a contiguous tablespace area called the doublewrite buffer. Only after the write and the flush to the doublewrite buffer has completed does InnoDB write the pages to their proper positions in the data file. If there is an operating system, storage subsystem, or unexpected mysqld process exit in the middle of a page write (causing a torn page condition), InnoDB can later find a good copy of the page from the doublewrite buffer during recovery.

14.12.2 File Space Management

The data files that you define in the configuration file using the innodb_data_file_path configuration option form the InnoDB system tablespace. The files are logically concatenated to form the system tablespace. There is no striping in use. You cannot define where within the system tablespace your tables are allocated. In a newly created system tablespace, InnoDB allocates space starting from the first data file.

To avoid the issues that come with storing all tables and indexes inside the system tablespace, you can enable the innodb_file_per_table configuration option (the default), which stores each newly created table in a separate tablespace file (with extension .ibd). For tables stored this way, there is less fragmentation within the disk file, and when the table is truncated, the space is returned to the operating system rather than still being reserved by InnoDB within the system tablespace.

Pages, Extents, Segments, and Tablespaces

Each tablespace consists of database pages. Every tablespace in a MySQL instance has the same page size. By default, all tablespaces have a page size of 16KB; you can reduce the page size to 8KB or 4KB by specifying the innodb_page_size option when you create the MySQL instance.

The pages are grouped into extents of size 1MB (64 consecutive 16KB pages, or 128 8KB pages, or 256 4KB pages). The “files” inside a tablespace are called segments in InnoDB. (These segments are different from the rollback segment, which actually contains many tablespace segments.)

When a segment grows inside the tablespace, InnoDB allocates the first 32 pages to it one at a time. After that, InnoDB starts to allocate whole extents to the segment. InnoDB can add up to 4 extents at a time to a large segment to ensure good sequentiality of data.

Two segments are allocated for each index in InnoDB. One is for nonleaf nodes of the B-tree, the other is for the leaf nodes. Keeping the leaf nodes contiguous on disk enables better sequential I/O operations, because these leaf nodes contain the actual table data.

Some pages in the tablespace contain bitmaps of other pages, and therefore a few extents in an InnoDB tablespace cannot be allocated to segments as a whole, but only as individual pages.

When you ask for available free space in the tablespace by issuing a SHOW TABLE STATUS statement, InnoDB reports the extents that are definitely free in the tablespace. InnoDB always reserves some extents for cleanup and other internal purposes; these reserved extents are not included in the free space.

When you delete data from a table, InnoDB contracts the corresponding B-tree indexes. Whether the freed space becomes available for other users depends on whether the pattern of deletes frees individual pages or extents to the tablespace. Dropping a table or deleting all rows from it is guaranteed to release the space to other users, but remember that deleted rows are physically removed only by the purge operation, which happens automatically some time after they are no longer needed for transaction rollbacks or consistent reads. (See Section 14.3, “InnoDB Multi-Versioning”.)

To see information about the tablespace, use the Tablespace Monitor. See Section 14.17, “InnoDB Monitors”.

How Pages Relate to Table Rows

The maximum row length is slightly less than half a database page. For example, the maximum row length is slightly less than 8KB for the default 16KB `InnoDB` page size, which is defined by the `innodb_page_size` configuration option.

If a row does not exceed the half page limit, all of it is stored locally within the page. If a row exceeds the half page limit, variable-length columns are chosen for external off-page storage until the row fits within half a page. External off-page storage for variable-length columns differs by row format:

- *COMPACT and REDUNDANT Row Formats*

  When a variable-length column is chosen for external off-page storage, `InnoDB` stores the first 768 bytes locally in the row, and the rest externally into overflow pages. Each such column has its own list of overflow pages. The 768-byte prefix is accompanied by a 20-byte value that stores the true length of the column and points into the overflow list where the rest of the value is stored. See Section 14.11, “InnoDB Row Formats”.

- *DYNAMIC and COMPRESSED Row Formats*

  When a variable-length column is chosen for external off-page storage, `InnoDB` stores a 20-byte pointer locally in the row, and the rest externally into overflow pages. See Section 14.11, “InnoDB Row Formats”.

`LONGBLOB` and `LONGTEXT` columns must be less than 4GB, and the total row length, including `BLOB` and `TEXT` columns, must be less than 4GB.

14.12.3 InnoDB Checkpoints

Making your log files very large may reduce disk I/O during checkpointing. It often makes sense to set the total size of the log files as large as the buffer pool or even larger. Although in the past large log files could make crash recovery take excessive time, starting with MySQL 5.5, performance enhancements to crash recovery make it possible to use large log files with fast startup after a crash. (Strictly speaking, this performance improvement is available for MySQL 5.1 with the InnoDB Plugin 1.0.7 and higher. It is with MySQL 5.5 that this improvement is available in the default InnoDB storage engine.)

How Checkpoint Processing Works

`InnoDB` implements a checkpoint mechanism known as fuzzy checkpointing. `InnoDB` flushes modified database pages from the buffer pool in small batches. There is no need to flush the buffer pool in one single batch, which would disrupt processing of user SQL statements during the checkpointing process.

During crash recovery, `InnoDB` looks for a checkpoint label written to the log files. It knows that all modifications to the database before the label are present in the disk image of the database. Then `InnoDB` scans the log files forward from the checkpoint, applying the logged modifications to the database.

14.12.4 Defragmenting a Table

Random insertions into or deletions from a secondary index can cause the index to become fragmented. Fragmentation means that the physical ordering of the index pages on the disk is not close to the index ordering of the records on the pages, or that there are many unused pages in the 64-page blocks that were allocated to the index.

One symptom of fragmentation is that a table takes more space than it “should” take. How much that is exactly, is difficult to determine. All `InnoDB` data and indexes are stored in B-trees, and their fill factor may vary from 50% to 100%. Another symptom of fragmentation is that a table scan such as this takes more time than it “should” take:

```
SELECT COUNT(*) FROM t WHERE non_indexed_column <> 12345;
```

The preceding query requires MySQL to perform a full table scan, the slowest type of query for a large table.

To speed up index scans, you can periodically perform a “null” `ALTER TABLE` operation, which causes MySQL to rebuild the table:

```
ALTER TABLE tbl_name ENGINE=INNODB
```

As of MySQL 5.6.3, you can also use `ALTER TABLE tbl_name FORCE` to perform a “null” alter operation that rebuilds the table. Previously the `FORCE` option was recognized but ignored.

As of MySQL 5.6.17, both `ALTER TABLE tbl_name ENGINE=INNODB` and `ALTER TABLE tbl_name FORCE` use online DDL. For more information, see Section 14.13, “InnoDB and Online DDL”.

Another way to perform a defragmentation operation is to use `mysqldump` to dump the table to a text file, drop the table, and reload it from the dump file.

If the insertions into an index are always ascending and records are deleted only from the end, the `InnoDB` filespace management algorithm guarantees that fragmentation in the index does not occur.

14.12.5 Reclaiming Disk Space with TRUNCATE TABLE

To reclaim operating system disk space when truncating an `InnoDB` table, the table must be stored in its own .ibd file. For a table to be stored in its own .ibd file, `innodb_file_per_table` must enabled when the table is created. Additionally, there cannot be a foreign key constraint between the table being truncated and other tables, otherwise the `TRUNCATE TABLE` operation fails. A foreign key constraint between two columns in the same table, however, is permitted.

When a table is truncated, it is dropped and re-created in a new `.ibd` file, and the freed space is returned to the operating system. This is in contrast to truncating `InnoDB` tables that are stored within the `InnoDB` system tablespace (tables created when `innodb_file_per_table=OFF`), where only `InnoDB` can use the freed space after the table is truncated.

The ability to truncate tables and return disk space to the operating system also means that physical backups can be smaller. Truncating tables that are stored in the system tablespace (tables created when `innodb_file_per_table=OFF`) leaves blocks of unused space in the system tablespace.

14.13 InnoDB and Online DDL

The online DDL feature provides support for in-place table alterations and concurrent DML. Benefits of this feature include:

- Improved responsiveness and availability in busy production environments, where making a table unavailable for minutes or hours is not practical.
- The ability to adjust the balance between performance and concurrency during DDL operations using the `LOCK` clause. See The LOCK clause.
- Less disk space usage and I/O overhead than the table-copy method.

The online DDL feature builds on the fast index creation feature that is available in MySQL 5.5, which optimized `CREATE INDEX` and `DROP INDEX` to avoid table-copying behavior.

The NDB Cluster `NDB` storage engine also supports online table schema changes, but uses its own syntax that is not compatible with the syntax used for `InnoDB` online DDL operations. For more information, see Section 18.5.11, “Online Operations with ALTER TABLE in NDB Cluster”.

Typically, you do not need to do anything special to enable online DDL. By default, MySQL performs the operation in place, as permitted, with as little locking as possible.

You can control aspects of a DDL operation using the `ALGORITHM` and `LOCK` clauses of the `ALTER TABLE` statement. These clauses are placed at the end of the statement, separated from the table and column specifications by commas. For example:

```
ALTER TABLE tbl_name ADD PRIMARY KEY (column), ALGORITHM=INPLACE, LOCK=NONE;
```

The `LOCK` clause is useful for fine-tuning the degree of concurrent access to the table. The `ALGORITHM` clause is primarily intended for performance comparisons and as a fallback to the older table-copying behavior in case you encounter any issues. For example:

- To avoid accidentally making the table unavailable for reads, writes, or both, specify a clause on the `ALTER TABLE` statement such as `LOCK=NONE` (permit reads and writes) or `LOCK=SHARED` (permit reads). The operation halts immediately if the requested level of concurrency is not available.
- To compare performance, run a statement with `ALGORITHM=INPLACE` and `ALGORITHM=COPY`. Alternatively, run a statement with the `old_alter_table` configuration option disabled and enabled.
- To avoid tying up the server with an `ALTER TABLE` operation that copies the table, include `ALGORITHM=INPLACE`. The statement halts immediately if it cannot use the in-place mechanism.

14.13.1 Online DDL Operations

Online support details, syntax examples, and usage notes for DDL operations are provided under the following topics in this section.

- Index Operations
- Primary Key Operations
- Column Operations
- Foreign Key Operations
- Table Operations
- Partitioning Operations

Index Operations

The following table provides an overview of online DDL support for index operations. An asterisk indicates additional information, an exception, or a dependency. For details, see Syntax and Usage Notes.

Table 14.8 Online DDL Support for Index Operations

Operation	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Creating or adding a secondary index	Yes	No	Yes	No
Dropping an index	Yes	No	Yes	Yes
Adding a `FULLTEXT` index	Yes*	No*	No	No
Changing the index type	Yes	No	Yes	Yes

Syntax and Usage Notes

- Creating or adding a secondary index

```
CREATE INDEX name ON table (col_list);
```

```
ALTER TABLE tbl_name ADD INDEX name (col_list);
```

The table remains available for read and write operations while the index is being created. The `CREATE INDEX` statement only finishes after all transactions that are accessing the table are completed, so that the initial state of the index reflects the most recent contents of the table.

Online DDL support for adding secondary indexes means that you can generally speed the overall process of creating and loading a table and associated indexes by creating the table without secondary indexes, then adding secondary indexes after the data is loaded.

A newly created secondary index contains only the committed data in the table at the time the `CREATE INDEX` or `ALTER TABLE` statement finishes executing. It does not contain any uncommitted values, old versions of values, or values marked for deletion but not yet removed from the old index.

If the server exits while creating a secondary index, upon recovery, MySQL drops any partially created indexes. You must re-run the `ALTER TABLE` or `CREATE INDEX` statement.

Some factors affect the performance, space usage, and semantics of this operation. For details, see Section 14.13.6, “Online DDL Limitations”.

- Dropping an index

```
DROP INDEX name ON table;
```

```
ALTER TABLE tbl_name DROP INDEX name;
```

The table remains available for read and write operations while the index is being dropped. The `DROP INDEX` statement only finishes after all transactions that are accessing the table are completed, so that the initial state of the index reflects the most recent contents of the table.

- Adding a `FULLTEXT` index

```
CREATE FULLTEXT INDEX name ON table(column);
```

Adding the first `FULLTEXT` index rebuilds the table if there is no user-defined `FTS_DOC_ID` column. Additional `FULLTEXT` indexes may be added without rebuilding the table.

- Changing the index type (`USING {BTREE | HASH}`)

```
ALTER TABLE tbl_name DROP INDEX i1, ADD INDEX i1(key_part,...) USING BTREE, ALGORITHM=INPLACE;
```

Primary Key Operations

The following table provides an overview of online DDL support for primary key operations. An asterisk indicates additional information, an exception, or a dependency. See Syntax and Usage Notes.

Table 14.9 Online DDL Support for Primary Key Operations

Operation	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Adding a primary key	Yes*	Yes*	Yes	No
Dropping a primary key	No	Yes	No	No
Dropping a primary key and adding another	Yes	Yes	Yes	No

Syntax and Usage Notes

- Adding a primary key

```
ALTER TABLE tbl_name ADD PRIMARY KEY (column), ALGORITHM=INPLACE, LOCK=NONE;
```

  Rebuilds the table in place. Data is reorganized substantially, making it an expensive operation. `ALGORITHM=INPLACE` is not permitted under certain conditions if columns have to be converted to `NOT NULL`.

  Restructuring the clustered index always requires copying table data. Thus, it is best to define the primary key when you create a table, rather than issuing `ALTER TABLE ... ADD PRIMARY KEY` later.

  When you create a `UNIQUE` or `PRIMARY KEY` index, MySQL must do some extra work. For `UNIQUE` indexes, MySQL checks that the table contains no duplicate values for the key. For a `PRIMARY KEY` index, MySQL also checks that none of the `PRIMARY KEY` columns contains a `NULL`.

  When you add a primary key using the `ALGORITHM=COPY` clause, MySQL converts `NULL` values in the associated columns to default values: 0 for numbers, an empty string for character-based columns and BLOBs, and 0000-00-00 00:00:00 for `DATETIME`. This is a non-standard behavior that Oracle recommends you not rely on. Adding a primary key using `ALGORITHM=INPLACE` is only permitted when the `SQL_MODE` setting includes the `strict_trans_tables` or `strict_all_tables` flags; when the `SQL_MODE` setting is strict, `ALGORITHM=INPLACE` is permitted, but the statement can still fail if the requested primary key columns contain `NULL` values. The `ALGORITHM=INPLACE` behavior is more standard-compliant.

  If you create a table without a primary key, `InnoDB` chooses one for you, which can be the first `UNIQUE` key defined on `NOT NULL` columns, or a system-generated key. To avoid uncertainty and the potential space requirement for an extra hidden column, specify the `PRIMARY KEY` clause as part of the `CREATE TABLE` statement.

  MySQL creates a new clustered index by copying the existing data from the original table to a temporary table that has the desired index structure. Once the data is completely copied to the temporary table, the original table is renamed with a different temporary table name. The temporary table comprising the new clustered index is renamed with the name of the original table, and the original table is dropped from the database.

  The online performance enhancements that apply to operations on secondary indexes do not apply to the primary key index. The rows of an InnoDB table are stored in a clustered index organized based on the primary key, forming what some database systems call an “index-organized table”. Because the table structure is closely tied to the primary key, redefining the primary key still requires copying the data.

  When an operation on the primary key uses `ALGORITHM=INPLACE`, even though the data is still copied, it is more efficient than using `ALGORITHM=COPY` because:

  - No undo logging or associated redo logging is required for `ALGORITHM=INPLACE`. These operations add overhead to DDL statements that use `ALGORITHM=COPY`.
  - The secondary index entries are pre-sorted, and so can be loaded in order.
  - The change buffer is not used, because there are no random-access inserts into the secondary indexes.

  If the server exits while creating a new clustered index, no data is lost, but you must complete the recovery process using the temporary tables that exist during the process. Since it is rare to re-create a clustered index or re-define primary keys on large tables, or to encounter a system crash during this operation, this manual does not provide information on recovering from this scenario.

- Dropping a primary key

```
ALTER TABLE tbl_name DROP PRIMARY KEY, ALGORITHM=COPY;
```

Only `ALGORITHM=COPY` supports dropping a primary key without adding a new one in the same `ALTER TABLE` statement.

- Dropping a primary key and adding another

```
ALTER TABLE tbl_name DROP PRIMARY KEY, ADD PRIMARY KEY (column), ALGORITHM=INPLACE, LOCK=NONE;
```

Data is reorganized substantially, making it an expensive operation.

Column Operations

The following table provides an overview of online DDL support for column operations. An asterisk indicates additional information, an exception, or a dependency. For details, see Syntax and Usage Notes.

Table 14.10 Online DDL Support for Column Operations

Operation	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Adding a column	Yes	Yes	Yes*	No
Dropping a column	Yes	Yes	Yes	No
Renaming a column	Yes	No	Yes*	Yes
Reordering columns	Yes	Yes	Yes	No
Setting a column default value	Yes	No	Yes	Yes
Changing the column data type	No	Yes	No	No
Dropping the column default value	Yes	No	Yes	Yes
Changing the auto-increment value	Yes	No	Yes	No*
Making a column `NULL`	Yes	Yes*	Yes	No
Making a column `NOT NULL`	Yes*	Yes*	Yes	No
Modifying the definition of an `ENUM` or `SET` column	Yes	No	Yes	Yes

Syntax and Usage Notes

- Adding a column

```
ALTER TABLE tbl_name ADD COLUMN column_name column_definition, ALGORITHM=INPLACE, LOCK=NONE;
```

Concurrent DML is not permitted when adding an auto-increment column. Data is reorganized substantially, making it an expensive operation. At a minimum, `ALGORITHM=INPLACE, LOCK=SHARED` is required.

- Dropping a column

```
ALTER TABLE tbl_name DROP COLUMN column_name, ALGORITHM=INPLACE, LOCK=NONE;
```

  Data is reorganized substantially, making it an expensive operation.

- Renaming a column

```
ALTER TABLE tbl CHANGE old_col_name new_col_name data_type, ALGORITHM=INPLACE, LOCK=NONE;
```

  To permit concurrent DML, keep the same data type and only change the column name.

  When you keep the same data type and `[NOT] NULL` attribute, only changing the column name, the operation can always be performed online.

  You can also rename a column that is part of a foreign key constraint. The foreign key definition is automatically updated to use the new column name. Renaming a column participating in a foreign key only works with `ALGORITHM=INPLACE`. If you use the `ALGORITHM=COPY` clause, or some other condition causes the command to use `ALGORITHM=COPY` behind the scenes, the `ALTER TABLE` statement fails.

- Reordering columns

  To reorder columns, use `FIRST` or `AFTER` in `CHANGE` or `MODIFY` operations.

```
ALTER TABLE tbl_name MODIFY COLUMN col_name column_definition FIRST, ALGORITHM=INPLACE, LOCK=NONE;
```

  Data is reorganized substantially, making it an expensive operation.

- Changing the column data type

```
ALTER TABLE tbl_name CHANGE c1 c1 BIGINT, ALGORITHM=COPY;
```

  Changing the column data type is only supported with `ALGORITHM=COPY`.

- Setting a column default value

```
ALTER TABLE tbl_name ALTER COLUMN col SET DEFAULT literal, ALGORITHM=INPLACE, LOCK=NONE;
```

  Only modifies table metadata. Default column values are stored in the .frm file for the table, not the `InnoDB` data dictionary.

- Dropping a column default value

```
ALTER TABLE tbl ALTER COLUMN col DROP DEFAULT, ALGORITHM=INPLACE, LOCK=NONE;
```

- Changing the auto-increment value

```
ALTER TABLE table AUTO_INCREMENT=next_value, ALGORITHM=INPLACE, LOCK=NONE;
```

  Modifies a value stored in memory, not the data file.

  In a distributed system using replication or sharding, you sometimes reset the auto-increment counter for a table to a specific value. The next row inserted into the table uses the specified value for its auto-increment column. You might also use this technique in a data warehousing environment where you periodically empty all the tables and reload them, and restart the auto-increment sequence from 1.

- Making a column `NULL`

```
ALTER TABLE tbl_name MODIFY COLUMN column_name data_type NULL, ALGORITHM=INPLACE, LOCK=NONE;
```

  Rebuilds the table in place. Data is reorganized substantially, making it an expensive operation.

- Making a column `NOT NULL`

```
ALTER TABLE tbl_name MODIFY COLUMN column_name data_type NOT NULL, ALGORITHM=INPLACE, LOCK=NONE;
```

Rebuilds the table in place. `STRICT_ALL_TABLES` or `STRICT_TRANS_TABLES` `SQL_MODE` is required for the operation to succeed. The operation fails if the column contains NULL values. The server prohibits changes to foreign key columns that have the potential to cause loss of referential integrity. See Section 13.1.7, “ALTER TABLE Statement”. Data is reorganized substantially, making it an expensive operation.

- Modifying the definition of an `ENUM` or `SET` column

```
CREATE TABLE t1 (c1 ENUM('a', 'b', 'c'));
ALTER TABLE t1 MODIFY COLUMN c1 ENUM('a', 'b', 'c', 'd'), ALGORITHM=INPLACE, LOCK=NONE;
```

Modifying the definition of an `ENUM` or `SET` column by adding new enumeration or set members to the *end* of the list of valid member values may be performed in place, as long as the storage size of the data type does not change. For example, adding a member to a `SET` column that has 8 members changes the required storage per value from 1 byte to 2 bytes; this requires a table copy. Adding members in the middle of the list causes renumbering of existing members, which requires a table copy.

Foreign Key Operations

The following table provides an overview of online DDL support for foreign key operations. An asterisk indicates additional information, an exception, or a dependency. For details, see Syntax and Usage Notes.

Table 14.11 Online DDL Support for Foreign Key Operations

Operation	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Adding a foreign key constraint	Yes*	No	Yes	Yes
Dropping a foreign key constraint	Yes	No	Yes	Yes

Syntax and Usage Notes

- Adding a foreign key constraint

  The `INPLACE` algorithm is supported when `foreign_key_checks` is disabled. Otherwise, only the `COPY` algorithm is supported.

```
ALTER TABLE tbl1 ADD CONSTRAINT fk_name FOREIGN KEY index (col1)
  REFERENCES tbl2(col2) referential_actions;
```

- Dropping a foreign key constraint

```
ALTER TABLE tbl DROP FOREIGN KEY fk_name;
```

Dropping a foreign key can be performed online with the `foreign_key_checks` option enabled or disabled.

If you do not know the names of the foreign key constraints on a particular table, issue the following statement and find the constraint name in the `CONSTRAINT` clause for each foreign key:

```
SHOW CREATE TABLE table\G
```

Or, query the `INFORMATION_SCHEMA.TABLE_CONSTRAINTS` table and use the `CONSTRAINT_NAME` and `CONSTRAINT_TYPE` columns to identify the foreign key names.

You can also drop a foreign key and its associated index in a single statement:

```
ALTER TABLE table DROP FOREIGN KEY constraint, DROP INDEX index;
```

Note

If foreign keys are already present in the table being altered (that is, it is a child table containing a `FOREIGN KEY ... REFERENCE` clause), additional restrictions apply to online DDL operations, even those not directly involving the foreign key columns:

- An `ALTER TABLE` on the child table could wait for another transaction to commit, if a change to the parent table causes associated changes in the child table through an `ON UPDATE` or `ON DELETE` clause using the `CASCADE` or `SET NULL` parameters.
- In the same way, if a table is the parent table in a foreign key relationship, even though it does not contain any `FOREIGN KEY` clauses, it could wait for the `ALTER TABLE` to complete if an `INSERT`, `UPDATE`, or `DELETE` statement causes an `ON UPDATE` or `ON DELETE` action in the child table.

Table Operations

The following table provides an overview of online DDL support for table operations. An asterisk indicates additional information, an exception, or a dependency. For details, see Syntax and Usage Notes.

Table 14.12 Online DDL Support for Table Operations

Operation	In Place	Rebuilds Table	Permits Concurrent DML	Only Modifies Metadata
Changing the `ROW_FORMAT`	Yes	Yes	Yes	No
Changing the `KEY_BLOCK_SIZE`	Yes	Yes	Yes	No
Setting persistent table statistics	Yes	No	Yes	Yes
Specifying a character set	Yes	Yes*	No	No
Converting a character set	No	Yes	No	No
Optimizing a table	Yes*	Yes	Yes	No
Rebuilding with the `FORCE` option	Yes*	Yes	Yes	No
Performing a null rebuild	Yes*	Yes	Yes	No
Renaming a table	Yes	No	Yes	Yes

Syntax and Usage Notes

- Changing the `ROW_FORMAT`

```
ALTER TABLE tbl_name ROW_FORMAT = row_format, ALGORITHM=INPLACE, LOCK=NONE;
```

Data is reorganized substantially, making it an expensive operation.

For additional information about the `ROW_FORMAT` option, see Table Options.

- Changing the `KEY_BLOCK_SIZE`

```
ALTER TABLE tbl_name KEY_BLOCK_SIZE = value, ALGORITHM=INPLACE, LOCK=NONE;
```

Data is reorganized substantially, making it an expensive operation.

For additional information about the KEY_BLOCK_SIZE option, see Table Options.

- Setting persistent table statistics options

```
ALTER TABLE tbl_name STATS_PERSISTENT=0, STATS_SAMPLE_PAGES=20, STATS_AUTO_RECALC=1, ALGORITHM=INPLACE, L
```

Only modifies table metadata.

Persistent statistics include STATS_PERSISTENT, STATS_AUTO_RECALC, and STATS_SAMPLE_PAGES. For more information, see Section 14.8.11.1, “Configuring Persistent Optimizer Statistics Parameters”.

- Specifying a character set

```
ALTER TABLE tbl_name CHARACTER SET = charset_name, ALGORITHM=INPLACE, LOCK=NONE;
```

Rebuilds the table if the new character encoding is different.

- Converting a character set

```
ALTER TABLE tbl_name CONVERT TO CHARACTER SET charset_name, ALGORITHM=COPY;
```

Rebuilds the table if the new character encoding is different.

- Optimizing a table

```
OPTIMIZE TABLE tbl_name;
```

Performed in-place as of MySQL 5.6.17. In-place operation is not supported for tables with FULLTEXT indexes. The operation uses the INPLACE algorithm, but ALGORITHM and LOCK syntax is not permitted.

- Rebuilding a table with the FORCE option

```
ALTER TABLE tbl_name FORCE, ALGORITHM=INPLACE, LOCK=NONE;
```

Uses ALGORITHM=INPLACE as of MySQL 5.6.17. ALGORITHM=INPLACE is not supported for tables with FULLTEXT indexes.

- Performing a "null" rebuild

```
ALTER TABLE tbl_name ENGINE=InnoDB, ALGORITHM=INPLACE, LOCK=NONE;
```

Uses ALGORITHM=INPLACE as of MySQL 5.6.17. ALGORITHM=INPLACE is not supported for tables with FULLTEXT indexes.

- Renaming a table

```
ALTER TABLE old_tbl_name RENAME TO new_tbl_name, ALGORITHM=INPLACE, LOCK=NONE;
```

MySQL renames files that correspond to the table *tbl_name* without making a copy. (You can also use the RENAME TABLE statement to rename tables. See Section 13.1.32, “RENAME TABLE Statement”.) Privileges granted specifically for the renamed table are not migrated to the new name. They must be changed manually.

Partitioning Operations

With the exception of ALTER TABLE partitioning clauses, online DDL operations for partitioned InnoDB tables follow the same rules that apply to regular InnoDB tables.

`ALTER TABLE` partitioning clauses do not go through the same internal online DDL API as regular non-partitioned `InnoDB` tables, and most do not support `ALGORITHM` and `LOCK` clauses.

If you use a partitioning clause in an `ALTER TABLE` statement, the partitioned table is repartitioned using the `ALTER TABLE COPY` algorithm. In other words, a new partitioned table is created with the new partitioning scheme. The newly created table includes any changes applied by the `ALTER TABLE` statement and the table data is copied into the new table structure.

If you do not modify table partitioning using `ALTER TABLE` partitioning clauses that use the `COPY` algorithm or perform any other partition management in your `ALTER TABLE` statement, `ALTER TABLE` performs supported `INPLACE` operations on each table partition. Be aware, however, that when `INPLACE ALTER TABLE` operations are performed on each partition, there is increased demand on system resources due to operations being performed on multiple partitions.

Even though partitioning clauses of the `ALTER TABLE` statement do not go through the same internal online DDL API as regular non-partitioned `InnoDB` tables, MySQL still attempts to minimize data copying and locking where possible:

- `ADD PARTITION` and `DROP PARTITION` for tables partitioned by `RANGE` or `LIST` do not copy existing data.
- `TRUNCATE PARTITION` does not copy existing data.
- Concurrent queries are allowed during `ADD PARTITION` and `COALESCE PARTITION` for tables partitioned by `HASH` or `LIST`. MySQL copies the data while holding a shared lock.
- For `REORGANIZE PARTITION`, `REBUILD PARTITION`, or `ADD PARTITION` or `COALESCE PARTITION` for a table partitioned by `LINEAR HASH` or `LIST`, concurrent queries are allowed. Data from the affected partitions is copied while holding a shared metadata (read) lock on the table.
- `PARTITION BY` and `REMOVE PARTITIONING` permit concurrent queries.

`ALTER TABLE` statements that change a table's partitioning cannot use `ALGORITHM` with `DEFAULT`, `COPY`, or `INPLACE`. (MySQL 5.6.11 and later support such statements using `ALGORITHM=1` or `ALGORITHM=2` when performing upgrades or downgrades.) Such `ALTER TABLE` statements also do not support a `LOCK` clause.

For additional information about `ALTER TABLE` partitioning clauses, see Partitioning Options, and Section 13.1.7.1, “ALTER TABLE Partition Operations”. For information about partitioning in general, see Chapter 19, *Partitioning*.

14.13.2 Online DDL Performance and Concurrency

Online DDL improves several aspects of MySQL operation:

- Applications that access the table are more responsive because queries and DML operations on the table can proceed while the DDL operation is in progress. Reduced locking and waiting for MySQL server resources leads to greater scalability, even for operations that are not involved in the DDL operation.
- In-place operations avoid the disk I/O and CPU cycles associated with the table-copy method, which minimizes overall load on the database. Minimizing load helps maintain good performance and high throughput during the DDL operation.
- In-place operations read less data into the buffer pool than the table-copy operations, which reduces purging of frequently accessed data from memory. Purging of frequently accessed data can cause a temporary performance dip after a DDL operation.

The LOCK clause

By default, MySQL uses as little locking as possible during a DDL operation. The `LOCK` clause can be specified to enforce more restrictive locking, if required. If the `LOCK` clause specifies a less restrictive

level of locking than is permitted for a particular DDL operation, the statement fails with an error. `LOCK` clauses are described below, in order of least to most restrictive:

- `LOCK=NONE`:

  Permits concurrent queries and DML.

  For example, use this clause for tables involving customer signups or purchases, to avoid making the tables unavailable during lengthy DDL operations.

- `LOCK=SHARED`:

  Permits concurrent queries but blocks DML.

  For example, use this clause on data warehouse tables, where you can delay data load operations until the DDL operation is finished, but queries cannot be delayed for long periods.

- `LOCK=DEFAULT`:

  Permits as much concurrency as possible (concurrent queries, DML, or both). Omitting the `LOCK` clause is the same as specifying `LOCK=DEFAULT`.

  Use this clause when you know that the default locking level of the DDL statement does not cause availability problems for the table.

- `LOCK=EXCLUSIVE`:

  Blocks concurrent queries and DML.

  Use this clause if the primary concern is finishing the DDL operation in the shortest amount of time possible, and concurrent query and DML access is not necessary. You might also use this clause if the server is supposed to be idle, to avoid unexpected table accesses.

Online DDL and Metadata Locks

Online DDL operations can be viewed as having three phases:

- *Phase 1: Initialization*

  In the initialization phase, the server determines how much concurrency is permitted during the operation, taking into account storage engine capabilities, operations specified in the statement, and user-specified `ALGORITHM` and `LOCK` options. During this phase, a shared upgradeable metadata lock is taken to protect the current table definition.

- *Phase 2: Execution*

  In this phase, the statement is prepared and executed. Whether the metadata lock is upgraded to exclusive depends on the factors assessed in the initialization phase. If an exclusive metadata lock is required, it is only taken briefly during statement preparation.

- *Phase 3: Commit Table Definition*

  In the commit table definition phase, the metadata lock is upgraded to exclusive to evict the old table definition and commit the new one. Once granted, the duration of the exclusive metadata lock is brief.

Due to the exclusive metadata lock requirements outlined above, an online DDL operation may have to wait for concurrent transactions that hold metadata locks on the table to commit or rollback. Transactions started before or during the DDL operation can hold metadata locks on the table being altered. In the case of a long running or inactive transaction, an online DDL operation can time out waiting for an exclusive metadata lock. Additionally, a pending exclusive metadata lock requested by an online DDL operation blocks subsequent transactions on the table.

The following example demonstrates an online DDL operation waiting for an exclusive metadata lock, and how a pending metadata lock blocks subsequent transactions on the table.

Session 1:

```
mysql> CREATE TABLE t1 (c1 INT) ENGINE=InnoDB;
mysql> START TRANSACTION;
mysql> SELECT * FROM t1;
```

The session 1 `SELECT` statement takes a shared metadata lock on table t1.

Session 2:

```
mysql> ALTER TABLE t1 ADD COLUMN x INT, ALGORITHM=INPLACE, LOCK=NONE;
```

The online DDL operation in session 2, which requires an exclusive metadata lock on table t1 to commit table definition changes, must wait for the session 1 transaction to commit or roll back.

Session 3:

```
mysql> SELECT * FROM t1;
```

The `SELECT` statement issued in session 3 is blocked waiting for the exclusive metadata lock requested by the `ALTER TABLE` operation in session 2 to be granted.

You can use `SHOW FULL PROCESSLIST` to determine if transactions are waiting for a metadata lock.

```
mysql> SHOW FULL PROCESSLIST\G
...
*************************** 2. row ***************************
     Id: 5
   User: root
   Host: localhost
     db: test
Command: Query
   Time: 44
  State: Waiting for table metadata lock
   Info: ALTER TABLE t1 ADD COLUMN x INT, ALGORITHM=INPLACE, LOCK=NONE
...
*************************** 4. row ***************************
     Id: 7
   User: root
   Host: localhost
     db: test
Command: Query
   Time: 5
  State: Waiting for table metadata lock
   Info: SELECT * FROM t1
4 rows in set (0.00 sec)
```

Online DDL Performance

The performance of a DDL operation is largely determined by whether the operation is performed in place and whether it rebuilds the table.

To assess the relative performance of a DDL operation, you can compare results using `ALGORITHM=INPLACE` with results using `ALGORITHM=COPY`. Alternatively, you can compare results with `old_alter_table` disabled and enabled.

For DDL operations that modify table data, you can determine whether a DDL operation performs changes in place or performs a table copy by looking at the “rows affected” value displayed after the command finishes. For example:

- Changing the default value of a column (fast, does not affect the table data):

  ```
  Query OK, 0 rows affected (0.07 sec)
  ```

- Adding an index (takes time, but `0 rows affected` shows that the table is not copied):

```
Query OK, 0 rows affected (21.42 sec)
```

- Changing the data type of a column (takes substantial time and requires rebuilding all the rows of the table):

```
Query OK, 1671168 rows affected (1 min 35.54 sec)
```

Before running a DDL operation on a large table, check whether the operation is fast or slow as follows:

1. Clone the table structure.
2. Populate the cloned table with a small amount of data.
3. Run the DDL operation on the cloned table.
4. Check whether the “rows affected” value is zero or not. A nonzero value means the operation copies table data, which might require special planning. For example, you might do the DDL operation during a period of scheduled downtime, or on each replica server one at a time.

Note

For a greater understanding of the MySQL processing associated with a DDL operation, examine `Performance Schema` and `INFORMATION_SCHEMA` tables related to `InnoDB` before and after DDL operations to see the number of physical reads, writes, memory allocations, and so on.

Because there is some processing work involved with recording the changes made by concurrent DML operations, then applying those changes at the end, an online DDL operation could take longer overall than the table-copy mechanism that blocks table access from other sessions. The reduction in raw performance is balanced against better responsiveness for applications that use the table. When evaluating the techniques for changing table structure, consider end-user perception of performance, based on factors such as load times for web pages.

14.13.3 Online DDL Space Requirements

Online DDL operations have the following space requirements:

- Space for temporary log files

  A temporary log file records concurrent DML when an online DDL operation creates an index or alters a table. The temporary log file is extended as required by the value of `innodb_sort_buffer_size` up to a maximum specified by `innodb_online_alter_log_max_size`. If a temporary log file exceeds the size limit, the online DDL operation fails, and uncommitted concurrent DML operations are rolled back. A large `innodb_online_alter_log_max_size` setting permits more DML during an online DDL operation, but it also extends the period of time at the end of the DDL operation when the table is locked to apply logged DML.

  If the operation takes a long time and concurrent DML modifies the table so much that the size of the temporary log file exceeds the value of `innodb_online_alter_log_max_size`, the online DDL operation fails with a `DB_ONLINE_LOG_TOO_BIG` error.

- Space for temporary sort files

  Online DDL operations that rebuild the table write temporary sort files to the MySQL temporary directory (`$TMPDIR` on Unix, `%TEMP%` on Windows, or the directory specified by `--tmpdir`) during index creation. Temporary sort files are not created in the directory that contains the original table. Each temporary sort file is large enough to hold one column of data, and each sort file is removed when its data is merged into the final table or index. Operations involving temporary sort files may require temporary space equal to the amount of data in the table plus indexes. An error is reported if online DDL operation uses all of the available disk space on the file system where the data directory resides.

If the MySQL temporary directory is not large enough to hold the sort files, set `tmpdir` to a different directory. Alternatively, define a separate temporary directory for online DDL operations using `innodb_tmpdir`. This option was introduced in MySQL 5.6.29 to help avoid temporary directory overflows that could occur as a result of large temporary sort files.

- Space for an intermediate table file

  Some online DDL operations that rebuild the table create a temporary intermediate table file in the same directory as the original table. An intermediate table file may require space equal to the size of the original table. Intermediate table file names begin with `#sql-ib` prefix and only appear briefly during the online DDL operation.

  The `innodb_tmpdir` option is not applicable to intermediate table files.

14.13.4 Simplifying DDL Statements with Online DDL

Before the introduction of online DDL, it was common practice to combine many DDL operations into a single `ALTER TABLE` statement. Because each `ALTER TABLE` statement involved copying and rebuilding the table, it was more efficient to make several changes to the same table at once, since those changes could all be done with a single rebuild operation for the table. The downside was that SQL code involving DDL operations was harder to maintain and to reuse in different scripts. If the specific changes were different each time, you might have to construct a new complex `ALTER TABLE` for each slightly different scenario.

For DDL operations that can be done in place, you can separate them into individual `ALTER TABLE` statements for easier scripting and maintenance, without sacrificing efficiency. For example, you might take a complicated statement such as:

```
ALTER TABLE t1 ADD INDEX i1(c1), ADD UNIQUE INDEX i2(c2),
  CHANGE c4_old_name c4_new_name INTEGER UNSIGNED;
```

and break it down into simpler parts that can be tested and performed independently, such as:

```
ALTER TABLE t1 ADD INDEX i1(c1);
ALTER TABLE t1 ADD UNIQUE INDEX i2(c2);
ALTER TABLE t1 CHANGE c4_old_name c4_new_name INTEGER UNSIGNED NOT NULL;
```

You might still use multi-part `ALTER TABLE` statements for:

- Operations that must be performed in a specific sequence, such as creating an index followed by a foreign key constraint that uses that index.
- Operations all using the same specific `LOCK` clause, that you want to either succeed or fail as a group.
- Operations that cannot be performed in place, that is, that still use the table-copy method.
- Operations for which you specify `ALGORITHM=COPY` or `old_alter_table=1`, to force the table-copying behavior if needed for precise backward-compatibility in specialized scenarios.

14.13.5 Online DDL Failure Conditions

The failure of an online DDL operation is typically due to one of the following conditions:

- An `ALGORITHM` clause specifies an algorithm that is not compatible with the particular type of DDL operation or storage engine.
- A `LOCK` clause specifies a low degree of locking (`SHARED` or `NONE`) that is not compatible with the particular type of DDL operation.
- A timeout occurs while waiting for an exclusive lock on the table, which may be needed briefly during the initial and final phases of the DDL operation.

- The `tmpdir` or `innodb_tmpdir` file system runs out of disk space, while MySQL writes temporary sort files on disk during index creation. For more information, see Section 14.13.3, “Online DDL Space Requirements”.
- The operation takes a long time and concurrent DML modifies the table so much that the size of the temporary online log exceeds the value of the `innodb_online_alter_log_max_size` configuration option. This condition causes a `DB_ONLINE_LOG_TOO_BIG` error.
- Concurrent DML makes changes to the table that are allowed with the original table definition, but not with the new one. The operation only fails at the very end, when MySQL tries to apply all the changes from concurrent DML statements. For example, you might insert duplicate values into a column while a unique index is being created, or you might insert `NULL` values into a column while creating a primary key index on that column. The changes made by the concurrent DML take precedence, and the `ALTER TABLE` operation is effectively rolled back.

14.13.6 Online DDL Limitations

The following limitations apply to online DDL operations:

- The table is copied when creating an index on a `TEMPORARY TABLE`.
- The `ALTER TABLE` clause `LOCK=NONE` is not permitted if there are `ON...CASCADE` or `ON...SET NULL` constraints on the table.
- Before an online DDL operation can finish, it must wait for transactions that hold metadata locks on the table to commit or roll back. An online DDL operation may briefly require an exclusive metadata lock on the table during its execution phase, and always requires one in the final phase of the operation when updating the table definition. Consequently, transactions holding metadata locks on the table can cause an online DDL operation to block. The transactions that hold metadata locks on the table may have been started before or during the online DDL operation. A long running or inactive transaction that holds a metadata lock on the table can cause an online DDL operation to timeout.
- An online DDL operation on a table in a foreign key relationship does not wait for a transaction executing on the other table in the foreign key relationship to commit or rollback. The transaction holds an exclusive metadata lock on the table it is updating and shared metadata lock on the foreign-key-related table (required for foreign key checking). The shared metadata lock permits the online DDL operation to proceed but blocks the operation in its final phase, when an exclusive metadata lock is required to update the table definition. This scenario can result in deadlocks as other transactions wait for the online DDL operation to finish.
- When running an online DDL operation, the thread that runs the `ALTER TABLE` statement applies an online log of DML operations that were run concurrently on the same table from other connection threads. When the DML operations are applied, it is possible to encounter a duplicate key entry error (`ERROR 1062 (23000): Duplicate entry`), even if the duplicate entry is only temporary and would be reverted by a later entry in the online log. This is similar to the idea of a foreign key constraint check in `InnoDB` in which constraints must hold during a transaction.
- `OPTIMIZE TABLE` for an `InnoDB` table is mapped to an `ALTER TABLE` operation to rebuild the table and update index statistics and free unused space in the clustered index. Prior to 5.6.17, there is no online DDL support for this operation. Secondary indexes are not created as efficiently because keys are inserted in the order they appeared in the primary key. As of 5.6.17, `OPTIMIZE TABLE` is supported with the addition of online DDL support for rebuilding regular and partitioned `InnoDB` tables.
- Tables created before MySQL 5.6 that include temporal columns (`DATE`, `DATETIME` or `TIMESTAMP`) and have not been rebuilt using `ALGORITHM=COPY` do not support `ALGORITHM=INPLACE`. In this case, an `ALTER TABLE ... ALGORITHM=INPLACE` operation returns the following error:

```
ERROR 1846 (0A000): ALGORITHM=INPLACE is not supported.
Reason: Cannot change column type INPLACE. Try ALGORITHM=COPY.
```

- The following limitations are generally applicable to online DDL operations on large tables that involve rebuilding the table:
  - There is no mechanism to pause an online DDL operation or to throttle I/O or CPU usage for an online DDL operation.
  - Progress monitoring capability for online DDL operations is limited until MySQL 5.7.6, which introduces Performance Schema stage events for monitoring `ALTER TABLE` progress. See Monitoring ALTER TABLE Progress for InnoDB Tables Using Performance Schema.
  - Rollback of an online DDL operation can be expensive should the operation fail.
  - Long running online DDL operations can cause replication lag. An online DDL operation must finish running on the source before it is run on the replica. Also, DML that was processed concurrently on the source is only processed on the replica after the DDL operation on the replica is completed.

  For additional information related to running online DDL operations on large tables, see Section 14.13.2, “Online DDL Performance and Concurrency”.

14.14 InnoDB Startup Options and System Variables

- System variables that are true or false can be enabled at server startup by naming them, or disabled by using a `--skip-` prefix. For example, to enable or disable the `InnoDB` adaptive hash index, you can use `--innodb-adaptive-hash-index` or `--skip-innodb-adaptive-hash-index` on the command line, or `innodb_adaptive_hash_index` or `skip_innodb_adaptive_hash_index` in an option file.
- System variables that take a numeric value can be specified as `--var_name=value` on the command line or as `var_name=value` in option files.
- Many system variables can be changed at runtime (see Section 5.1.8.2, “Dynamic System Variables”).
- For information about `GLOBAL` and `SESSION` variable scope modifiers, refer to the `SET` statement documentation.
- Certain options control the locations and layout of the `InnoDB` data files. Section 14.8.1, “InnoDB Startup Configuration” explains how to use these options.
- Some options, which you might not use initially, help tune `InnoDB` performance characteristics based on machine capacity and your database workload.
- For more information on specifying options and system variables, see Section 4.2.2, “Specifying Program Options”.

Table 14.13 InnoDB Option and Variable Reference

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
daemon_memcached_enable_binlog	Yes	Yes	Yes		Global	No
daemon_memcached_engine_lib_name	Yes	Yes	Yes		Global	No
daemon_memcached_engine_lib_path	Yes	Yes	Yes		Global	No
daemon_memcached_option	Yes	Yes	Yes		Global	No
daemon_memcached_r_batch_size	Yes	Yes	Yes		Global	No
daemon_memcached_w_batch_size	Yes	Yes	Yes		Global	No
foreign_key_checks			Yes		Both	Ye
ignore_builtin_innodb	Yes	Yes	Yes		Global	No
innodb	Yes	Yes				

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
innodb_adaptive_flushing	Yes	Yes	Yes		Global	Yes
innodb_adaptive_flushing_lwm	Yes	Yes	Yes		Global	Yes
innodb_adaptive_hash_index	Yes	Yes	Yes		Global	Yes
innodb_adaptive_max_sleep_delay	Yes	Yes	Yes		Global	Yes
innodb_additional_mem_pool_size	Yes	Yes	Yes		Global	No
innodb_api_bk_commit_interval	Yes	Yes	Yes		Global	Yes
innodb_api_disable_rowlock	Yes	Yes	Yes		Global	No
innodb_api_enable_binlog	Yes	Yes	Yes		Global	No
innodb_api_enable_mdl	Yes	Yes	Yes		Global	No
innodb_api_trx_level	Yes	Yes	Yes		Global	Yes
innodb_autoextend_increment	Yes	Yes	Yes		Global	Yes
innodb_autoinc_lock_mode	Yes	Yes	Yes		Global	No
Innodb_available_undo_logs				Yes	Global	No
Innodb_buffer_pool_bytes_data				Yes	Global	No
Innodb_buffer_pool_bytes_dirty				Yes	Global	No
innodb_buffer_pool_dump_at_shutdown	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_dump_now	Yes	Yes	Yes		Global	Yes
Innodb_buffer_pool_dump_status				Yes	Global	No
innodb_buffer_pool_filename	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_instances	Yes	Yes	Yes		Global	No
innodb_buffer_pool_load_abort	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_load_at_startup	Yes	Yes	Yes		Global	No
innodb_buffer_pool_load_now	Yes	Yes	Yes		Global	Yes
Innodb_buffer_pool_load_status				Yes	Global	No
Innodb_buffer_pool_pages_data				Yes	Global	No
Innodb_buffer_pool_pages_dirty				Yes	Global	No
Innodb_buffer_pool_pages_flushed				Yes	Global	No
Innodb_buffer_pool_pages_free				Yes	Global	No
Innodb_buffer_pool_pages_latched				Yes	Global	No
Innodb_buffer_pool_pages_misc				Yes	Global	No
Innodb_buffer_pool_pages_total				Yes	Global	No
Innodb_buffer_pool_read_ahead				Yes	Global	No
Innodb_buffer_pool_read_ahead_evicted				Yes	Global	No
Innodb_buffer_pool_read_ahead_rnd				Yes	Global	No
Innodb_buffer_pool_read_requests				Yes	Global	No
Innodb_buffer_pool_reads				Yes	Global	No
innodb_buffer_pool_size	Yes	Yes	Yes		Global	No
Innodb_buffer_pool_wait_free				Yes	Global	No
Innodb_buffer_pool_write_requests				Yes	Global	No
innodb_change_buffer_max_size	Yes	Yes	Yes		Global	Yes
innodb_change_buffering	Yes	Yes	Yes		Global	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
innodb_change_buffering_debug	Yes	Yes	Yes		Global	Yes
innodb_checksum_algorithm	Yes	Yes	Yes		Global	Yes
innodb_checksums	Yes	Yes	Yes		Global	No
innodb_cmp_per_index_enabled	Yes	Yes	Yes		Global	Yes
innodb_commit_concurrency	Yes	Yes	Yes		Global	Yes
innodb_compression_failure_threshold_pct	Yes	Yes	Yes		Global	Yes
innodb_compression_level	Yes	Yes	Yes		Global	Yes
innodb_compression_pad_pct_max	Yes	Yes	Yes		Global	Yes
innodb_concurrency_tickets	Yes	Yes	Yes		Global	Yes
innodb_data_file_path	Yes	Yes	Yes		Global	No
Innodb_data_fsyncs				Yes	Global	No
innodb_data_home_dir	Yes	Yes	Yes		Global	No
Innodb_data_pending_fsyncs				Yes	Global	No
Innodb_data_pending_reads				Yes	Global	No
Innodb_data_pending_writes				Yes	Global	No
Innodb_data_read				Yes	Global	No
Innodb_data_reads				Yes	Global	No
Innodb_data_writes				Yes	Global	No
Innodb_data_written				Yes	Global	No
Innodb_dblwr_pages_written				Yes	Global	No
Innodb_dblwr_writes				Yes	Global	No
innodb_disable_sort_file_cache	Yes	Yes	Yes		Global	Yes
innodb_doublewrite	Yes	Yes	Yes		Global	No
innodb_fast_shutdown	Yes	Yes	Yes		Global	Yes
innodb_fil_make_page_dirty_debug	Yes	Yes	Yes		Global	Yes
innodb_file_format	Yes	Yes	Yes		Global	Yes
innodb_file_format_check	Yes	Yes	Yes		Global	No
innodb_file_format_max	Yes	Yes	Yes		Global	Yes
innodb_file_per_table	Yes	Yes	Yes		Global	Yes
innodb_flush_log_at_timeout	Yes	Yes	Yes		Global	Yes
innodb_flush_log_at_trx_commit	Yes	Yes	Yes		Global	Yes
innodb_flush_method	Yes	Yes	Yes		Global	No
innodb_flush_neighbors	Yes	Yes	Yes		Global	Yes
innodb_flushing_avg_loops	Yes	Yes	Yes		Global	Yes
innodb_force_load_corrupted	Yes	Yes	Yes		Global	No
innodb_force_recovery	Yes	Yes	Yes		Global	No
innodb_ft_aux_table			Yes		Global	Yes
innodb_ft_cache_size	Yes	Yes	Yes		Global	No
innodb_ft_enable_diag_print	Yes	Yes	Yes		Global	Yes
innodb_ft_enable_stopword	Yes	Yes	Yes		Both	Yes
innodb_ft_max_token_size	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
innodb_ft_min_token_size	Yes	Yes	Yes		Global	No
innodb_ft_num_word_optimize	Yes	Yes	Yes		Global	Yes
innodb_ft_result_cache_limit	Yes	Yes	Yes		Global	Yes
innodb_ft_server_stopword_table	Yes	Yes	Yes		Global	Yes
innodb_ft_sort_pll_degree	Yes	Yes	Yes		Global	No
innodb_ft_total_cache_size	Yes	Yes	Yes		Global	No
innodb_ft_user_stopword_table	Yes	Yes	Yes		Both	Yes
Innodb_have_atomic_builtins				Yes	Global	No
innodb_io_capacity	Yes	Yes	Yes		Global	Yes
innodb_io_capacity_max	Yes	Yes	Yes		Global	Yes
innodb_large_prefix	Yes	Yes	Yes		Global	Yes
innodb_limit_optimistic_insert_debug	Yes	Yes	Yes		Global	Yes
innodb_lock_wait_timeout	Yes	Yes	Yes		Both	Yes
innodb_locks_unsafe_for_binlog	Yes	Yes	Yes		Global	No
innodb_log_buffer_size	Yes	Yes	Yes		Global	No
innodb_log_checkpoint_now	Yes	Yes	Yes		Global	Yes
innodb_log_compressed_pages	Yes	Yes	Yes		Global	Yes
innodb_log_file_size	Yes	Yes	Yes		Global	No
innodb_log_files_in_group	Yes	Yes	Yes		Global	No
innodb_log_group_home_dir	Yes	Yes	Yes		Global	No
Innodb_log_waits				Yes	Global	No
Innodb_log_write_requests				Yes	Global	No
Innodb_log_writes				Yes	Global	No
innodb_lru_scan_depth	Yes	Yes	Yes		Global	Yes
innodb_max_dirty_pages_pct	Yes	Yes	Yes		Global	Yes
innodb_max_dirty_pages_pct_lwm	Yes	Yes	Yes		Global	Yes
innodb_max_purge_lag	Yes	Yes	Yes		Global	Yes
innodb_max_purge_lag_delay	Yes	Yes	Yes		Global	Yes
innodb_mirrored_log_groups	Yes	Yes	Yes		Global	No
innodb_monitor_disable	Yes	Yes	Yes		Global	Yes
innodb_monitor_enable	Yes	Yes	Yes		Global	Yes
innodb_monitor_reset	Yes	Yes	Yes		Global	Yes
innodb_monitor_reset_all	Yes	Yes	Yes		Global	Yes
Innodb_num_open_files				Yes	Global	No
innodb_numa_interleave	Yes	Yes	Yes		Global	No
innodb_old_blocks_pct	Yes	Yes	Yes		Global	Yes
innodb_old_blocks_time	Yes	Yes	Yes		Global	Yes
innodb_online_alter_log_max_size	Yes	Yes	Yes		Global	Yes
innodb_open_files	Yes	Yes	Yes		Global	No
innodb_optimize_fulltext_only	Yes	Yes	Yes		Global	Yes
Innodb_os_log_fsyncs				Yes	Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
Innodb_os_log_pending_fsyncs				Yes	Global	No
Innodb_os_log_pending_writes				Yes	Global	No
Innodb_os_log_written				Yes	Global	No
Innodb_page_size				Yes	Global	No
innodb_page_size	Yes	Yes	Yes		Global	No
Innodb_pages_created				Yes	Global	No
Innodb_pages_read				Yes	Global	No
Innodb_pages_written				Yes	Global	No
innodb_print_all_deadlocks	Yes	Yes	Yes		Global	Ye
innodb_purge_batch_size	Yes	Yes	Yes		Global	Ye
innodb_purge_threads	Yes	Yes	Yes		Global	No
innodb_random_read_ahead	Yes	Yes	Yes		Global	Ye
innodb_read_ahead_threshold	Yes	Yes	Yes		Global	Ye
innodb_read_io_threads	Yes	Yes	Yes		Global	No
innodb_read_only	Yes	Yes	Yes		Global	No
innodb_replication_delay	Yes	Yes	Yes		Global	Ye
innodb_rollback_on_timeout	Yes	Yes	Yes		Global	No
innodb_rollback_segments	Yes	Yes	Yes		Global	Ye
Innodb_row_lock_current_waits				Yes	Global	No
Innodb_row_lock_time				Yes	Global	No
Innodb_row_lock_time_avg				Yes	Global	No
Innodb_row_lock_time_max				Yes	Global	No
Innodb_row_lock_waits				Yes	Global	No
Innodb_rows_deleted				Yes	Global	No
Innodb_rows_inserted				Yes	Global	No
Innodb_rows_read				Yes	Global	No
Innodb_rows_updated				Yes	Global	No
innodb_saved_page_number_debug	Yes	Yes	Yes		Global	Ye
innodb_sort_buffer_size	Yes	Yes	Yes		Global	No
innodb_spin_wait_delay	Yes	Yes	Yes		Global	Ye
innodb_stats_auto_recalc	Yes	Yes	Yes		Global	Ye
innodb_stats_include_delete_marked	Yes	Yes	Yes		Global	Ye
innodb_stats_method	Yes	Yes	Yes		Global	Ye
innodb_stats_on_metadata	Yes	Yes	Yes		Global	Ye
innodb_stats_persistent	Yes	Yes	Yes		Global	Ye
innodb_stats_persistent_sample_pages	Yes	Yes	Yes		Global	Ye
innodb_stats_sample_pages	Yes	Yes	Yes		Global	Ye
innodb_stats_transient_sample_pages	Yes	Yes	Yes		Global	Ye
innodb-status-file	Yes	Yes				
innodb_status_output	Yes	Yes	Yes		Global	Ye
innodb_status_output_locks	Yes	Yes	Yes		Global	Ye

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
innodb_strict_mode	Yes	Yes	Yes		Both	Yes
innodb_support_xa	Yes	Yes	Yes		Both	Yes
innodb_sync_array_size	Yes	Yes	Yes		Global	No
innodb_sync_spin_loops	Yes	Yes	Yes		Global	Yes
innodb_table_locks	Yes	Yes	Yes		Both	Yes
innodb_thread_concurrency	Yes	Yes	Yes		Global	Yes
innodb_thread_sleep_delay	Yes	Yes	Yes		Global	Yes
innodb_tmpdir	Yes	Yes	Yes		Both	Yes
Innodb_truncated_status_writes				Yes	Global	No
innodb_trx_purge_view_update_only_debug	Yes	Yes	Yes		Global	Yes
innodb_trx_rseg_n_slots_debug	Yes	Yes	Yes		Global	Yes
innodb_undo_directory	Yes	Yes	Yes		Global	No
innodb_undo_logs	Yes	Yes	Yes		Global	Yes
innodb_undo_tablespaces	Yes	Yes	Yes		Global	No
innodb_use_native_aio	Yes	Yes	Yes		Global	No
innodb_use_sys_malloc	Yes	Yes	Yes		Global	No
innodb_version			Yes		Global	No
innodb_write_io_threads	Yes	Yes	Yes		Global	No
timed_mutexes	Yes	Yes	Yes		Global	Yes
unique_checks			Yes		Both	Yes

InnoDB Command Options

- `--innodb[=value]`

Command-Line Format	`--innodb[=value]`
Deprecated	5.6.21
Type	Enumeration
Default Value	`ON`

Valid Values	`OFF`
	`ON`
	`FORCE`

Controls loading of the `InnoDB` storage engine, if the server was compiled with `InnoDB` support. This option has a tristate format, with possible values of `OFF`, `ON`, or `FORCE`. See Section 5.5.1, “Installing and Uninstalling Plugins”.

To disable `InnoDB`, use `--innodb=OFF` or `--skip-innodb`. In this case, because the default storage engine is `InnoDB`, the server does not start unless you also use `--default-storage-engine` and `--default-tmp-storage-engine` to set the default to some other engine for both permanent and `TEMPORARY` tables.

As of MySQL 5.6.21, `--innodb=OFF` and `--skip-innodb` options are deprecated and their use results in a warning. Expect these options to be removed in a future MySQL release.

- `--innodb-status-file`

Command-Line Format	`--innodb-status-file[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

The `--innodb-status-file` startup option controls whether `InnoDB` creates a file named `innodb_status.pid` in the data directory and writes `SHOW ENGINE INNODB STATUS` output to it every 15 seconds, approximately.

The `innodb_status.pid` file is not created by default. To create it, start `mysqld` with the `--innodb-status-file` option. `InnoDB` removes the file when the server is shut down normally. If an abnormal shutdown occurs, the status file may have to be removed manually.

The `--innodb-status-file` option is intended for temporary use, as `SHOW ENGINE INNODB STATUS` output generation can affect performance, and the `innodb_status.pid` file can become quite large over time.

For related information, see Section 14.17.2, “Enabling InnoDB Monitors”.

- `--skip-innodb`

  Disable the `InnoDB` storage engine. See the description of `--innodb`.

InnoDB System Variables

- `daemon_memcached_enable_binlog`

Command-Line Format	`--daemon-memcached-enable-binlog[={OFF|ON}]`
System Variable	`daemon_memcached_enable_binlog`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`OFF`

Enable this option on the source server to use the `InnoDB memcached` plugin (`daemon_memcached`) with the MySQL binary log. This option can only be set at server startup. You must also enable the MySQL binary log on the source server using the `--log-bin` option.

For more information, see Section 14.20.6, “The InnoDB memcached Plugin and Replication”.

- `daemon_memcached_engine_lib_name`

Command-Line Format	`--daemon-memcached-engine-lib-name=file_name`
System Variable	`daemon_memcached_engine_lib_name`
Scope	Global
Dynamic	No
Type	File name
Default Value	`innodb_engine.so`

Specifies the shared library that implements the `InnoDB memcached` plugin.

For more information, see Section 14.20.3, “Setting Up the InnoDB memcached Plugin”.

- `daemon_memcached_engine_lib_path`

Command-Line Format	`--daemon-memcached-engine-lib-path=dir_name`
System Variable	`daemon_memcached_engine_lib_path`
Scope	Global
Dynamic	No
Type	Directory name
Default Value	`NULL`

The path of the directory containing the shared library that implements the `InnoDB memcached` plugin. The default value is NULL, representing the MySQL plugin directory. You should not need to modify this parameter unless specifying a `memcached` plugin for a different storage engine that is located outside of the MySQL plugin directory.

For more information, see Section 14.20.3, “Setting Up the InnoDB memcached Plugin”.

- `daemon_memcached_option`

Command-Line Format	`--daemon-memcached-option=options`
System Variable	`daemon_memcached_option`
Scope	Global
Dynamic	No
Type	String
Default Value	

Used to pass space-separated memcached options to the underlying `memcached` memory object caching daemon on startup. For example, you might change the port that `memcached` listens on, reduce the maximum number of simultaneous connections, change the maximum memory size for a key-value pair, or enable debugging messages for the error log.

See Section 14.20.3, “Setting Up the InnoDB memcached Plugin” for usage details. For information about `memcached` options, refer to the `memcached` man page.

- `daemon_memcached_r_batch_size`

Command-Line Format	`--daemon-memcached-r-batch-size=#`
System Variable	`daemon_memcached_r_batch_size`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`1`

Specifies how many `memcached` read operations (`get` operations) to perform before doing a `COMMIT` to start a new transaction. Counterpart of `daemon_memcached_w_batch_size`.

This value is set to 1 by default, so that any changes made to the table through SQL statements are immediately visible to `memcached` operations. You might increase it to reduce the overhead from frequent commits on a system where the underlying table is only being accessed through the `memcached` interface. If you set the value too large, the amount of undo or redo data could impose some storage overhead, as with any long-running transaction.

For more information, see Section 14.20.3, “Setting Up the InnoDB memcached Plugin”.

- `daemon_memcached_w_batch_size`

Command-Line Format	`--daemon-memcached-w-batch-size=#`
System Variable	`daemon_memcached_w_batch_size`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`1`

Specifies how many `memcached` write operations, such as `add`, `set`, and `incr`, to perform before doing a `COMMIT` to start a new transaction. Counterpart of `daemon_memcached_r_batch_size`.

This value is set to 1 by default, on the assumption that data being stored is important to preserve in case of an outage and should immediately be committed. When storing non-critical data, you might increase this value to reduce the overhead from frequent commits; but then the last *N*-1 uncommitted write operations could be lost if an unexpected exit occurs.

For more information, see Section 14.20.3, “Setting Up the InnoDB memcached Plugin”.

- `ignore_builtin_innodb`

Command-Line Format	`--ignore-builtin-innodb[={OFF|ON}]`
Deprecated	Yes
System Variable	`ignore_builtin_innodb`
Scope	Global
Dynamic	No
Type	Boolean

In earlier versions of MySQL, enabling this variable caused the server to behave as if the built-in `InnoDB` were not present, which enabled the `InnoDB Plugin` to be used instead. In MySQL 5.6, `InnoDB` is the default storage engine and `InnoDB Plugin` is not used, so this variable has no effect. As of MySQL 5.6.5, it is ignored.

- `innodb_adaptive_flushing`

Command-Line Format	`--innodb-adaptive-flushing[={OFF|ON}]`
System Variable	`innodb_adaptive_flushing`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`ON`

Specifies whether to dynamically adjust the rate of flushing dirty pages in the `InnoDB` buffer pool based on the workload. Adjusting the flush rate dynamically is intended to avoid bursts of I/O activity. This setting is enabled by default. See Section 14.8.3.4, “Configuring Buffer Pool Flushing” for more information. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_adaptive_flushing_lwm`

Command-Line Format	`--innodb-adaptive-flushing-lwm=#`
System Variable	`innodb_adaptive_flushing_lwm`

Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`10`
Minimum Value	`0`
Maximum Value	`70`

Defines the low water mark representing percentage of redo log capacity at which adaptive flushing is enabled. For more information, see Section 14.8.3.4, “Configuring Buffer Pool Flushing”.

- `innodb_adaptive_hash_index`

Command-Line Format	`--innodb-adaptive-hash-index[={OFF|ON}]`
System Variable	`innodb_adaptive_hash_index`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`ON`

Whether the `InnoDB` adaptive hash index is enabled or disabled. It may be desirable, depending on your workload, to dynamically enable or disable adaptive hash indexing to improve query performance. Because the adaptive hash index may not be useful for all workloads, conduct benchmarks with it both enabled and disabled, using realistic workloads. See Section 14.5.3, “Adaptive Hash Index” for details.

This variable is enabled by default. You can modify this parameter using the `SET GLOBAL` statement, without restarting the server. Changing the setting at runtime requires privileges sufficient to set global system variables. See Section 5.1.8.1, “System Variable Privileges”. You can also use `--skip-innodb-adaptive-hash-index` at server startup to disable it.

Disabling the adaptive hash index empties the hash table immediately. Normal operations can continue while the hash table is emptied, and executing queries that were using the hash table access the index B-trees directly instead. When the adaptive hash index is re-enabled, the hash table is populated again during normal operation.

- `innodb_adaptive_max_sleep_delay`

Command-Line Format	`--innodb-adaptive-max-sleep-delay=#`
System Variable	`innodb_adaptive_max_sleep_delay`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`150000`
Minimum Value	`0`
Maximum Value	`1000000`

Permits `InnoDB` to automatically adjust the value of `innodb_thread_sleep_delay` up or down according to the current workload. Any nonzero value enables automated, dynamic adjustment of the `innodb_thread_sleep_delay` value, up to the maximum value specified in the `innodb_adaptive_max_sleep_delay` option. The value represents the number of microseconds.

This option can be useful in busy systems, with greater than 16 InnoDB threads. (In practice, it is most valuable for MySQL systems with hundreds or thousands of simultaneous connections.)

For more information, see Section 14.8.5, “Configuring Thread Concurrency for InnoDB”.

- innodb_additional_mem_pool_size

Command-Line Format	--innodb-additional-mem-pool-size=#
Deprecated	Yes
System Variable	innodb_additional_mem_pool_size
Scope	Global
Dynamic	No
Type	Integer
Default Value	8388608
Minimum Value	2097152
Maximum Value	4294967295

The size in bytes of a memory pool InnoDB uses to store data dictionary information and other internal data structures. The more tables you have in your application, the more memory you allocate here. If InnoDB runs out of memory in this pool, it starts to allocate memory from the operating system and writes warning messages to the MySQL error log. The default value is 8MB.

This variable relates to the InnoDB internal memory allocator, which is unused if innodb_use_sys_malloc is enabled. For more information, see Section 14.8.4, “Configuring the Memory Allocator for InnoDB”.

As of MySQL 5.6.3, innodb_additional_mem_pool_size is deprecated; expect it to be removed in a future MySQL release.

- innodb_api_bk_commit_interval

Command-Line Format	--innodb-api-bk-commit-interval=#
System Variable	innodb_api_bk_commit_interval
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	5
Minimum Value	1
Maximum Value	1073741824

How often to auto-commit idle connections that use the InnoDB memcached interface, in seconds. For more information, see Section 14.20.5.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”.

- innodb_api_disable_rowlock

Command-Line Format	--innodb-api-disable-rowlock[={OFF|ON}]
System Variable	innodb_api_disable_rowlock
Scope	Global
Dynamic	No
Type	Boolean

Default Value	OFF

Use this option to disable row locks when InnoDB memcached performs DML operations. By default, innodb_api_disable_rowlock is disabled, which means that memcached requests row locks for get and set operations. When innodb_api_disable_rowlock is enabled, memcached requests a table lock instead of row locks.

innodb_api_disable_rowlock is not dynamic. It must be specified on the mysqld command line or entered in the MySQL configuration file. Configuration takes effect when the plugin is installed, which occurs when the MySQL server is started.

For more information, see Section 14.20.5.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”.

- innodb_api_enable_binlog

Command-Line Format	--innodb-api-enable-binlog[={OFF|ON}]
System Variable	innodb_api_enable_binlog
Scope	Global
Dynamic	No
Type	Boolean
Default Value	OFF

Lets you use the InnoDB memcached plugin with the MySQL binary log. For more information, see Enabling the InnoDB memcached Binary Log.

- innodb_api_enable_mdl

Command-Line Format	--innodb-api-enable-mdl[={OFF|ON}]
System Variable	innodb_api_enable_mdl
Scope	Global
Dynamic	No
Type	Boolean
Default Value	OFF

Locks the table used by the InnoDB memcached plugin, so that it cannot be dropped or altered by DDL through the SQL interface. For more information, see Section 14.20.5.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”.

- innodb_api_trx_level

Command-Line Format	--innodb-api-trx-level=#
System Variable	innodb_api_trx_level
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	0

Controls the transaction isolation level on queries processed by the memcached interface. The constants corresponding to the familiar names are:

- 0 = READ_UNCOMMITTED

- 1 = `READ COMMITTED`
- 2 = `REPEATABLE READ`
- 3 = `SERIALIZABLE`

For more information, see Section 14.20.5.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”.

- `innodb_autoextend_increment`

Command-Line Format	`--innodb-autoextend-increment=#`
System Variable	`innodb_autoextend_increment`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`64`
Minimum Value	`1`
Maximum Value	`1000`

The increment size (in megabytes) for extending the size of an auto-extending `InnoDB` system tablespace file when it becomes full. The default value is 64. For related information, see System Tablespace Data File Configuration, and Resizing the System Tablespace.

The `innodb_autoextend_increment` setting does not affect file-per-table tablespace files. These files are auto-extending regardless of of the `innodb_autoextend_increment` setting. The initial extensions are by small amounts, after which extensions occur in increments of 4MB.

- `innodb_autoinc_lock_mode`

Command-Line Format	`--innodb-autoinc-lock-mode=#`
System Variable	`innodb_autoinc_lock_mode`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`1`

Valid Values	`0`
	`1`
	`2`

The lock mode to use for generating auto-increment values. Permissible values are 0, 1, or 2, for traditional, consecutive, or interleaved, respectively. The default setting is 1 (consecutive). For the characteristics of each lock mode, see InnoDB AUTO_INCREMENT Lock Modes.

- `innodb_buffer_pool_dump_at_shutdown`

Command-Line Format	`--innodb-buffer-pool-dump-at-shutdown[={OFF|ON}]`
System Variable	`innodb_buffer_pool_dump_at_shutdown`
Scope	Global
Dynamic	Yes

Type	Boolean
Default Value	`OFF`

Specifies whether to record the pages cached in the `InnoDB` buffer pool when the MySQL server is shut down, to shorten the warmup process at the next restart. Typically used in combination with `innodb_buffer_pool_load_at_startup`.

For more information, see Section 14.8.3.5, “Saving and Restoring the Buffer Pool State”.

- `innodb_buffer_pool_dump_now`

Command-Line Format	`--innodb-buffer-pool-dump-now[={OFF|ON}]`
System Variable	`innodb_buffer_pool_dump_now`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

Immediately records the pages cached in the `InnoDB` buffer pool. Typically used in combination with `innodb_buffer_pool_load_now`.

For more information, see Section 14.8.3.5, “Saving and Restoring the Buffer Pool State”.

- `innodb_buffer_pool_filename`

Command-Line Format	`--innodb-buffer-pool-filename=file_name`
System Variable	`innodb_buffer_pool_filename`
Scope	Global
Dynamic	Yes
Type	File name
Default Value	`ib_buffer_pool`

Specifies the name of the file that holds the list of tablespace IDs and page IDs produced by `innodb_buffer_pool_dump_at_shutdown` or `innodb_buffer_pool_dump_now`. Tablespace IDs and page IDs are saved in the following format: `space, page_id`. By default, the file is named `ib_buffer_pool` and is located in the `InnoDB` data directory. A non-default location must be specified relative to the data directory.

A file name can be specified at runtime, using a `SET` statement:

```
SET GLOBAL innodb_buffer_pool_filename='file_name';
```

You can also specify a file name at startup, in a startup string or MySQL configuration file. When specifying a file name at startup, the file must exist or `InnoDB` returns a startup error indicating that there is no such file or directory.

For more information, see Section 14.8.3.5, “Saving and Restoring the Buffer Pool State”.

- `innodb_buffer_pool_instances`

Command-Line Format	`--innodb-buffer-pool-instances=#`
System Variable	`innodb_buffer_pool_instances`
Scope	Global
Dynamic	No

Type	Integer
Default Value (Other)	`8 (or 1 if innodb_buffer_pool_size < 1GB`
Default Value (Windows, 32-bit platforms)	`(autosized)`
Minimum Value	`1`
Maximum Value	`64`

The number of regions that the `InnoDB` buffer pool is divided into. For systems with buffer pools in the multi-gigabyte range, dividing the buffer pool into separate instances can improve concurrency, by reducing contention as different threads read and write to cached pages. Each page that is stored in or read from the buffer pool is assigned to one of the buffer pool instances randomly, using a hashing function. Each buffer pool manages its own free lists, flush lists, LRUs, and all other data structures connected to a buffer pool, and is protected by its own buffer pool mutex.

This option only takes effect when setting `innodb_buffer_pool_size` to a size of 1GB or more. The total size you specify is divided among all the buffer pools. For best efficiency, specify a combination of `innodb_buffer_pool_instances` and `innodb_buffer_pool_size` so that each buffer pool instance is at least 1GB.

Before MySQL 5.6.6, the default is 1. The default value in MySQL 5.6.6 and higher on 32-bit Windows systems depends on the value of `innodb_buffer_pool_size`, as described below:

- If `innodb_buffer_pool_size` is greater than 1.3GB, the default for `innodb_buffer_pool_instances` is `innodb_buffer_pool_size`/128MB, with individual memory allocation requests for each chunk. 1.3GB was chosen as the boundary at which there is significant risk for 32-bit Windows to be unable to allocate the contiguous address space needed for a single buffer pool.
- Otherwise, the default is 1.

On all other platforms, the default value in MySQL 5.6.6 and higher is 8 when `innodb_buffer_pool_size` is greater than or equal to 1GB. Otherwise, the default is 1.

Note

A bug in MySQL 5.6 causes `SHOW VARIABLES` to report an `innodb_buffer_pool_instances` value of 8 when `innodb_buffer_pool_size` is less than 1GB and only one buffer pool instance is present (Bug #18343670). As an alternative, you can use `SHOW ENGINE INNODB STATUS` to check the number of buffer pool instances. If there are multiple buffer pool instances, `SHOW ENGINE INNODB STATUS` output includes an `INDIVIDUAL BUFFER POOL INFO` section.

- `innodb_buffer_pool_load_abort`

Command-Line Format	`--innodb-buffer-pool-load-abort[={OFF|ON}]`
System Variable	`innodb_buffer_pool_load_abort`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

Interrupts the process of restoring `InnoDB` buffer pool contents triggered by `innodb_buffer_pool_load_at_startup` or `innodb_buffer_pool_load_now`.

For more information, see Section 14.8.3.5, “Saving and Restoring the Buffer Pool State”.

- `innodb_buffer_pool_load_at_startup`

Command-Line Format	`--innodb-buffer-pool-load-at-startup[={OFF|ON}]`
System Variable	`innodb_buffer_pool_load_at_startup`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`OFF`

Specifies that, on MySQL server startup, the `InnoDB` buffer pool is automatically warmed up by loading the same pages it held at an earlier time. Typically used in combination with `innodb_buffer_pool_dump_at_shutdown`.

For more information, see Section 14.8.3.5, “Saving and Restoring the Buffer Pool State”.

- `innodb_buffer_pool_load_now`

Command-Line Format	`--innodb-buffer-pool-load-now[={OFF|ON}]`
System Variable	`innodb_buffer_pool_load_now`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

Immediately warms up the `InnoDB` buffer pool by loading a set of data pages, without waiting for a server restart. Can be useful to bring cache memory back to a known state during benchmarking, or to ready the MySQL server to resume its normal workload after running queries for reports or maintenance.

For more information, see Section 14.8.3.5, “Saving and Restoring the Buffer Pool State”.

- `innodb_buffer_pool_size`

Command-Line Format	`--innodb-buffer-pool-size=#`
System Variable	`innodb_buffer_pool_size`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`134217728`
Minimum Value	`5242880`
Maximum Value (64-bit platforms)	`2**64-1`
Maximum Value (32-bit platforms)	`2**32-1`

The size in bytes of the buffer pool, the memory area where `InnoDB` caches table and index data. The default value is 134217728 bytes (128MB). The maximum value depends on the CPU architecture; the maximum is 4294967295 (2^{32}-1) on 32-bit systems and 18446744073709551615 (2^{64}-1) on 64-bit systems. On 32-bit systems, the CPU architecture and operating system may impose a lower practical maximum size than the stated maximum. When the size of the buffer pool is greater than 1GB, setting `innodb_buffer_pool_instances` to a value greater than 1 can improve the scalability on a busy server.

A larger buffer pool requires less disk I/O to access the same table data more than once. On a dedicated database server, you might set the buffer pool size to 80% of the machine's physical memory size. Be aware of the following potential issues when configuring buffer pool size, and be prepared to scale back the size of the buffer pool if necessary.

- Competition for physical memory can cause paging in the operating system.
- `InnoDB` reserves additional memory for buffers and control structures, so that the total allocated space is approximately 10% greater than the specified buffer pool size.
- Address space for the buffer pool must be contiguous, which can be an issue on Windows systems with DLLs that load at specific addresses.
- The time to initialize the buffer pool is roughly proportional to its size. On instances with large buffer pools, initialization time might be significant. To reduce the initialization period, you can save the buffer pool state at server shutdown and restore it at server startup. See Section 14.8.3.5, “Saving and Restoring the Buffer Pool State”.

- `innodb_change_buffer_max_size`

Command-Line Format	`--innodb-change-buffer-max-size=#`
System Variable	`innodb_change_buffer_max_size`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`25`
Minimum Value	`0`
Maximum Value	`50`

Maximum size for the `InnoDB` change buffer, as a percentage of the total size of the buffer pool. You might increase this value for a MySQL server with heavy insert, update, and delete activity, or decrease it for a MySQL server with unchanging data used for reporting. For more information, see Section 14.5.2, “Change Buffer”. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_change_buffering`

Command-Line Format	`--innodb-change-buffering=value`
System Variable	`innodb_change_buffering`
Scope	Global
Dynamic	Yes
Type	Enumeration
Default Value	`all`

Valid Values	`none`
	`inserts`
	`deletes`
	`changes`
	`purges`

	`all`

Whether `InnoDB` performs change buffering, an optimization that delays write operations to secondary indexes so that the I/O operations can be performed sequentially. Permitted values are described in the following table.

Table 14.14 Permitted Values for innodb_change_buffering

Value	Description
`none`	Do not buffer any operations.
`inserts`	Buffer insert operations.
`deletes`	Buffer delete marking operations; strictly speaking, the writes that mark index records for later deletion during a purge operation.
`changes`	Buffer inserts and delete-marking operations.
`purges`	Buffer the physical deletion operations that happen in the background.
`all`	The default. Buffer inserts, delete-marking operations, and purges.

For more information, see Section 14.5.2, “Change Buffer”. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_change_buffering_debug`

Command-Line Format	`--innodb-change-buffering-debug=#`
System Variable	`innodb_change_buffering_debug`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`0`
Maximum Value	`2`

Sets a debug flag for `InnoDB` change buffering. A value of 1 forces all changes to the change buffer. A value of 2 causes an unexpected exit at merge. A default value of 0 indicates that the change buffering debug flag is not set. This option is only available when debugging support is compiled in using the `WITH_DEBUG CMake` option.

- `innodb_checksum_algorithm`

Command-Line Format	`--innodb-checksum-algorithm=value`
System Variable	`innodb_checksum_algorithm`
Scope	Global
Dynamic	Yes
Type	Enumeration
Default Value	`innodb`

Valid Values	`innodb`
	`crc32`
	`none`

	`strict_innodb`
	`strict_crc32`
	`strict_none`

Specifies how to generate and verify the checksum stored in the disk blocks of `InnoDB` tablespaces.

`innodb_checksum_algorithm` replaces the `innodb_checksums` option in MySQL 5.6.3. The following values are provided for compatibility:

- `innodb_checksum_algorithm=innodb` is the same as `innodb_checksums=ON`
- `innodb_checksum_algorithm=none` is the same as `innodb_checksums=OFF`

To avoid conflicts, remove references to `innodb_checksums` from MySQL configuration files and startup scripts.

The value `innodb` is backward-compatible with earlier versions of MySQL. The value `crc32` uses an algorithm that is faster to compute the checksum for every modified block, and to check the checksums for each disk read. It scans blocks 32 bits at a time, which is faster than the `innodb` checksum algorithm, which scans blocks 8 bits at a time. The value `none` writes a constant value in the checksum field rather than computing a value based on the block data. The blocks in a tablespace can use a mix of old, new, and no checksum values, being updated gradually as the data is modified; once blocks in a tablespace are modified to use the `crc32` algorithm, the associated tables cannot be read by earlier versions of MySQL.

The strict form of a checksum algorithm reports an error if it encounters a valid but non-matching checksum value in a tablespace. It is recommended that you only use strict settings in a new instance, to set up tablespaces for the first time. Strict settings are somewhat faster, because they do not need to compute all checksum values during disk reads.

Note

Prior to MySQL 5.6.25, a strict mode setting for `innodb_checksum_algorithm` caused `InnoDB` to halt when encountering a *valid* but non-matching checksum. In MySQL 5.6.25 and later, only an error message is printed, and the page is accepted as valid if it has a valid `innodb`, `crc32` or `none` checksum.

The following table shows the difference between the `none`, `innodb`, and `crc32` option values, and their strict counterparts. `none`, `innodb`, and `crc32` write the specified type of checksum value into each data block, but for compatibility accept other checksum values when verifying a block during a read operation. Strict settings also accept valid checksum values but print an error message when a valid non-matching checksum value is encountered. Using the strict form can make verification faster if all `InnoDB` data files in an instance are created under an identical `innodb_checksum_algorithm` value.

Table 14.15 Permitted innodb_checksum_algorithm Values

Value	Generated checksum (when writing)	Permitted checksums (when reading)
none	A constant number.	Any of the checksums generated by `none`, `innodb`, or `crc32`.
innodb	A checksum calculated in software, using the original algorithm from `InnoDB`.	Any of the checksums generated by `none`, `innodb`, or `crc32`.

Value	Generated checksum (when writing)	Permitted checksums (when reading)
crc32	A checksum calculated using the `crc32` algorithm, possibly done with a hardware assist.	Any of the checksums generated by `none`, `innodb`, or `crc32`.
strict_none	A constant number	Any of the checksums generated by `none`, `innodb`, or `crc32`. `InnoDB` prints an error message if a valid but non-matching checksum is encountered.
strict_innodb	A checksum calculated in software, using the original algorithm from `InnoDB`.	Any of the checksums generated by `none`, `innodb`, or `crc32`. `InnoDB` prints an error message if a valid but non-matching checksum is encountered.
strict_crc32	A checksum calculated using the `crc32` algorithm, possibly done with a hardware assist.	Any of the checksums generated by `none`, `innodb`, or `crc32`. `InnoDB` prints an error message if a valid but non-matching checksum is encountered.

The default value for `innodb_checksum_algorithm` was changed from `innodb` to `crc32` in MySQL 5.6.6, but switched back to `innodb` in 5.6.7 for backward compatibility of `InnoDB` data files, and for use with MySQL Enterprise Backup. The limitations encountered included:

- `.ibd` files containing CRC32 checksums could cause problems downgrading to MySQL versions prior to 5.6.3. MySQL 5.6.3 and up recognizes either the new or old checksum values for the block as correct when reading the block from disk, ensuring that data blocks are compatible during upgrade and downgrade regardless of the algorithm setting. If data written with new checksum values is processed by a version of MySQL earlier than 5.6.3, it could be reported as corrupted.
- Versions of MySQL Enterprise Backup up to 3.8.0 do not support backing up tablespaces that use CRC32 checksums. MySQL Enterprise Backup adds CRC32 checksum support in 3.8.1, with some limitations. Refer to the MySQL Enterprise Backup 3.8.1 Change History for more information.

- `innodb_checksums`

Command-Line Format	`--innodb-checksums[={OFF|ON}]`
Deprecated	Yes
System Variable	`innodb_checksums`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`ON`

`InnoDB` can use checksum validation on all tablespace pages read from disk to ensure extra fault tolerance against hardware faults or corrupted data files. This validation is enabled by default. Under specialized circumstances (such as when running benchmarks) this safety feature can be disabled with `--skip-innodb-checksums`. You can specify the method of calculating the checksum using the `innodb_checksum_algorithm` option.

In MySQL 5.6.3 and higher, `innodb_checksums` is deprecated, replaced by `innodb_checksum_algorithm`. `innodb_checksum_algorithm=innodb` is the same as

`innodb_checksums=ON` (the default). `innodb_checksum_algorithm=none` is the same as `innodb_checksums=OFF`. Remove `innodb_checksums` options from your configuration files and startup scripts to avoid conflicts with `innodb_checksum_algorithm`: `innodb_checksums=OFF` automatically sets `innodb_checksum_algorithm=none`; `innodb_checksums=ON` is ignored and overridden by any other setting for `innodb_checksum_algorithm`.

- `innodb_cmp_per_index_enabled`

Command-Line Format	`--innodb-cmp-per-index-enabled[={OFF|ON}]`
System Variable	`innodb_cmp_per_index_enabled`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

Enables per-index compression-related statistics in the `INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX` table. Because these statistics can be expensive to gather, only enable this option on development, test, or replica instances during performance tuning related to `InnoDB` compressed tables.

For more information, see Section 21.30.6, “The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables”, and Section 14.9.4, “Monitoring InnoDB Table Compression at Runtime”.

- `innodb_commit_concurrency`

Command-Line Format	`--innodb-commit-concurrency=#`
System Variable	`innodb_commit_concurrency`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1000`

The number of threads that can commit at the same time. A value of 0 (the default) permits any number of transactions to commit simultaneously.

The value of `innodb_commit_concurrency` cannot be changed at runtime from zero to nonzero or vice versa. The value can be changed from one nonzero value to another.

- `innodb_compression_failure_threshold_pct`

Command-Line Format	`--innodb-compression-failure-threshold-pct=#`
System Variable	`innodb_compression_failure_threshold_pct`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`5`
Minimum Value	`0`

Maximum Value	100

Defines the compression failure rate threshold for a table, as a percentage, at which point MySQL begins adding padding within compressed pages to avoid expensive compression failures. When this threshold is passed, MySQL begins to leave additional free space within each new compressed page, dynamically adjusting the amount of free space up to the percentage of page size specified by `innodb_compression_pad_pct_max`. A value of zero disables the mechanism that monitors compression efficiency and dynamically adjusts the padding amount.

For more information, see Section 14.9.6, “Compression for OLTP Workloads”.

- `innodb_compression_level`

Command-Line Format	`--innodb-compression-level=#`
System Variable	`innodb_compression_level`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	6
Minimum Value	0
Maximum Value	9

Specifies the level of zlib compression to use for `InnoDB` compressed tables and indexes. A higher value lets you fit more data onto a storage device, at the expense of more CPU overhead during compression. A lower value lets you reduce CPU overhead when storage space is not critical, or you expect the data is not especially compressible.

For more information, see Section 14.9.6, “Compression for OLTP Workloads”.

- `innodb_compression_pad_pct_max`

Command-Line Format	`--innodb-compression-pad-pct-max=#`
System Variable	`innodb_compression_pad_pct_max`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	50
Minimum Value	0
Maximum Value	75

Specifies the maximum percentage that can be reserved as free space within each compressed page, allowing room to reorganize the data and modification log within the page when a compressed table or index is updated and the data might be recompressed. Only applies when `innodb_compression_failure_threshold_pct` is set to a nonzero value, and the rate of compression failures passes the cutoff point.

For more information, see Section 14.9.6, “Compression for OLTP Workloads”.

- `innodb_concurrency_tickets`

Command-Line Format	`--innodb-concurrency-tickets=#`
System Variable	`innodb_concurrency_tickets`
Scope	Global

Dynamic	Yes
Type	Integer
Default Value	5000
Minimum Value	1
Maximum Value	4294967295

Determines the number of threads that can enter InnoDB concurrently. A thread is placed in a queue when it tries to enter InnoDB if the number of threads has already reached the concurrency limit. When a thread is permitted to enter InnoDB, it is given a number of " tickets" equal to the value of innodb_concurrency_tickets, and the thread can enter and leave InnoDB freely until it has used up its tickets. After that point, the thread again becomes subject to the concurrency check (and possible queuing) the next time it tries to enter InnoDB. The default value is 5000 as of MySQL 5.6.6, 500 before that.

With a small innodb_concurrency_tickets value, small transactions that only need to process a few rows compete fairly with larger transactions that process many rows. The disadvantage of a small innodb_concurrency_tickets value is that large transactions must loop through the queue many times before they can complete, which extends the amount of time required to complete their task.

With a large innodb_concurrency_tickets value, large transactions spend less time waiting for a position at the end of the queue (controlled by innodb_thread_concurrency) and more time retrieving rows. Large transactions also require fewer trips through the queue to complete their task. The disadvantage of a large innodb_concurrency_tickets value is that too many large transactions running at the same time can starve smaller transactions by making them wait a longer time before executing.

With a nonzero innodb_thread_concurrency value, you may need to adjust the innodb_concurrency_tickets value up or down to find the optimal balance between larger and smaller transactions. The SHOW ENGINE INNODB STATUS report shows the number of tickets remaining for an executing transaction in its current pass through the queue. This data may also be obtained from the TRX_CONCURRENCY_TICKETS column of the INFORMATION_SCHEMA.INNODB_TRX table.

For more information, see Section 14.8.5, "Configuring Thread Concurrency for InnoDB".

- innodb_data_file_path

Command-Line Format	--innodb-data-file-path=file_name
System Variable	innodb_data_file_path
Scope	Global
Dynamic	No
Type	String
Default Value	ibdata1:12M:autoextend

Defines the name, size, and attributes of InnoDB system tablespace data files. If you do not specify a value for innodb_data_file_path, the default behavior is to create a single auto-extending data file, slightly larger than 12MB, named ibdata1.

The full syntax for a data file specification includes the file name, file size, autoextend attribute, and max attribute:

```
file_name:file_size[:autoextend[:max:max_file_size]]
```

File sizes are specified in kilobytes, megabytes, or gigabytes by appending K, M or G to the size value. If specifying the data file size in kilobytes, do so in multiples of 1024. Otherwise, KB values are

rounded to nearest megabyte (MB) boundary. The sum of file sizes must be, at a minimum, slightly larger than 12MB.

For additional configuration information, see System Tablespace Data File Configuration. For resizing instructions, see Resizing the System Tablespace.

- `innodb_data_home_dir`

Command-Line Format	`--innodb-data-home-dir=dir_name`
System Variable	`innodb_data_home_dir`
Scope	Global
Dynamic	No
Type	Directory name

The common part of the directory path for `InnoDB` system tablespace data files. The default value is the MySQL `data` directory. The setting is concatenated with the `innodb_data_file_path` setting. If you specify the value as an empty string, you can specify an absolute path for `innodb_data_file_path`.

A trailing slash is required when specifying a value for `innodb_data_home_dir`. For example:

```
[mysqld]
innodb_data_home_dir = /path/to/myibdata/
```

This setting does not affect the location of file-per-table tablespaces.

For related information, see Section 14.8.1, “InnoDB Startup Configuration”.

- `innodb_disable_sort_file_cache`

Command-Line Format	`--innodb-disable-sort-file-cache[={OFF|ON}]`
System Variable	`innodb_disable_sort_file_cache`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

Disables the operating system file system cache for merge-sort temporary files. The effect is to open such files with the equivalent of `O_DIRECT`.

- `innodb_doublewrite`

Command-Line Format	`--innodb-doublewrite[={OFF|ON}]`
System Variable	`innodb_doublewrite`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`ON`

When enabled (the default), `InnoDB` stores all data twice, first to the doublewrite buffer, then to the actual data files. This variable can be turned off with `--skip-innodb-doublewrite` for benchmarks or cases when top performance is needed rather than concern for data integrity or possible failures.

For related information, see Section 14.6.5, “Doublewrite Buffer”.

- `innodb_fast_shutdown`

Command-Line Format	`--innodb-fast-shutdown=#`
System Variable	`innodb_fast_shutdown`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`1`

Valid Values	`0`
	`1`
	`2`

The `InnoDB` shutdown mode. If the value is 0, `InnoDB` does a slow shutdown, a full purge and a change buffer merge before shutting down. If the value is 1 (the default), `InnoDB` skips these operations at shutdown, a process known as a fast shutdown. If the value is 2, `InnoDB` flushes its logs and shuts down cold, as if MySQL had crashed; no committed transactions are lost, but the crash recovery operation makes the next startup take longer.

The slow shutdown can take minutes, or even hours in extreme cases where substantial amounts of data are still buffered. Use the slow shutdown technique before upgrading or downgrading between MySQL major releases, so that all data files are fully prepared in case the upgrade process updates the file format.

Use `innodb_fast_shutdown=2` in emergency or troubleshooting situations, to get the absolute fastest shutdown if data is at risk of corruption.

- `innodb_fil_make_page_dirty_debug`

Command-Line Format	`--innodb-fil-make-page-dirty-debug=#`
Introduced	5.6.17
System Variable	`innodb_fil_make_page_dirty_debug`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`0`
Maximum Value	`2**32-1`

By default, setting `innodb_fil_make_page_dirty_debug` to the ID of a tablespace immediately dirties the first page of the tablespace. If `innodb_saved_page_number_debug` is set to a non-default value, setting `innodb_fil_make_page_dirty_debug` dirties the specified page. The `innodb_fil_make_page_dirty_debug` option is only available if debugging support is compiled in using the `WITH_DEBUG` `CMake` option.

- `innodb_file_format`

Command-Line Format	`--innodb-file-format=value`
System Variable	`innodb_file_format`
Scope	Global
Dynamic	Yes
Type	String

Default Value	Antelope
Valid Values	Antelope Barracuda

Enables an InnoDB file format for file-per-table tablespaces. Supported file formats are Antelope and Barracuda. Antelope is the original InnoDB file format, which supports REDUNDANT and COMPACT row formats for InnoDB tables. Barracuda is the newer file format, which supports COMPRESSED and DYNAMIC row formats.

COMPRESSED and DYNAMIC row formats enable important storage features for InnoDB tables. See Section 14.11, “InnoDB Row Formats”.

To create tables that use COMPRESSED or DYNAMIC row format, the Barracuda file format and innodb_file_per_table must be enabled.

Changing the innodb_file_format setting does not affect the file format of existing InnoDB tablespace files.

For more information, see Section 14.10, “InnoDB File-Format Management”.

- innodb_file_format_check

Command-Line Format	--innodb-file-format-check[={OFF|ON}]
System Variable	innodb_file_format_check
Scope	Global
Dynamic	No
Type	Boolean
Default Value	ON

This variable can be set to 1 or 0 at server startup to enable or disable whether InnoDB checks the file format tag in the system tablespace (for example, Antelope or Barracuda). If the tag is checked and is higher than that supported by the current version of InnoDB, an error occurs and InnoDB does not start. If the tag is not higher, InnoDB sets the value of innodb_file_format_max to the file format tag.

Note

Despite the default value sometimes being displayed as ON or OFF, always use the numeric values 1 or 0 to turn this option on or off in your configuration file or command line string.

For more information, see Section 14.10.2.1, “Compatibility Check When InnoDB Is Started”.

- innodb_file_format_max

Command-Line Format	--innodb-file-format-max=value
System Variable	innodb_file_format_max
Scope	Global
Dynamic	Yes
Type	String
Default Value	Antelope
Valid Values	Antelope

	`Barracuda`

At server startup, `InnoDB` sets the value of this variable to the file format tag in the system tablespace (for example, `Antelope` or `Barracuda`). If the server creates or opens a table with a "higher" file format, it sets the value of `innodb_file_format_max` to that format.

For related information, see Section 14.10, "InnoDB File-Format Management".

- `innodb_file_per_table`

Command-Line Format	`--innodb-file-per-table[={OFF|ON}]`
System Variable	`innodb_file_per_table`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`ON`

When `innodb_file_per_table` is enabled, tables are created in file-per-table tablespaces. When disabled, tables are created in the system tablespace. For information about file-per-table tablespaces, see Section 14.6.3.2, "File-Per-Table Tablespaces". For information about the `InnoDB` system tablespace, see Section 14.6.3.1, "The System Tablespace".

The `innodb_file_per_table` variable can be configured at runtime using a `SET GLOBAL` statement, specified on the command line at startup, or specified in an option file. Configuration at runtime requires privileges sufficient to set global system variables (see Section 5.1.8.1, "System Variable Privileges") and immediately affects the operation of all connections.

When a table that resides in a file-per-table tablespace is truncated or dropped, the freed space is returned to the operating system. Truncating or dropping a table that resides in the system tablespace only frees space in the system tablespace. Freed space in the system tablespace can be used again for `InnoDB` data but is not returned to the operating system, as system tablespace data files never shrink.

When `innodb_file_per_table` is enabled, a table-copying `ALTER TABLE` operation on a table that resides in the system tablespace implicitly re-creates the table in a file-per-table tablespace. To prevent this from occurring, disable `innodb_file_per_table` before executing table-copying `ALTER TABLE` operations on tables that reside in the system tablespace.

- `innodb_flush_log_at_timeout`

Command-Line Format	`--innodb-flush-log-at-timeout=#`
System Variable	`innodb_flush_log_at_timeout`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`1`
Minimum Value	`1`
Maximum Value	`2700`

Write and flush the logs every *`N`* seconds. `innodb_flush_log_at_timeout` allows the timeout period between flushes to be increased in order to reduce flushing and avoid impacting performance of binary log group commit. Prior to the introduction of this configuration option, flushing frequency was once per second. The default setting for `innodb_flush_log_at_timeout` is also once per second.

- `innodb_flush_log_at_trx_commit`

Command-Line Format	`--innodb-flush-log-at-trx-commit=#`
System Variable	`innodb_flush_log_at_trx_commit`
Scope	Global
Dynamic	Yes
Type	Enumeration
Default Value	`1`

Valid Values	`0`
	`1`
	`2`

Controls the balance between strict ACID compliance for commit operations and higher performance that is possible when commit-related I/O operations are rearranged and done in batches. You can achieve better performance by changing the default value but then you can lose transactions in a crash.

- The default setting of 1 is required for full ACID compliance. Logs are written and flushed to disk at each transaction commit.
- With a setting of 0, logs are written and flushed to disk once per second. Transactions for which logs have not been flushed can be lost in a crash.
- With a setting of 2, logs are written after each transaction commit and flushed to disk once per second. Transactions for which logs have not been flushed can be lost in a crash.
- For settings 0 and 2, once-per-second flushing is not 100% guaranteed. Flushing may occur more frequently due to DDL changes and other internal `InnoDB` activities that cause logs to be flushed independently of the `innodb_flush_log_at_trx_commit` setting, and sometimes less frequently due to scheduling issues. If logs are flushed once per second, up to one second of transactions can be lost in a crash. If logs are flushed more or less frequently than once per second, the amount of transactions that can be lost varies accordingly.
- Log flushing frequency is controlled by `innodb_flush_log_at_timeout`, which allows you to set log flushing frequency to *`N`* seconds (where *`N`* is `1 ... 2700`, with a default value of 1). However, any unexpected `mysqld` process exit can erase up to *`N`* seconds of transactions.
- DDL changes and other internal `InnoDB` activities flush the log independently of the `innodb_flush_log_at_trx_commit` setting.
- `InnoDB` crash recovery works regardless of the `innodb_flush_log_at_trx_commit` setting. Transactions are either applied entirely or erased entirely.

For durability and consistency in a replication setup that uses `InnoDB` with transactions:

- If binary logging is enabled, set `sync_binlog=1`.
- Always set `innodb_flush_log_at_trx_commit=1`.

Caution

Many operating systems and some disk hardware fool the flush-to-disk operation. They may tell `mysqld` that the flush has taken place, even though it has not. In this case, the durability of transactions is not guaranteed even with the recommended settings, and in the worst case, a power outage can corrupt `InnoDB` data. Using a battery-backed disk cache in the SCSI disk

controller or in the disk itself speeds up file flushes, and makes the operation safer. You can also try to disable the caching of disk writes in hardware caches.

- `innodb_flush_method`

Command-Line Format	`--innodb-flush-method=value`
System Variable	`innodb_flush_method`
Scope	Global
Dynamic	No
Type	String
Default Value	`NULL`

Valid Values (Windows)	`async_unbuffered`
	`normal`
	`unbuffered`

Valid Values (Unix)	`fsync`
	`O_DSYNC`
	`littlesync`
	`nosync`
	`O_DIRECT`
	`O_DIRECT_NO_FSYNC`

Defines the method used to flush data to `InnoDB` data files and log files, which can affect I/O throughput.

If `innodb_flush_method` is set to `NULL` on a Unix-like system, the `fsync` option is used by default. If `innodb_flush_method` is set to `NULL` on Windows, the `async_unbuffered` option is used by default.

The `innodb_flush_method` options for Unix-like systems include:

- `fsync`: `InnoDB` uses the `fsync()` system call to flush both the data and log files. `fsync` is the default setting.
- `O_DSYNC`: `InnoDB` uses `O_SYNC` to open and flush the log files, and `fsync()` to flush the data files. `InnoDB` does not use `O_DSYNC` directly because there have been problems with it on many varieties of Unix.
- `littlesync`: This option is used for internal performance testing and is currently unsupported. Use at your own risk.
- `nosync`: This option is used for internal performance testing and is currently unsupported. Use at your own risk.
- `O_DIRECT`: `InnoDB` uses `O_DIRECT` (or `directio()` on Solaris) to open the data files, and uses `fsync()` to flush both the data and log files. This option is available on some GNU/Linux versions, FreeBSD, and Solaris.

- `O_DIRECT_NO_FSYNC`: `InnoDB` uses `O_DIRECT` during flushing I/O, but skips the `fsync()` system call after each write operation.

  This setting is not suitable for file systems such as XFS and EXT4, which require an `fsync()` system call to synchronize file system metadata changes. If you are not sure whether your file system requires an `fsync()` system call to synchronize file system metadata changes, use `O_DIRECT` instead.

  Data loss is possible if redo log files and data files reside on different storage devices, and an unexpected exit occurs before data file writes are flushed from a device cache that is not battery-backed. If you use or intend to use different storage devices for redo log files and data files, and your data files reside on a device with a cache that is not battery-backed, use `O_DIRECT` instead.

The `innodb_flush_method` options for Windows systems include:

- `async_unbuffered`: `InnoDB` uses Windows asynchronous I/O and non-buffered I/O. `async_unbuffered` is the default setting on Windows systems.

  Running MySQL server on a 4K sector hard drive on Windows is not supported with `async_unbuffered`. The workaround is to use `innodb_flush_method=normal`.

- `normal`: `InnoDB` uses simulated asynchronous I/O and buffered I/O. This option is used for internal performance testing and is currently unsupported. Use at your own risk.

- `unbuffered`: `InnoDB` uses simulated asynchronous I/O and non-buffered I/O. This option is used for internal performance testing and is currently unsupported. Use at your own risk.

How each setting affects performance depends on hardware configuration and workload. Benchmark your particular configuration to decide which setting to use, or whether to keep the default setting. Examine the `Innodb_data_fsyncs` status variable to see the overall number of `fsync()` calls for each setting. The mix of read and write operations in your workload can affect how a setting performs. For example, on a system with a hardware RAID controller and battery-backed write cache, `O_DIRECT` can help to avoid double buffering between the `InnoDB` buffer pool and the operating system file system cache. On some systems where `InnoDB` data and log files are located on a SAN, the default value or `O_DSYNC` might be faster for a read-heavy workload with mostly `SELECT` statements. Always test this parameter with hardware and workload that reflect your production environment. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_flush_neighbors`

Command-Line Format	`--innodb-flush-neighbors=#`
System Variable	`innodb_flush_neighbors`
Scope	Global
Dynamic	Yes
Type	Enumeration
Default Value	`1`

Valid Values	`0`
	
	`1`

	2

Specifies whether flushing a page from the InnoDB buffer pool also flushes other dirty pages in the same extent.

- A setting of 0 disables innodb_flush_neighbors. Dirty pages in the same extent are not flushed.
- The default setting of 1 flushes contiguous dirty pages in the same extent.
- A setting of 2 flushes dirty pages in the same extent.

When the table data is stored on a traditional HDD storage device, flushing such neighbor pages in one operation reduces I/O overhead (primarily for disk seek operations) compared to flushing individual pages at different times. For table data stored on SSD, seek time is not a significant factor and you can turn this setting off to spread out write operations. For related information, see Section 14.8.3.4, “Configuring Buffer Pool Flushing”.

- innodb_flushing_avg_loops

Command-Line Format	--innodb-flushing-avg-loops=#
System Variable	innodb_flushing_avg_loops
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	30
Minimum Value	1
Maximum Value	1000

Number of iterations for which InnoDB keeps the previously calculated snapshot of the flushing state, controlling how quickly adaptive flushing responds to changing workloads. Increasing the value makes the rate of flush operations change smoothly and gradually as the workload changes. Decreasing the value makes adaptive flushing adjust quickly to workload changes, which can cause spikes in flushing activity if the workload increases and decreases suddenly.

For related information, see Section 14.8.3.4, “Configuring Buffer Pool Flushing”.

- innodb_force_load_corrupted

Command-Line Format	--innodb-force-load-corrupted[={OFF|ON}]
System Variable	innodb_force_load_corrupted
Scope	Global
Dynamic	No
Type	Boolean
Default Value	OFF

Permits InnoDB to load tables at startup that are marked as corrupted. Use only during troubleshooting, to recover data that is otherwise inaccessible. When troubleshooting is complete, disable this setting and restart the server.

- innodb_force_recovery

Command-Line Format	--innodb-force-recovery=#

System Variable	innodb_force_recovery
Scope	Global
Dynamic	No
Type	Integer
Default Value	0
Minimum Value	0
Maximum Value	6

The crash recovery mode, typically only changed in serious troubleshooting situations. Possible values are from 0 to 6. For the meanings of these values and important information about innodb_force_recovery, see Section 14.21.2, “Forcing InnoDB Recovery”.

Warning

Only set this variable to a value greater than 0 in an emergency situation so that you can start InnoDB and dump your tables. As a safety measure, InnoDB prevents INSERT, UPDATE, or DELETE operations when innodb_force_recovery is greater than 0. As of 5.6.15, an innodb_force_recovery setting of 4 or greater places InnoDB into read-only mode.

These restrictions may cause replication administration commands to fail with an error because replication settings such as relay_log_info_repository=TABLE and master_info_repository=TABLE store information in InnoDB tables.

- innodb_ft_aux_table

System Variable	innodb_ft_aux_table
Scope	Global
Dynamic	Yes
Type	String

Specifies the qualified name of an InnoDB table containing a FULLTEXT index. This variable is intended for diagnostic purposes and can only be set at runtime. For example:

```
SET GLOBAL innodb_ft_aux_table = 'test/t1';
```

After you set this variable to a name in the format *db_name/table_name*, the INFORMATION_SCHEMA tables INNODB_FT_INDEX_TABLE, INNODB_FT_INDEX_CACHE, INNODB_FT_CONFIG, INNODB_FT_DELETED, and INNODB_FT_BEING_DELETED show information about the search index for the specified table.

For more information, see Section 14.15.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”.

- innodb_ft_cache_size

Command-Line Format	--innodb-ft-cache-size=#
System Variable	innodb_ft_cache_size
Scope	Global
Dynamic	No
Type	Integer
Default Value	8000000

Minimum Value	1600000
Maximum Value	80000000

The memory allocated, in bytes, for the InnoDB FULLTEXT search index cache, which holds a parsed document in memory while creating an InnoDB FULLTEXT index. Index inserts and updates are only committed to disk when the innodb_ft_cache_size size limit is reached. innodb_ft_cache_size defines the cache size on a per table basis. To set a global limit for all tables, see innodb_ft_total_cache_size.

For more information, see InnoDB Full-Text Index Cache.

- innodb_ft_enable_diag_print

Command-Line Format	--innodb-ft-enable-diag-print[={OFF|ON}]
System Variable	innodb_ft_enable_diag_print
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	OFF

Whether to enable additional full-text search (FTS) diagnostic output. This option is primarily intended for advanced FTS debugging and is not of interest to most users. Output is printed to the error log and includes information such as:

- FTS index sync progress (when the FTS cache limit is reached). For example:

```
FTS SYNC for table test, deleted count: 100 size: 10000 bytes
SYNC words: 100
```

- FTS optimize progress. For example:

```
FTS start optimize test
FTS_OPTIMIZE: optimize "mysql"
FTS_OPTIMIZE: processed "mysql"
```

- FTS index build progress. For example:

```
Number of doc processed: 1000
```

- For FTS queries, the query parsing tree, word weight, query processing time, and memory usage are printed. For example:

```
FTS Search Processing time: 1 secs: 100 millisec: row(s) 10000
Full Search Memory: 245666 (bytes),  Row: 10000
```

- innodb_ft_enable_stopword

Command-Line Format	--innodb-ft-enable-stopword[={OFF|ON}]
System Variable	innodb_ft_enable_stopword
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	ON

Specifies that a set of stopwords is associated with an InnoDB FULLTEXT index at the time the index is created. If the innodb_ft_user_stopword_table option is set, the stopwords are taken

from that table. Else, if the `innodb_ft_server_stopword_table` option is set, the stopwords are taken from that table. Otherwise, a built-in set of default stopwords is used.

For more information, see Section 12.10.4, “Full-Text Stopwords”.

- `innodb_ft_max_token_size`

Command-Line Format	`--innodb-ft-max-token-size=#`
System Variable	`innodb_ft_max_token_size`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`84`
Minimum Value	`10`
Maximum Value (≥ 5.6.14)	`84`
Maximum Value (≤ 5.6.13)	`252`

Maximum character length of words that are stored in an

InnoDB

`FULLTEXT` index. Setting a limit on this value reduces the size of the index, thus speeding up queries, by omitting long keywords or arbitrary collections of letters that are not real words and are not likely to be search terms.

For more information, see Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”.

- `innodb_ft_min_token_size`

Command-Line Format	`--innodb-ft-min-token-size=#`
System Variable	`innodb_ft_min_token_size`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`3`
Minimum Value	`0`
Maximum Value	`16`

Minimum length of words that are stored in an `InnoDB` `FULLTEXT` index. Increasing this value reduces the size of the index, thus speeding up queries, by omitting common words that are unlikely to be significant in a search context, such as the English words “a” and “to”. For content using a CJK (Chinese, Japanese, Korean) character set, specify a value of 1.

For more information, see Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”.

- `innodb_ft_num_word_optimize`

Command-Line Format	`--innodb-ft-num-word-optimize=#`
System Variable	`innodb_ft_num_word_optimize`
Scope	Global
Dynamic	Yes
Type	Integer

Default Value	2000
Minimum Value	1000
Maximum Value	10000

Number of words to process during each OPTIMIZE TABLE operation on an InnoDB FULLTEXT index. Because a bulk insert or update operation to a table containing a full-text search index could require substantial index maintenance to incorporate all changes, you might do a series of OPTIMIZE TABLE statements, each picking up where the last left off.

For more information, see Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”.

- innodb_ft_result_cache_limit

Command-Line Format	--innodb-ft-result-cache-limit=#
Introduced	5.6.13
System Variable	innodb_ft_result_cache_limit
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	2000000000
Minimum Value	1000000
Maximum Value (Windows, ≥ 5.6.13, ≤ 5.6.16)	2**32-1
Maximum Value (Unix, 64-bit platforms, ≥ 5.6.13, ≤ 5.6.16)	2**64-1
Maximum Value (Unix, 32-bit platforms, ≥ 5.6.13, ≤ 5.6.16)	2**32-1
Maximum Value (≥ 5.6.17)	2**32-1

The InnoDB full-text search query result cache limit (defined in bytes) per full-text search query or per thread. Intermediate and final InnoDB full-text search query results are handled in memory. Use innodb_ft_result_cache_limit to place a size limit on the full-text search query result cache to avoid excessive memory consumption in case of very large InnoDB full-text search query results (millions or hundreds of millions of rows, for example). Memory is allocated as required when a full-text search query is processed. If the result cache size limit is reached, an error is returned indicating that the query exceeds the maximum allowed memory.

As of MySQL 5.6.17, the maximum value of innodb_ft_result_cache_limit for all platform types and bit sizes is 2**32-1.

- innodb_ft_server_stopword_table

Command-Line Format	--innodb-ft-server-stopword-table=db_name/table_name
System Variable	innodb_ft_server_stopword_table
Scope	Global
Dynamic	Yes
Type	String

Default Value	NULL

This option is used to specify your own InnoDB FULLTEXT index stopword list for all InnoDB tables. To configure your own stopword list for a specific InnoDB table, use innodb_ft_user_stopword_table.

Set innodb_ft_server_stopword_table to the name of the table containing a list of stopwords, in the format *db_name/table_name*.

The stopword table must exist before you configure innodb_ft_server_stopword_table. innodb_ft_enable_stopword must be enabled and innodb_ft_server_stopword_table option must be configured before you create the FULLTEXT index.

The stopword table must be an InnoDB table, containing a single VARCHAR column named value.

For more information, see Section 12.10.4, “Full-Text Stopwords”.

- innodb_ft_sort_pll_degree

Command-Line Format	--innodb-ft-sort-pll-degree=#
System Variable	innodb_ft_sort_pll_degree
Scope	Global
Dynamic	No
Type	Integer
Default Value	2
Minimum Value	1
Maximum Value	32

Number of threads used in parallel to index and tokenize text in an InnoDB FULLTEXT index when building a search index.

For related information, see Section 14.6.2.3, “InnoDB FULLTEXT Indexes”, and innodb_sort_buffer_size.

- innodb_ft_total_cache_size

Command-Line Format	--innodb-ft-total-cache-size=#
Introduced	5.6.13
System Variable	innodb_ft_total_cache_size
Scope	Global
Dynamic	No
Type	Integer
Default Value	640000000
Minimum Value	32000000
Maximum Value	1600000000

The total memory allocated, in bytes, for the InnoDB full-text search index cache for all tables. Creating numerous tables, each with a FULLTEXT search index, could consume a significant portion of available memory. innodb_ft_total_cache_size defines a global memory limit for all full-text search indexes to help avoid excessive memory consumption. If the global limit is reached by an index operation, a forced sync is triggered.

For more information, see InnoDB Full-Text Index Cache.

- `innodb_ft_user_stopword_table`

Command-Line Format	`--innodb-ft-user-stopword-table=db_name/table_name`
System Variable	`innodb_ft_user_stopword_table`
Scope	Global, Session
Dynamic	Yes
Type	String
Default Value	`NULL`

This option is used to specify your own `InnoDB` `FULLTEXT` index stopword list on a specific table. To configure your own stopword list for all `InnoDB` tables, use `innodb_ft_server_stopword_table`.

Set `innodb_ft_user_stopword_table` to the name of the table containing a list of stopwords, in the format *`db_name/table_name`*.

The stopword table must exist before you configure `innodb_ft_user_stopword_table`. `innodb_ft_enable_stopword` must be enabled and `innodb_ft_user_stopword_table` must be configured before you create the `FULLTEXT` index.

The stopword table must be an `InnoDB` table, containing a single `VARCHAR` column named `value`.

For more information, see Section 12.10.4, “Full-Text Stopwords”.

- `innodb_io_capacity`

Command-Line Format	`--innodb-io-capacity=#`
System Variable	`innodb_io_capacity`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`200`
Minimum Value	`100`
Maximum Value (64-bit platforms)	`2**64-1`
Maximum Value (32-bit platforms)	`2**32-1`

The `innodb_io_capacity` variable defines the number of I/O operations per second (IOPS) available to `InnoDB` background tasks, such as flushing pages from the buffer pool and merging data from the change buffer.

For information about configuring the `innodb_io_capacity` variable, see Section 14.8.8, “Configuring InnoDB I/O Capacity”.

- `innodb_io_capacity_max`

Command-Line Format	`--innodb-io-capacity-max=#`
System Variable	`innodb_io_capacity_max`
Scope	Global
Dynamic	Yes
Type	Integer

Default Value	see description
Minimum Value	100
Maximum Value (Windows, 64-bit platforms)	2**32-1
Maximum Value (Unix, 64-bit platforms)	2**64-1
Maximum Value (32-bit platforms)	2**32-1

If flushing activity falls behind, InnoDB can flush more aggressively, at a higher rate of I/O operations per second (IOPS) than defined by the innodb_io_capacity variable. The innodb_io_capacity_max variable defines a maximum number of IOPS performed by InnoDB background tasks in such situations.

For information about configuring the innodb_io_capacity_max variable, see Section 14.8.8, "Configuring InnoDB I/O Capacity".

- innodb_large_prefix

Command-Line Format	--innodb-large-prefix[={OFF|ON}]
System Variable	innodb_large_prefix
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	OFF

Enable this option to allow index key prefixes longer than 767 bytes (up to 3072 bytes) for InnoDB tables that use DYNAMIC or COMPRESSED row format. (Creating such tables also requires the option values innodb_file_format=barracuda and innodb_file_per_table=true.) See Section 14.22, "InnoDB Limits" for maximums associated with index key prefixes under various settings.

For tables that use REDUNDANT or COMPACT row format, this option does not affect the permitted index key prefix length.

- innodb_limit_optimistic_insert_debug

Command-Line Format	--innodb-limit-optimistic-insert-debug=#
System Variable	innodb_limit_optimistic_insert_debug
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	0
Minimum Value	0
Maximum Value	2**32-1

Limits the number of records per B-tree page. A default value of 0 means that no limit is imposed. This option is only available if debugging support is compiled in using the WITH_DEBUG CMake option.

- `innodb_lock_wait_timeout`

Command-Line Format	`--innodb-lock-wait-timeout=#`
System Variable	`innodb_lock_wait_timeout`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`50`
Minimum Value	`1`
Maximum Value	`1073741824`

The length of time in seconds an `InnoDB` transaction waits for a row lock before giving up. The default value is 50 seconds. A transaction that tries to access a row that is locked by another `InnoDB` transaction waits at most this many seconds for write access to the row before issuing the following error:

```
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

When a lock wait timeout occurs, the current statement is rolled back (not the entire transaction). To have the entire transaction roll back, start the server with the `--innodb-rollback-on-timeout` option. See also Section 14.21.4, “InnoDB Error Handling”.

You might decrease this value for highly interactive applications or OLTP systems, to display user feedback quickly or put the update into a queue for processing later. You might increase this value for long-running back-end operations, such as a transform step in a data warehouse that waits for other large insert or update operations to finish.

`innodb_lock_wait_timeout` applies to `InnoDB` row locks only. A MySQL table lock does not happen inside `InnoDB` and this timeout does not apply to waits for table locks.

The lock wait timeout value does not apply to deadlocks, because `InnoDB` detects them immediately and rolls back one of the deadlocked transactions. See Section 14.7.5.2, “Deadlock Detection”.

`innodb_lock_wait_timeout` can be set at runtime with the `SET GLOBAL` or `SET SESSION` statement. Changing the `GLOBAL` setting requires privileges sufficient to set global system variables (see Section 5.1.8.1, “System Variable Privileges”) and affects the operation of all clients that subsequently connect. Any client can change the `SESSION` setting for `innodb_lock_wait_timeout`, which affects only that client.

- `innodb_locks_unsafe_for_binlog`

Command-Line Format	`--innodb-locks-unsafe-for-binlog[={OFF|ON}]`
Deprecated	Yes
System Variable	`innodb_locks_unsafe_for_binlog`
Scope	Global
Dynamic	No
Type	Boolean

Default Value	OFF

This variable affects how InnoDB uses gap locking for searches and index scans. innodb_locks_unsafe_for_binlog is deprecated; expect it to be removed in a future MySQL release.

Normally, InnoDB uses an algorithm called next-key locking that combines index-row locking with gap locking. InnoDB performs row-level locking in such a way that when it searches or scans a table index, it sets shared or exclusive locks on the index records it encounters. Thus, row-level locks are actually index-record locks. In addition, a next-key lock on an index record also affects the gap before the index record. That is, a next-key lock is an index-record lock plus a gap lock on the gap preceding the index record. If one session has a shared or exclusive lock on record R in an index, another session cannot insert a new index record in the gap immediately before R in the index order. See Section 14.7.1, “InnoDB Locking”.

By default, the value of innodb_locks_unsafe_for_binlog is 0 (disabled), which means that gap locking is enabled: InnoDB uses next-key locks for searches and index scans. To enable the variable, set it to 1. This causes gap locking to be disabled: InnoDB uses only index-record locks for searches and index scans.

Enabling innodb_locks_unsafe_for_binlog does not disable the use of gap locking for foreign-key constraint checking or duplicate-key checking.

The effects of enabling innodb_locks_unsafe_for_binlog are the same as setting the transaction isolation level to READ COMMITTED, with these exceptions:

- Enabling innodb_locks_unsafe_for_binlog is a global setting and affects all sessions, whereas the isolation level can be set globally for all sessions, or individually per session.
- innodb_locks_unsafe_for_binlog can be set only at server startup, whereas the isolation level can be set at startup or changed at runtime.

READ COMMITTED therefore offers finer and more flexible control than innodb_locks_unsafe_for_binlog. For more information about the effect of isolation level on gap locking, see Section 14.7.2.1, “Transaction Isolation Levels”.

Enabling innodb_locks_unsafe_for_binlog may cause phantom problems because other sessions can insert new rows into the gaps when gap locking is disabled. Suppose that there is an index on the id column of the child table and that you want to read and lock all rows from the table having an identifier value larger than 100, with the intention of updating some column in the selected rows later:

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

The query scans the index starting from the first record where the id is greater than 100. If the locks set on the index records in that range do not lock out inserts made in the gaps, another session can insert a new row into the table. Consequently, if you were to execute the same SELECT again within the same transaction, you would see a new row in the result set returned by the query. This also means that if new items are added to the database, InnoDB does not guarantee serializability. Therefore, if innodb_locks_unsafe_for_binlog is enabled, InnoDB guarantees at most an

isolation level of `READ COMMITTED`. (Conflict serializability is still guaranteed.) For more information about phantoms, see Section 14.7.4, “Phantom Rows”.

Enabling `innodb_locks_unsafe_for_binlog` has additional effects:

- For `UPDATE` or `DELETE` statements, `InnoDB` holds locks only for rows that it updates or deletes. Record locks for nonmatching rows are released after MySQL has evaluated the `WHERE` condition. This greatly reduces the probability of deadlocks, but they can still happen.
- For `UPDATE` statements, if a row is already locked, `InnoDB` performs a “semi-consistent” read, returning the latest committed version to MySQL so that MySQL can determine whether the row matches the `WHERE` condition of the `UPDATE`. If the row matches (must be updated), MySQL reads the row again and this time `InnoDB` either locks it or waits for a lock on it.

Consider the following example, beginning with this table:

```
CREATE TABLE t (a INT NOT NULL, b INT) ENGINE = InnoDB;
INSERT INTO t VALUES (1,2),(2,3),(3,2),(4,3),(5,2);
COMMIT;
```

In this case, table has no indexes, so searches and index scans use the hidden clustered index for record locking (see Section 14.6.2.1, “Clustered and Secondary Indexes”).

Suppose that one client performs an `UPDATE` using these statements:

```
SET autocommit = 0;
UPDATE t SET b = 5 WHERE b = 3;
```

Suppose also that a second client performs an `UPDATE` by executing these statements following those of the first client:

```
SET autocommit = 0;
UPDATE t SET b = 4 WHERE b = 2;
```

As `InnoDB` executes each `UPDATE`, it first acquires an exclusive lock for each row, and then determines whether to modify it. If `InnoDB` does not modify the row and `innodb_locks_unsafe_for_binlog` is enabled, it releases the lock. Otherwise, `InnoDB` retains the lock until the end of the transaction. This affects transaction processing as follows.

If `innodb_locks_unsafe_for_binlog` is disabled, the first `UPDATE` acquires x-locks and does not release any of them:

```
x-lock(1,2); retain x-lock
x-lock(2,3); update(2,3) to (2,5); retain x-lock
x-lock(3,2); retain x-lock
x-lock(4,3); update(4,3) to (4,5); retain x-lock
x-lock(5,2); retain x-lock
```

The second `UPDATE` blocks as soon as it tries to acquire any locks (because the first update has retained locks on all rows), and does not proceed until the first `UPDATE` commits or rolls back:

```
x-lock(1,2); block and wait for first UPDATE to commit or roll back
```

If `innodb_locks_unsafe_for_binlog` is enabled, the first `UPDATE` acquires x-locks and releases those for rows that it does not modify:

```
x-lock(1,2); unlock(1,2)
x-lock(2,3); update(2,3) to (2,5); retain x-lock
x-lock(3,2); unlock(3,2)
x-lock(4,3); update(4,3) to (4,5); retain x-lock
```

```
x-lock(5,2); unlock(5,2)
```

For the second UPDATE, InnoDB does a “semi-consistent” read, returning the latest committed version of each row to MySQL so that MySQL can determine whether the row matches the WHERE condition of the UPDATE:

```
x-lock(1,2); update(1,2) to (1,4); retain x-lock
x-lock(2,3); unlock(2,3)
x-lock(3,2); update(3,2) to (3,4); retain x-lock
x-lock(4,3); unlock(4,3)
x-lock(5,2); update(5,2) to (5,4); retain x-lock
```

- innodb_log_buffer_size

Command-Line Format	--innodb-log-buffer-size=#
System Variable	innodb_log_buffer_size
Scope	Global
Dynamic	No
Type	Integer
Default Value	8388608
Minimum Value	262144
Maximum Value	4294967295

The size in bytes of the buffer that InnoDB uses to write to the log files on disk. The default value is 8MB. A large log buffer enables large transactions to run without the need to write the log to disk before the transactions commit. Thus, if you have transactions that update, insert, or delete many rows, making the log buffer larger saves disk I/O. For related information, see Memory Configuration, and Section 8.5.4, “Optimizing InnoDB Redo Logging”. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- innodb_log_checkpoint_now

Command-Line Format	--innodb-log-checkpoint-now[={OFF|ON}]
Introduced	5.6.12
System Variable	innodb_log_checkpoint_now
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	OFF

Enable this debug option to force InnoDB to write a checkpoint. This option is only available if debugging support is compiled in using the WITH_DEBUG CMake option.

- innodb_log_compressed_pages

Command-Line Format	--innodb-log-compressed-pages[={OFF|ON}]
Introduced	5.6.11
System Variable	innodb_log_compressed_pages
Scope	Global
Dynamic	Yes
Type	Boolean

Default Value	ON

Specifies whether images of re-compressed pages are written to the redo log. Re-compression may occur when changes are made to compressed data.

innodb_log_compressed_pages is enabled by default to prevent corruption that could occur if a different version of the zlib compression algorithm is used during recovery. If you are certain that the zlib version is not subject to change, you can disable innodb_log_compressed_pages to reduce redo log generation for workloads that modify compressed data.

To measure the effect of enabling or disabling innodb_log_compressed_pages, compare redo log generation for both settings under the same workload. Options for measuring redo log generation include observing the Log sequence number (LSN) in the LOG section of SHOW ENGINE INNODB STATUS output, or monitoring Innodb_os_log_written status for the number of bytes written to the redo log files.

For related information, see Section 14.9.6, “Compression for OLTP Workloads”.

- innodb_log_file_size

Command-Line Format	--innodb-log-file-size=#
System Variable	innodb_log_file_size
Scope	Global
Dynamic	No
Type	Integer
Default Value	50331648
Minimum Value	1048576
Maximum Value	512GB / innodb_log_files_in_group

The size in bytes of each log file in a log group. The combined size of log files (innodb_log_file_size * innodb_log_files_in_group) cannot exceed a maximum value that is slightly less than 512GB. A pair of 255 GB log files, for example, approaches the limit but does not exceed it. The default value is 48MB.

Generally, the combined size of the log files should be large enough that the server can smooth out peaks and troughs in workload activity, which often means that there is enough redo log space to handle more than an hour of write activity. The larger the value, the less checkpoint flush activity is required in the buffer pool, saving disk I/O. Larger log files also make crash recovery slower.

Important

Due to Bug #69477, redo log writes for large, externally stored BLOB fields could overwrite the most recent checkpoint. To address this bug, a patch introduced in MySQL 5.6.20 limits the size of redo log BLOB writes to 10% of the redo log file size. As a result of this limit, innodb_log_file_size should be set to a value greater than 10 times the largest BLOB data size found in the rows of your tables plus the length of other variable length fields (VARCHAR, VARBINARY, and TEXT type fields).

In MySQL 5.6.22, the redo log BLOB write limit is relaxed to 10% of the *total redo log size* (innodb_log_file_size * innodb_log_files_in_group). (Bug #19498877)

For related information, see Redo Log File Configuration. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- innodb_log_files_in_group

Command-Line Format	`--innodb-log-files-in-group=#`
System Variable	`innodb_log_files_in_group`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`2`
Minimum Value	`2`
Maximum Value	`100`

The number of log files in the log group. `InnoDB` writes to the files in a circular fashion. The default (and recommended) value is 2. The location of the files is specified by `innodb_log_group_home_dir`. The combined size of log files (`innodb_log_file_size` * `innodb_log_files_in_group`) can be up to 512GB.

For related information, see Redo Log File Configuration.

- `innodb_log_group_home_dir`

Command-Line Format	`--innodb-log-group-home-dir=dir_name`
System Variable	`innodb_log_group_home_dir`
Scope	Global
Dynamic	No
Type	Directory name

The directory path to the `InnoDB` redo log files, whose number is specified by `innodb_log_files_in_group`. If you do not specify any `InnoDB` log variables, the default is to create two files named `ib_logfile0` and `ib_logfile1` in the MySQL data directory. Log file size is given by the `innodb_log_file_size` system variable.

For related information, see Redo Log File Configuration.

- `innodb_lru_scan_depth`

Command-Line Format	`--innodb-lru-scan-depth=#`
System Variable	`innodb_lru_scan_depth`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`1024`
Minimum Value	`100`
Maximum Value (64-bit platforms)	`2**64-1`
Maximum Value (32-bit platforms)	`2**32-1`

A parameter that influences the algorithms and heuristics for the flush operation for the `InnoDB` buffer pool. Primarily of interest to performance experts tuning I/O-intensive workloads. It specifies,

per buffer pool instance, how far down the buffer pool LRU page list the page cleaner thread scans looking for dirty pages to flush. This is a background operation performed once per second.

A setting smaller than the default is generally suitable for most workloads. A value that is much higher than necessary may impact performance. Only consider increasing the value if you have spare I/O capacity under a typical workload. Conversely, if a write-intensive workload saturates your I/O capacity, decrease the value, especially in the case of a large buffer pool.

When tuning `innodb_lru_scan_depth`, start with a low value and configure the setting upward with the goal of rarely seeing zero free pages. Also, consider adjusting `innodb_lru_scan_depth` when changing the number of buffer pool instances, since `innodb_lru_scan_depth` * `innodb_buffer_pool_instances` defines the amount of work performed by the page cleaner thread each second.

For related information, see Section 14.8.3.4, “Configuring Buffer Pool Flushing”. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_max_dirty_pages_pct`

Command-Line Format	`--innodb-max-dirty-pages-pct=#`
System Variable	`innodb_max_dirty_pages_pct`
Scope	Global
Dynamic	Yes
Type	Numeric
Default Value	`75`
Minimum Value	`0`
Maximum Value	`99`

`InnoDB` tries to flush data from the buffer pool so that the percentage of dirty pages does not exceed this value. Specify an integer in the range from 0 to 99. The default value is 75.

The `innodb_max_dirty_pages_pct` setting establishes a target for flushing activity. It does not affect the rate of flushing. For information about managing the rate of flushing, see Section 14.8.3.4, “Configuring Buffer Pool Flushing”.

For related information, see Section 14.8.3.4, “Configuring Buffer Pool Flushing”. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_max_dirty_pages_pct_lwm`

Command-Line Format	`--innodb-max-dirty-pages-pct-lwm=#`
System Variable	`innodb_max_dirty_pages_pct_lwm`
Scope	Global
Dynamic	Yes
Type	Numeric
Default Value	`0`
Minimum Value	`0`
Maximum Value	`99`

Defines a low water mark representing the percentage of dirty pages at which preflushing is enabled to control the dirty page ratio. The default of 0 disables the pre-flushing behavior entirely. For more information, see Section 14.8.3.4, “Configuring Buffer Pool Flushing”.

- `innodb_max_purge_lag`

Command-Line Format	`--innodb-max-purge-lag=#`
System Variable	`innodb_max_purge_lag`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`4294967295`

Defines the desired maximum purge lag. If this value is exceeded, a delay is imposed on `INSERT`, `UPDATE`, and `DELETE` operations to allow time for purge to catch up. The default value is 0, which means there is no maximum purge lag and no delay.

For more information, see Section 14.8.10, “Purge Configuration”.

- `innodb_max_purge_lag_delay`

Command-Line Format	`--innodb-max-purge-lag-delay=#`
System Variable	`innodb_max_purge_lag_delay`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`10000000`

Specifies the maximum delay in microseconds for the delay imposed when the `innodb_max_purge_lag` threshold is exceeded. The specified `innodb_max_purge_lag_delay` value is an upper limit on the delay period calculated by the `innodb_max_purge_lag` formula.

For more information, see Section 14.8.10, “Purge Configuration”.

- `innodb_mirrored_log_groups`

Has no effect. This variable is deprecated as of MySQL 5.6.11; expect it to be removed in a future MySQL release.

- `innodb_monitor_disable`

Command-Line Format	`--innodb-monitor-disable={counter|module|pattern|all}`
System Variable	`innodb_monitor_disable`
Scope	Global
Dynamic	Yes
Type	String

Disables `InnoDB` metrics counters. Counter data may be queried using the `INFORMATION_SCHEMA.INNODB_METRICS` table. For usage information, see Section 14.15.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

- `innodb_monitor_enable`

Command-Line Format	`--innodb-monitor-enable={counter|module|pattern|all}`
System Variable	`innodb_monitor_enable`
Scope	Global
Dynamic	Yes
Type	String

Enables `InnoDB` metrics counters. Counter data may be queried using the `INFORMATION_SCHEMA.INNODB_METRICS` table. For usage information, see Section 14.15.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

- `innodb_monitor_reset`

Command-Line Format	`--innodb-monitor-reset={counter|module|pattern|all}`
System Variable	`innodb_monitor_reset`
Scope	Global
Dynamic	Yes
Type	Enumeration
Default Value	`empty string`

Valid Values	`counter`
	`module`
	`pattern`
	`all`

Resets the count value for `InnoDB` metrics counters to zero. Counter data may be queried using the `INFORMATION_SCHEMA.INNODB_METRICS` table. For usage information, see Section 14.15.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

- `innodb_monitor_reset_all`

Command-Line Format	`--innodb-monitor-reset-all={counter|module|pattern|all}`
System Variable	`innodb_monitor_reset_all`
Scope	Global
Dynamic	Yes
Type	Enumeration
Default Value	`empty string`

Valid Values	`counter`
	`module`
	`pattern`
	`all`

Resets all values (minimum, maximum, and so on) for `InnoDB` metrics counters. Counter data may be queried using the `INFORMATION_SCHEMA.INNODB_METRICS` table. For usage information, see Section 14.15.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

- `innodb_numa_interleave`

Command-Line Format	`--innodb-numa-interleave[={OFF|ON}]`
Introduced	5.6.27
System Variable	`innodb_numa_interleave`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`OFF`

Enables the NUMA interleave memory policy for allocation of the `InnoDB` buffer pool. When `innodb_numa_interleave` is enabled, the NUMA memory policy is set to `MPOL_INTERLEAVE` for the `mysqld` process. After the `InnoDB` buffer pool is allocated, the NUMA memory policy is set back to `MPOL_DEFAULT`. For the `innodb_numa_interleave` option to be available, MySQL must be compiled on a NUMA-enabled Linux system.

- `innodb_old_blocks_pct`

Command-Line Format	`--innodb-old-blocks-pct=#`
System Variable	`innodb_old_blocks_pct`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`37`
Minimum Value	`5`
Maximum Value	`95`

Specifies the approximate percentage of the `InnoDB` buffer pool used for the old block sublist. The range of values is 5 to 95. The default value is 37 (that is, 3/8 of the pool). Often used in combination with `innodb_old_blocks_time`.

For more information, see Section 14.8.3.2, “Making the Buffer Pool Scan Resistant”. For information about buffer pool management, the LRU algorithm, and eviction policies, see Section 14.5.1, “Buffer Pool”.

- `innodb_old_blocks_time`

Command-Line Format	`--innodb-old-blocks-time=#`
System Variable	`innodb_old_blocks_time`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`1000`
Minimum Value	`0`
Maximum Value	`2**32-1`

Non-zero values protect against the buffer pool being filled by data that is referenced only for a brief period, such as during a full table scan. Increasing this value offers more protection against full table scans interfering with data cached in the buffer pool.

Specifies how long in milliseconds a block inserted into the old sublist must stay there after its first access before it can be moved to the new sublist. If the value is 0, a block inserted into the old sublist

moves immediately to the new sublist the first time it is accessed, no matter how soon after insertion the access occurs. If the value is greater than 0, blocks remain in the old sublist until an access occurs at least that many milliseconds after the first access. For example, a value of 1000 causes blocks to stay in the old sublist for 1 second after the first access before they become eligible to move to the new sublist.

The default value is 1000 as of MySQL 5.6.6, 0 before that.

This variable is often used in combination with `innodb_old_blocks_pct`. For more information, see Section 14.8.3.2, "Making the Buffer Pool Scan Resistant". For information about buffer pool management, the LRU algorithm, and eviction policies, see Section 14.5.1, "Buffer Pool".

- `innodb_online_alter_log_max_size`

Command-Line Format	`--innodb-online-alter-log-max-size=#`
System Variable	`innodb_online_alter_log_max_size`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`134217728`
Minimum Value	`65536`
Maximum Value	`2**64-1`

Specifies an upper limit in bytes on the size of the temporary log files used during online DDL operations for `InnoDB` tables. There is one such log file for each index being created or table being altered. This log file stores data inserted, updated, or deleted in the table during the DDL operation. The temporary log file is extended when needed by the value of `innodb_sort_buffer_size`, up to the maximum specified by `innodb_online_alter_log_max_size`. If a temporary log file exceeds the upper size limit, the `ALTER TABLE` operation fails and all uncommitted concurrent DML operations are rolled back. Thus, a large value for this option allows more DML to happen during an online DDL operation, but also extends the period of time at the end of the DDL operation when the table is locked to apply the data from the log.

- `innodb_open_files`

Command-Line Format	`--innodb-open-files=#`
System Variable	`innodb_open_files`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)
Minimum Value	`10`
Maximum Value	`4294967295`

This variable is only relevant if you use multiple `InnoDB` tablespaces. It specifies the maximum number of `.ibd` files that MySQL can keep open at one time. The minimum value is 10. As of MySQL 5.6.6, the default value is 300 if `innodb_file_per_table` is not enabled, and the higher of 300 and `table_open_cache` otherwise. Before 5.6.6, the default value is 300.

The file descriptors used for `.ibd` files are for `InnoDB` tables only. They are independent of those specified by the `open_files_limit` system variable, and do not affect the operation of the table cache. For general I/O tuning advice, see Section 8.5.8, "Optimizing InnoDB Disk I/O".

- `innodb_optimize_fulltext_only`

Command-Line Format	`--innodb-optimize-fulltext-only[={OFF|ON}]`
System Variable	`innodb_optimize_fulltext_only`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

Changes the way `OPTIMIZE TABLE` operates on `InnoDB` tables. Intended to be enabled temporarily, during maintenance operations for `InnoDB` tables with `FULLTEXT` indexes.

By default, `OPTIMIZE TABLE` reorganizes data in the clustered index of the table. When this option is enabled, `OPTIMIZE TABLE` skips the reorganization of table data, and instead processes newly added, deleted, and updated token data for `InnoDB FULLTEXT` indexes. For more information, see Optimizing InnoDB Full-Text Indexes.

- `innodb_page_size`

Command-Line Format	`--innodb-page-size=#`
System Variable	`innodb_page_size`
Scope	Global
Dynamic	No
Type	Enumeration
Default Value	`16384`

Valid Values	`4096`
	`8192`
	`16384`

Specifies the page size for `InnoDB` tablespaces. Values can be specified in bytes or kilobytes. For example, a 16 kilobyte page size value can be specified as 16384, 16KB, or 16k.

`innodb_page_size` can only be configured prior to initializing the MySQL instance and cannot be changed afterward. If no value is specified, the instance is initialized using the default page size. See Section 14.8.1, “InnoDB Startup Configuration”.

The default `16KB` page size is appropriate for a wide range of workloads, particularly for queries involving table scans and DML operations involving bulk updates. Smaller page sizes might be more efficient for OLTP workloads involving many small writes, where contention can be an issue when single pages contain many rows. Smaller pages might also be efficient with SSD storage devices, which typically use small block sizes. Keeping the `InnoDB` page size close to the storage device block size minimizes the amount of unchanged data that is rewritten to disk. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_print_all_deadlocks`

Command-Line Format	`--innodb-print-all-deadlocks[={OFF|ON}]`
System Variable	`innodb_print_all_deadlocks`
Scope	Global
Dynamic	Yes
Type	Boolean

Default Value	OFF

When this option is enabled, information about all deadlocks in InnoDB user transactions is recorded in the mysqld error log. Otherwise, you see information about only the last deadlock, using the SHOW ENGINE INNODB STATUS command. An occasional InnoDB deadlock is not necessarily an issue, because InnoDB detects the condition immediately and rolls back one of the transactions automatically. You might use this option to troubleshoot why deadlocks are occurring if an application does not have appropriate error-handling logic to detect the rollback and retry its operation. A large number of deadlocks might indicate the need to restructure transactions that issue DML or SELECT ... FOR UPDATE statements for multiple tables, so that each transaction accesses the tables in the same order, thus avoiding the deadlock condition.

For related information, see Section 14.7.5, “Deadlocks in InnoDB”.

- innodb_purge_batch_size

Command-Line Format	--innodb-purge-batch-size=#
System Variable	innodb_purge_batch_size
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	300
Minimum Value	1
Maximum Value	5000

Defines the number of undo log pages that purge parses and processes in one batch from the history list. In a multithreaded purge configuration, the coordinator purge thread divides innodb_purge_batch_size by innodb_purge_threads and assigns that number of pages to each purge thread. The innodb_purge_batch_size variable also defines the number of undo log pages that purge frees after every 128 iterations through the undo logs.

The innodb_purge_batch_size option is intended for advanced performance tuning in combination with the innodb_purge_threads setting. Most users need not change innodb_purge_batch_size from its default value.

For related information, see Section 14.8.10, “Purge Configuration”.

- innodb_purge_threads

Command-Line Format	--innodb-purge-threads=#
System Variable	innodb_purge_threads
Scope	Global
Dynamic	No
Type	Integer
Default Value	1
Minimum Value	1
Maximum Value	32

The number of background threads devoted to the InnoDB purge operation. Increasing the value creates additional purge threads, which can improve efficiency on systems where DML operations are performed on multiple tables.

For related information, see Section 14.8.10, “Purge Configuration”.

- `innodb_random_read_ahead`

Command-Line Format	`--innodb-random-read-ahead[={OFF|ON}]`
System Variable	`innodb_random_read_ahead`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

Enables the random read-ahead technique for optimizing `InnoDB` I/O.

For details about performance considerations for different types of read-ahead requests, see Section 14.8.3.3, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_read_ahead_threshold`

Command-Line Format	`--innodb-read-ahead-threshold=#`
System Variable	`innodb_read_ahead_threshold`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`56`
Minimum Value	`0`
Maximum Value	`64`

Controls the sensitivity of linear read-ahead that `InnoDB` uses to prefetch pages into the buffer pool. If `InnoDB` reads at least `innodb_read_ahead_threshold` pages sequentially from an extent (64 pages), it initiates an asynchronous read for the entire following extent. The permissible range of values is 0 to 64. As of MySQL 5.6.1, a value of 0 disables read-ahead. Prior to 5.6.1, a value of 0 would trigger a read-ahead upon reading the boundary page of a 64 page extent. For the default of 56, `InnoDB` must read at least 56 pages sequentially from an extent to initiate an asynchronous read for the following extent.

Knowing how many pages are read through the read-ahead mechanism, and how many of these pages are evicted from the buffer pool without ever being accessed, can be useful when fine-tuning the `innodb_read_ahead_threshold` setting. `SHOW ENGINE INNODB STATUS` output displays counter information from the `Innodb_buffer_pool_read_ahead` and `Innodb_buffer_pool_read_ahead_evicted` global status variables, which report the number of pages brought into the buffer pool by read-ahead requests, and the number of such pages evicted from the buffer pool without ever being accessed, respectively. The status variables report global values since the last server restart.

`SHOW ENGINE INNODB STATUS` also shows the rate at which the read-ahead pages are read and the rate at which such pages are evicted without being accessed. The per-second averages are based on the statistics collected since the last invocation of `SHOW ENGINE INNODB STATUS` and are displayed in the `BUFFER POOL AND MEMORY` section of the `SHOW ENGINE INNODB STATUS` output.

For more information, see Section 14.8.3.3, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_read_io_threads`

Command-Line Format	`--innodb-read-io-threads=#`
System Variable	`innodb_read_io_threads`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`4`
Minimum Value	`1`
Maximum Value	`64`

The number of I/O threads for read operations in `InnoDB`. Its counterpart for write threads is `innodb_write_io_threads`. For more information, see Section 14.8.6, “Configuring the Number of Background InnoDB I/O Threads”. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

Note

On Linux systems, running multiple MySQL servers (typically more than 12) with default settings for `innodb_read_io_threads`, `innodb_write_io_threads`, and the Linux `aio-max-nr` setting can exceed system limits. Ideally, increase the `aio-max-nr` setting; as a workaround, you might reduce the settings for one or both of the MySQL variables.

- `innodb_read_only`

Command-Line Format	`--innodb-read-only[={OFF|ON}]`
System Variable	`innodb_read_only`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`OFF`

Starts `InnoDB` in read-only mode. For distributing database applications or data sets on read-only media. Can also be used in data warehouses to share the same data directory between multiple instances. For more information, see Section 14.8.2, “Configuring InnoDB for Read-Only Operation”.

- `innodb_replication_delay`

Command-Line Format	`--innodb-replication-delay=#`
System Variable	`innodb_replication_delay`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`4294967295`

The replication thread delay in milliseconds on a replica server if `innodb_thread_concurrency` is reached.

- `innodb_rollback_on_timeout`

Command-Line Format	`--innodb-rollback-on-timeout[={OFF|ON}]`
System Variable	`innodb_rollback_on_timeout`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`OFF`

`InnoDB` rolls back only the last statement on a transaction timeout by default. If `--innodb-rollback-on-timeout` is specified, a transaction timeout causes `InnoDB` to abort and roll back the entire transaction (the same behavior as in MySQL 4.1).

For more information, see Section 14.21.4, “InnoDB Error Handling”.

- `innodb_rollback_segments`

Command-Line Format	`--innodb-rollback-segments=#`
System Variable	`innodb_rollback_segments`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`128`
Minimum Value	`1`
Maximum Value	`128`

Defines the number of rollback segments used by `InnoDB` for transactions that generate undo records. The number of transactions that each rollback segment supports depends on the `InnoDB` page size and the number of undo logs assigned to each transaction. For more information, see Section 14.6.7, “Undo Logs”.

This setting is appropriate for tuning performance if you observe mutex contention related to the undo logs.

Although you can increase or decrease the number of rollback segments used by `InnoDB`, the number of rollback segments physically present in the system never decreases. Thus, you might start with a low value and gradually increase it to avoid allocating rollback segments that are not required. The `innodb_rollback_segments` default and maximum value is 128.

For related information, see Section 14.3, “InnoDB Multi-Versioning”. For information about configuring separate undo tablespaces, see Section 14.6.3.3, “Undo Tablespaces”.

- `innodb_saved_page_number_debug`

Command-Line Format	`--innodb-saved-page-number-debug=#`
Introduced	5.6.17
System Variable	`innodb_saved_page_number_debug`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`0`
Maximum Value	`2**23-1`

Saves a page number. Setting the `innodb_fil_make_page_dirty_debug` option dirties the page defined by `innodb_saved_page_number_debug`. The `innodb_saved_page_number_debug` option is only available if debugging support is compiled in using the `WITH_DEBUG CMake` option.

- `innodb_sort_buffer_size`

Command-Line Format	`--innodb-sort-buffer-size=#`
System Variable	`innodb_sort_buffer_size`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`1048576`
Minimum Value	`65536`
Maximum Value	`67108864`

Specifies the size of sort buffers used to sort data during creation of an `InnoDB` index. The specified size defines the amount of data that is read into memory for internal sorting and then written out to disk. This process is referred to as a “run”. During the merge phase, pairs of buffers of the specified size are read and merged. The larger the setting, the fewer runs and merges there are.

This sort area is only used for merge sorts during index creation, not during later index maintenance operations. Buffers are deallocated when index creation completes.

The value of this option also controls the amount by which the temporary log file is extended to record concurrent DML during online DDL operations.

Before this setting was made configurable, the size was hardcoded to 1048576 bytes (1MB), which remains the default.

During an `ALTER TABLE` or `CREATE TABLE` statement that creates an index, 3 buffers are allocated, each with a size defined by this option. Additionally, auxiliary pointers are allocated to rows in the sort buffer so that the sort can run on pointers (as opposed to moving rows during the sort operation).

For a typical sort operation, a formula such as this one can be used to estimate memory consumption:

```
(6 /*FTS_NUM_AUX_INDEX*/ * (3*@@GLOBAL.innodb_sort_buffer_size)
+ 2 * number_of_partitions * number_of_secondary_indexes_created
* (@@GLOBAL.innodb_sort_buffer_size/dict_index_get_min_size(index)*/)
* 8 /*64-bit sizeof *buf->tuples*/")
```

`@@GLOBAL.innodb_sort_buffer_size/dict_index_get_min_size(index)` indicates the maximum tuples held. `2 * (@@GLOBAL.innodb_sort_buffer_size/ *dict_index_get_min_size(index)*/) * 8 /*64-bit size of *buf->tuples*/` indicates auxiliary pointers allocated.

Note

For 32-bit, multiply by 4 instead of 8.

For parallel sorts on a full-text index, multiply by the `innodb_ft_sort_pll_degree` setting:

```
(6 /*FTS_NUM_AUX_INDEX*/ * @@GLOBAL.innodb_ft_sort_pll_degree)
```

- `innodb_spin_wait_delay`

Command-Line Format	`--innodb-spin-wait-delay=#`
System Variable	`innodb_spin_wait_delay`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`6`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`2**64-1`
Maximum Value (32-bit platforms)	`2**32-1`

The maximum delay between polls for a spin lock. The low-level implementation of this mechanism varies depending on the combination of hardware and operating system, so the delay does not correspond to a fixed time interval. For more information, see Section 14.8.9, “Configuring Spin Lock Polling”.

- `innodb_stats_auto_recalc`

Command-Line Format	`--innodb-stats-auto-recalc[={OFF|ON}]`
System Variable	`innodb_stats_auto_recalc`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`ON`

Causes `InnoDB` to automatically recalculate persistent statistics after the data in a table is changed substantially. The threshold value is 10% of the rows in the table. This setting applies to tables created when the `innodb_stats_persistent` option is enabled. Automatic statistics recalculation may also be configured by specifying `STATS_PERSISTENT=1` in a `CREATE TABLE` or `ALTER TABLE` statement. The amount of data sampled to produce the statistics is controlled by the `innodb_stats_persistent_sample_pages` variable.

For more information, see Section 14.8.11.1, “Configuring Persistent Optimizer Statistics Parameters”.

- `innodb_stats_include_delete_marked`

Command-Line Format	`--innodb-stats-include-delete-marked[={OFF|ON}]`
Introduced	5.6.35
System Variable	`innodb_stats_include_delete_marked`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

By default, `InnoDB` reads uncommitted data when calculating statistics. In the case of an uncommitted transaction that deletes rows from a table, `InnoDB` excludes records that are delete-marked when calculating row estimates and index statistics, which can lead to non-

optimal execution plans for other transactions that are operating on the table concurrently using a transaction isolation level other than `READ UNCOMMITTED`. To avoid this scenario, `innodb_stats_include_delete_marked` can be enabled to ensure that `InnoDB` includes delete-marked records when calculating persistent optimizer statistics.

When `innodb_stats_include_delete_marked` is enabled, `ANALYZE TABLE` considers delete-marked records when recalculating statistics.

`innodb_stats_include_delete_marked` is a global setting that affects all `InnoDB` tables. It is only applicable to persistent optimizer statistics.

For related information, see Section 14.8.11.1, “Configuring Persistent Optimizer Statistics Parameters”.

- `innodb_stats_method`

Command-Line Format	`--innodb-stats-method=value`
System Variable	`innodb_stats_method`
Scope	Global
Dynamic	Yes
Type	Enumeration
Default Value	`nulls_equal`

Valid Values	`nulls_equal`
	`nulls_unequal`
	`nulls_ignored`

How the server treats `NULL` values when collecting statistics about the distribution of index values for `InnoDB` tables. Permitted values are `nulls_equal`, `nulls_unequal`, and `nulls_ignored`. For `nulls_equal`, all `NULL` index values are considered equal and form a single value group with a size equal to the number of `NULL` values. For `nulls_unequal`, `NULL` values are considered unequal, and each `NULL` forms a distinct value group of size 1. For `nulls_ignored`, `NULL` values are ignored.

The method used to generate table statistics influences how the optimizer chooses indexes for query execution, as described in Section 8.3.7, “InnoDB and MyISAM Index Statistics Collection”.

- `innodb_stats_on_metadata`

Command-Line Format	`--innodb-stats-on-metadata[={OFF|ON}]`
System Variable	`innodb_stats_on_metadata`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

This option only applies when optimizer statistics are configured to be non-persistent. Optimizer statistics are not persisted to disk when `innodb_stats_persistent` is disabled or when individual tables are created or altered with `STATS_PERSISTENT=0`. For more information, see Section 14.8.11.2, “Configuring Non-Persistent Optimizer Statistics Parameters”.

When `innodb_stats_on_metadata` is enabled, `InnoDB` updates non-persistent statistics when metadata statements such as `SHOW TABLE STATUS` or `SHOW INDEX` are run, or when accessing the `INFORMATION_SCHEMA.TABLES` or `INFORMATION_SCHEMA.STATISTICS` tables. (These

updates are similar to what happens for `ANALYZE TABLE`.) When disabled, `InnoDB` does not update statistics during these operations. Leaving the setting disabled can improve access speed for schemas that have a large number of tables or indexes. It can also improve the stability of execution plans for queries that involve `InnoDB` tables.

To change the setting, issue the statement `SET GLOBAL innodb_stats_on_metadata=mode`, where *`mode`* is either `ON` or `OFF` (or `1` or `0`). Changing the setting requires privileges sufficient to set global system variables (see Section 5.1.8.1, “System Variable Privileges”) and immediately affects the operation of all connections.

- `innodb_stats_persistent`

Command-Line Format	`--innodb-stats-persistent[={OFF|ON}]`
System Variable	`innodb_stats_persistent`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`ON`

Specifies whether `InnoDB` index statistics are persisted to disk. Otherwise, statistics may be recalculated frequently which can lead to variations in query execution plans. This setting is stored with each table when the table is created. You can set `innodb_stats_persistent` at the global level before creating a table, or use the `STATS_PERSISTENT` clause of the `CREATE TABLE` and `ALTER TABLE` statements to override the system-wide setting and configure persistent statistics for individual tables.

For more information, see Section 14.8.11.1, “Configuring Persistent Optimizer Statistics Parameters”.

- `innodb_stats_persistent_sample_pages`

Command-Line Format	`--innodb-stats-persistent-sample-pages=#`
System Variable	`innodb_stats_persistent_sample_pages`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`20`

The number of index pages to sample when estimating cardinality and other statistics for an indexed column, such as those calculated by `ANALYZE TABLE`. Increasing the value improves the accuracy of index statistics, which can improve the query execution plan, at the expense of increased I/O during the execution of `ANALYZE TABLE` for an `InnoDB` table. For more information, see Section 14.8.11.1, “Configuring Persistent Optimizer Statistics Parameters”.

Note

Setting a high value for `innodb_stats_persistent_sample_pages` could result in lengthy `ANALYZE TABLE` execution time. To estimate the number of database pages accessed by `ANALYZE TABLE`, see Section 14.8.11.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”.

`innodb_stats_persistent_sample_pages` only applies when `innodb_stats_persistent` is enabled for a table; when `innodb_stats_persistent` is disabled, `innodb_stats_transient_sample_pages` applies instead.

- `innodb_stats_sample_pages`

Command-Line Format	`--innodb-stats-sample-pages=#`
Deprecated	Yes
System Variable	`innodb_stats_sample_pages`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`8`
Minimum Value	`1`
Maximum Value	`2**64-1`

Deprecated. Use `innodb_stats_transient_sample_pages` instead.

- `innodb_stats_transient_sample_pages`

Command-Line Format	`--innodb-stats-transient-sample-pages=#`
System Variable	`innodb_stats_transient_sample_pages`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`8`

The number of index pages to sample when estimating cardinality and other statistics for an indexed column, such as those calculated by `ANALYZE TABLE`. The default value is 8. Increasing the value improves the accuracy of index statistics, which can improve the query execution plan, at the expense of increased I/O when opening an `InnoDB` table or recalculating statistics. For more information, see Section 14.8.11.2, “Configuring Non-Persistent Optimizer Statistics Parameters”.

Note

Setting a high value for `innodb_stats_transient_sample_pages` could result in lengthy `ANALYZE TABLE` execution time. To estimate the number of database pages accessed by `ANALYZE TABLE`, see Section 14.8.11.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”.

`innodb_stats_transient_sample_pages` only applies when `innodb_stats_persistent` is disabled for a table; when `innodb_stats_persistent` is enabled, `innodb_stats_persistent_sample_pages` applies instead. Takes the place of `innodb_stats_sample_pages`. For more information, see Section 14.8.11.2, “Configuring Non-Persistent Optimizer Statistics Parameters”.

- `innodb_status_output`

Command-Line Format	`--innodb-status-output[={OFF|ON}]`
Introduced	5.6.16
System Variable	`innodb_status_output`
Scope	Global
Dynamic	Yes
Type	Boolean

Default Value	OFF

Enables or disables periodic output for the standard InnoDB Monitor. Also used in combination with innodb_status_output_locks to enable or disable periodic output for the InnoDB Lock Monitor. For more information, see Section 14.17.2, “Enabling InnoDB Monitors”.

- innodb_status_output_locks

Command-Line Format	--innodb-status-output-locks[={OFF|ON}]
Introduced	5.6.16
System Variable	innodb_status_output_locks
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	OFF

Enables or disables the InnoDB Lock Monitor. When enabled, the InnoDB Lock Monitor prints additional information about locks in SHOW ENGINE INNODB STATUS output and in periodic output printed to the MySQL error log. Periodic output for the InnoDB Lock Monitor is printed as part of the standard InnoDB Monitor output. The standard InnoDB Monitor must therefore be enabled for the InnoDB Lock Monitor to print data to the MySQL error log periodically. For more information, see Section 14.17.2, “Enabling InnoDB Monitors”.

- innodb_strict_mode

Command-Line Format	--innodb-strict-mode[={OFF|ON}]
System Variable	innodb_strict_mode
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	OFF

When innodb_strict_mode is enabled, InnoDB returns errors rather than warnings for certain conditions.

Strict mode helps guard against ignored typos and syntax errors in SQL, or other unintended consequences of various combinations of operational modes and SQL statements. When innodb_strict_mode is enabled, InnoDB raises error conditions in certain cases, rather than issuing a warning and processing the specified statement (perhaps with unintended behavior). This is analogous to sql_mode in MySQL, which controls what SQL syntax MySQL accepts, and determines whether it silently ignores errors, or validates input syntax and data values.

The innodb_strict_mode setting affects the handling of syntax errors for CREATE TABLE, ALTER TABLE and CREATE INDEX statements. innodb_strict_mode also enables a record size check, so that an INSERT or UPDATE never fails due to the record being too large for the selected page size.

Oracle recommends enabling innodb_strict_mode when using ROW_FORMAT and KEY_BLOCK_SIZE clauses in CREATE TABLE, ALTER TABLE, and CREATE INDEX statements. When innodb_strict_mode is disabled, InnoDB ignores conflicting clauses and creates the table or index with only a warning in the message log. The resulting table might have different characteristics than intended, such as lack of compression support when attempting to create a

compressed table. When innodb_strict_mode is enabled, such problems generate an immediate error and the table or index is not created.

You can enable or disable innodb_strict_mode on the command line when starting mysqld, or in a MySQL configuration file. You can also enable or disable innodb_strict_mode at runtime with the statement SET [GLOBAL|SESSION] innodb_strict_mode=*mode*, where *mode* is either ON or OFF. Changing the GLOBAL setting requires privileges sufficient to set global system variables (see Section 5.1.8.1, "System Variable Privileges") and affects the operation of all clients that subsequently connect. Any client can change the SESSION setting for innodb_strict_mode, and the setting affects only that client.

- innodb_support_xa

Command-Line Format	--innodb-support-xa[={OFF|ON}]
System Variable	innodb_support_xa
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	ON

Enables InnoDB support for two-phase commit in XA transactions, causing an extra disk flush for transaction preparation. The XA mechanism is used internally and is essential for any server that has its binary log turned on and is accepting changes to its data from more than one thread. If you disable innodb_support_xa, transactions can be written to the binary log in a different order than the live database is committing them, which can produce different data when the binary log is replayed in disaster recovery or on a replica. Do not disable innodb_support_xa on a replication source server unless you have an unusual setup where only one thread is able to change data.

For a server that is accepting data changes from only one thread, it is safe and recommended to disable this option to improve performance for InnoDB tables. For example, you can turn it off on replicas where only the replication SQL thread is changing data.

You can also disable this option if you do not need it for safe binary logging or replication, and you also do not use an external XA transaction manager.

- innodb_sync_array_size

Command-Line Format	--innodb-sync-array-size=#
System Variable	innodb_sync_array_size
Scope	Global
Dynamic	No
Type	Integer
Default Value	1
Minimum Value	1
Maximum Value	1024

Defines the size of the mutex/lock wait array. Increasing the value splits the internal data structure used to coordinate threads, for higher concurrency in workloads with large numbers of waiting threads. This setting must be configured when the MySQL instance is starting up, and cannot be changed afterward. Increasing the value is recommended for workloads that frequently produce a large number of waiting threads, typically greater than 768.

- `innodb_sync_spin_loops`

Command-Line Format	`--innodb-sync-spin-loops=#`
System Variable	`innodb_sync_spin_loops`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`30`
Minimum Value	`0`
Maximum Value	`4294967295`

The number of times a thread waits for an `InnoDB` mutex to be freed before the thread is suspended.

- `innodb_table_locks`

Command-Line Format	`--innodb-table-locks[={OFF|ON}]`
System Variable	`innodb_table_locks`
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	`ON`

If `autocommit = 0`, `InnoDB` honors `LOCK TABLES`; MySQL does not return from `LOCK TABLES ... WRITE` until all other threads have released all their locks to the table. The default value of `innodb_table_locks` is 1, which means that `LOCK TABLES` causes InnoDB to lock a table internally if `autocommit = 0`.

`innodb_table_locks = 0` has no effect for tables locked explicitly with `LOCK TABLES ... WRITE`. It does have an effect for tables locked for read or write by `LOCK TABLES ... WRITE` implicitly (for example, through triggers) or by `LOCK TABLES ... READ`.

For related information, see Section 14.7, “InnoDB Locking and Transaction Model”.

- `innodb_thread_concurrency`

Command-Line Format	`--innodb-thread-concurrency=#`
System Variable	`innodb_thread_concurrency`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1000`

Defines the maximum number of threads permitted inside of `InnoDB`. A value of 0 (the default) is interpreted as infinite concurrency (no limit). This variable is intended for performance tuning on high concurrency systems.

`InnoDB` tries to keep the number of threads inside `InnoDB` less than or equal to the `innodb_thread_concurrency` limit. Once the limit is reached, additional threads are placed into

a “First In, First Out” (FIFO) queue for waiting threads. Threads waiting for locks are not counted in the number of concurrently executing threads.

The correct setting depends on workload and computing environment. Consider setting this variable if your MySQL instance shares CPU resources with other applications or if your workload or number of concurrent users is growing. Test a range of values to determine the setting that provides the best performance. `innodb_thread_concurrency` is a dynamic variable, which permits experimenting with different settings on a live test system. If a particular setting performs poorly, you can quickly set `innodb_thread_concurrency` back to 0.

Use the following guidelines to help find and maintain an appropriate setting:

- If the number of concurrent user threads for a workload is consistently small and does not affect performance, set `innodb_thread_concurrency=0` (no limit).
- If your workload is consistently heavy or occasionally spikes, set an `innodb_thread_concurrency` value and adjust it until you find the number of threads that provides the best performance. For example, suppose that your system typically has 40 to 50 users, but periodically the number increases to 60, 70, or more. Through testing, you find that performance remains largely stable with a limit of 80 concurrent users. In this case, set `innodb_thread_concurrency` to 80.
- If you do not want `InnoDB` to use more than a certain number of virtual CPUs for user threads (20 virtual CPUs, for example), set `innodb_thread_concurrency` to this number (or possibly lower, depending on performance testing). If your goal is to isolate MySQL from other applications, consider binding the `mysqld` process exclusively to the virtual CPUs. Be aware, however, that exclusive binding can result in non-optimal hardware usage if the `mysqld` process is not consistently busy. In this case, you can bind the `mysqld` process to the virtual CPUs but allow other applications to use some or all of the virtual CPUs.

  Note

  From an operating system perspective, using a resource management solution to manage how CPU time is shared among applications may be preferable to binding the `mysqld` process. For example, you could assign 90% of virtual CPU time to a given application while other critical processes *are not* running, and scale that value back to 40% when other critical processes *are* running.

- In some cases, the optimal `innodb_thread_concurrency` setting can be smaller than the number of virtual CPUs.
- An `innodb_thread_concurrency` value that is too high can cause performance regression due to increased contention on system internals and resources.
- Monitor and analyze your system regularly. Changes to workload, number of users, or computing environment may require that you adjust the `innodb_thread_concurrency` setting.

A value of 0 disables the `queries inside InnoDB` and `queries in queue` counters in the `ROW OPERATIONS` section of `SHOW ENGINE INNODB STATUS` output.

For related information, see Section 14.8.5, “Configuring Thread Concurrency for InnoDB”.

- `innodb_thread_sleep_delay`

Command-Line Format	`--innodb-thread-sleep-delay=#`
System Variable	`innodb_thread_sleep_delay`
Scope	Global
Dynamic	Yes

Type	Integer
Default Value	`10000`
Minimum Value	`0`
Maximum Value (64-bit platforms, ≤ 5.6.16)	`18446744073709551615`
Maximum Value (32-bit platforms, ≤ 5.6.16)	`4294967295`
Maximum Value (≥ 5.6.17)	`1000000`

Defines how long `InnoDB` threads sleep before joining the `InnoDB` queue, in microseconds. The default value is 10000. A value of 0 disables sleep. You can set `innodb_adaptive_max_sleep_delay` to the highest value you would allow for `innodb_thread_sleep_delay`, and `InnoDB` automatically adjusts `innodb_thread_sleep_delay` up or down depending on current thread-scheduling activity. This dynamic adjustment helps the thread scheduling mechanism to work smoothly during times when the system is lightly loaded or when it is operating near full capacity.

For more information, see Section 14.8.5, “Configuring Thread Concurrency for InnoDB”.

- `innodb_tmpdir`

Command-Line Format	`--innodb-tmpdir=dir_name`
Introduced	5.6.29
System Variable	`innodb_tmpdir`
Scope	Global, Session
Dynamic	Yes
Type	Directory name
Default Value	`NULL`

Used to define an alternate directory for temporary sort files created during online `ALTER TABLE` operations that rebuild the table.

Online `ALTER TABLE` operations that rebuild the table also create an *intermediate* table file in the same directory as the original table. The `innodb_tmpdir` option is not applicable to intermediate table files.

A valid value is any directory path other than the MySQL data directory path. If the value is NULL (the default), temporary files are created MySQL temporary directory (`$TMPDIR` on Unix, `%TEMP%` on Windows, or the directory specified by the `--tmpdir` configuration option). If a directory is specified, existence of the directory and permissions are only checked when `innodb_tmpdir` is configured using a `SET` statement. If a symlink is provided in a directory string, the symlink is resolved and stored as an absolute path. The path should not exceed 512 bytes. An online `ALTER TABLE` operation reports an error if `innodb_tmpdir` is set to an invalid directory. `innodb_tmpdir` overrides the MySQL `tmpdir` setting but only for online `ALTER TABLE` operations.

The `FILE` privilege is required to configure `innodb_tmpdir`.

The `innodb_tmpdir` option was introduced to help avoid overflowing a temporary file directory located on a `tmpfs` file system. Such overflows could occur as a result of large temporary sort files created during online `ALTER TABLE` operations that rebuild the table.

In replication environments, only consider replicating the `innodb_tmpdir` setting if all servers have the same operating system environment. Otherwise, replicating the `innodb_tmpdir` setting could result in a replication failure when running online `ALTER TABLE` operations that rebuild the table.

If server operating environments differ, it is recommended that you configure `innodb_tmpdir` on each server individually.

For more information, see Section 14.13.3, “Online DDL Space Requirements”. For information about online `ALTER TABLE` operations, see Section 14.13, “InnoDB and Online DDL”.

- `innodb_trx_purge_view_update_only_debug`

Command-Line Format	`--innodb-trx-purge-view-update-only-debug[={OFF|ON}]`
System Variable	`innodb_trx_purge_view_update_only_debug`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

Pauses purging of delete-marked records while allowing the purge view to be updated. This option artificially creates a situation in which the purge view is updated but purges have not yet been performed. This option is only available if debugging support is compiled in using the `WITH_DEBUG CMake` option.

- `innodb_trx_rseg_n_slots_debug`

Command-Line Format	`--innodb-trx-rseg-n-slots-debug=#`
System Variable	`innodb_trx_rseg_n_slots_debug`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`0`
Maximum Value	`1024`

Sets a debug flag that limits `TRX_RSEG_N_SLOTS` to a given value for the `trx_rsegf_undo_find_free` function that looks for free slots for undo log segments. This option is only available if debugging support is compiled in using the `WITH_DEBUG CMake` option.

- `innodb_undo_directory`

Command-Line Format	`--innodb-undo-directory=dir_name`
System Variable	`innodb_undo_directory`
Scope	Global
Dynamic	No
Type	Directory name
Default Value	`.`

The relative or absolute directory path where `InnoDB` creates undo tablespaces.
Typically used to place undo logs on a different storage device. Used in conjunction with

`innodb_rollback_segments` and `innodb_undo_tablespaces`. The default value of "`.`" represents the same directory where `InnoDB` creates its other log files by default.

Note

An absolute directory path must be set for embedded MySQL installations. Otherwise, the server may not be able to locate undo tablespaces that are created when the MySQL instance is initialized.

For more information, see Section 14.6.3.3, "Undo Tablespaces".

- `innodb_undo_logs`

Command-Line Format	`--innodb-undo-logs=#`
System Variable	`innodb_undo_logs`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`128`
Minimum Value	`1`
Maximum Value	`128`

Defines the number of rollback segments used by `InnoDB`. The `innodb_undo_logs` option is an alias for `innodb_rollback_segments`. For more information, see the description of `innodb_rollback_segments`.

- `innodb_undo_tablespaces`

Command-Line Format	`--innodb-undo-tablespaces=#`
System Variable	`innodb_undo_tablespaces`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`126`

The number of undo tablespaces used by `InnoDB`. The default value is 0.

Because undo logs can become large during long-running transactions, having undo logs in multiple tablespaces reduces the maximum size of any one tablespace. The undo tablespace files are created in the location defined by `innodb_undo_directory`, with names in the form of `undo`*`N`*, where *`N`* is a sequential series of integers (including leading zeros) representing the space ID. The default size of an undo tablespace file is 10MiB.

Important

`innodb_undo_tablespaces` can only be configured prior to initializing the MySQL instance and cannot be changed afterward. If no value is specified, the instance is initialized using the default setting of 0. Attempting to restart `InnoDB` with a greater number of undo tablespaces than specified when the

MySQL instance was initialized results in a startup failure and an error stating that InnoDB did not find the expected number of undo tablespaces.

For more information, see Section 14.6.3.3, “Undo Tablespaces”.

- innodb_use_native_aio

Command-Line Format	--innodb-use-native-aio[={OFF|ON}]
System Variable	innodb_use_native_aio
Scope	Global
Dynamic	No
Type	Boolean
Default Value	ON

Specifies whether to use the Linux asynchronous I/O subsystem. This variable applies to Linux systems only, and cannot be changed while the server is running. Normally, you do not need to configure this option, because it is enabled by default.

The asynchronous I/O capability that InnoDB has on Windows systems is available on Linux systems. (Other Unix-like systems continue to use synchronous I/O calls.) This feature improves the scalability of heavily I/O-bound systems, which typically show many pending reads/writes in SHOW ENGINE INNODB STATUS\G output.

Running with a large number of InnoDB I/O threads, and especially running multiple such instances on the same server machine, can exceed capacity limits on Linux systems. In this case, you may receive the following error:

```
EAGAIN: The specified maxevents exceeds the user's limit of available events.
```

You can typically address this error by writing a higher limit to /proc/sys/fs/aio-max-nr.

However, if a problem with the asynchronous I/O subsystem in the OS prevents InnoDB from starting, you can start the server with innodb_use_native_aio=0. This option may also be disabled automatically during startup if InnoDB detects a potential problem such as a combination of tmpdir location, tmpfs file system, and Linux kernel that does not support AIO on tmpfs.

For more information, see Section 14.8.7, “Using Asynchronous I/O on Linux”.

- innodb_use_sys_malloc

Command-Line Format	--innodb-use-sys-malloc[={OFF|ON}]
Deprecated	Yes
System Variable	innodb_use_sys_malloc
Scope	Global
Dynamic	No
Type	Boolean
Default Value	ON

Enables the operating system memory allocator. If disabled, InnoDB uses its own allocator. The default value is ON. For more information, see Section 14.8.4, “Configuring the Memory Allocator for InnoDB”.

innodb_use_sys_malloc is deprecated; expect it to be removed in a future MySQL release.

- `innodb_version`

  The `InnoDB` version number. Starting in MySQL 5.6.11, separate version numbering for `InnoDB` is discontinued and this value is the same the `version` number of the server.

- `innodb_write_io_threads`

Command-Line Format	`--innodb-write-io-threads=#`
System Variable	`innodb_write_io_threads`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`4`
Minimum Value	`1`
Maximum Value	`64`

  The number of I/O threads for write operations in `InnoDB`. The default value is 4. Its counterpart for read threads is `innodb_read_io_threads`. For more information, see Section 14.8.6, "Configuring the Number of Background InnoDB I/O Threads". For general I/O tuning advice, see Section 8.5.8, "Optimizing InnoDB Disk I/O".

Note

On Linux systems, running multiple MySQL servers (typically more than 12) with default settings for `innodb_read_io_threads`, `innodb_write_io_threads`, and the Linux `aio-max-nr` setting can exceed system limits. Ideally, increase the `aio-max-nr` setting; as a workaround, you might reduce the settings for one or both of the MySQL variables.

Also take into consideration the value of `sync_binlog`, which controls synchronization of the binary log to disk.

For general I/O tuning advice, see Section 8.5.8, "Optimizing InnoDB Disk I/O".

14.15 InnoDB INFORMATION_SCHEMA Tables

This section provides information and usage examples for `InnoDB INFORMATION_SCHEMA` tables.

`InnoDB INFORMATION_SCHEMA` tables provide metadata, status information, and statistics about various aspects of the `InnoDB` storage engine. You can view a list of `InnoDB INFORMATION_SCHEMA` tables by issuing a `SHOW TABLES` statement on the `INFORMATION_SCHEMA` database:

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA LIKE 'INNODB%';
```

For table definitions, see Section 21.30, "INFORMATION_SCHEMA InnoDB Tables". For general information regarding the `MySQL INFORMATION_SCHEMA` database, see Chapter 21, *INFORMATION_SCHEMA Tables*.

14.15.1 InnoDB INFORMATION_SCHEMA Tables about Compression

There are two pairs of `InnoDB INFORMATION_SCHEMA` tables about compression that can provide insight into how well compression is working overall:

- `INNODB_CMP` and `INNODB_CMP_RESET` provide information about the number of compression operations and the amount of time spent performing compression.

- `INNODB_CMPMEM` and `INNODB_CMPMEM_RESET` provide information about the way memory is allocated for compression.

14.15.1.1 INNODB_CMP and INNODB_CMP_RESET

The `INNODB_CMP` and `INNODB_CMP_RESET` tables provide status information about operations related to compressed tables, which are described in Section 14.9, “InnoDB Table Compression”. The `PAGE_SIZE` column reports the compressed page size.

These two tables have identical contents, but reading from `INNODB_CMP_RESET` resets the statistics on compression and uncompression operations. For example, if you archive the output of `INNODB_CMP_RESET` every 60 minutes, you see the statistics for each hourly period. If you monitor the output of `INNODB_CMP` (making sure never to read `INNODB_CMP_RESET`), you see the cumulative statistics since InnoDB was started.

For the table definition, see Section 21.30.4, “The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables”.

14.15.1.2 INNODB_CMPMEM and INNODB_CMPMEM_RESET

The `INNODB_CMPMEM` and `INNODB_CMPMEM_RESET` tables provide status information about compressed pages that reside in the buffer pool. Please consult Section 14.9, “InnoDB Table Compression” for further information on compressed tables and the use of the buffer pool. The `INNODB_CMP` and `INNODB_CMP_RESET` tables should provide more useful statistics on compression.

Internal Details

`InnoDB` uses a buddy allocator system to manage memory allocated to pages of various sizes, from 1KB to 16KB. Each row of the two tables described here corresponds to a single page size.

The `INNODB_CMPMEM` and `INNODB_CMPMEM_RESET` tables have identical contents, but reading from `INNODB_CMPMEM_RESET` resets the statistics on relocation operations. For example, if every 60 minutes you archived the output of `INNODB_CMPMEM_RESET`, it would show the hourly statistics. If you never read `INNODB_CMPMEM_RESET` and monitored the output of `INNODB_CMPMEM` instead, it would show the cumulative statistics since `InnoDB` was started.

For the table definition, see Section 21.30.5, “The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables”.

14.15.1.3 Using the Compression Information Schema Tables

Example 14.1 Using the Compression Information Schema Tables

The following is sample output from a database that contains compressed tables (see Section 14.9, “InnoDB Table Compression”, `INNODB_CMP`, `INNODB_CMP_PER_INDEX`, and `INNODB_CMPMEM`).

The following table shows the contents of `INFORMATION_SCHEMA.INNODB_CMP` under a light workload. The only compressed page size that the buffer pool contains is 8K. Compressing or uncompressing pages has consumed less than a second since the time the statistics were reset, because the columns `COMPRESS_TIME` and `UNCOMPRESS_TIME` are zero.

page size	compress ops	compress ops ok	compress time	uncompress ops	uncompress time
1024	0	0	0	0	0
2048	0	0	0	0	0
4096	0	0	0	0	0
8192	1048	921	0	61	0
16384	0	0	0	0	0

According to INNODB_CMPMEM, there are 6169 compressed 8KB pages in the buffer pool. The only other allocated block size is 64 bytes. The smallest PAGE_SIZE in INNODB_CMPMEM is used for block descriptors of those compressed pages for which no uncompressed page exists in the buffer pool. We see that there are 5910 such pages. Indirectly, we see that 259 (6169-5910) compressed pages also exist in the buffer pool in uncompressed form.

The following table shows the contents of INFORMATION_SCHEMA.INNODB_CMPMEM under a light workload. Some memory is unusable due to fragmentation of the memory allocator for compressed pages: SUM(PAGE_SIZE*PAGES_FREE)=6784. This is because small memory allocation requests are fulfilled by splitting bigger blocks, starting from the 16K blocks that are allocated from the main buffer pool, using the buddy allocation system. The fragmentation is this low because some allocated blocks have been relocated (copied) to form bigger adjacent free blocks. This copying of SUM(PAGE_SIZE*RELOCATION_OPS) bytes has consumed less than a second (SUM(RELOCATION_TIME)=0).

page size	pages used	pages free	relocation ops	relocation time
64	5910	0	2436	0
128	0	1	0	0
256	0	0	0	0
512	0	1	0	0
1024	0	0	0	0
2048	0	1	0	0
4096	0	1	0	0
8192	6169	0	5	0
16384	0	0	0	0

14.15.2 InnoDB INFORMATION_SCHEMA Transaction and Locking Information

Three InnoDB INFORMATION_SCHEMA tables enable you to monitor transactions and diagnose potential locking problems:

- INNODB_TRX: Provides information about every transaction currently executing inside InnoDB, including the transaction state (for example, whether it is running or waiting for a lock), when the transaction started, and the particular SQL statement the transaction is executing.
- INNODB_LOCKS: Each transaction in InnoDB that is waiting for another transaction to release a lock (INNODB_TRX.TRX_STATE is LOCK WAIT) is blocked by exactly one blocking lock request. That blocking lock request is for a row or table lock held by another transaction in an incompatible mode. A lock that blocks a transaction is always held in a mode incompatible with the mode of requested lock (read vs. write, shared vs. exclusive). The blocked transaction cannot proceed until the other transaction commits or rolls back, thereby releasing the requested lock. For every blocked transaction, INNODB_LOCKS contains one row that describes each lock the transaction has requested, and for which it is waiting. INNODB_LOCKS also contains one row for each lock that is blocking another transaction, whatever the state of the transaction that holds the lock (INNODB_TRX.TRX_STATE is RUNNING, LOCK WAIT, ROLLING BACK or COMMITTING).
- INNODB_LOCK_WAITS: This table indicates which transactions are waiting for a given lock, or for which lock a given transaction is waiting. This table contains one or more rows for each blocked transaction, indicating the lock it has requested and any locks that are blocking that request. The REQUESTED_LOCK_ID value refers to the lock requested by a transaction, and the BLOCKING_LOCK_ID value refers to the lock (held by another transaction) that prevents the first transaction from proceeding. For any given blocked transaction, all rows in INNODB_LOCK_WAITS have the same value for REQUESTED_LOCK_ID and different values for BLOCKING_LOCK_ID.

For more information about the preceding tables, see Section 21.30.25, “The INFORMATION_SCHEMA INNODB_TRX Table”, Section 21.30.13, “The INFORMATION_SCHEMA INNODB_LOCKS Table”, and Section 21.30.14, “The INFORMATION_SCHEMA INNODB_LOCK_WAITS Table”.

14.15.2.1 Using InnoDB Transaction and Locking Information

Identifying Blocking Transactions

It is sometimes helpful to identify which transaction blocks another. The tables that contain information about `InnoDB` transactions and data locks enable you to determine which transaction is waiting for another, and which resource is being requested. (For descriptions of these tables, see Section 14.15.2, “InnoDB INFORMATION_SCHEMA Transaction and Locking Information”.)

Suppose that three sessions are running concurrently. Each session corresponds to a MySQL thread, and executes one transaction after another. Consider the state of the system when these sessions have issued the following statements, but none has yet committed its transaction:

- Session A:

```
BEGIN;
SELECT a FROM t FOR UPDATE;
SELECT SLEEP(100);
```

- Session B:

```
SELECT b FROM t FOR UPDATE;
```

- Session C:

```
SELECT c FROM t FOR UPDATE;
```

In this scenario, use the following query to see which transactions are waiting and which transactions are blocking them:

```
SELECT
  r.trx_id waiting_trx_id,
  r.trx_mysql_thread_id waiting_thread,
  r.trx_query waiting_query,
  b.trx_id blocking_trx_id,
  b.trx_mysql_thread_id blocking_thread,
  b.trx_query blocking_query
FROM       information_schema.innodb_lock_waits w
INNER JOIN information_schema.innodb_trx b
  ON b.trx_id = w.blocking_trx_id
INNER JOIN information_schema.innodb_trx r
  ON r.trx_id = w.requesting_trx_id;
```

waiting trx id	waiting thread	waiting query	blocking trx id	blocking thread	blocking query
A4	6	SELECT b FROM t FOR UPDATE	A3	5	SELECT SLEEP(100)
A5	7	SELECT c FROM t FOR UPDATE	A3	5	SELECT SLEEP(100)
A5	7	SELECT c FROM t FOR UPDATE	A4	6	SELECT b FROM t FOR UPDATE

In the preceding table, you can identify sessions by the “waiting query” or “blocking query” columns. As you can see:

- Session B (trx id A4, thread 6) and Session C (trx id A5, thread 7) are both waiting for Session A (trx id A3, thread 5).
- Session C is waiting for Session B as well as Session A.

You can see the underlying data in the tables `INNODB_TRX`, `INNODB_LOCKS`, and `INNODB_LOCK_WAITS`.

The following table shows some sample contents of `INFORMATION_SCHEMA.INNODB_TRX`.

trx id	trx state	trx started
A3	RUNNING	2008-01-15 16:44:54
A4	LOCK WAIT	2008-01-15 16:45:09
A5	LOCK WAIT	2008-01-15 16:45:14

The following table shows some sample contents of `INFORMATION_SCHEMA.INNODB_LOCKS`.

lock id	lock trx id	lock mode	lock type
A3:1:3:2	A3	X	RECORD
A4:1:3:2	A4	X	RECORD
A5:1:3:2	A5	X	RECORD

The following table shows some sample contents of `INFORMATION_SCHEMA.INNODB_LOCK_WAITS`.

requesting trx id	requested lock id	blocking trx id	blocking lock id
A4	A4:1:3:2	A3	A3:1:3:2
A5	A5:1:3:2	A3	A3:1:3:2
A5	A5:1:3:2	A4	A4:1:3:2

Correlating InnoDB Transactions with MySQL Sessions

Sometimes it is useful to correlate internal `InnoDB` locking information with the session-level information maintained by MySQL. For example, you might like to know, for a given `InnoDB` transaction ID, the corresponding MySQL session ID and name of the session that may be holding a lock, and thus blocking other transactions.

The following output from the `INFORMATION_SCHEMA` tables is taken from a somewhat loaded system. As can be seen, there are several transactions running.

The following `INNODB_LOCKS` and `INNODB_LOCK_WAITS` tables show that:

- Transaction `77F` (executing an `INSERT`) is waiting for transactions `77E`, `77D`, and `77B` to commit.
- Transaction `77E` (executing an `INSERT`) is waiting for transactions `77D` and `77B` to commit.
- Transaction `77D` (executing an `INSERT`) is waiting for transaction `77B` to commit.
- Transaction `77B` (executing an `INSERT`) is waiting for transaction `77A` to commit.
- Transaction `77A` is running, currently executing `SELECT`.
- Transaction `E56` (executing an `INSERT`) is waiting for transaction `E55` to commit.
- Transaction `E55` (executing an `INSERT`) is waiting for transaction `19C` to commit.
- Transaction `19C` is running, currently executing an `INSERT`.

Note

There may be inconsistencies between queries shown in the `INFORMATION_SCHEMA PROCESSLIST` and `INNODB_TRX` tables. For an explanation, see Section 14.15.2.3, “Persistence and Consistency of InnoDB Transaction and Locking Information”.

The following table shows the contents of `INFORMATION_SCHEMA.PROCESSLIST` for a system running a heavy workload.

ID	USER	HOST	DB	COMMAND	TIME	STATE
384	root	localhost	test	Query	10	update
257	root	localhost	test	Query	3	update
130	root	localhost	test	Query	0	update
61	root	localhost	test	Query	1	update
8	root	localhost	test	Query	1	update
4	root	localhost	test	Query	0	prepar
2	root	localhost	test	Sleep	566	

The following table shows the contents of `INFORMATION_SCHEMA.INNODB_TRX` for a system running a heavy workload.

trx id	trx state	trx started	trx requested lock id	trx wait started	trx weigh
77F	LOCK WAIT	2008-01-15 13:10:16	77F	2008-01-15 13:10:16	1
77E	LOCK WAIT	2008-01-15 13:10:16	77E	2008-01-15 13:10:16	1
77D	LOCK WAIT	2008-01-15 13:10:16	77D	2008-01-15 13:10:16	1
77B	LOCK WAIT	2008-01-15 13:10:16	77B:733:12:1	2008-01-15 13:10:16	4
77A	RUNNING	2008-01-15 13:10:16	NULL	NULL	4
E56	LOCK WAIT	2008-01-15 13:10:06	E56:743:6:2	2008-01-15 13:10:06	5
E55	LOCK WAIT	2008-01-15 13:10:06	E55:743:38:2	2008-01-15 13:10:13	965
19C	RUNNING	2008-01-15 13:09:10	NULL	NULL	2900
E15	RUNNING	2008-01-15 13:08:59	NULL	NULL	5395
51D	RUNNING	2008-01-15 13:08:47	NULL	NULL	9807

The following table shows the contents of `INFORMATION_SCHEMA.INNODB_LOCK_WAITS` for a system running a heavy workload.

requesting trx id	requested lock id	blocking trx id	blocking lock id
77F	77F:806	77E	77E:806
77F	77F:806	77D	77D:806
77F	77F:806	77B	77B:806
77E	77E:806	77D	77D:806
77E	77E:806	77B	77B:806
77D	77D:806	77B	77B:806
77B	77B:733:12:1	77A	77A:733:12:1

requesting trx id	requested lock id	blocking trx id	blocking lock id
E56	E56:743:6:2	E55	E55:743:6:2
E55	E55:743:38:2	19C	19C:743:38:2

The following table shows the contents of INFORMATION_SCHEMA.INNODB_LOCKS for a system running a heavy workload.

lock id	lock trx id	lock mode	lock type	lock table	lock index	lock data
77F:806	77F	AUTO_INC	TABLE	test.t09	NULL	NULL
77E:806	77E	AUTO_INC	TABLE	test.t09	NULL	NULL
77D:806	77D	AUTO_INC	TABLE	test.t09	NULL	NULL
77B:806	77B	AUTO_INC	TABLE	test.t09	NULL	NULL
77B:733:12:1	77B	X	RECORD	test.t09	PRIMARY	supremum pseudo-record
77A:733:12:1	77A	X	RECORD	test.t09	PRIMARY	supremum pseudo-record
E56:743:6:2	E56	S	RECORD	test.t2	PRIMARY	0, 0
E55:743:6:2	E55	X	RECORD	test.t2	PRIMARY	0, 0
E55:743:38:2	E55	S	RECORD	test.t2	PRIMARY	1922, 1922
19C:743:38:2	19C	X	RECORD	test.t2	PRIMARY	1922, 1922

14.15.2.2 InnoDB Lock and Lock-Wait Information

When a transaction updates a row in a table, or locks it with SELECT FOR UPDATE, InnoDB establishes a list or queue of locks on that row. Similarly, InnoDB maintains a list of locks on a table for table-level locks. If a second transaction wants to update a row or lock a table already locked by a prior transaction in an incompatible mode, InnoDB adds a lock request for the row to the corresponding queue. For a lock to be acquired by a transaction, all incompatible lock requests previously entered into the lock queue for that row or table must be removed (which occurs when the transactions holding or requesting those locks either commit or roll back).

A transaction may have any number of lock requests for different rows or tables. At any given time, a transaction may request a lock that is held by another transaction, in which case it is blocked by that other transaction. The requesting transaction must wait for the transaction that holds the blocking lock to commit or roll back. If a transaction is not waiting for a lock, it is in a RUNNING state. If a transaction is waiting for a lock, it is in a LOCK WAIT state. (The INFORMATION_SCHEMA INNODB_TRX table indicates transaction state values.)

The INNODB_LOCKS table holds one or more rows for each LOCK WAIT transaction, indicating any lock requests that prevent its progress. This table also contains one row describing each lock in a queue of locks pending for a given row or table. The INNODB_LOCK_WAITS table shows which locks already held by a transaction are blocking locks requested by other transactions.

14.15.2.3 Persistence and Consistency of InnoDB Transaction and Locking Information

The data exposed by the transaction and locking tables (INNODB_TRX, INNODB_LOCKS, and INNODB_LOCK_WAITS) represents a glimpse into fast-changing data. This is not like user tables, where the data changes only when application-initiated updates occur. The underlying data is internal system-managed data, and can change very quickly.

For performance reasons, and to minimize the chance of misleading joins between the transaction and locking tables, InnoDB collects the required transaction and locking information into an intermediate

buffer whenever a SELECT on any of the tables is issued. This buffer is refreshed only if more than 0.1 seconds has elapsed since the last time the buffer was read. The data needed to fill the three tables is fetched atomically and consistently and is saved in this global internal buffer, forming a point-in-time “snapshot”. If multiple table accesses occur within 0.1 seconds (as they almost certainly do when MySQL processes a join among these tables), then the same snapshot is used to satisfy the query.

A correct result is returned when you join any of these tables together in a single query, because the data for the three tables comes from the same snapshot. Because the buffer is not refreshed with every query of any of these tables, if you issue separate queries against these tables within a tenth of a second, the results are the same from query to query. On the other hand, two separate queries of the same or different tables issued more than a tenth of a second apart may see different results, since the data come from different snapshots.

Because InnoDB must temporarily stall while the transaction and locking data is collected, too frequent queries of these tables can negatively impact performance as seen by other users.

As these tables contain sensitive information (at least INNODB_LOCKS.LOCK_DATA and INNODB_TRX.TRX_QUERY), for security reasons, only the users with the PROCESS privilege are allowed to SELECT from them.

As described previously, the data that fills the transaction and locking tables (INNODB_TRX, INNODB_LOCKS and INNODB_LOCK_WAITS) is fetched automatically and saved to an intermediate buffer that provides a “point-in-time” snapshot. The data across all three tables is consistent when queried from the same snapshot. However, the underlying data changes so fast that similar glimpses at other, similarly fast-changing data, may not be in synchrony. Thus, you should be careful when comparing data in the InnoDB transaction and locking tables with data in the PROCESSLIST table. The data from the PROCESSLIST table does not come from the same snapshot as the data about locking and transactions. Even if you issue a single SELECT (joining INNODB_TRX and PROCESSLIST, for example), the content of those tables is generally not consistent. INNODB_TRX may reference rows that are not present in PROCESSLIST or the currently executing SQL query of a transaction shown in INNODB_TRX.TRX_QUERY may differ from the one in PROCESSLIST.INFO.

14.15.3 InnoDB INFORMATION_SCHEMA System Tables

As of MySQL 5.6, you can extract metadata about schema objects managed by InnoDB using InnoDB INFORMATION_SCHEMA system tables. This information comes from the InnoDB internal system tables (also referred to as the InnoDB data dictionary), which cannot be queried directly like regular InnoDB tables. Traditionally, you would get this type of information using the techniques from Section 14.17, “InnoDB Monitors”, setting up InnoDB monitors and parsing the output from the SHOW ENGINE INNODB STATUS statement. The InnoDB INFORMATION_SCHEMA table interface allows you to query this data using SQL.

With the exception of INNODB_SYS_TABLESTATS, for which there is no corresponding internal system table, InnoDB INFORMATION_SCHEMA system tables are populated with data read directly from internal InnoDB system tables rather than from metadata that is cached in memory.

InnoDB INFORMATION_SCHEMA system tables include the tables listed below. INNODB_SYS_DATAFILES and INNODB_SYS_TABLESPACES were added in MySQL 5.6.6 with the introduction of support for the DATA DIRECTORY='*directory*' clause of the CREATE TABLE statement, which allows InnoDB file-per-table tablespaces (.ibd files) to be created in a location outside the MySQL data directory.

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA LIKE 'INNODB_SYS%';
+-----------------------------------------+
| Tables_in_information_schema (INNODB_SYS%) |
+-----------------------------------------+
| INNODB_SYS_DATAFILES                    |
| INNODB_SYS_TABLESTATS                   |
| INNODB_SYS_FOREIGN                      |
| INNODB_SYS_COLUMNS                      |
| INNODB_SYS_INDEXES                      |
| INNODB_SYS_FIELDS                       |
```

```
| INNODB_SYS_TABLESPACES                    |
| INNODB_SYS_FOREIGN_COLS                   |
| INNODB_SYS_TABLES                         |
+-------------------------------------------+
```

The table names are indicative of the type of data provided:

- `INNODB_SYS_TABLES` provides metadata about `InnoDB` tables, equivalent to the information in the `SYS_TABLES` table in the `InnoDB` data dictionary.
- `INNODB_SYS_COLUMNS` provides metadata about `InnoDB` table columns, equivalent to the information in the `SYS_COLUMNS` table in the `InnoDB` data dictionary.
- `INNODB_SYS_INDEXES` provides metadata about `InnoDB` indexes, equivalent to the information in the `SYS_INDEXES` table in the `InnoDB` data dictionary.
- `INNODB_SYS_FIELDS` provides metadata about the key columns (fields) of `InnoDB` indexes, equivalent to the information in the `SYS_FIELDS` table in the `InnoDB` data dictionary.
- `INNODB_SYS_TABLESTATS` provides a view of low-level status information about `InnoDB` tables that is derived from in-memory data structures. There is no corresponding internal `InnoDB` system table.
- `INNODB_SYS_DATAFILES` provides data file path information for `InnoDB` file-per-table tablespaces, equivalent to information in the `SYS_DATAFILES` table in the `InnoDB` data dictionary.
- `INNODB_SYS_TABLESPACES` provides metadata about `InnoDB` file-per-table tablespaces, equivalent to the information in the `SYS_TABLESPACES` table in the `InnoDB` data dictionary.
- `INNODB_SYS_FOREIGN` provides metadata about foreign keys defined on `InnoDB` tables, equivalent to the information in the `SYS_FOREIGN` table in the `InnoDB` data dictionary.
- `INNODB_SYS_FOREIGN_COLS` provides metadata about the columns of foreign keys that are defined on `InnoDB` tables, equivalent to the information in the `SYS_FOREIGN_COLS` table in the `InnoDB` data dictionary.

`InnoDB` `INFORMATION_SCHEMA` system tables can be joined together through fields such as `TABLE_ID`, `INDEX_ID`, and `SPACE`, allowing you to easily retrieve all available data for an object you want to study or monitor.

Refer to the `InnoDB` INFORMATION_SCHEMA documentation for information about the columns of each table.

Example 14.2 InnoDB INFORMATION_SCHEMA System Tables

This example uses a simple table (`t1`) with a single index (`i1`) to demonstrate the type of metadata found in the `InnoDB` `INFORMATION_SCHEMA` system tables.

1. Create a test database and table `t1`:

```
mysql> CREATE DATABASE test;

mysql> USE test;

mysql> CREATE TABLE t1 (
       col1 INT,
       col2 CHAR(10),
       col3 VARCHAR(10))
       ENGINE = InnoDB;

mysql> CREATE INDEX i1 ON t1(col1);
```

2. After creating the table `t1`, query `INNODB_SYS_TABLES` to locate the metadata for `test/t1`:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_TABLES WHERE NAME='test/t1' \G
```

```
*************************** 1. row ***************************
     TABLE_ID: 71
         NAME: test/t1
         FLAG: 1
       N_COLS: 6
        SPACE: 57
  FILE_FORMAT: Antelope
   ROW_FORMAT: Compact
ZIP_PAGE_SIZE: 0
...
```

Table `t1` has a `TABLE_ID` of 71. The `FLAG` field provides bit level information about table format and storage characteristics. There are six columns, three of which are hidden columns created by `InnoDB` (`DB_ROW_ID`, `DB_TRX_ID`, and `DB_ROLL_PTR`). The ID of the table's `SPACE` is 57 (a value of 0 would indicate that the table resides in the system tablespace). The `FILE_FORMAT` is Antelope, and the `ROW_FORMAT` is Compact. `ZIP_PAGE_SIZE` only applies to tables with a `Compressed` row format.

3. Using the `TABLE_ID` information from `INNODB_SYS_TABLES`, query the `INNODB_SYS_COLUMNS` table for information about the table's columns.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_COLUMNS where TABLE_ID = 71 \G
*************************** 1. row ***************************
TABLE_ID: 71
    NAME: col1
     POS: 0
   MTYPE: 6
  PRTYPE: 1027
     LEN: 4
*************************** 2. row ***************************
TABLE_ID: 71
    NAME: col2
     POS: 1
   MTYPE: 2
  PRTYPE: 524542
     LEN: 10
*************************** 3. row ***************************
TABLE_ID: 71
    NAME: col3
     POS: 2
   MTYPE: 1
  PRTYPE: 524303
     LEN: 10
```

In addition to the `TABLE_ID` and column `NAME`, `INNODB_SYS_COLUMNS` provides the ordinal position (`POS`) of each column (starting from 0 and incrementing sequentially), the column `MTYPE` or “main type” (6 = INT, 2 = CHAR, 1 = VARCHAR), the `PRTYPE` or “precise type” (a binary value with bits that represent the MySQL data type, character set code, and nullability), and the column length (`LEN`).

4. Using the `TABLE_ID` information from `INNODB_SYS_TABLES` once again, query `INNODB_SYS_INDEXES` for information about the indexes associated with table `t1`.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_INDEXES WHERE TABLE_ID = 71 \G
*************************** 1. row ***************************
INDEX_ID: 111
    NAME: GEN_CLUST_INDEX
TABLE_ID: 71
    TYPE: 1
N_FIELDS: 0
 PAGE_NO: 3
   SPACE: 57
*************************** 2. row ***************************
INDEX_ID: 112
    NAME: i1
TABLE_ID: 71
    TYPE: 0
N_FIELDS: 1
 PAGE_NO: 4
```

```
   SPACE: 57
```

INNODB_SYS_INDEXES returns data for two indexes. The first index is GEN_CLUST_INDEX, which is a clustered index created by InnoDB if the table does not have a user-defined clustered index. The second index (i1) is the user-defined secondary index.

The INDEX_ID is an identifier for the index that is unique across all databases in an instance. The TABLE_ID identifies the table that the index is associated with. The index TYPE value indicates the type of index (1 = Clustered Index, 0 = Secondary index). The N_FILEDS value is the number of fields that comprise the index. PAGE_NO is the root page number of the index B-tree, and SPACE is the ID of the tablespace where the index resides. A nonzero value indicates that the index does not reside in the system tablespace.

5. Using the INDEX_ID information from INNODB_SYS_INDEXES, query INNODB_SYS_FIELDS for information about the fields of index i1.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_FIELDS where INDEX_ID = 112 \G
*************************** 1. row ***************************
INDEX_ID: 112
    NAME: col1
     POS: 0
```

INNODB_SYS_FIELDS provides the NAME of the indexed field and its ordinal position within the index. If the index (i1) had been defined on multiple fields, INNODB_SYS_FIELDS would provide metadata for each of the indexed fields.

6. Using the SPACE information from INNODB_SYS_TABLES, query INNODB_SYS_TABLESPACES table for information about the table's tablespace.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_TABLESPACES WHERE SPACE = 57 \G
*************************** 1. row ***************************
        SPACE: 57
         NAME: test/t1
         FLAG: 0
  FILE_FORMAT: Antelope
   ROW_FORMAT: Compact or Redundant
    PAGE_SIZE: 16384
ZIP_PAGE_SIZE: 0
```

In addition to the SPACE ID of the tablespace and the NAME of the associated table, INNODB_SYS_TABLESPACES provides tablespace FLAG data, which is bit level information about tablespace format and storage characteristics. Also provided are tablespace FILE_FORMAT, ROW_FORMAT, PAGE_SIZE, and several other tablespace metadata items.

7. Using the SPACE information from INNODB_SYS_TABLES once again, query INNODB_SYS_DATAFILES for the location of the tablespace data file.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_DATAFILES WHERE SPACE = 57 \G
*************************** 1. row ***************************
SPACE: 57
 PATH: ./test/t1.ibd
```

The datafile is located in the test directory under MySQL's data directory. If a file-per-table tablespace were created in a location outside the MySQL data directory using the DATA DIRECTORY clause of the CREATE TABLE statement, the tablespace PATH would be a fully qualified directory path.

8. As a final step, insert a row into table t1 (TABLE_ID = 71) and view the data in the INNODB_SYS_TABLESTATS table. The data in this table is used by the MySQL optimizer to calculate which index to use when querying an InnoDB table. This information is derived from in-memory data structures. There is no corresponding internal InnoDB system table.

```
mysql> INSERT INTO t1 VALUES(5, 'abc', 'def');
Query OK, 1 row affected (0.06 sec)
```

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_TABLESTATS where TABLE_ID = 71 \G
*************************** 1. row ***************************
        TABLE_ID: 71
            NAME: test/t1
STATS_INITIALIZED: Initialized
        NUM_ROWS: 1
 CLUST_INDEX_SIZE: 1
 OTHER_INDEX_SIZE: 0
 MODIFIED_COUNTER: 1
          AUTOINC: 0
       REF_COUNT: 1
```

The STATS_INITIALIZED field indicates whether or not statistics have been collected for the table. NUM_ROWS is the current estimated number of rows in the table. The CLUST_INDEX_SIZE and OTHER_INDEX_SIZE fields report the number of pages on disk that store clustered and secondary indexes for the table, respectively. The MODIFIED_COUNTER value shows the number of rows modified by DML operations and cascade operations from foreign keys. The AUTOINC value is the next number to be issued for any autoincrement-based operation. There are no autoincrement columns defined on table t1, so the value is 0. The REF_COUNT value is a counter. When the counter reaches 0, it signifies that the table metadata can be evicted from the table cache.

Example 14.3 Foreign Key INFORMATION_SCHEMA System Tables

The INNODB_SYS_FOREIGN and INNODB_SYS_FOREIGN_COLS tables provide data about foreign key relationships. This example uses a parent table and child table with a foreign key relationship to demonstrate the data found in the INNODB_SYS_FOREIGN and INNODB_SYS_FOREIGN_COLS tables.

1. Create the test database with parent and child tables:

```
mysql> CREATE DATABASE test;

mysql> USE test;

mysql> CREATE TABLE parent (id INT NOT NULL,
       PRIMARY KEY (id)) ENGINE=INNODB;

mysql> CREATE TABLE child (id INT, parent_id INT,
       INDEX par_ind (parent_id),
       CONSTRAINT fk1
       FOREIGN KEY (parent_id) REFERENCES parent(id)
       ON DELETE CASCADE) ENGINE=INNODB;
```

2. After the parent and child tables are created, query INNODB_SYS_FOREIGN and locate the foreign key data for the test/child and test/parent foreign key relationship:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_FOREIGN \G
*************************** 1. row ***************************
      ID: test/fk1
FOR_NAME: test/child
REF_NAME: test/parent
  N_COLS: 1
    TYPE: 1
```

Metadata includes the foreign key ID (fk1), which is named for the CONSTRAINT that was defined on the child table. The FOR_NAME is the name of the child table where the foreign key is defined. REF_NAME is the name of the parent table (the “referenced” table). N_COLS is the number of columns in the foreign key index. TYPE is a numerical value representing bit flags that provide additional information about the foreign key column. In this case, the TYPE value is 1, which indicates that the ON DELETE CASCADE option was specified for the foreign key. See the INNODB_SYS_FOREIGN table definition for more information about TYPE values.

3. Using the foreign key ID, query INNODB_SYS_FOREIGN_COLS to view data about the columns of the foreign key.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_FOREIGN_COLS WHERE ID = 'test/fk1' \G
*************************** 1. row ***************************
```

```
          ID: test/fk1
FOR_COL_NAME: parent_id
REF_COL_NAME: id
         POS: 0
```

`FOR_COL_NAME` is the name of the foreign key column in the child table, and `REF_COL_NAME` is the name of the referenced column in the parent table. The `POS` value is the ordinal position of the key field within the foreign key index, starting at zero.

Example 14.4 Joining InnoDB INFORMATION_SCHEMA System Tables

This example demonstrates joining three `InnoDB INFORMATION_SCHEMA` system tables (`INNODB_SYS_TABLES`, `INNODB_SYS_TABLESPACES`, and `INNODB_SYS_TABLESTATS`) to gather file format, row format, page size, and index size information about tables in the employees sample database.

The following table name aliases are used to shorten the query string:

- `INFORMATION_SCHEMA.INNODB_SYS_TABLES`: a
- `INFORMATION_SCHEMA.INNODB_SYS_TABLESPACES`: b
- `INFORMATION_SCHEMA.INNODB_SYS_TABLESTATS`: c

An `IF()` control flow function is used to account for compressed tables. If a table is compressed, the index size is calculated using `ZIP_PAGE_SIZE` rather than `PAGE_SIZE`. `CLUST_INDEX_SIZE` and `OTHER_INDEX_SIZE`, which are reported in bytes, are divided by `1024*1024` to provide index sizes in megabytes (MBs). MB values are rounded to zero decimal spaces using the `ROUND()` function.

```
mysql> SELECT a.NAME, a.FILE_FORMAT, a.ROW_FORMAT,
        @page_size :=
         IF(a.ROW_FORMAT='Compressed',
          b.ZIP_PAGE_SIZE, b.PAGE_SIZE)
          AS page_size,
         ROUND((@page_size * c.CLUST_INDEX_SIZE)
          /(1024*1024)) AS pk_mb,
         ROUND((@page_size * c.OTHER_INDEX_SIZE)
          /(1024*1024)) AS secidx_mb
       FROM INFORMATION_SCHEMA.INNODB_SYS_TABLES a
       INNER JOIN INFORMATION_SCHEMA.INNODB_SYS_TABLESPACES b on a.NAME = b.NAME
       INNER JOIN INFORMATION_SCHEMA.INNODB_SYS_TABLESTATS c on b.NAME = c.NAME
       WHERE a.NAME LIKE 'employees/%'
       ORDER BY a.NAME DESC;
+------------------------+-------------+------------+-----------+-------+-----------+
| NAME                   | FILE_FORMAT | ROW_FORMAT | page_size | pk_mb | secidx_mb |
+------------------------+-------------+------------+-----------+-------+-----------+
| employees/titles       | Antelope    | Compact    |     16384 |    20 |        11 |
| employees/salaries     | Antelope    | Compact    |     16384 |    91 |        33 |
| employees/employees    | Antelope    | Compact    |     16384 |    15 |         0 |
| employees/dept_manager | Antelope    | Compact    |     16384 |     0 |         0 |
| employees/dept_emp     | Antelope    | Compact    |     16384 |    12 |        10 |
| employees/departments  | Antelope    | Compact    |     16384 |     0 |         0 |
+------------------------+-------------+------------+-----------+-------+-----------+
```

14.15.4 InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables

The following tables provide metadata for `FULLTEXT` indexes:

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA LIKE 'INNODB_FT%';
+----------------------------------------+
| Tables_in_INFORMATION_SCHEMA (INNODB_FT%) |
+----------------------------------------+
| INNODB_FT_CONFIG                       |
| INNODB_FT_BEING_DELETED                |
| INNODB_FT_DELETED                      |
| INNODB_FT_DEFAULT_STOPWORD             |
| INNODB_FT_INDEX_TABLE                  |
| INNODB_FT_INDEX_CACHE                  |
```

```
+----------------------------------------+
```

Table Overview

- `INNODB_FT_CONFIG`: Provides metadata about the `FULLTEXT` index and associated processing for an `InnoDB` table.
- Provides a snapshot of the `INNODB_FT_DELETED` table; it is used only during an `OPTIMIZE TABLE` maintenance operation. When `OPTIMIZE TABLE` is run, the `INNODB_FT_BEING_DELETED` table is emptied, and `DOC_ID` values are removed from the `INNODB_FT_DELETED` table. Because the contents of `INNODB_FT_BEING_DELETED` typically have a short lifetime, this table has limited utility for monitoring or debugging. For information about running `OPTIMIZE TABLE` on tables with `FULLTEXT` indexes, see Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”.
- `INNODB_FT_DELETED`: Stores rows that are deleted from the `FULLTEXT` index for an `InnoDB` table. To avoid expensive index reorganization during DML operations for an `InnoDB FULLTEXT` index, the information about newly deleted words is stored separately, filtered out of search results when you do a text search, and removed from the main search index only when you issue an `OPTIMIZE TABLE` statement for the `InnoDB` table.
- `INNODB_FT_DEFAULT_STOPWORD`: Holds a list of stopwords that are used by default when creating a `FULLTEXT` index on `InnoDB` tables.

  For information about the `INNODB_FT_DEFAULT_STOPWORD` table, see Section 12.10.4, “Full-Text Stopwords”.
- `INNODB_FT_INDEX_TABLE`: Provides information about the inverted index used to process text searches against the `FULLTEXT` index of an `InnoDB` table.
- `INNODB_FT_INDEX_CACHE`: Provides token information about newly inserted rows in a `FULLTEXT` index. To avoid expensive index reorganization during DML operations, the information about newly indexed words is stored separately, and combined with the main search index only when `OPTIMIZE TABLE` is run, when the server is shut down, or when the cache size exceeds a limit defined by the `innodb_ft_cache_size` or `innodb_ft_total_cache_size` system variable.

Note

With the exception of the `INNODB_FT_DEFAULT_STOPWORD` table, these tables are empty initially. Before querying any of them, set the value of the `innodb_ft_aux_table` system variable to the name (including the database name) of the table that contains the `FULLTEXT` index (for example, `test/articles`).

Example 14.5 InnoDB FULLTEXT Index INFORMATION_SCHEMA Tables

This example uses a table with a `FULLTEXT` index to demonstrate the data contained in the `FULLTEXT` index `INFORMATION_SCHEMA` tables.

1. Create a table with a `FULLTEXT` index and insert some data:

```
mysql> CREATE TABLE articles (
         id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
         title VARCHAR(200),
         body TEXT,
         FULLTEXT (title,body)
       ) ENGINE=InnoDB;

mysql> INSERT INTO articles (title,body) VALUES
       ('MySQL Tutorial','DBMS stands for DataBase ...'),
       ('How To Use MySQL Well','After you went through a ...'),
       ('Optimizing MySQL','In this tutorial we show ...'),
       ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
       ('MySQL vs. YourSQL','In the following database comparison ...'),
       ('MySQL Security','When configured properly, MySQL ...');
```

2. Set the `innodb_ft_aux_table` variable to the name of the table with the `FULLTEXT` index. If this variable is not set, the `InnoDB` `FULLTEXT` `INFORMATION_SCHEMA` tables are empty, with the exception of `INNODB_FT_DEFAULT_STOPWORD`.

```
SET GLOBAL innodb_ft_aux_table = 'test/articles';
```

3. Query the `INNODB_FT_INDEX_CACHE` table, which shows information about newly inserted rows in a `FULLTEXT` index. To avoid expensive index reorganization during DML operations, data for newly inserted rows remains in the `FULLTEXT` index cache until `OPTIMIZE TABLE` is run (or until the server is shut down or cache limits are exceeded).

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_INDEX_CACHE LIMIT 5;
+------------+--------------+-------------+-----------+--------+----------+
| WORD       | FIRST_DOC_ID | LAST_DOC_ID | DOC_COUNT | DOC_ID | POSITION |
+------------+--------------+-------------+-----------+--------+----------+
| 1001       |            5 |           5 |         1 |      5 |        0 |
| after      |            3 |           3 |         1 |      3 |       22 |
| comparison |            6 |           6 |         1 |      6 |       44 |
| configured |            7 |           7 |         1 |      7 |       20 |
| database   |            2 |           6 |         2 |      2 |       31 |
+------------+--------------+-------------+-----------+--------+----------+
```

4. Enable the `innodb_optimize_fulltext_only` system variable and run `OPTIMIZE TABLE` on the table that contains the `FULLTEXT` index. This operation flushes the contents of the `FULLTEXT` index cache to the main `FULLTEXT` index. `innodb_optimize_fulltext_only` changes the way the `OPTIMIZE TABLE` statement operates on `InnoDB` tables, and is intended to be enabled temporarily, during maintenance operations on `InnoDB` tables with `FULLTEXT` indexes.

```
mysql> SET GLOBAL innodb_optimize_fulltext_only=ON;

mysql> OPTIMIZE TABLE articles;
+---------------+----------+----------+----------+
| Table         | Op       | Msg_type | Msg_text |
+---------------+----------+----------+----------+
| test.articles | optimize | status   | OK       |
+---------------+----------+----------+----------+
```

5. Query the `INNODB_FT_INDEX_TABLE` table to view information about data in the main `FULLTEXT` index, including information about the data that was just flushed from the `FULLTEXT` index cache.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_INDEX_TABLE LIMIT 5;
+------------+--------------+-------------+-----------+--------+----------+
| WORD       | FIRST_DOC_ID | LAST_DOC_ID | DOC_COUNT | DOC_ID | POSITION |
+------------+--------------+-------------+-----------+--------+----------+
| 1001       |            5 |           5 |         1 |      5 |        0 |
| after      |            3 |           3 |         1 |      3 |       22 |
| comparison |            6 |           6 |         1 |      6 |       44 |
| configured |            7 |           7 |         1 |      7 |       20 |
| database   |            2 |           6 |         2 |      2 |       31 |
+------------+--------------+-------------+-----------+--------+----------+
```

   The `INNODB_FT_INDEX_CACHE` table is now empty since the `OPTIMIZE TABLE` operation flushed the `FULLTEXT` index cache.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_INDEX_CACHE LIMIT 5;
Empty set (0.00 sec)
```

6. Delete some records from the `test/articles` table.

```
mysql> DELETE FROM test.articles WHERE id < 4;
```

7. Query the `INNODB_FT_DELETED` table. This table records rows that are deleted from the `FULLTEXT` index. To avoid expensive index reorganization during DML operations, information about newly deleted records is stored separately, filtered out of search results when you do a text search, and removed from the main search index when you run `OPTIMIZE TABLE`.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_DELETED;
+--------+
```

```
| DOC_ID |
+--------+
|      2 |
|      3 |
|      4 |
+--------+
```

8. Run `OPTIMIZE TABLE` to remove the deleted records.

```
mysql> OPTIMIZE TABLE articles;
+---------------+----------+----------+----------+
| Table         | Op       | Msg_type | Msg_text |
+---------------+----------+----------+----------+
| test.articles | optimize | status   | OK       |
+---------------+----------+----------+----------+
```

   The `INNODB_FT_DELETED` table should now be empty.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_DELETED;
Empty set (0.00 sec)
```

9. Query the `INNODB_FT_CONFIG` table. This table contains metadata about the `FULLTEXT` index and related processing:

   - `optimize_checkpoint_limit`: The number of seconds after which an `OPTIMIZE TABLE` run stops.
   - `synced_doc_id`: The next `DOC_ID` to be issued.
   - `stopword_table_name`: The *`database/table`* name for a user-defined stopword table. The `VALUE` column is empty if there is no user-defined stopword table.
   - `use_stopword`: Indicates whether a stopword table is used, which is defined when the `FULLTEXT` index is created.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_CONFIG;
+---------------------------+-------+
| KEY                       | VALUE |
+---------------------------+-------+
| optimize_checkpoint_limit | 180   |
| synced_doc_id             | 8     |
| stopword_table_name       |       |
| use_stopword              | 1     |
+---------------------------+-------+
```

10. Disable `innodb_optimize_fulltext_only`, since it is intended to be enabled only temporarily:

```
mysql> SET GLOBAL innodb_optimize_fulltext_only=OFF;
```

14.15.5 InnoDB INFORMATION_SCHEMA Buffer Pool Tables

The `InnoDB` `INFORMATION_SCHEMA` buffer pool tables provide buffer pool status information and metadata about the pages within the `InnoDB` buffer pool.

The `InnoDB` `INFORMATION_SCHEMA` buffer pool tables include those listed below:

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA LIKE 'INNODB_BUFFER%';
+-----------------------------------------------+
| Tables_in_INFORMATION_SCHEMA (INNODB_BUFFER%) |
+-----------------------------------------------+
| INNODB_BUFFER_PAGE_LRU                        |
| INNODB_BUFFER_PAGE                            |
| INNODB_BUFFER_POOL_STATS                      |
+-----------------------------------------------+
```

Table Overview

- `INNODB_BUFFER_PAGE`: Holds information about each page in the `InnoDB` buffer pool.

- `INNODB_BUFFER_PAGE_LRU`: Holds information about the pages in the `InnoDB` buffer pool, in particular how they are ordered in the LRU list that determines which pages to evict from the buffer pool when it becomes full. The `INNODB_BUFFER_PAGE_LRU` table has the same columns as the `INNODB_BUFFER_PAGE` table, except that the `INNODB_BUFFER_PAGE_LRU` table has an `LRU_POSITION` column instead of a `BLOCK_ID` column.

- `INNODB_BUFFER_POOL_STATS`: Provides buffer pool status information. Much of the same information is provided by `SHOW ENGINE INNODB STATUS` output, or may be obtained using `InnoDB` buffer pool server status variables.

Warning

Querying the `INNODB_BUFFER_PAGE` or `INNODB_BUFFER_PAGE_LRU` table can affect performance. Do not query these tables on a production system unless you are aware of the performance impact and have determined it to be acceptable. To avoid impacting performance on a production system, reproduce the issue you want to investigate and query buffer pool statistics on a test instance.

Example 14.6 Querying System Data in the INNODB_BUFFER_PAGE Table

This query provides an approximate count of pages that contain system data by excluding pages where the `TABLE_NAME` value is either `NULL` or includes a slash `/` or period `.` in the table name, which indicates a user-defined table.

```
mysql> SELECT COUNT(*) FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
       WHERE TABLE_NAME IS NULL OR (INSTR(TABLE_NAME, '/') = 0 AND INSTR(TABLE_NAME, '.') = 0);
+----------+
| COUNT(*) |
+----------+
|     1320 |
+----------+
```

This query returns the approximate number of pages that contain system data, the total number of buffer pool pages, and an approximate percentage of pages that contain system data.

```
mysql> SELECT
       (SELECT COUNT(*) FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
       WHERE TABLE_NAME IS NULL OR (INSTR(TABLE_NAME, '/') = 0 AND INSTR(TABLE_NAME, '.') = 0)
       ) AS system_pages,
       (
       SELECT COUNT(*)
       FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
       ) AS total_pages,
       (
       SELECT ROUND((system_pages/total_pages) * 100)
       ) AS system_page_percentage;
+--------------+-------------+-----------------------+
| system_pages | total_pages | system_page_percentage |
+--------------+-------------+-----------------------+
|         1320 |        8192 |                    16 |
+--------------+-------------+-----------------------+
```

The type of system data in the buffer pool can be determined by querying the `PAGE_TYPE` value. For example, the following query returns eight distinct `PAGE_TYPE` values among the pages that contain system data:

```
mysql> SELECT DISTINCT PAGE_TYPE FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
       WHERE TABLE_NAME IS NULL OR (INSTR(TABLE_NAME, '/') = 0 AND INSTR(TABLE_NAME, '.') = 0);
+-------------------+
| PAGE_TYPE         |
+-------------------+
| SYSTEM            |
| IBUF_BITMAP       |
| UNDO_LOG          |
| UNKNOWN           |
```

```
| FILE_SPACE_HEADER |
| INODE             |
| ALLOCATED         |
| TRX_SYSTEM        |
+-------------------+
```

Example 14.7 Querying User Data in the INNODB_BUFFER_PAGE Table

This query provides an approximate count of pages containing user data by counting pages where the `TABLE_NAME` value is `NOT NULL` and `NOT LIKE '%INNODB_SYS_TABLES%'`.

```
mysql> SELECT COUNT(*) FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
       WHERE TABLE_NAME IS NOT NULL;
+----------+
| COUNT(*) |
+----------+
|     6872 |
+----------+
```

This query returns the approximate number of pages that contain user data, the total number of buffer pool pages, and an approximate percentage of pages that contain user data.

```
mysql> SELECT
       (SELECT COUNT(*) FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
       WHERE TABLE_NAME IS NOT NULL AND (INSTR(TABLE_NAME, '/') > 0 OR INSTR(TABLE_NAME, '.') > 0)
       ) AS user_pages,
       (
       SELECT COUNT(*)
       FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
       ) AS total_pages,
       (
       SELECT ROUND((user_pages/total_pages) * 100)
       ) AS user_page_percentage;
+------------+-------------+----------------------+
| user_pages | total_pages | user_page_percentage |
+------------+-------------+----------------------+
|       6872 |        8192 |                   84 |
+------------+-------------+----------------------+
```

This query identifies user-defined tables with pages in the buffer pool:

```
mysql> SELECT DISTINCT TABLE_NAME FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
       WHERE TABLE_NAME IS NOT NULL AND (INSTR(TABLE_NAME, '/') > 0 OR INSTR(TABLE_NAME, '.') > 0)
       AND TABLE_NAME NOT LIKE '`mysql`.`innodb_%';
+-------------------------+
| TABLE_NAME              |
+-------------------------+
| `employees`.`salaries`  |
| `employees`.`employees` |
+-------------------------+
```

Example 14.8 Querying Index Data in the INNODB_BUFFER_PAGE Table

For information about index pages, query the `INDEX_NAME` column using the name of the index. For example, the following query returns the number of pages and total data size of pages for the `emp_no` index that is defined on the `employees.salaries` table:

```
mysql> SELECT INDEX_NAME, COUNT(*) AS Pages,
       ROUND(SUM(IF(COMPRESSED_SIZE = 0, @@GLOBAL.innodb_page_size, COMPRESSED_SIZE))/1024/1024)
       AS 'Total Data (MB)'
       FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
       WHERE INDEX_NAME='emp_no' AND TABLE_NAME = '`employees`.`salaries`';
+------------+-------+-----------------+
| INDEX_NAME | Pages | Total Data (MB) |
+------------+-------+-----------------+
| emp_no     |  1756 |              27 |
+------------+-------+-----------------+
```

This query returns the number of pages and total data size of pages for all indexes defined on the `employees.salaries` table:

```
mysql> SELECT INDEX_NAME, COUNT(*) AS Pages,
       ROUND(SUM(IF(COMPRESSED_SIZE = 0, @@GLOBAL.innodb_page_size, COMPRESSED_SIZE))/1024/1024)
       AS 'Total Data (MB)'
       FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
       WHERE TABLE_NAME = '`employees`.`salaries`'
       GROUP BY INDEX_NAME;
+------------+-------+-----------------+
| INDEX_NAME | Pages | Total Data (MB) |
+------------+-------+-----------------+
| emp_no     |  1756 |              27 |
| PRIMARY    |  4838 |              76 |
+------------+-------+-----------------+
```

Example 14.9 Querying LRU_POSITION Data in the INNODB_BUFFER_PAGE_LRU Table

The `INNODB_BUFFER_PAGE_LRU` table holds information about the pages in the `InnoDB` buffer pool, in particular how they are ordered that determines which pages to evict from the buffer pool when it becomes full. The definition for this page is the same as for `INNODB_BUFFER_PAGE`, except this table has an `LRU_POSITION` column instead of a `BLOCK_ID` column.

This query counts the number of positions at a specific location in the LRU list occupied by pages of the `employees.employees` table.

```
mysql> SELECT COUNT(LRU_POSITION) FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE_LRU
       WHERE TABLE_NAME='`employees`.`employees`' AND LRU_POSITION < 3072;
+---------------------+
| COUNT(LRU_POSITION) |
+---------------------+
|                 275 |
+---------------------+
```

Example 14.10 Querying the INNODB_BUFFER_POOL_STATS Table

The `INNODB_BUFFER_POOL_STATS` table provides information similar to `SHOW ENGINE INNODB STATUS` and `InnoDB` buffer pool status variables.

```
mysql> SELECT * FROM information_schema.INNODB_BUFFER_POOL_STATS \G
*************************** 1. row ***************************
                         POOL_ID: 0
                       POOL_SIZE: 8192
                    FREE_BUFFERS: 1024
                  DATABASE_PAGES: 7029
              OLD_DATABASE_PAGES: 2574
         MODIFIED_DATABASE_PAGES: 0
              PENDING_DECOMPRESS: 0
                   PENDING_READS: 0
               PENDING_FLUSH_LRU: 0
              PENDING_FLUSH_LIST: 0
                PAGES_MADE_YOUNG: 173
            PAGES_NOT_MADE_YOUNG: 3721891
           PAGES_MADE_YOUNG_RATE: 0
       PAGES_MADE_NOT_YOUNG_RATE: 0
               NUMBER_PAGES_READ: 1075
            NUMBER_PAGES_CREATED: 12594
            NUMBER_PAGES_WRITTEN: 13525
                 PAGES_READ_RATE: 0
               PAGES_CREATE_RATE: 0
              PAGES_WRITTEN_RATE: 0
                NUMBER_PAGES_GET: 27873240
                        HIT_RATE: 0
    YOUNG_MAKE_PER_THOUSAND_GETS: 0
NOT_YOUNG_MAKE_PER_THOUSAND_GETS: 0
         NUMBER_PAGES_READ_AHEAD: 576
       NUMBER_READ_AHEAD_EVICTED: 0
                 READ_AHEAD_RATE: 0
         READ_AHEAD_EVICTED_RATE: 0
                    LRU_IO_TOTAL: 0
                  LRU_IO_CURRENT: 0
                UNCOMPRESS_TOTAL: 0
              UNCOMPRESS_CURRENT: 0
```

For comparison, `SHOW ENGINE INNODB STATUS` output and `InnoDB` buffer pool status variable output is shown below, based on the same data set.

For more information about `SHOW ENGINE INNODB STATUS` output, see Section 14.17.3, “InnoDB Standard Monitor and Lock Monitor Output”.

```
mysql> SHOW ENGINE INNODB STATUS \G
...
----------------------
BUFFER POOL AND MEMORY
----------------------
Total memory allocated 137363456; in additional pool allocated 0
Dictionary memory allocated 99725
Buffer pool size   8192
Free buffers       1024
Database pages     7029
Old database pages 2574
Modified db pages  0
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 173, not young 3721891
0.00 youngs/s, 0.00 non-youngs/s
Pages read 1075, created 12594, written 13525
0.00 reads/s, 0.00 creates/s, 0.00 writes/s
No buffer pool page gets since the last printout
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead 0.00/s
LRU len: 7029, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0]
...
```

For status variable descriptions, see Section 5.1.9, “Server Status Variables”.

```
mysql> SHOW STATUS LIKE 'Innodb_buffer%';
+---------------------------------------+-------------+
| Variable_name                         | Value       |
+---------------------------------------+-------------+
| Innodb_buffer_pool_dump_status        | not started |
| Innodb_buffer_pool_load_status        | not started |
| Innodb_buffer_pool_pages_data         | 7029        |
| Innodb_buffer_pool_bytes_data         | 115163136   |
| Innodb_buffer_pool_pages_dirty        | 0           |
| Innodb_buffer_pool_bytes_dirty        | 0           |
| Innodb_buffer_pool_pages_flushed      | 13525       |
| Innodb_buffer_pool_pages_free         | 1024        |
| Innodb_buffer_pool_pages_misc         | 139         |
| Innodb_buffer_pool_pages_total        | 8192        |
| Innodb_buffer_pool_read_ahead_rnd     | 0           |
| Innodb_buffer_pool_read_ahead         | 576         |
| Innodb_buffer_pool_read_ahead_evicted | 0           |
| Innodb_buffer_pool_read_requests      | 27873240    |
| Innodb_buffer_pool_reads              | 500         |
| Innodb_buffer_pool_wait_free          | 0           |
| Innodb_buffer_pool_write_requests     | 11966441    |
+---------------------------------------+-------------+
```

14.15.6 InnoDB INFORMATION_SCHEMA Metrics Table

The `INNODB_METRICS` table provides information about `InnoDB` performance and resource-related counters.

`INNODB_METRICS` table columns are shown below. For column descriptions, see Section 21.30.15, “The INFORMATION_SCHEMA INNODB_METRICS Table”.

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME="dml_inserts" \G
*************************** 1. row ***************************
           NAME: dml_inserts
      SUBSYSTEM: dml
          COUNT: 46273
      MAX_COUNT: 46273
      MIN_COUNT: NULL
```

```
      AVG_COUNT: 492.2659574468085
    COUNT_RESET: 46273
MAX_COUNT_RESET: 46273
MIN_COUNT_RESET: NULL
AVG_COUNT_RESET: NULL
   TIME_ENABLED: 2014-11-28 16:07:53
  TIME_DISABLED: NULL
   TIME_ELAPSED: 94
     TIME_RESET: NULL
         STATUS: enabled
           TYPE: status_counter
        COMMENT: Number of rows inserted
```

Enabling, Disabling, and Resetting Counters

You can enable, disable, and reset counters using the following variables:

- `innodb_monitor_enable`: Enables counters.

  ```
  SET GLOBAL innodb_monitor_enable = [counter-name|module_name|pattern|all];
  ```

- `innodb_monitor_disable`: Disables counters.

  ```
  SET GLOBAL innodb_monitor_disable = [counter-name|module_name|pattern|all];
  ```

- `innodb_monitor_reset`: Resets counter values to zero.

  ```
  SET GLOBAL innodb_monitor_reset = [counter-name|module_name|pattern|all];
  ```

- `innodb_monitor_reset_all`: Resets all counter values. A counter must be disabled before using `innodb_monitor_reset_all`.

  ```
  SET GLOBAL innodb_monitor_reset_all = [counter-name|module_name|pattern|all];
  ```

Counters and counter modules can also be enabled at startup using the MySQL server configuration file. For example, to enable the `log` module, `metadata_table_handles_opened` and `metadata_table_handles_closed` counters, enter the following line in the `[mysqld]` section of the MySQL server configuration file.

```
[mysqld]
innodb_monitor_enable = module_recovery,metadata_table_handles_opened,metadata_table_handles_closed
```

When enabling multiple counters or modules in a configuration file, specify the `innodb_monitor_enable` variable followed by counter and module names separated by a comma, as shown above. Only the `innodb_monitor_enable` variable can be used in a configuration file. The `innodb_monitor_disable` and `innodb_monitor_reset` variables are supported on the command line only.

Note

Because each counter adds a degree of runtime overhead, use counters conservatively on production servers to diagnose specific issues or monitor specific functionality. A test or development server is recommended for more extensive use of counters.

Counters

The list of available counters is subject to change. Query the `INFORMATION_SCHEMA.INNODB_METRICS` table for counters available in your MySQL server version.

The counters enabled by default correspond to those shown in `SHOW ENGINE INNODB STATUS` output. Counters shown in `SHOW ENGINE INNODB STATUS` output are always enabled at a system level but can be disable for the `INNODB_METRICS` table. Counter status is not persistent. Unless configured otherwise, counters revert to their default enabled or disabled status when the server is restarted.

If you run programs that would be affected by the addition or removal of counters, it is recommended that you review the releases notes and query the `INNODB_METRICS` table to identify those changes as part of your upgrade process.

```
mysql> SELECT name, subsystem, status FROM INFORMATION_SCHEMA.INNODB_METRICS ORDER BY NAME;
+----------------------------------------+--------------------+----------+
| name                                   | subsystem          | status   |
+----------------------------------------+--------------------+----------+
| adaptive_hash_pages_added              | adaptive_hash_index | disabled |
| adaptive_hash_pages_removed            | adaptive_hash_index | disabled |
| adaptive_hash_rows_added               | adaptive_hash_index | disabled |
| adaptive_hash_rows_deleted_no_hash_entry | adaptive_hash_index | disabled |
| adaptive_hash_rows_removed             | adaptive_hash_index | disabled |
| adaptive_hash_rows_updated             | adaptive_hash_index | disabled |
| adaptive_hash_searches                 | adaptive_hash_index | enabled  |
| adaptive_hash_searches_btree           | adaptive_hash_index | disabled |
| buffer_data_reads                      | buffer              | enabled  |
| buffer_data_written                    | buffer              | enabled  |
| buffer_flush_adaptive                  | buffer              | disabled |
| buffer_flush_adaptive_pages            | buffer              | disabled |
| buffer_flush_adaptive_total_pages      | buffer              | disabled |
| buffer_flush_avg_page_rate             | buffer              | disabled |
| buffer_flush_background                | buffer              | disabled |
| buffer_flush_background_pages          | buffer              | disabled |
| buffer_flush_background_total_pages    | buffer              | disabled |
| buffer_flush_batches                   | buffer              | disabled |
| buffer_flush_batch_num_scan            | buffer              | disabled |
| buffer_flush_batch_pages               | buffer              | disabled |
| buffer_flush_batch_rescan              | buffer              | disabled |
| buffer_flush_batch_scanned             | buffer              | disabled |
| buffer_flush_batch_scanned_per_call    | buffer              | disabled |
| buffer_flush_batch_total_pages         | buffer              | disabled |
| buffer_flush_lsn_avg_rate              | buffer              | disabled |
| buffer_flush_neighbor                  | buffer              | disabled |
| buffer_flush_neighbor_pages            | buffer              | disabled |
| buffer_flush_neighbor_total_pages      | buffer              | disabled |
| buffer_flush_n_to_flush_requested      | buffer              | disabled |
| buffer_flush_pct_for_dirty             | buffer              | disabled |
| buffer_flush_pct_for_lsn               | buffer              | disabled |
| buffer_flush_sync                      | buffer              | disabled |
| buffer_flush_sync_pages                | buffer              | disabled |
| buffer_flush_sync_total_pages          | buffer              | disabled |
| buffer_flush_sync_waits                | buffer              | disabled |
| buffer_LRU_batches                     | buffer              | disabled |
| buffer_LRU_batch_num_scan              | buffer              | disabled |
| buffer_LRU_batch_pages                 | buffer              | disabled |
| buffer_LRU_batch_scanned               | buffer              | disabled |
| buffer_LRU_batch_scanned_per_call      | buffer              | disabled |
| buffer_LRU_batch_total_pages           | buffer              | disabled |
| buffer_LRU_get_free_search             | Buffer              | disabled |
| buffer_LRU_search_num_scan             | buffer              | disabled |
| buffer_LRU_search_scanned              | buffer              | disabled |
| buffer_LRU_search_scanned_per_call     | buffer              | disabled |
| buffer_LRU_single_flush_failure_count  | Buffer              | disabled |
| buffer_LRU_single_flush_num_scan       | buffer              | disabled |
| buffer_LRU_single_flush_scanned        | buffer              | disabled |
| buffer_LRU_single_flush_scanned_per_call | buffer              | disabled |
| buffer_LRU_unzip_search_num_scan       | buffer              | disabled |
| buffer_LRU_unzip_search_scanned        | buffer              | disabled |
| buffer_LRU_unzip_search_scanned_per_call | buffer              | disabled |
| buffer_pages_created                   | buffer              | enabled  |
| buffer_pages_read                      | buffer              | enabled  |
| buffer_pages_written                   | buffer              | enabled  |
| buffer_page_read_blob                  | buffer_page_io      | disabled |
| buffer_page_read_fsp_hdr               | buffer_page_io      | disabled |
| buffer_page_read_ibuf_bitmap           | buffer_page_io      | disabled |
| buffer_page_read_ibuf_free_list        | buffer_page_io      | disabled |
| buffer_page_read_index_ibuf_leaf       | buffer_page_io      | disabled |
| buffer_page_read_index_ibuf_non_leaf   | buffer_page_io      | disabled |
| buffer_page_read_index_inode           | buffer_page_io      | disabled |
| buffer_page_read_index_leaf            | buffer_page_io      | disabled |
| buffer_page_read_index_non_leaf        | buffer_page_io      | disabled |
```

```
| buffer_page_read_other                  | buffer_page_io      | disabled |
| buffer_page_read_system_page            | buffer_page_io      | disabled |
| buffer_page_read_trx_system             | buffer_page_io      | disabled |
| buffer_page_read_undo_log               | buffer_page_io      | disabled |
| buffer_page_read_xdes                   | buffer_page_io      | disabled |
| buffer_page_read_zblob                  | buffer_page_io      | disabled |
| buffer_page_read_zblob2                 | buffer_page_io      | disabled |
| buffer_page_written_blob                | buffer_page_io      | disabled |
| buffer_page_written_fsp_hdr             | buffer_page_io      | disabled |
| buffer_page_written_ibuf_bitmap         | buffer_page_io      | disabled |
| buffer_page_written_ibuf_free_list      | buffer_page_io      | disabled |
| buffer_page_written_index_ibuf_leaf     | buffer_page_io      | disabled |
| buffer_page_written_index_ibuf_non_leaf | buffer_page_io      | disabled |
| buffer_page_written_index_inode         | buffer_page_io      | disabled |
| buffer_page_written_index_leaf          | buffer_page_io      | disabled |
| buffer_page_written_index_non_leaf      | buffer_page_io      | disabled |
| buffer_page_written_other               | buffer_page_io      | disabled |
| buffer_page_written_system_page         | buffer_page_io      | disabled |
| buffer_page_written_trx_system          | buffer_page_io      | disabled |
| buffer_page_written_undo_log            | buffer_page_io      | disabled |
| buffer_page_written_xdes                | buffer_page_io      | disabled |
| buffer_page_written_zblob               | buffer_page_io      | disabled |
| buffer_page_written_zblob2              | buffer_page_io      | disabled |
| buffer_pool_bytes_data                  | buffer              | enabled  |
| buffer_pool_bytes_dirty                 | buffer              | enabled  |
| buffer_pool_pages_data                  | buffer              | enabled  |
| buffer_pool_pages_dirty                 | buffer              | enabled  |
| buffer_pool_pages_free                  | buffer              | enabled  |
| buffer_pool_pages_misc                  | buffer              | enabled  |
| buffer_pool_pages_total                 | buffer              | enabled  |
| buffer_pool_reads                       | buffer              | enabled  |
| buffer_pool_read_ahead                  | buffer              | enabled  |
| buffer_pool_read_ahead_evicted          | buffer              | enabled  |
| buffer_pool_read_requests               | buffer              | enabled  |
| buffer_pool_size                        | server              | enabled  |
| buffer_pool_wait_free                   | buffer              | enabled  |
| buffer_pool_write_requests              | buffer              | enabled  |
| compression_pad_decrements              | compression         | disabled |
| compression_pad_increments              | compression         | disabled |
| compress_pages_compressed               | compression         | disabled |
| compress_pages_decompressed             | compression         | disabled |
| ddl_background_drop_indexes             | ddl                 | disabled |
| ddl_background_drop_tables              | ddl                 | disabled |
| ddl_online_create_index                 | ddl                 | disabled |
| ddl_pending_alter_table                 | ddl                 | disabled |
| dml_deletes                             | dml                 | enabled  |
| dml_inserts                             | dml                 | enabled  |
| dml_reads                               | dml                 | enabled  |
| dml_updates                             | dml                 | enabled  |
| file_num_open_files                     | file_system         | enabled  |
| ibuf_merges                             | change_buffer       | enabled  |
| ibuf_merges_delete                      | change_buffer       | enabled  |
| ibuf_merges_delete_mark                 | change_buffer       | enabled  |
| ibuf_merges_discard_delete              | change_buffer       | enabled  |
| ibuf_merges_discard_delete_mark         | change_buffer       | enabled  |
| ibuf_merges_discard_insert              | change_buffer       | enabled  |
| ibuf_merges_insert                      | change_buffer       | enabled  |
| ibuf_size                               | change_buffer       | enabled  |
| icp_attempts                            | icp                 | disabled |
| icp_match                               | icp                 | disabled |
| icp_no_match                            | icp                 | disabled |
| icp_out_of_range                        | icp                 | disabled |
| index_page_discards                     | index               | disabled |
| index_page_merge_attempts               | index               | disabled |
| index_page_merge_successful             | index               | disabled |
| index_page_reorg_attempts               | index               | disabled |
| index_page_reorg_successful             | index               | disabled |
| index_page_splits                       | index               | disabled |
| innodb_activity_count                   | server              | enabled  |
| innodb_background_drop_table_usec       | server              | disabled |
| innodb_checkpoint_usec                  | server              | disabled |
| innodb_dblwr_pages_written              | server              | enabled  |
```

```
| innodb_dblwr_writes                  | server             | enabled  |
| innodb_dict_lru_usec                 | server             | disabled |
| innodb_ibuf_merge_usec               | server             | disabled |
| innodb_log_flush_usec                | server             | disabled |
| innodb_master_active_loops           | server             | disabled |
| innodb_master_idle_loops             | server             | disabled |
| innodb_master_purge_usec             | server             | disabled |
| innodb_master_thread_sleeps          | server             | disabled |
| innodb_mem_validate_usec             | server             | disabled |
| innodb_page_size                     | server             | enabled  |
| innodb_rwlock_s_os_waits             | server             | enabled  |
| innodb_rwlock_s_spin_rounds          | server             | enabled  |
| innodb_rwlock_s_spin_waits           | server             | enabled  |
| innodb_rwlock_x_os_waits             | server             | enabled  |
| innodb_rwlock_x_spin_rounds          | server             | enabled  |
| innodb_rwlock_x_spin_waits           | server             | enabled  |
| lock_deadlocks                       | lock               | enabled  |
| lock_rec_locks                       | lock               | disabled |
| lock_rec_lock_created                | lock               | disabled |
| lock_rec_lock_removed                | lock               | disabled |
| lock_rec_lock_requests               | lock               | disabled |
| lock_rec_lock_waits                  | lock               | disabled |
| lock_row_lock_current_waits          | lock               | enabled  |
| lock_row_lock_time                   | lock               | enabled  |
| lock_row_lock_time_avg               | lock               | enabled  |
| lock_row_lock_time_max               | lock               | enabled  |
| lock_row_lock_waits                  | lock               | enabled  |
| lock_table_locks                     | lock               | disabled |
| lock_table_lock_created              | lock               | disabled |
| lock_table_lock_removed              | lock               | disabled |
| lock_table_lock_waits                | lock               | disabled |
| lock_timeouts                        | lock               | enabled  |
| log_checkpoints                      | recovery           | disabled |
| log_lsn_buf_pool_oldest              | recovery           | disabled |
| log_lsn_checkpoint_age               | recovery           | disabled |
| log_lsn_current                      | recovery           | disabled |
| log_lsn_last_checkpoint              | recovery           | disabled |
| log_lsn_last_flush                   | recovery           | disabled |
| log_max_modified_age_async           | recovery           | disabled |
| log_max_modified_age_sync            | recovery           | disabled |
| log_num_log_io                       | recovery           | disabled |
| log_pending_checkpoint_writes        | recovery           | disabled |
| log_pending_log_writes               | recovery           | disabled |
| log_waits                            | recovery           | enabled  |
| log_writes                           | recovery           | enabled  |
| log_write_requests                   | recovery           | enabled  |
| metadata_mem_pool_size               | metadata           | enabled  |
| metadata_table_handles_closed        | metadata           | disabled |
| metadata_table_handles_opened        | metadata           | disabled |
| metadata_table_reference_count       | metadata           | disabled |
| os_data_fsyncs                       | os                 | enabled  |
| os_data_reads                        | os                 | enabled  |
| os_data_writes                       | os                 | enabled  |
| os_log_bytes_written                 | os                 | enabled  |
| os_log_fsyncs                        | os                 | enabled  |
| os_log_pending_fsyncs                | os                 | enabled  |
| os_log_pending_writes                | os                 | enabled  |
| os_pending_reads                     | os                 | disabled |
| os_pending_writes                    | os                 | disabled |
| purge_del_mark_records               | purge              | disabled |
| purge_dml_delay_usec                 | purge              | disabled |
| purge_invoked                        | purge              | disabled |
| purge_resume_count                   | purge              | disabled |
| purge_stop_count                     | purge              | disabled |
| purge_undo_log_pages                 | purge              | disabled |
| purge_upd_exist_or_extern_records    | purge              | disabled |
| trx_active_transactions              | transaction        | disabled |
| trx_commits_insert_update            | transaction        | disabled |
| trx_nl_ro_commits                    | transaction        | disabled |
| trx_rollbacks                        | transaction        | disabled |
| trx_rollbacks_savepoint              | transaction        | disabled |
| trx_rollback_active                  | transaction        | disabled |
```

```
| trx_ro_commits                          | transaction         | disabled |
| trx_rseg_current_size                   | transaction         | disabled |
| trx_rseg_history_len                    | transaction         | enabled  |
| trx_rw_commits                          | transaction         | disabled |
| trx_undo_slots_cached                   | transaction         | disabled |
| trx_undo_slots_used                     | transaction         | disabled |
+-----------------------------------------+---------------------+----------+
214 rows in set (0.00 sec)
```

Counter Modules

Each counter is associated with a particular module. Module names can be used to enable, disable, or reset all counters for a particular subsystem. For example, use `module_dml` to enable all counters associated with the `dml` subsystem.

```
mysql> SET GLOBAL innodb_monitor_enable = module_dml;

mysql> SELECT name, subsystem, status FROM INFORMATION_SCHEMA.INNODB_METRICS
       WHERE subsystem ='dml';
+-------------+-----------+---------+
| name        | subsystem | status  |
+-------------+-----------+---------+
| dml_reads   | dml       | enabled |
| dml_inserts | dml       | enabled |
| dml_deletes | dml       | enabled |
| dml_updates | dml       | enabled |
+-------------+-----------+---------+
```

Module names can be used with `innodb_monitor_enable` and related variables.

Module names and corresponding `SUBSYSTEM` names are listed below.

- `module_adaptive_hash` (subsystem = `adaptive_hash_index`)
- `module_buffer` (subsystem = `buffer`)
- `module_buffer_page` (subsystem = `buffer_page_io`)
- `module_compress` (subsystem = `compression`)
- `module_ddl` (subsystem = `ddl`)
- `module_dml` (subsystem = `dml`)
- `module_file` (subsystem = `file_system`)
- `module_ibuf_system` (subsystem = `change_buffer`)
- `module_icp` (subsystem = `icp`)
- `module_index` (subsystem = `index`)
- `module_innodb` (subsystem = `innodb`)
- `module_lock` (subsystem = `lock`)
- `module_log` (subsystem = `recovery`)
- `module_metadata` (subsystem = `metadata`)
- `module_os` (subsystem = `os`)
- `module_purge` (subsystem = `purge`)
- `module_trx` (subsystem = `transaction`)

Example 14.11 Working with INNODB_METRICS Table Counters

This example demonstrates enabling, disabling, and resetting a counter, and querying counter data in the `INNODB_METRICS` table.

1. Create a simple `InnoDB` table:

```
mysql> USE test;
Database changed

mysql> CREATE TABLE t1 (c1 INT) ENGINE=INNODB;
Query OK, 0 rows affected (0.02 sec)
```

2. Enable the `dml_inserts` counter.

```
mysql> SET GLOBAL innodb_monitor_enable = dml_inserts;
Query OK, 0 rows affected (0.01 sec)
```

   A description of the `dml_inserts` counter can be found in the `COMMENT` column of the `INNODB_METRICS` table:

```
mysql> SELECT NAME, COMMENT FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME="dml_inserts";
+-------------+------------------------+
| NAME        | COMMENT                |
+-------------+------------------------+
| dml_inserts | Number of rows inserted |
+-------------+------------------------+
```

3. Query the `INNODB_METRICS` table for the `dml_inserts` counter data. Because no DML operations have been performed, the counter values are zero or NULL. The `TIME_ENABLED` and `TIME_ELAPSED` values indicate when the counter was last enabled and how many seconds have elapsed since that time.

```
mysql>  SELECT * FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME="dml_inserts" \G
*************************** 1. row ***************************
           NAME: dml_inserts
      SUBSYSTEM: dml
          COUNT: 0
      MAX_COUNT: 0
      MIN_COUNT: NULL
      AVG_COUNT: 0
    COUNT_RESET: 0
MAX_COUNT_RESET: 0
MIN_COUNT_RESET: NULL
AVG_COUNT_RESET: NULL
   TIME_ENABLED: 2014-12-04 14:18:28
  TIME_DISABLED: NULL
   TIME_ELAPSED: 28
     TIME_RESET: NULL
         STATUS: enabled
           TYPE: status_counter
        COMMENT: Number of rows inserted
```

4. Insert three rows of data into the table.

```
mysql> INSERT INTO t1 values(1);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO t1 values(2);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO t1 values(3);
Query OK, 1 row affected (0.00 sec)
```

5. Query the `INNODB_METRICS` table again for the `dml_inserts` counter data. A number of counter values have now incremented including `COUNT`, `MAX_COUNT`, `AVG_COUNT`, and `COUNT_RESET`. Refer to the `INNODB_METRICS` table definition for descriptions of these values.

```
mysql>  SELECT * FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME="dml_inserts"\G
```

```
*************************** 1. row ***************************
           NAME: dml_inserts
      SUBSYSTEM: dml
          COUNT: 3
      MAX_COUNT: 3
      MIN_COUNT: NULL
      AVG_COUNT: 0.046153846153846156
    COUNT_RESET: 3
MAX_COUNT_RESET: 3
MIN_COUNT_RESET: NULL
AVG_COUNT_RESET: NULL
   TIME_ENABLED: 2014-12-04 14:18:28
  TIME_DISABLED: NULL
   TIME_ELAPSED: 65
     TIME_RESET: NULL
         STATUS: enabled
           TYPE: status_counter
        COMMENT: Number of rows inserted
```

6. Reset the `dml_inserts` counter and query the `INNODB_METRICS` table again for the `dml_inserts` counter data. The `%_RESET` values that were reported previously, such as `COUNT_RESET` and `MAX_RESET`, are set back to zero. Values such as `COUNT`, `MAX_COUNT`, and `AVG_COUNT`, which cumulatively collect data from the time the counter is enabled, are unaffected by the reset.

```
mysql> SET GLOBAL innodb_monitor_reset = dml_inserts;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME="dml_inserts"\G
*************************** 1. row ***************************
           NAME: dml_inserts
      SUBSYSTEM: dml
          COUNT: 3
      MAX_COUNT: 3
      MIN_COUNT: NULL
      AVG_COUNT: 0.03529411764705882
    COUNT_RESET: 0
MAX_COUNT_RESET: 0
MIN_COUNT_RESET: NULL
AVG_COUNT_RESET: 0
   TIME_ENABLED: 2014-12-04 14:18:28
  TIME_DISABLED: NULL
   TIME_ELAPSED: 85
     TIME_RESET: 2014-12-04 14:19:44
         STATUS: enabled
           TYPE: status_counter
        COMMENT: Number of rows inserted
```

7. To reset all counter values, you must first disable the counter. Disabling the counter sets the `STATUS` value to `disabled`.

```
mysql> SET GLOBAL innodb_monitor_disable = dml_inserts;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME="dml_inserts"\G
*************************** 1. row ***************************
           NAME: dml_inserts
      SUBSYSTEM: dml
          COUNT: 3
      MAX_COUNT: 3
      MIN_COUNT: NULL
      AVG_COUNT: 0.030612244897959183
    COUNT_RESET: 0
MAX_COUNT_RESET: 0
MIN_COUNT_RESET: NULL
AVG_COUNT_RESET: 0
   TIME_ENABLED: 2014-12-04 14:18:28
  TIME_DISABLED: 2014-12-04 14:20:06
   TIME_ELAPSED: 98
     TIME_RESET: NULL
         STATUS: disabled
```

```
          TYPE: status_counter
       COMMENT: Number of rows inserted
```

Note

Wildcard match is supported for counter and module names. For example, instead of specifying the full `dml_inserts` counter name, you can specify `dml_i%`. You can also enable, disable, or reset multiple counters or modules at once using a wildcard match. For example, specify `dml_%` to enable, disable, or reset all counters that begin with `dml_`.

8. After the counter is disabled, you can reset all counter values using the `innodb_monitor_reset_all` option. All values are set to zero or NULL.

```
mysql> SET GLOBAL innodb_monitor_reset_all = dml_inserts;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME="dml_inserts"\G
*************************** 1. row ***************************
           NAME: dml_inserts
      SUBSYSTEM: dml
          COUNT: 0
      MAX_COUNT: NULL
      MIN_COUNT: NULL
      AVG_COUNT: NULL
    COUNT_RESET: 0
MAX_COUNT_RESET: NULL
MIN_COUNT_RESET: NULL
AVG_COUNT_RESET: NULL
   TIME_ENABLED: NULL
  TIME_DISABLED: NULL
   TIME_ELAPSED: NULL
     TIME_RESET: NULL
         STATUS: disabled
           TYPE: status_counter
        COMMENT: Number of rows inserted
```

14.16 InnoDB Integration with MySQL Performance Schema

This section provides a brief introduction to `InnoDB` integration with Performance Schema. For comprehensive Performance Schema documentation, see Chapter 22, *MySQL Performance Schema*.

You can profile certain internal `InnoDB` operations using the MySQL Performance Schema feature. This type of tuning is primarily for expert users who evaluate optimization strategies to overcome performance bottlenecks. DBAs can also use this feature for capacity planning, to see whether their typical workload encounters any performance bottlenecks with a particular combination of CPU, RAM, and disk storage; and if so, to judge whether performance can be improved by increasing the capacity of some part of the system.

To use this feature to examine `InnoDB` performance:

- You must be generally familiar with how to use the Performance Schema feature. For example, you should know how enable instruments and consumers, and how to query `performance_schema` tables to retrieve data. For an introductory overview, see Section 22.1, “Performance Schema Quick Start”.
- You should be familiar with Performance Schema instruments that are available for `InnoDB`. To view `InnoDB`-related instruments, you can query the `setup_instruments` table for instrument names that contain '`innodb`'.

```
mysql> SELECT *
       FROM performance_schema.setup_instruments
       WHERE NAME LIKE '%innodb%';
+---------------------------------------------------------+---------+-------+
| NAME                                                    | ENABLED | TIMED |
+---------------------------------------------------------+---------+-------+
```

```
| wait/synch/mutex/innodb/commit_cond_mutex                | NO      | NO    |
| wait/synch/mutex/innodb/innobase_share_mutex             | NO      | NO    |
| wait/synch/mutex/innodb/autoinc_mutex                    | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_mutex                   | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_zip_mutex               | NO      | NO    |
| wait/synch/mutex/innodb/cache_last_read_mutex            | NO      | NO    |
| wait/synch/mutex/innodb/dict_foreign_err_mutex           | NO      | NO    |
| wait/synch/mutex/innodb/dict_sys_mutex                   | NO      | NO    |
| wait/synch/mutex/innodb/file_format_max_mutex            | NO      | NO    |
| wait/synch/mutex/innodb/fil_system_mutex                 | NO      | NO    |
| wait/synch/mutex/innodb/flush_list_mutex                 | NO      | NO    |
| wait/synch/mutex/innodb/fts_bg_threads_mutex             | NO      | NO    |
| wait/synch/mutex/innodb/fts_delete_mutex                 | NO      | NO    |
| wait/synch/mutex/innodb/fts_optimize_mutex               | NO      | NO    |
...
| wait/synch/rwlock/innodb/btr_search_latch                | NO      | NO    |
| wait/synch/rwlock/innodb/dict_operation_lock             | NO      | NO    |
| wait/synch/rwlock/innodb/fil_space_latch                 | NO      | NO    |
| wait/synch/rwlock/innodb/checkpoint_lock                 | NO      | NO    |
| wait/synch/rwlock/innodb/fts_cache_rw_lock               | NO      | NO    |
| wait/synch/rwlock/innodb/fts_cache_init_rw_lock          | NO      | NO    |
| wait/synch/rwlock/innodb/trx_i_s_cache_lock              | NO      | NO    |
| wait/synch/rwlock/innodb/trx_purge_latch                 | NO      | NO    |
| wait/synch/rwlock/innodb/index_tree_rw_lock              | NO      | NO    |
| wait/synch/rwlock/innodb/index_online_log                | NO      | NO    |
| wait/synch/rwlock/innodb/dict_table_stats                | NO      | NO    |
| wait/synch/rwlock/innodb/hash_table_locks                | NO      | NO    |
| wait/synch/cond/innodb/commit_cond                       | NO      | NO    |
| wait/io/file/innodb/innodb_data_file                     | YES     | YES   |
| wait/io/file/innodb/innodb_log_file                      | YES     | YES   |
| wait/io/file/innodb/innodb_temp_file                     | YES     | YES   |
+----------------------------------------------------------+---------+-------+
62 rows in set (0.00 sec)
```

For additional information about the instrumented `InnoDB` objects, you can query Performance Schema instances tables, which provide additional information about instrumented objects. Instance tables relevant to `InnoDB` include:

- The `mutex_instances` table
- The `rwlock_instances` table
- The `cond_instances` table
- The `file_instances` table

Note

Mutexes and RW-locks related to the `InnoDB` buffer pool are not included in this coverage; the same applies to the output of the `SHOW ENGINE INNODB MUTEX` command.

For example, to view information about instrumented `InnoDB` file objects seen by the Performance Schema when executing file I/O instrumentation, you might issue the following query:

```
mysql> SELECT *
       FROM performance_schema.file_instances
       WHERE EVENT_NAME LIKE '%innodb%'\G
*************************** 1. row ***************************
 FILE_NAME: /path/to/mysql-5.6/data/ibdata1
EVENT_NAME: wait/io/file/innodb/innodb_data_file
OPEN_COUNT: 2
*************************** 2. row ***************************
 FILE_NAME: /path/to/mysql-5.6/data/ib_logfile0
EVENT_NAME: wait/io/file/innodb/innodb_log_file
OPEN_COUNT: 2
*************************** 3. row ***************************
 FILE_NAME: /path/to/mysql-5.6/data/ib_logfile1
EVENT_NAME: wait/io/file/innodb/innodb_log_file
```

```
OPEN_COUNT: 2
...
```

- You should be familiar with `performance_schema` tables that store `InnoDB` event data. Tables relevant to `InnoDB`-related events include:

  - The Wait Event tables, which store wait events.

  - The Summary tables, which provide aggregated information for terminated events over time. Summary tables include file I/O summary tables, which aggregate information about I/O operations.

  If you are only interested in `InnoDB`-related objects, use the clause `WHERE EVENT_NAME LIKE '%innodb%'` or `WHERE NAME LIKE '%innodb%'` (as required) when querying these tables.

14.16.1 Monitoring InnoDB Mutex Waits Using Performance Schema

A mutex is a synchronization mechanism used in the code to enforce that only one thread at a given time can have access to a common resource. When two or more threads executing in the server need to access the same resource, the threads compete against each other. The first thread to obtain a lock on the mutex causes the other threads to wait until the lock is released.

For `InnoDB` mutexes that are instrumented, mutex waits can be monitored using Performance Schema. Wait event data collected in Performance Schema tables can help identify mutexes with the most waits or the greatest total wait time, for example.

The following example demonstrates how to enable `InnoDB` mutex wait instruments, how to enable associated consumers, and how to query wait event data.

1. To view available `InnoDB` mutex wait instruments, query the Performance Schema `setup_instruments` table, as shown below. All `InnoDB` mutex wait instruments are disabled by default.

```
mysql> SELECT *
       FROM performance_schema.setup_instruments
       WHERE NAME LIKE '%wait/synch/mutex/innodb%';
+-----------------------------------------------------------+---------+-------+
| NAME                                                      | ENABLED | TIMED |
+-----------------------------------------------------------+---------+-------+
| wait/synch/mutex/innodb/commit_cond_mutex                 | NO      | NO    |
| wait/synch/mutex/innodb/innobase_share_mutex              | NO      | NO    |
| wait/synch/mutex/innodb/autoinc_mutex                     | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_mutex                    | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_zip_mutex                | NO      | NO    |
| wait/synch/mutex/innodb/cache_last_read_mutex             | NO      | NO    |
| wait/synch/mutex/innodb/dict_foreign_err_mutex            | NO      | NO    |
| wait/synch/mutex/innodb/dict_sys_mutex                    | NO      | NO    |
| wait/synch/mutex/innodb/file_format_max_mutex             | NO      | NO    |
| wait/synch/mutex/innodb/fil_system_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/flush_list_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/fts_bg_threads_mutex              | NO      | NO    |
| wait/synch/mutex/innodb/fts_delete_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/fts_optimize_mutex                | NO      | NO    |
| wait/synch/mutex/innodb/fts_doc_id_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/fts_pll_tokenize_mutex            | NO      | NO    |
| wait/synch/mutex/innodb/log_flush_order_mutex             | NO      | NO    |
| wait/synch/mutex/innodb/hash_table_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/ibuf_bitmap_mutex                 | NO      | NO    |
| wait/synch/mutex/innodb/ibuf_mutex                        | NO      | NO    |
| wait/synch/mutex/innodb/ibuf_pessimistic_insert_mutex     | NO      | NO    |
| wait/synch/mutex/innodb/log_sys_mutex                     | NO      | NO    |
| wait/synch/mutex/innodb/mem_pool_mutex                    | NO      | NO    |
| wait/synch/mutex/innodb/mutex_list_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/page_zip_stat_per_index_mutex     | NO      | NO    |
| wait/synch/mutex/innodb/purge_sys_bh_mutex                | NO      | NO    |
| wait/synch/mutex/innodb/recv_sys_mutex                    | NO      | NO    |
| wait/synch/mutex/innodb/recv_writer_mutex                 | NO      | NO    |
```

```
| wait/synch/mutex/innodb/rseg_mutex                     | NO      | NO    |
| wait/synch/mutex/innodb/rw_lock_list_mutex             | NO      | NO    |
| wait/synch/mutex/innodb/rw_lock_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/srv_dict_tmpfile_mutex         | NO      | NO    |
| wait/synch/mutex/innodb/srv_innodb_monitor_mutex       | NO      | NO    |
| wait/synch/mutex/innodb/srv_misc_tmpfile_mutex         | NO      | NO    |
| wait/synch/mutex/innodb/srv_monitor_file_mutex         | NO      | NO    |
| wait/synch/mutex/innodb/buf_dblwr_mutex                | NO      | NO    |
| wait/synch/mutex/innodb/trx_undo_mutex                 | NO      | NO    |
| wait/synch/mutex/innodb/srv_sys_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/lock_mutex                     | NO      | NO    |
| wait/synch/mutex/innodb/lock_wait_mutex                | NO      | NO    |
| wait/synch/mutex/innodb/trx_mutex                      | NO      | NO    |
| wait/synch/mutex/innodb/srv_threads_mutex              | NO      | NO    |
| wait/synch/mutex/innodb/os_mutex                       | NO      | NO    |
| wait/synch/mutex/innodb/ut_list_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/trx_sys_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/zip_pad_mutex                  | NO      | NO    |
+--------------------------------------------------------+---------+-------+
46 rows in set (0.00 sec)
```

2. Some `InnoDB` mutex instances are created at server startup and are only instrumented if the associated instrument is also enabled at server startup. To ensure that all `InnoDB` mutex instances are instrumented and enabled, add the following `performance-schema-instrument` rule to your MySQL configuration file:

```
performance-schema-instrument='wait/synch/mutex/innodb/%=ON'
```

If you do not require wait event data for all `InnoDB` mutexes, you can disable specific instruments by adding additional `performance-schema-instrument` rules to your MySQL configuration file. For example, to disable `InnoDB` mutex wait event instruments related to full-text search, add the following rule:

```
performance-schema-instrument='wait/synch/mutex/innodb/fts%=OFF'
```

Note

Rules with a longer prefix such as `wait/synch/mutex/innodb/fts%` take precedence over rules with shorter prefixes such as `wait/synch/mutex/innodb/%`.

After adding the `performance-schema-instrument` rules to your configuration file, restart the server. All the `InnoDB` mutexes except for those related to full text search are enabled. To verify, query the `setup_instruments` table. The `ENABLED` and `TIMED` columns should be set to `YES` for the instruments that you enabled.

```
mysql> SELECT *
       FROM performance_schema.setup_instruments
       WHERE NAME LIKE '%wait/synch/mutex/innodb%';
+--------------------------------------------------------+---------+-------+
| NAME                                                   | ENABLED | TIMED |
+--------------------------------------------------------+---------+-------+
| wait/synch/mutex/innodb/commit_cond_mutex              | YES     | YES   |
| wait/synch/mutex/innodb/innobase_share_mutex           | YES     | YES   |
| wait/synch/mutex/innodb/autoinc_mutex                  | YES     | YES   |
...
| wait/synch/mutex/innodb/zip_pad_mutex                  | YES     | YES   |
+--------------------------------------------------------+---------+-------+
46 rows in set (0.00 sec)
```

3. Enable wait event consumers by updating the `setup_consumers` table. Wait event consumers are disabled by default.

```
mysql> UPDATE performance_schema.setup_consumers
       SET enabled = 'YES'
       WHERE name like 'events_waits%';
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

You can verify that wait event consumers are enabled by querying the `setup_consumers` table. The `events_waits_current`, `events_waits_history`, and `events_waits_history_long` consumers should be enabled.

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+--------------------------------+---------+
| NAME                           | ENABLED |
+--------------------------------+---------+
| events_stages_current          | NO      |
| events_stages_history          | NO      |
| events_stages_history_long     | NO      |
| events_statements_current      | YES     |
| events_statements_history      | NO      |
| events_statements_history_long | NO      |
| events_waits_current           | YES     |
| events_waits_history           | YES     |
| events_waits_history_long      | YES     |
| global_instrumentation         | YES     |
| thread_instrumentation         | YES     |
| statements_digest              | YES     |
+--------------------------------+---------+
12 rows in set (0.00 sec)
```

4. Once instruments and consumers are enabled, run the workload that you want to monitor. In this example, the `mysqlslap` load emulation client is used to simulate a workload.

```
shell> ./mysqlslap --auto-generate-sql --concurrency=100 --iterations=10
       --number-of-queries=1000 --number-char-cols=6 --number-int-cols=6;
```

5. Query the wait event data. In this example, wait event data is queried from the `events_waits_summary_global_by_event_name` table which aggregates data found in the `events_waits_current`, `events_waits_history`, and `events_waits_history_long` tables. Data is summarized by event name (`EVENT_NAME`), which is the name of the instrument that produced the event. Summarized data includes:

   - `COUNT_STAR`

     The number of summarized wait events.

   - `SUM_TIMER_WAIT`

     The total wait time of the summarized timed wait events.

   - `MIN_TIMER_WAIT`

     The minimum wait time of the summarized timed wait events.

   - `AVG_TIMER_WAIT`

     The average wait time of the summarized timed wait events.

   - `MAX_TIMER_WAIT`

     The maximum wait time of the summarized timed wait events.

   The following query returns the instrument name (`EVENT_NAME`), the number of wait events (`COUNT_STAR`), and the total wait time for the events for that instrument (`SUM_TIMER_WAIT`). Because waits are timed in picoseconds (trillionths of a second) by default, wait times are divided by 1000000000 to show wait times in milliseconds. Data is presented in descending order, by the number of summarized wait events (`COUNT_STAR`). You can adjust the `ORDER BY` clause to order the data by total wait time.

```
mysql> SELECT EVENT_NAME, COUNT_STAR, SUM_TIMER_WAIT/1000000000 SUM_TIMER_WAIT_MS
       FROM performance_schema.events_waits_summary_global_by_event_name
       WHERE SUM_TIMER_WAIT > 0 AND EVENT_NAME LIKE 'wait/synch/mutex/innodb/%'
```

```
       ORDER BY COUNT_STAR DESC;
+----------------------------------------------------+------------+-------------------+
| EVENT_NAME                                         | COUNT_STAR | SUM_TIMER_WAIT_MS |
+----------------------------------------------------+------------+-------------------+
| wait/synch/mutex/innodb/srv_threads_mutex          |     454474 |           16.3490 |
| wait/synch/mutex/innodb/trx_mutex                  |     107586 |            8.2415 |
| wait/synch/mutex/innodb/log_sys_mutex              |      37292 |          460.9484 |
| wait/synch/mutex/innodb/redo_rseg_mutex            |      31077 |            3.2212 |
| wait/synch/mutex/innodb/trx_sys_mutex              |      20005 |          447.8246 |
| wait/synch/mutex/innodb/lock_mutex                 |      19293 |          728.4862 |
| wait/synch/mutex/innodb/fil_system_mutex           |      15789 |            1.3710 |
| wait/synch/mutex/innodb/dict_sys_mutex             |       9606 |          319.7662 |
| wait/synch/mutex/innodb/rw_lock_list_mutex         |       8351 |            0.2046 |
| wait/synch/mutex/innodb/trx_undo_mutex             |       6274 |            0.5573 |
| wait/synch/mutex/innodb/buf_pool_mutex             |       3230 |            7.3861 |
| wait/synch/mutex/innodb/trx_pool_mutex             |       2225 |         9591.4174 |
| wait/synch/mutex/innodb/innobase_share_mutex       |       2025 |            0.1891 |
| wait/synch/mutex/innodb/flush_list_mutex           |       1421 |            0.1452 |
| wait/synch/mutex/innodb/trx_pool_manager_mutex     |       1115 |            0.1503 |
| wait/synch/mutex/innodb/file_format_max_mutex      |       1032 |            0.0445 |
| wait/synch/mutex/innodb/buf_dblwr_mutex            |        594 |            0.0950 |
| wait/synch/mutex/innodb/log_flush_order_mutex      |        502 |            0.0489 |
| wait/synch/mutex/innodb/srv_sys_mutex              |        417 |            0.6856 |
| wait/synch/mutex/innodb/recalc_pool_mutex          |        406 |            0.0842 |
| wait/synch/mutex/innodb/purge_sys_pq_mutex         |        365 |            0.0302 |
| wait/synch/mutex/innodb/recv_sys_mutex             |        273 |            0.0121 |
| wait/synch/mutex/innodb/lock_wait_mutex            |        170 |            0.0852 |
| wait/synch/mutex/innodb/noredo_rseg_mutex          |         32 |            0.0008 |
| wait/synch/mutex/innodb/ibuf_mutex                 |         10 |            0.0024 |
| wait/synch/mutex/innodb/srv_innodb_monitor_mutex   |          2 |            0.0005 |
| wait/synch/mutex/innodb/autoinc_mutex              |          2 |            0.0004 |
| wait/synch/mutex/innodb/recv_writer_mutex          |          1 |            0.0001 |
+----------------------------------------------------+------------+-------------------+
28 rows in set (0.03 sec)
```

Note

The preceding result set includes wait event data produced during the startup process. To exclude this data, you can truncate the `events_waits_summary_global_by_event_name` table immediately after startup and before running your workload. However, the truncate operation itself may produce a negligible amount wait event data.

```
mysql> TRUNCATE performance_schema.events_waits_summary_global_by_event_name;
```

14.17 InnoDB Monitors

`InnoDB` monitors provide information about the `InnoDB` internal state. This information is useful for performance tuning.

14.17.1 InnoDB Monitor Types

There are four types of `InnoDB` monitors:

- The standard `InnoDB` Monitor displays the following types of information:
  - Work done by the main background thread
  - Semaphore waits
  - Data about the most recent foreign key and deadlock errors
  - Lock waits for transactions
  - Table and record locks held by active transactions
  - Pending I/O operations and related statistics

- Insert buffer and adaptive hash index statistics
- Redo log data
- Buffer pool statistics
- Row operation data

- The `InnoDB` Lock Monitor prints additional lock information as part of the standard `InnoDB` Monitor output.
- The `InnoDB` Tablespace Monitor prints a list of file segments in the shared tablespace and validates the tablespace allocation data structures.
- The `InnoDB` Table Monitor prints the contents of the `InnoDB` internal data dictionary.

Note

The Tablespace Monitor and Table Monitor are deprecated; expect them to be removed in a future MySQL release. Similar information for the Table Monitor can be obtained from `InnoDB` `INFORMATION_SCHEMA` tables. See Section 21.30, “INFORMATION_SCHEMA InnoDB Tables”.

For additional information about `InnoDB` table and tablespace monitors, see Mark Leith: InnoDB Table and Tablespace Monitors.

14.17.2 Enabling InnoDB Monitors

When `InnoDB` monitors are enabled for periodic output, `InnoDB` writes the output to `mysqld` server standard error output (`stderr`) every 15 seconds, approximately.

`InnoDB` sends the monitor output to `stderr` rather than to `stdout` or fixed-size memory buffers to avoid potential buffer overflows.

On Windows, `stderr` is directed to the default log file unless configured otherwise. If you want to direct the output to the console window rather than to the error log, start the server from a command prompt in a console window with the `--console` option. For more information, see Section 5.4.2.1, “Error Logging on Windows”.

On Unix and Unix-like systems, `stderr` is typically directed to the terminal unless configured otherwise. For more information, see Section 5.4.2.2, “Error Logging on Unix and Unix-Like Systems”.

`InnoDB` monitors should only be enabled when you actually want to see monitor information because output generation causes some performance decrement. Also, if monitor output is directed to the error log, the log may become quite large if you forget to disable the monitor later by dropping the monitor table.

Note

To assist with troubleshooting, `InnoDB` temporarily enables standard `InnoDB` Monitor output under certain conditions. For more information, see Section 14.21, “InnoDB Troubleshooting”.

Each monitor begins with a header containing a timestamp and the monitor name. For example:

```
===================================
2014-10-16 16:28:15 7feee43c5700 INNODB MONITOR OUTPUT
===================================
```

The header for the standard `InnoDB` Monitor (`INNODB MONITOR OUTPUT`) is also used for the Lock Monitor because the latter produces the same output with the addition of extra lock information.

Enabling the InnoDB Standard Monitor and Lock Monitor for periodic output may be performed in one of two ways:

- Using a CREATE TABLE statement to create a specially named InnoDB table that is associated with the monitor. For example, to enable the standard InnoDB Monitor, you would create an InnoDB table named innodb_monitor. The Tablespace Monitor and Table Monitor are also enabled using the CREATE TABLE method.

  Using CREATE TABLE syntax is just a way to pass a command to the InnoDB engine through MySQL's SQL parser. The only things that matter are the table name and that it be an InnoDB table. The structure of the table and the database where the table is created are not relevant. If you shut down the server, the monitor does not restart automatically when you restart the server. Drop the monitor table and issue a new CREATE TABLE statement to start the monitor.

  Note

  The CREATE TABLE method of enabling InnoDB monitors is deprecated and may be removed in a future release. As of MySQL 5.6.16, you can enable the standard InnoDB Monitor and InnoDB Lock Monitor using the innodb_status_output and innodb_status_output_locks system variables.

- Using the innodb_status_output and innodb_status_output_locks system variables, introduced in MySQL 5.6.16.

The PROCESS privilege is required to enable or disable InnoDB Monitors.

Enabling the Standard InnoDB Monitor

To enable the standard InnoDB Monitor for periodic output, create the innodb_monitor table:

```
CREATE TABLE innodb_monitor (a INT) ENGINE=INNODB;
```

To disable the standard InnoDB Monitor, drop the table:

```
DROP TABLE innodb_monitor;
```

As of MySQL 5.6.16, you can also enable the standard InnoDB Monitor by setting the innodb_status_output system variable to ON.

```
SET GLOBAL innodb_status_output=ON;
```

To disable the standard InnoDB Monitor, set innodb_status_output to OFF.

When you shut down the server, the innodb_status_output variable is set to the default OFF value.

Enabling the InnoDB Lock Monitor

To enable the InnoDB Lock Monitor for periodic output, create the innodb_lock_monitor table:

```
CREATE TABLE innodb_lock_monitor (a INT) ENGINE=INNODB;
```

To disable the InnoDB Lock Monitor, drop the table:

```
DROP TABLE innodb_lock_monitor;
```

As of MySQL 5.6.16, you can also enable the InnoDB Lock Monitor by setting the innodb_status_output_locks system variable to ON. As with the CREATE TABLE method for enabling InnoDB Monitors, both the InnoDB standard Monitor and InnoDB Lock Monitor must be enabled to have InnoDBLock Monitor data printed periodically:

```
SET GLOBAL innodb_status_output=ON;
SET GLOBAL innodb_status_output_locks=ON;
```

When you shut down the server, the `innodb_status_output` and `innodb_status_output_locks` variables are set to the default `OFF` value.

To disable the `InnoDB` Lock Monitor, set `innodb_status_output_locks` to `OFF`. Set `innodb_status_output` to OFF to also disable the standard `InnoDB` Monitor.

Note

To enable the `InnoDB` Lock Monitor for `SHOW ENGINE INNODB STATUS` output, you are only required to enable `innodb_status_output_locks`.

Obtaining Standard InnoDB Monitor Output On Demand

As an alternative to enabling the standard `InnoDB` Monitor for periodic output, you can obtain standard `InnoDB` Monitor output on demand using the `SHOW ENGINE INNODB STATUS` SQL statement, which fetches the output to your client program. If you are using the `mysql` interactive client, the output is more readable if you replace the usual semicolon statement terminator with `\G`:

```
mysql> SHOW ENGINE INNODB STATUS\G
```

`SHOW ENGINE INNODB STATUS` output also includes `InnoDB` Lock Monitor data if the `InnoDB` Lock Monitor is enabled.

Directing Standard InnoDB Monitor Output to a Status File

Standard `InnoDB` Monitor output can be enabled and directed to a status file by specifying the `--innodb-status-file` option at startup. When this option is used, `InnoDB` creates a file named `innodb_status.pid` in the data directory and writes output to it every 15 seconds, approximately.

`InnoDB` removes the status file when the server is shut down normally. If an abnormal shutdown occurs, the status file may have to be removed manually.

The `--innodb-status-file` option is intended for temporary use, as output generation can affect performance, and the `innodb_status.pid` file can become quite large over time.

Enabling the InnoDB Tablespace Monitor

To enable the `InnoDB` Tablespace Monitor for periodic output, create the `innodb_tablespace_monitor` table:

```
CREATE TABLE innodb_tablespace_monitor (a INT) ENGINE=INNODB;
```

To disable the standard `InnoDB` Tablespace Monitor, drop the table:

```
DROP TABLE innodb_tablespace_monitor;
```

Note

The Tablespace Monitor is deprecated; expect it to be removed in a future MySQL release.

Enabling the InnoDB Table Monitor

To enable the `InnoDB` Table Monitor for periodic output, create the `innodb_table_monitor` table:

```
CREATE TABLE innodb_table_monitor (a INT) ENGINE=INNODB;
```

To disable the `InnoDB` Table Monitor, drop the table:

```
DROP TABLE innodb_table_monitor;
```

Note

The Table Monitor is deprecated; expect it to be removed in a future MySQL release.

14.17.3 InnoDB Standard Monitor and Lock Monitor Output

The Lock Monitor is the same as the Standard Monitor except that it includes additional lock information. Enabling either monitor for periodic output turns on the same output stream, but the stream includes extra information if the Lock Monitor is enabled. For example, if you enable the Standard Monitor and Lock Monitor, that turns on a single output stream. The stream includes extra lock information until you disable the Lock Monitor.

Standard Monitor output is limited to 1MB when produced using the `SHOW ENGINE INNODB STATUS` statement. This limit does not apply to output written to server standard error output (`stderr`).

Example Standard Monitor output:

```
mysql> SHOW ENGINE INNODB STATUS\G
*************************** 1. row ***************************
  Type: InnoDB
  Name:
Status:
=====================================
2014-10-17 10:33:50 7f47bcd64700 INNODB MONITOR OUTPUT
=====================================
Per second averages calculated from the last 6 seconds
-----------------
BACKGROUND THREAD
-----------------
srv_master_thread loops: 167 srv_active, 0 srv_shutdown, 3023 srv_idle
srv_master_thread log flush and writes: 3190
----------
SEMAPHORES
----------
OS WAIT ARRAY INFO: reservation count 1040
OS WAIT ARRAY INFO: signal count 959
Mutex spin waits 677, rounds 20336, OS waits 644
RW-shared spins 180, rounds 5400, OS waits 180
RW-excl spins 0, rounds 6420, OS waits 214
Spin rounds per wait: 30.04 mutex, 30.00 RW-shared, 6420.00 RW-excl
------------------------
LATEST FOREIGN KEY ERROR
------------------------
2014-10-17 09:51:31 7f47bcde6700 Transaction:
TRANSACTION 436786, ACTIVE 0 sec inserting
mysql tables in use 1, locked 1
4 lock struct(s), heap size 1184, 3 row lock(s), undo log entries 3
MySQL thread id 1, OS thread handle 0x7f47bcde6700, query id 96 localhost
root update
INSERT INTO child VALUES
    (NULL, 1)
    , (NULL, 2)
    , (NULL, 3)
    , (NULL, 4)
    , (NULL, 5)
    , (NULL, 6)
Foreign key constraint fails for table `mysql`.`child`:
,
  CONSTRAINT `child_ibfk_1` FOREIGN KEY (`parent_id`) REFERENCES `parent`
  (`id`)
 ON DELETE CASCADE ON UPDATE CASCADE
Trying to add in child table, in index `par_ind` tuple:
DATA TUPLE: 2 fields;
 0: len 4; hex 80000003; asc     ;;
 1: len 4; hex 80000003; asc     ;;

But in parent table `mysql`.`parent`, in index `PRIMARY`,
the closest match we can find is record:
PHYSICAL RECORD: n_fields 3; compact format; info bits 0
 0: len 4; hex 80000004; asc     ;;
 1: len 6; hex 00000006aa26; asc      &;;
 2: len 7; hex 9d000001610137; asc     a 7;;
------------------------
LATEST DETECTED DEADLOCK
```

```
------------------------
2014-10-17 09:52:38 7f47bcde6700
*** (1) TRANSACTION:
TRANSACTION 436801, ACTIVE 12 sec starting index read
mysql tables in use 1, locked 1
LOCK WAIT 2 lock struct(s), heap size 360, 1 row lock(s)
MySQL thread id 2, OS thread handle 0x7f47bcda5700, query id 102 localhost
root updating
DELETE FROM t WHERE i = 1
*** (1) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 3693 page no 3 n bits 72 index `GEN_CLUST_INDEX` of
table `mysql`.`t` trx id 436801 lock_mode X waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 4; compact format; info
bits 0
 0: len 6; hex 000000003a00; asc     : ;;
 1: len 6; hex 00000006aa3f; asc      ?;;
 2: len 7; hex ad0000021d0110; asc        ;;
 3: len 4; hex 80000001; asc     ;;

*** (2) TRANSACTION:
TRANSACTION 436800, ACTIVE 34 sec starting index read
mysql tables in use 1, locked 1
4 lock struct(s), heap size 1184, 3 row lock(s)
MySQL thread id 1, OS thread handle 0x7f47bcde6700, query id 103 localhost
root updating
DELETE FROM t WHERE i = 1
*** (2) HOLDS THE LOCK(S):
RECORD LOCKS space id 3693 page no 3 n bits 72 index `GEN_CLUST_INDEX` of
table `mysql`.`t` trx id 436800 lock mode S
Record lock, heap no 1 PHYSICAL RECORD: n_fields 1; compact format; info
bits 0 0: len 8; hex 73757072656d756d; asc supremum;;

Record lock, heap no 2 PHYSICAL RECORD: n_fields 4; compact format; info
bits 0
 0: len 6; hex 000000003a00; asc     : ;;
 1: len 6; hex 00000006aa3f; asc      ?;;
 2: len 7; hex ad0000021d0110; asc        ;;
 3: len 4; hex 80000001; asc     ;;

*** (2) WAITING FOR THIS LOCK TO BE GRANTED:
RECORD LOCKS space id 3693 page no 3 n bits 72 index `GEN_CLUST_INDEX` of
table `mysql`.`t` trx id 436800 lock_mode X waiting
Record lock, heap no 2 PHYSICAL RECORD: n_fields 4; compact format; info
bits 0
 0: len 6; hex 000000003a00; asc     : ;;
 1: len 6; hex 00000006aa3f; asc      ?;;
 2: len 7; hex ad0000021d0110; asc        ;;
 3: len 4; hex 80000001; asc     ;;

*** WE ROLL BACK TRANSACTION (1)
------------
TRANSACTIONS
------------
Trx id counter 437661
Purge done for trx's n:o < 437657 undo n:o < 0 state: running but
idle History list length 371
LIST OF TRANSACTIONS FOR EACH SESSION:
---TRANSACTION 0, not started
MySQL thread id 10, OS thread handle 0x7f47bcd64700, query id 1001 localhost
root init
SHOW ENGINE INNODB STATUS
---TRANSACTION 436801, not started
MySQL thread id 2, OS thread handle 0x7f47bcda5700, query id 102 localhost
root ceaning up
---TRANSACTION 437660, ACTIVE 0 sec inserting
mysql tables in use 1, locked 1
43 lock struct(s), heap size 6544, 6474 row lock(s), undo log entries 7124
MySQL thread id 14, OS thread handle 0x7f47bcde6700, query id 1000 localhost
root update
INSERT INTO `dept_emp` VALUES (100258,'d002','1994-03-21','9999-01-01'),
(100259, 'd005','1998-11-04','9999-01-01'),(100259,'d008','1988-02-03',
'1998-11-04'),(100 260,'d005','1998-09-18','9999-01-01'),(100261,'d004',
```

```
'1989-03-11','9999-01-01'), (100262,'d008','1996-08-12','9999-01-01'),
(100263,'d002','1998-06-24','1998-10-0 5'),(100264,'d005','1989-11-09',
'9999-01-01'),(100265,'d001','1992-06-27','9999- 01-01'),(100266,'d009',
'1990-09-10','9999-01-01'),(100267,'d009','1992-04-14','9 999-01-01'),
(100268,'d005','1998-05-01','2000-04-07'),(100269,'d007','1994-01-02',
'1999-09-18'),(100269,'d009','1999-09-
--------
FILE I/O
--------
I/O thread 0 state: waiting for completed aio requests (insert buffer thread)
I/O thread 1 state: waiting for completed aio requests (log thread)
I/O thread 2 state: waiting for completed aio requests (read thread)
I/O thread 3 state: waiting for completed aio requests (read thread)
I/O thread 4 state: waiting for completed aio requests (read thread)
I/O thread 5 state: waiting for completed aio requests (read thread)
I/O thread 6 state: waiting for completed aio requests (write thread)
I/O thread 7 state: waiting for completed aio requests (write thread)
I/O thread 8 state: waiting for completed aio requests (write thread)
I/O thread 9 state: waiting for completed aio requests (write thread)
Pending normal aio reads: 0 [0, 0, 0, 0] , aio writes: 0 [0, 0, 0, 0] ,
 ibuf aio reads: 0, log i/o's: 0, sync i/o's: 0
Pending flushes (fsync) log: 0; buffer pool: 0
344 OS file reads, 45666 OS file writes, 4030 OS fsyncs
0.00 reads/s, 0 avg bytes/read, 202.80 writes/s, 48.33 fsyncs/s
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf: size 1, free list len 0, seg size 2, 0 merges
merged operations:
 insert 0, delete mark 0, delete 0
discarded operations:
 insert 0, delete mark 0, delete 0
Hash table size 4425293, node heap has 143 buffer(s)
137083.82 hash searches/s, 2495.92 non-hash searches/s
---
LOG
---
Log sequence number 3091027710
Log flushed up to   3090240098
Pages flushed up to 3074432960
Last checkpoint at  3050856266
0 pending log writes, 0 pending chkp writes
1187 log i/o's done, 14.67 log i/o's/second
----------------------
BUFFER POOL AND MEMORY
----------------------
Total memory allocated 2197815296; in additional pool allocated 0
Dictionary memory allocated 155455
Buffer pool size   131071
Free buffers       92158
Database pages     38770
Old database pages 14271
Modified db pages  619
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 4, not young 0
0.00 youngs/s, 0.00 non-youngs/s
Pages read 322, created 38448, written 42083
0.00 reads/s, 222.30 creates/s, 159.47 writes/s
Buffer pool hit rate 1000 / 1000, young-making rate 0 / 1000 not 0 / 1000
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead
0.00/s
LRU len: 38770, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0]
----------------------
INDIVIDUAL BUFFER POOL INFO
----------------------
---BUFFER POOL 0
Buffer pool size   65536
Free buffers       46120
Database pages     19345
Old database pages 7121
```

```
Modified db pages  291
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 3, not young 0
0.00 youngs/s, 0.00 non-youngs/s
Pages read 163, created 19182, written 21149
0.00 reads/s, 103.48 creates/s, 83.15 writes/s
Buffer pool hit rate 1000 / 1000, young-making rate 0 / 1000 not 0 / 1000
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead
0.00/s
LRU len: 19345, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0]
---BUFFER POOL 1
Buffer pool size   65535
Free buffers       46038
Database pages     19425
Old database pages 7150
Modified db pages  328
Pending reads 0
Pending writes: LRU 0, flush list 0, single page 0
Pages made young 1, not young 0
0.00 youngs/s, 0.00 non-youngs/s
Pages read 159, created 19266, written 20934
0.00 reads/s, 118.81 creates/s, 76.32 writes/s
Buffer pool hit rate 1000 / 1000, young-making rate 0 / 1000 not 0 / 1000
Pages read ahead 0.00/s, evicted without access 0.00/s, Random read ahead
0.00/s
LRU len: 19425, unzip_LRU len: 0
I/O sum[0]:cur[0], unzip sum[0]:cur[0]
--------------
ROW OPERATIONS
--------------
0 queries inside InnoDB, 0 queries in queue
0 read views open inside InnoDB
Main thread process no. 54607, id 139946075744000, state: sleeping
Number of rows inserted 12163964, updated 0, deleted 3, read 4
67807.03 inserts/s, 0.00 updates/s, 0.00 deletes/s, 0.00 reads/s
----------------------------
END OF INNODB MONITOR OUTPUT
============================
```

Standard Monitor Output Sections

For a description of each metric reported by the Standard Monitor, refer to the Metrics chapter in the Oracle Enterprise Manager for MySQL Database User's Guide.

- `Status`

  This section shows the timestamp, the monitor name, and the number of seconds that per-second averages are based on. The number of seconds is the elapsed time between the current time and the last time `InnoDB` Monitor output was printed.

- `BACKGROUND THREAD`

  The `srv_master_thread` lines shows work done by the main background thread.

- `SEMAPHORES`

  This section reports threads waiting for a semaphore and statistics on how many times threads have needed a spin or a wait on a mutex or a rw-lock semaphore. A large number of threads waiting for semaphores may be a result of disk I/O, or contention problems inside `InnoDB`. Contention can be due to heavy parallelism of queries or problems in operating system thread scheduling. Setting the `innodb_thread_concurrency` system variable smaller than the default value might help in such situations. The `Spin rounds per wait` line shows the number of spinlock rounds per OS wait for a mutex.

- `LATEST FOREIGN KEY ERROR`

This section provides information about the most recent foreign key constraint error. It is not present if no such error has occurred. The contents include the statement that failed as well as information about the constraint that failed and the referenced and referencing tables.

- `LATEST DETECTED DEADLOCK`

  This section provides information about the most recent deadlock. It is not present if no deadlock has occurred. The contents show which transactions are involved, the statement each was attempting to execute, the locks they have and need, and which transaction `InnoDB` decided to roll back to break the deadlock. The lock modes reported in this section are explained in Section 14.7.1, “InnoDB Locking”.

- `TRANSACTIONS`

  If this section reports lock waits, your applications might have lock contention. The output can also help to trace the reasons for transaction deadlocks.

- `FILE I/O`

  This section provides information about threads that `InnoDB` uses to perform various types of I/O. The first few of these are dedicated to general `InnoDB` processing. The contents also display information for pending I/O operations and statistics for I/O performance.

  The number of these threads are controlled by the `innodb_read_io_threads` and `innodb_write_io_threads` parameters. See Section 14.14, “InnoDB Startup Options and System Variables”.

- `INSERT BUFFER AND ADAPTIVE HASH INDEX`

  This section shows the status of the `InnoDB` insert buffer (also referred to as the change buffer) and the adaptive hash index.

  For related information, see Section 14.5.2, “Change Buffer”, and Section 14.5.3, “Adaptive Hash Index”.

- `LOG`

  This section displays information about the `InnoDB` log. The contents include the current log sequence number, how far the log has been flushed to disk, and the position at which `InnoDB` last took a checkpoint. (See Section 14.12.3, “InnoDB Checkpoints”.) The section also displays information about pending writes and write performance statistics.

- `BUFFER POOL AND MEMORY`

  This section gives you statistics on pages read and written. You can calculate from these numbers how many data file I/O operations your queries currently are doing.

  For buffer pool statistics descriptions, see Monitoring the Buffer Pool Using the InnoDB Standard Monitor. For additional information about the operation of the buffer pool, see Section 14.5.1, “Buffer Pool”.

- `ROW OPERATIONS`

  This section shows what the main thread is doing, including the number and performance rate for each type of row operation.

14.17.4 InnoDB Tablespace Monitor Output

Note

The `InnoDB` Tablespace Monitor is deprecated and may be removed in a future release.

The `InnoDB` Tablespace Monitor prints information about the file segments in the shared tablespace and validates the tablespace allocation data structures. The Tablespace Monitor does not describe file-per-table tablespaces created with the `innodb_file_per_table` option.

Example `InnoDB` Tablespace Monitor output:

```
=====================================================
090408 21:28:09 INNODB TABLESPACE MONITOR OUTPUT
=====================================================
FILE SPACE INFO: id 0
size 13440, free limit 3136, free extents 28
not full frag extents 2: used pages 78, full frag extents 3
first seg id not used 0 23845
SEGMENT id 0 1 space 0; page 2; res 96 used 46; full ext 0
fragm pages 32; free extents 0; not full extents 1: pages 14
SEGMENT id 0 2 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
SEGMENT id 0 3 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
...
SEGMENT id 0 15 space 0; page 2; res 160 used 160; full ext 2
fragm pages 32; free extents 0; not full extents 0: pages 0
SEGMENT id 0 488 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
SEGMENT id 0 17 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
...
SEGMENT id 0 171 space 0; page 2; res 592 used 481; full ext 7
fragm pages 16; free extents 0; not full extents 2: pages 17
SEGMENT id 0 172 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
SEGMENT id 0 173 space 0; page 2; res 96 used 44; full ext 0
fragm pages 32; free extents 0; not full extents 1: pages 12
...
SEGMENT id 0 601 space 0; page 2; res 1 used 1; full ext 0
fragm pages 1; free extents 0; not full extents 0: pages 0
NUMBER of file segments: 73
Validating tablespace
Validation ok
---------------------------------------
END OF INNODB TABLESPACE MONITOR OUTPUT
=======================================
```

The Tablespace Monitor output includes information about the shared tablespace as a whole, followed by a list containing a breakdown for each segment within the tablespace.

In this example using the default page size, the tablespace consists of database pages that are 16KB each. The pages are grouped into extents of size 1MB (64 consecutive pages).

The initial part of the output that displays overall tablespace information has this format:

```
FILE SPACE INFO: id 0
size 13440, free limit 3136, free extents 28
not full frag extents 2: used pages 78, full frag extents 3
first seg id not used 0 23845
```

Overall tablespace information includes these values:

- `id`

  The tablespace ID. A value of 0 refers to the shared tablespace.

- `size`

The current tablespace size in pages.

- `free limit`

  The minimum page number for which the free list has not been initialized. Pages at or above this limit are free.

- `free extents`

  The number of free extents.

- `not full frag extents`, `used pages`

  The number of fragment extents that are not completely filled, and the number of pages in those extents that have been allocated.

- `full frag extents`

  The number of completely full fragment extents.

- `first seg id not used`

  The first unused segment ID.

Individual segment information has this format:

```
SEGMENT id 0 15 space 0; page 2; res 160 used 160; full ext 2
fragm pages 32; free extents 0; not full extents 0: pages 0
```

Segment information includes these values:

- `id`:

  The segment ID.

- `space`, `page`

  The tablespace number and page within the tablespace where the segment “inode” is located. A tablespace number of 0 indicates the shared tablespace. `InnoDB` uses inodes to keep track of segments in the tablespace. The other fields displayed for a segment (`id`, `res`, and so forth) are derived from information in the inode.

- `res`

  The number of pages allocated (reserved) for the segment.

- `used`

  The number of allocated pages in use by the segment.

- `full ext`

  The number of extents allocated for the segment that are completely used.

- `fragm pages`

  The number of initial pages that have been allocated to the segment.

- `free extents`

  The number of extents allocated for the segment that are completely unused.

- `not full extents`

  The number of extents allocated for the segment that are partially used.

- `pages`

  The number of pages used within the not-full extents.

When a segment grows, it starts as a single page, and `InnoDB` allocates the first pages for it one at a time, up to 32 pages (this is the `fragm pages` value). After that, `InnoDB` allocates complete extents. `InnoDB` can add up to 4 extents at a time to a large segment to ensure good sequentiality of data.

For the example segment shown earlier, it has 32 fragment pages, plus 2 full extents (64 pages each), for a total of 160 pages used out of 160 pages allocated. The following segment has 32 fragment pages and one partially full extent using 14 pages for a total of 46 pages used out of 96 pages allocated:

```
SEGMENT id 0 1 space 0; page 2; res 96 used 46; full ext 0
fragm pages 32; free extents 0; not full extents 1: pages 14
```

It is possible for a segment that has extents allocated to it to have a `fragm pages` value less than 32 if some of the individual pages have been deallocated subsequent to extent allocation.

14.17.5 InnoDB Table Monitor Output

Note

The `InnoDB` Table Monitor is deprecated and may be removed in a future release. Similar information can be obtained from `InnoDB` `INFORMATION_SCHEMA` tables. See Section 21.30, “INFORMATION_SCHEMA InnoDB Tables”.

The `InnoDB` Table Monitor prints the contents of the `InnoDB` internal data dictionary.

The output contains one section per table. The `SYS_FOREIGN` and `SYS_FOREIGN_COLS` sections are for internal data dictionary tables that maintain information about foreign keys. There are also sections for the Table Monitor table and each user-created `InnoDB` table. Suppose that the following two tables have been created in the `test` database:

```
CREATE TABLE parent
(
  par_id    INT NOT NULL,
  fname      CHAR(20),
  lname      CHAR(20),
  PRIMARY KEY (par_id),
  UNIQUE INDEX (lname, fname)
) ENGINE = INNODB;

CREATE TABLE child
(
  par_id      INT NOT NULL,
  child_id    INT NOT NULL,
  name        VARCHAR(40),
  birth       DATE,
  weight      DECIMAL(10,2),
  misc_info   VARCHAR(255),
  last_update TIMESTAMP,
  PRIMARY KEY (par_id, child_id),
  INDEX (name),
  FOREIGN KEY (par_id) REFERENCES parent (par_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
) ENGINE = INNODB;
```

Then the Table Monitor output looks something like this (reformatted slightly):

```
===========================================
090420 12:09:32 INNODB TABLE MONITOR OUTPUT
===========================================
-------------------------------------
TABLE: name SYS_FOREIGN, id 0 11, columns 7, indexes 3, appr.rows 1
  COLUMNS: ID: DATA_VARCHAR DATA_ENGLISH len 0;
```

```
           FOR_NAME: DATA_VARCHAR DATA_ENGLISH len 0;
           REF_NAME: DATA_VARCHAR DATA_ENGLISH len 0;
           N_COLS: DATA_INT len 4;
           DB_ROW_ID: DATA_SYS prtype 256 len 6;
           DB_TRX_ID: DATA_SYS prtype 257 len 6;
  INDEX: name ID_IND, id 0 11, fields 1/6, uniq 1, type 3
   root page 46, appr.key vals 1, leaf pages 1, size pages 1
   FIELDS:  ID DB_TRX_ID DB_ROLL_PTR FOR_NAME REF_NAME N_COLS
  INDEX: name FOR_IND, id 0 12, fields 1/2, uniq 2, type 0
   root page 47, appr.key vals 1, leaf pages 1, size pages 1
   FIELDS:  FOR_NAME ID
  INDEX: name REF_IND, id 0 13, fields 1/2, uniq 2, type 0
   root page 48, appr.key vals 1, leaf pages 1, size pages 1
   FIELDS:  REF_NAME ID
-------------------------------------
TABLE: name SYS_FOREIGN_COLS, id 0 12, columns 7, indexes 1, appr.rows 1
  COLUMNS: ID: DATA_VARCHAR DATA_ENGLISH len 0;
           POS: DATA_INT len 4;
           FOR_COL_NAME: DATA_VARCHAR DATA_ENGLISH len 0;
           REF_COL_NAME: DATA_VARCHAR DATA_ENGLISH len 0;
           DB_ROW_ID: DATA_SYS prtype 256 len 6;
           DB_TRX_ID: DATA_SYS prtype 257 len 6;
  INDEX: name ID_IND, id 0 14, fields 2/6, uniq 2, type 3
   root page 49, appr.key vals 1, leaf pages 1, size pages 1
   FIELDS:  ID POS DB_TRX_ID DB_ROLL_PTR FOR_COL_NAME REF_COL_NAME
-------------------------------------
TABLE: name test/child, id 0 14, columns 10, indexes 2, appr.rows 201
  COLUMNS: par_id: DATA_INT DATA_BINARY_TYPE DATA_NOT_NULL len 4;
           child_id: DATA_INT DATA_BINARY_TYPE DATA_NOT_NULL len 4;
           name: DATA_VARCHAR prtype 524303 len 40;
           birth: DATA_INT DATA_BINARY_TYPE len 3;
           weight: DATA_FIXBINARY DATA_BINARY_TYPE len 5;
           misc_info: DATA_VARCHAR prtype 524303 len 255;
           last_update: DATA_INT DATA_UNSIGNED DATA_BINARY_TYPE DATA_NOT_NULL len 4;
           DB_ROW_ID: DATA_SYS prtype 256 len 6;
           DB_TRX_ID: DATA_SYS prtype 257 len 6;
  INDEX: name PRIMARY, id 0 17, fields 2/9, uniq 2, type 3
   root page 52, appr.key vals 201, leaf pages 5, size pages 6
   FIELDS:  par_id child_id DB_TRX_ID DB_ROLL_PTR name birth weight misc_info last_update
  INDEX: name name, id 0 18, fields 1/3, uniq 3, type 0
   root page 53, appr.key vals 210, leaf pages 1, size pages 1
   FIELDS:  name par_id child_id
  FOREIGN KEY CONSTRAINT test/child_ibfk_1: test/child ( par_id )
             REFERENCES test/parent ( par_id )
-------------------------------------
TABLE: name test/innodb_table_monitor, id 0 15, columns 4, indexes 1, appr.rows 0
  COLUMNS: i: DATA_INT DATA_BINARY_TYPE len 4;
           DB_ROW_ID: DATA_SYS prtype 256 len 6;
           DB_TRX_ID: DATA_SYS prtype 257 len 6;
  INDEX: name GEN_CLUST_INDEX, id 0 19, fields 0/4, uniq 1, type 1
   root page 193, appr.key vals 0, leaf pages 1, size pages 1
   FIELDS:  DB_ROW_ID DB_TRX_ID DB_ROLL_PTR i
-------------------------------------
TABLE: name test/parent, id 0 13, columns 6, indexes 2, appr.rows 299
  COLUMNS: par_id: DATA_INT DATA_BINARY_TYPE DATA_NOT_NULL len 4;
           fname: DATA_CHAR prtype 524542 len 20;
           lname: DATA_CHAR prtype 524542 len 20;
           DB_ROW_ID: DATA_SYS prtype 256 len 6;
           DB_TRX_ID: DATA_SYS prtype 257 len 6;
  INDEX: name PRIMARY, id 0 15, fields 1/5, uniq 1, type 3
   root page 50, appr.key vals 299, leaf pages 2, size pages 3
   FIELDS:  par_id DB_TRX_ID DB_ROLL_PTR fname lname
  INDEX: name lname, id 0 16, fields 2/3, uniq 2, type 2
   root page 51, appr.key vals 300, leaf pages 1, size pages 1
   FIELDS:  lname fname par_id
  FOREIGN KEY CONSTRAINT test/child_ibfk_1: test/child ( par_id )
             REFERENCES test/parent ( par_id )
-----------------------------------
END OF INNODB TABLE MONITOR OUTPUT
===================================
```

For each table, Table Monitor output contains a section that displays general information about the table and specific information about its columns, indexes, and foreign keys.

The general information for each table includes the table name (in *db_name/tbl_name* format except for internal tables), its ID, the number of columns and indexes, and an approximate row count.

The `COLUMNS` part of a table section lists each column in the table. Information for each column indicates its name and data type characteristics. Some internal columns are added by `InnoDB`, such as `DB_ROW_ID` (row ID), `DB_TRX_ID` (transaction ID), and `DB_ROLL_PTR` (a pointer to the rollback/undo data).

- `DATA_`*`xxx`*

  These symbols indicate the data type. There may be multiple `DATA_`*`xxx`* symbols for a given column.

- `prtype`

  The column's “precise” type. This field includes information such as the column data type, character set code, nullability, signedness, and whether it is a binary string. This field is described in the `innobase/include/data0type.h` source file.

- `len`

  The column length in bytes.

Each `INDEX` part of the table section provides the name and characteristics of one table index:

- `name`

  The index name. If the name is `PRIMARY`, the index is a primary key. If the name is `GEN_CLUST_INDEX`, the index is the clustered index that is created automatically if the table definition doesn't include a primary key or non-`NULL` unique index. See Section 14.6.2.1, “Clustered and Secondary Indexes”.

- `id`

  The index ID.

- `fields`

  The number of fields in the index, as a value in *`m/n`* format:

  - *`m`* is the number of user-defined columns; that is, the number of columns you would see in the index definition in a `CREATE TABLE` statement.

  - *`n`* is the total number of index columns, including those added internally. For the clustered index, the total includes the other columns in the table definition, plus any columns added internally. For a secondary index, the total includes the columns from the primary key that are not part of the secondary index.

- `uniq`

  The number of leading fields that are enough to determine index values uniquely.

- `type`

  The index type. This is a bit field. For example, 1 indicates a clustered index and 2 indicates a unique index, so a clustered index (which always contains unique values), has the `type` value 3. An index with `type` value 0 is neither clustered nor unique. The flag values are defined in the `innobase/include/dict0mem.h` source file.

- `root page`

The index root page number.

- `appr. key vals`

  The approximate index cardinality.

- `leaf pages`

  The approximate number of leaf pages in the index.

- `size pages`

  The approximate total number of pages in the index.

- `FIELDS`

  The names of the fields in the index. For a clustered index that was generated automatically, the field list begins with the internal `DB_ROW_ID` (row ID) field. `DB_TRX_ID` and `DB_ROLL_PTR` are always added internally to the clustered index, following the fields that comprise the primary key. For a secondary index, the final fields are those from the primary key that are not part of the secondary index.

The end of the table section lists the `FOREIGN KEY` definitions that apply to the table. This information appears whether the table is a referencing or referenced table.

14.18 InnoDB Backup and Recovery

This section covers topics related to `InnoDB` backup and recovery.

- For information about backup techniques applicable to `InnoDB`, see Section 14.18.1, “InnoDB Backup”.
- For information about point-in-time recovery, recovery from disk failure or corruption, and how `InnoDB` performs crash recovery, see Section 14.18.2, “InnoDB Recovery”.

14.18.1 InnoDB Backup

The key to safe database management is making regular backups. Depending on your data volume, number of MySQL servers, and database workload, you can use these backup techniques, alone or in combination: hot backup with MySQL Enterprise Backup; cold backup by copying files while the MySQL server is shut down; logical backup with `mysqldump` for smaller data volumes or to record the structure of schema objects. Hot and cold backups are physical backups that copy actual data files, which can be used directly by the `mysqld` server for faster restore.

Using *MySQL Enterprise Backup* is the recommended method for backing up `InnoDB` data.

Note

`InnoDB` does not support databases that are restored using third-party backup tools.

Hot Backups

The `mysqlbackup` command, part of the MySQL Enterprise Backup component, lets you back up a running MySQL instance, including `InnoDB` tables, with minimal disruption to operations while producing a consistent snapshot of the database. When `mysqlbackup` is copying `InnoDB` tables, reads and writes to `InnoDB` tables can continue. MySQL Enterprise Backup can also create compressed backup files, and back up subsets of tables and databases. In conjunction with the MySQL binary log, users can perform point-in-time recovery. MySQL Enterprise Backup is part of the MySQL Enterprise subscription. For more details, see Section 24.2, “MySQL Enterprise Backup Overview”.

Cold Backups

If you can shut down the MySQL server, you can make a physical backup that consists of all files used by `InnoDB` to manage its tables. Use the following procedure:

1. Perform a slow shutdown of the MySQL server and make sure that it stops without errors.
2. Copy all `InnoDB` data files (`ibdata` files and `.ibd` files) into a safe place.
3. Copy all the `.frm` files for `InnoDB` tables to a safe place.
4. Copy all `InnoDB` log files (`ib_logfile` files) to a safe place.
5. Copy your `my.cnf` configuration file or files to a safe place.

Logical Backups Using mysqldump

In addition to physical backups, it is recommended that you regularly create logical backups by dumping your tables using `mysqldump`. A binary file might be corrupted without you noticing it. Dumped tables are stored into text files that are human-readable, so spotting table corruption becomes easier. Also, because the format is simpler, the chance for serious data corruption is smaller. `mysqldump` also has a `--single-transaction` option for making a consistent snapshot without locking out other clients. See Section 7.3.1, “Establishing a Backup Policy”.

Replication works with `InnoDB` tables, so you can use MySQL replication capabilities to keep a copy of your database at database sites requiring high availability. See Section 14.19, “InnoDB and MySQL Replication”.

14.18.2 InnoDB Recovery

This section describes `InnoDB` recovery. Topics include:

- Point-in-Time Recovery
- Recovery from Data Corruption or Disk Failure
- InnoDB Crash Recovery

Point-in-Time Recovery

To recover an `InnoDB` database to the present from the time at which the physical backup was made, you must run MySQL server with binary logging enabled, even before taking the backup. To achieve point-in-time recovery after restoring a backup, you can apply changes from the binary log that occurred after the backup was made. See Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”.

Recovery from Data Corruption or Disk Failure

If your database becomes corrupted or disk failure occurs, you must perform the recovery using a backup. In the case of corruption, first find a backup that is not corrupted. After restoring the base backup, do a point-in-time recovery from the binary log files using `mysqlbinlog` and `mysql` to restore the changes that occurred after the backup was made.

In some cases of database corruption, it is enough to dump, drop, and re-create one or a few corrupt tables. You can use the `CHECK TABLE` statement to check whether a table is corrupt, although `CHECK TABLE` naturally cannot detect every possible kind of corruption. You can use the Tablespace Monitor to check the integrity of the file space management inside the tablespace files.

In some cases, apparent database page corruption is actually due to the operating system corrupting its own file cache, and the data on disk may be okay. It is best to try restarting the computer first. Doing so may eliminate errors that appeared to be database page corruption. If MySQL still has trouble

starting because of InnoDB consistency problems, see Section 14.21.2, “Forcing InnoDB Recovery” for steps to start the instance in recovery mode, which permits you to dump the data.

InnoDB Crash Recovery

To recover from an unexpected MySQL server exit, the only requirement is to restart the MySQL server. InnoDB automatically checks the logs and performs a roll-forward of the database to the present. InnoDB automatically rolls back uncommitted transactions that were present at the time of the crash. During recovery, mysqld displays output similar to this:

```
InnoDB: Log scan progressed past the checkpoint lsn 430875675
InnoDB: Database was not shutdown normally!
InnoDB: Starting crash recovery.
InnoDB: Reading tablespace information from the .ibd files...
InnoDB: Restoring possible half-written data pages
InnoDB: from the doublewrite buffer...
InnoDB: Doing recovery: scanned up to log sequence number 436118528
InnoDB: Doing recovery: scanned up to log sequence number 441361408
InnoDB: Doing recovery: scanned up to log sequence number 446604288
InnoDB: Doing recovery: scanned up to log sequence number 451847168
InnoDB: Doing recovery: scanned up to log sequence number 457090048
InnoDB: Doing recovery: scanned up to log sequence number 462332928
InnoDB: Doing recovery: scanned up to log sequence number 467575808
InnoDB: Doing recovery: scanned up to log sequence number 472818688
InnoDB: Doing recovery: scanned up to log sequence number 478061568
InnoDB: Starting an apply batch of log records to the database...
InnoDB: Progress in percent: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
InnoDB: Apply batch completed
InnoDB: Doing recovery: scanned up to log sequence number 483304448
InnoDB: Doing recovery: scanned up to log sequence number 488547328
InnoDB: Doing recovery: scanned up to log sequence number 493790208
InnoDB: Doing recovery: scanned up to log sequence number 496426509
InnoDB: 1 transaction(s) which must be rolled back or cleaned up
InnoDB: in total 1441473 row operations to undo
InnoDB: Trx id counter is 2304
InnoDB: Starting an apply batch of log records to the database...
InnoDB: Progress in percent: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37
38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59
60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81
82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99
InnoDB: Apply batch completed
...
InnoDB: Waiting for purge to start
Starting in background the rollback of uncommitted transactions
InnoDB: Rolling back trx with id 2022, 1441473 rows to undo
...
InnoDB: 5.6.36 started; log sequence number 496426509
...
./mysqld: ready for connections.
```

The InnoDB crash recovery process consists of several steps:

- Redo log application

  Redo log application is the first step and is performed during initialization, before accepting any connections. If all changes are flushed from the buffer pool to the tablespaces (ibdata* and *.ibd files) at the time of the shutdown or crash, redo log application is skipped. InnoDB also skips redo log application if redo log files are missing at startup.

  Removing redo logs to speed up recovery is not recommended, even if some data loss is acceptable. Removing redo logs should only be considered after a clean shutdown, with innodb_fast_shutdown set to 0 or 1.

- Roll back of incomplete transactions

  Incomplete transactions are any transactions that were active at the time of unexpected exit or fast shutdown. The time it takes to roll back an incomplete transaction can be three or four times the amount of time a transaction is active before it is interrupted, depending on server load.

  You cannot cancel transactions that are being rolled back. In extreme cases, when rolling back transactions is expected to take an exceptionally long time, it may be faster to start `InnoDB` with an `innodb_force_recovery` setting of `3` or greater. See Section 14.21.2, “Forcing InnoDB Recovery”.

- Change buffer merge

  Applying changes from the change buffer (part of the system tablespace) to leaf pages of secondary indexes, as the index pages are read to the buffer pool.

- Purge

  Deleting delete-marked records that are no longer visible to active transactions.

The steps that follow redo log application do not depend on the redo log (other than for logging the writes) and are performed in parallel with normal processing. Of these, only rollback of incomplete transactions is special to crash recovery. The insert buffer merge and the purge are performed during normal processing.

After redo log application, `InnoDB` attempts to accept connections as early as possible, to reduce downtime. As part of crash recovery, `InnoDB` rolls back transactions that were not committed or in `XA PREPARE` state when the server exited. The rollback is performed by a background thread, executed in parallel with transactions from new connections. Until the rollback operation is completed, new connections may encounter locking conflicts with recovered transactions.

In most situations, even if the MySQL server was killed unexpectedly in the middle of heavy activity, the recovery process happens automatically and no action is required of the DBA. If a hardware failure or severe system error corrupted `InnoDB` data, MySQL might refuse to start. In this case, see Section 14.21.2, “Forcing InnoDB Recovery”.

For information about the binary log and `InnoDB` crash recovery, see Section 5.4.4, “The Binary Log”.

14.19 InnoDB and MySQL Replication

It is possible to use replication in a way where the storage engine on the replica is not the same as the storage engine on the source. For example, you can replicate modifications to an `InnoDB` table on the source to a `MyISAM` table on the replica. For more information see, Section 17.3.3, “Using Replication with Different Source and Replica Storage Engines”.

For information about setting up a replica, see Section 17.1.1.8, “Setting Up Replication with Existing Data”. To make a new replica without taking down the source or an existing replica, use the MySQL Enterprise Backup product.

Transactions that fail on the source do not affect replication. MySQL replication is based on the binary log where MySQL writes SQL statements that modify data. A transaction that fails (for example, because of a foreign key violation, or because it is rolled back) is not written to the binary log, so it is not sent to replicas. See Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”.

Replication and CASCADE. Cascading actions for `InnoDB` tables on the source are replicated on the replica *only* if the tables sharing the foreign key relation use `InnoDB` on both the source and replica. This is true whether you are using statement-based or row-based replication. Suppose that you have started replication, and then create two tables on the source using the following `CREATE TABLE` statements:

```
CREATE TABLE fc1 (
    i INT PRIMARY KEY,
    j INT
) ENGINE = InnoDB;

CREATE TABLE fc2 (
    m INT PRIMARY KEY,
    n INT,
    FOREIGN KEY ni (n) REFERENCES fc1 (i)
        ON DELETE CASCADE
) ENGINE = InnoDB;
```

If the replica does not have `InnoDB` support enabled, the tables on the replica are created, but they use the `MyISAM` storage engine, and the `FOREIGN KEY` option is ignored. Now we insert some rows into the tables on the source:

```
source> INSERT INTO fc1 VALUES (1, 1), (2, 2);
Query OK, 2 rows affected (0.09 sec)
Records: 2  Duplicates: 0  Warnings: 0

source> INSERT INTO fc2 VALUES (1, 1), (2, 2), (3, 1);
Query OK, 3 rows affected (0.19 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

At this point, on both the source and the replica, table `fc1` contains 2 rows, and table `fc2` contains 3 rows, as shown here:

```
source> SELECT * FROM fc1;
+---+------+
| i | j    |
+---+------+
| 1 |    1 |
| 2 |    2 |
+---+------+
2 rows in set (0.00 sec)

source> SELECT * FROM fc2;
+---+------+
| m | n    |
+---+------+
| 1 |    1 |
| 2 |    2 |
| 3 |    1 |
+---+------+
3 rows in set (0.00 sec)

replica> SELECT * FROM fc1;
+---+------+
| i | j    |
+---+------+
| 1 |    1 |
| 2 |    2 |
+---+------+
2 rows in set (0.00 sec)

replica> SELECT * FROM fc2;
+---+------+
| m | n    |
+---+------+
| 1 |    1 |
| 2 |    2 |
| 3 |    1 |
+---+------+
3 rows in set (0.00 sec)
```

Now suppose that you perform the following `DELETE` statement on the source:

```
source> DELETE FROM fc1 WHERE i=1;
Query OK, 1 row affected (0.09 sec)
```

Due to the cascade, table `fc2` on the source now contains only 1 row:

```
source> SELECT * FROM fc2;
+---+---+
| m | n |
+---+---+
| 2 | 2 |
+---+---+
1 row in set (0.00 sec)
```

However, the cascade does not propagate on the replica because on the replica the `DELETE` for `fc1` deletes no rows from `fc2`. The replica's copy of `fc2` still contains all of the rows that were originally inserted:

```
replica> SELECT * FROM fc2;
+---+---+
| m | n |
+---+---+
| 1 | 1 |
| 3 | 1 |
| 2 | 2 |
+---+---+
3 rows in set (0.00 sec)
```

This difference is due to the fact that the cascading deletes are handled internally by the `InnoDB` storage engine, which means that none of the changes are logged.

14.20 InnoDB memcached Plugin

The `InnoDB` `memcached` plugin (`daemon_memcached`) provides an integrated `memcached` daemon that automatically stores and retrieves data from `InnoDB` tables, turning the MySQL server into a fast "key-value store". Instead of formulating queries in SQL, you can use simple `get`, `set`, and `incr` operations that avoid the performance overhead associated with SQL parsing and constructing a query optimization plan. You can also access the same `InnoDB` tables through SQL for convenience, complex queries, bulk operations, and other strengths of traditional database software.

This "NoSQL-style" interface uses the `memcached` API to speed up database operations, letting `InnoDB` handle memory caching using its buffer pool mechanism. Data modified through `memcached` operations such as `add`, `set`, and `incr` are stored to disk, in `InnoDB` tables. The combination of `memcached` simplicity and `InnoDB` reliability and consistency provides users with the best of both worlds, as explained in Section 14.20.1, "Benefits of the InnoDB memcached Plugin". For an architectural overview, see Section 14.20.2, "InnoDB memcached Architecture".

14.20.1 Benefits of the InnoDB memcached Plugin

This section outlines advantages the `daemon_memcached` plugin. The combination of `InnoDB` tables and `memcached` offers advantages over using either by themselves.

- Direct access to the `InnoDB` storage engine avoids the parsing and planning overhead of SQL.
- Running `memcached` in the same process space as the MySQL server avoids the network overhead of passing requests back and forth.
- Data written using the `memcached` protocol is transparently written to an `InnoDB` table, without going through the MySQL SQL layer. You can control frequency of writes to achieve higher raw performance when updating non-critical data.
- Data requested through the `memcached` protocol is transparently queried from an `InnoDB` table, without going through the MySQL SQL layer.
- Subsequent requests for the same data is served from the `InnoDB` buffer pool. The buffer pool handles the in-memory caching. You can tune performance of data-intensive operations using `InnoDB` configuration options.
- Data can be unstructured or structured, depending on the type of application. You can create a new table for data, or use existing tables.

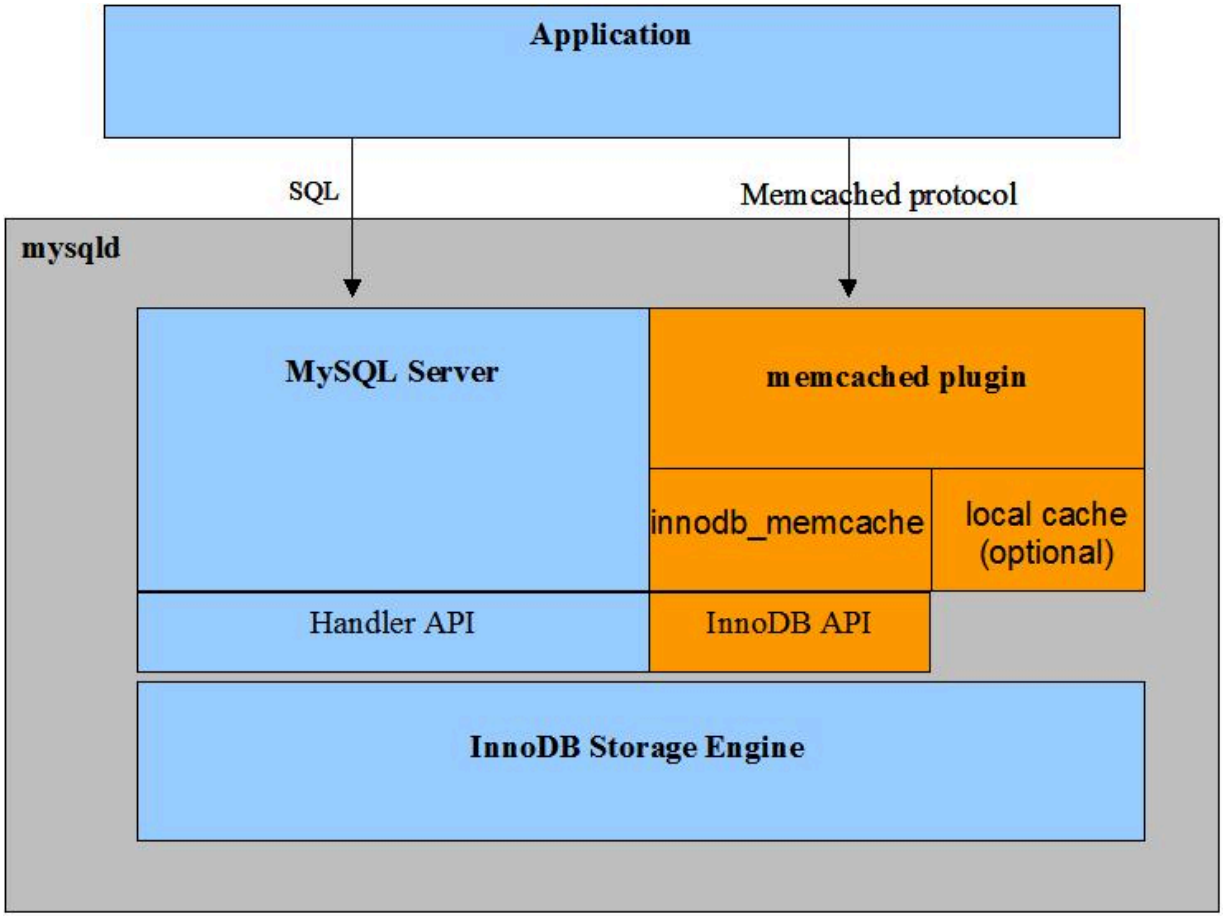
- `InnoDB` can handle composing and decomposing multiple column values into a single `memcached` item value, reducing the amount of string parsing and concatenation required in your application. For example, you can store the string value `2|4|6|8` in the `memcached` cache, and have `InnoDB` split the value based on a separator character, then store the result in four numeric columns.
- The transfer between memory and disk is handled automatically, simplifying application logic.
- Data is stored in a MySQL database to protect against crashes, outages, and corruption.
- You can access the underlying `InnoDB` table through SQL for reporting, analysis, ad hoc queries, bulk loading, multi-step transactional computations, set operations such as union and intersection, and other operations suited to the expressiveness and flexibility of SQL.
- You can ensure high availability by using the `daemon_memcached` plugin on a source server in combination with MySQL replication.
- The integration of `memcached` with MySQL provides a way to make in-memory data persistent, so you can use it for more significant kinds of data. You can use more `add`, `incr`, and similar write operations in your application without concern that data could be lost. You can stop and start the `memcached` server without losing updates made to cached data. To guard against unexpected outages, you can take advantage of `InnoDB` crash recovery, replication, and backup capabilities.
- The way `InnoDB` does fast primary key lookups is a natural fit for `memcached` single-item queries. The direct, low-level database access path used by the `daemon_memcached` plugin is much more efficient for key-value lookups than equivalent SQL queries.
- The serialization features of `memcached`, which can turn complex data structures, binary files, or even code blocks into storeable strings, offer a simple way to get such objects into a database.
- Because you can access the underlying data through SQL, you can produce reports, search or update across multiple keys, and call functions such as `AVG()` and `MAX()` on `memcached` data. All of these operations are expensive or complicated using `memcached` by itself.
- You do not need to manually load data into `memcached` at startup. As particular keys are requested by an application, values are retrieved from the database automatically, and cached in memory using the `InnoDB` buffer pool.
- Because `memcached` consumes relatively little CPU, and its memory footprint is easy to control, it can run comfortably alongside a MySQL instance on the same system.
- Because data consistency is enforced by mechanisms used for regular `InnoDB` tables, you do not have to worry about stale `memcached` data or fallback logic to query the database in the case of a missing key.

14.20.2 InnoDB memcached Architecture

The `InnoDB` `memcached` plugin implements `memcached` as a MySQL plugin daemon that accesses the `InnoDB` storage engine directly, bypassing the MySQL SQL layer.

The following diagram illustrates how an application accesses data through the `daemon_memcached` plugin, compared with SQL.

Figure 14.4 MySQL Server with Integrated `memcached` Server



Features of the `daemon_memcached` plugin:

- `memcached` as a daemon plugin of `mysqld`. Both `mysqld` and `memcached` run in the same process space, with very low latency access to data.
- Direct access to `InnoDB` tables, bypassing the SQL parser, the optimizer, and even the Handler API layer.
- Standard `memcached` protocols, including the text-based protocol and the binary protocol. The `daemon_memcached` plugin passes all 55 compatibility tests of the `memcapable` command.
- Multi-column support. You can map multiple columns into the “value” part of the key-value store, with column values delimited by a user-specified separator character.
- By default, the `memcached` protocol is used to read and write data directly to `InnoDB`, letting MySQL manage in-memory caching using the `InnoDB` buffer pool. The default settings represent a combination of high reliability and the fewest surprises for database applications. For example, default settings avoid uncommitted data on the database side, or stale data returned for `memcached get` requests.
- Advanced users can configure the system as a traditional `memcached` server, with all data cached only in the `memcached` engine (memory caching), or use a combination of the “`memcached` engine” (memory caching) and the `InnoDB` `memcached` engine (`InnoDB` as back-end persistent storage).
- Control over how often data is passed back and forth between `InnoDB` and `memcached` operations through the `innodb_api_bk_commit_interval`, `daemon_memcached_r_batch_size`, and `daemon_memcached_w_batch_size` configuration options. Batch size options default to a value of 1 for maximum reliability.

- The ability to specify `memcached` options through the `daemon_memcached_option` configuration parameter. For example, you can change the port that `memcached` listens on, reduce the maximum number of simultaneous connections, change the maximum memory size for a key-value pair, or enable debugging messages for the error log.
- The `innodb_api_trx_level` configuration option controls the transaction isolation level on queries processed by `memcached`. Although `memcached` has no concept of transactions, you can use this option to control how soon `memcached` sees changes caused by SQL statements issued on the table used by the `daemon_memcached` plugin. By default, `innodb_api_trx_level` is set to `READ UNCOMMITTED`.
- The `innodb_api_enable_mdl` option can be used to lock the table at the MySQL level, so that the mapped table cannot be dropped or altered by DDL through the SQL interface. Without the lock, the table can be dropped from the MySQL layer, but kept in `InnoDB` storage until `memcached` or some other user stops using it. “MDL” stands for “metadata locking”.

Differences Between InnoDB memcached and Traditional memcached

You may already be familiar with using `memcached` with MySQL, as described in Section 16.2, “Using MySQL with `memcached`”. This section describes how features of the integrated `InnoDB` `memcached` plugin differ from traditional `memcached`.

- Installation: The `memcached` library comes with the MySQL server, making installation and setup relatively easy. Installation involves running the `innodb_memcached_config.sql` script to create a `demo_test` table for `memcached` to use, issuing an `INSTALL PLUGIN` statement to enable the `daemon_memcached` plugin, and adding desired `memcached` options to a MySQL configuration file or startup script. You might still install the traditional `memcached` distribution for additional utilities such as `memcp`, `memcat`, and `memcapable`.

  For comparison with traditional `memcached`, see Section 16.2.1, “Installing `memcached`”.

- Deployment: With traditional `memcached`, it is typical to run large numbers of low-capacity `memcached` servers. A typical deployment of the `daemon_memcached` plugin, however, involves a smaller number of moderate or high-powered servers that are already running MySQL. The benefit of this configuration is in improving efficiency of individual database servers rather than exploiting unused memory or distributing lookups across large numbers of servers. In the default configuration, very little memory is used for `memcached`, and in-memory lookups are served from the `InnoDB` buffer pool, which automatically caches the most recently and frequently used data. As with a traditional MySQL server instance, keep the value of the `innodb_buffer_pool_size` configuration option as high as practical (without causing paging at the OS level), so that as much work as possible is performed in memory.

  For comparison with traditional `memcached`, see Section 16.2.2.2, “`memcached` Deployment”.

- Expiry: By default (that is, using the `innodb_only` caching policy), the latest data from the `InnoDB` table is always returned, so the expiry options have no practical effect. If you change the caching policy to `caching` or `cache_only`, the expiry options work as usual, but requested data might be stale if it is updated in the underlying table before it expires from the memory cache.

  For comparison with traditional `memcached`, see Section 16.2.2.4, “Data Expiry”.

- Namespaces: `memcached` is like a large directory where you give files elaborate names with prefixes and suffixes to keep the files from conflicting. The `daemon_memcached` plugin lets you use similar naming conventions for keys, with one addition. Key names in the format `@@table_id.key.table_id` are decoded to reference a specific a table, using mapping data from the `innodb_memcache.containers` table. The *`key`* is looked up in or written to the specified table.

  The `@@` notation only works for individual calls to `get`, `add`, and `set` functions, but not others such as `incr` or `delete`. To designate a default table for subsequent `memcached` operations within a

session, perform a `get` request using the `@@` notation with a *`table_id`*, but without the key portion. For example:

```
get @@table_id
```

Subsequent `get`, `set`, `incr`, `delete`, and other operations use the table designated by *`table_id`* in the `innodb_memcache.containers.name` column.

For comparison with traditional `memcached`, see Section 16.2.2.3, “Using Namespaces”.

- Hashing and distribution: The default configuration, which uses the `innodb_only` caching policy, is suitable for a traditional deployment configuration where all data is available on all servers, such as a set of replica servers.

  If you physically divide data, as in a sharded configuration, you can split data across several machines running the `daemon_memcached` plugin, and use the traditional `memcached` hashing mechanism to route requests to a particular machine. On the MySQL side, you would typically let all data be inserted by `add` requests to `memcached` so that appropriate values are stored in the database on the appropriate server.

  For comparison with traditional `memcached`, see Section 16.2.2.5, “`memcached` Hashing/Distribution Types”.

- Memory usage: By default (with the `innodb_only` caching policy), the `memcached` protocol passes information back and forth with `InnoDB` tables, and the `InnoDB` buffer pool handles in-memory lookups instead of `memcached` memory usage growing and shrinking. Relatively little memory is used on the `memcached` side.

  If you switch the caching policy to `caching` or `cache_only`, the normal rules of `memcached` memory usage apply. Memory for `memcached` data values is allocated in terms of “slabs”. You can control slab size and maximum memory used for `memcached`.

  Either way, you can monitor and troubleshoot the `daemon_memcached` plugin using the familiar statistics system, accessed through the standard protocol, over a `telnet` session, for example. Extra utilities are not included with the `daemon_memcached` plugin. You can use the `memcached-tool` script to install a full `memcached` distribution.

  For comparison with traditional `memcached`, see Section 16.2.2.7, “Memory Allocation within `memcached`”.

- Thread usage: MySQL threads and `memcached` threads co-exist on the same server. Limits imposed on threads by the operating system apply to the total number of threads.

  For comparison with traditional `memcached`, see Section 16.2.2.8, “`memcached` Thread Support”.

- Log usage: Because the `memcached` daemon is run alongside the MySQL server and writes to `stderr`, the `-v`, `-vv`, and `-vvv` options for logging write output to the MySQL error log.

  For comparison with traditional `memcached`, see Section 16.2.2.9, “`memcached` Logs”.

- `memcached` operations: Familiar `memcached` operations such as `get`, `set`, `add`, and `delete` are available. Serialization (that is, the exact string format representing complex data structures) depends on the language interface.

  For comparison with traditional `memcached`, see Section 16.2.3.1, “Basic `memcached` Operations”.

- Using `memcached` as a MySQL front end: This is the primary purpose of the `InnoDB` `memcached` plugin. An integrated `memcached` daemon improves application performance, and having `InnoDB` handle data transfers between memory and disk simplifies application logic.

  For comparison with traditional `memcached`, see Section 16.2.3.2, “Using `memcached` as a MySQL Caching Layer”.

- Utilities: The MySQL server includes the `libmemcached` library but not additional command-line utilities. To use commands such as `memcp`, `memcat`, and `memcapable` commands, install a full `memcached` distribution. When `memrm` and `memflush` remove items from the cache, the items are also removed from the underlying `InnoDB` table.

  For comparison with traditional `memcached`, see `libmemcached` Command-Line Utilities.

- Programming interfaces: You can access the MySQL server through the `daemon_memcached` plugin using all supported languages: C and C++, Java, Perl, Python, PHP, and Ruby. Specify the server hostname and port as with a traditional `memcached` server. By default, the `daemon_memcached` plugin listens on port `11211`. You can use both the text and binary protocols. You can customize the behavior of `memcached` functions at runtime. Serialization (that is, the exact string format representing complex data structures) depends on the language interface.

  For comparison with traditional `memcached`, see Section 16.2.3, “Developing a `memcached` Application”.

- Frequently asked questions: MySQL has an extensive FAQ for traditional `memcached`. The FAQ is mostly applicable, except that using `InnoDB` tables as a storage medium for `memcached` data means that you can use `memcached` for more write-intensive applications than before, rather than as a read-only cache.

  See Section 16.2.5, “`memcached` FAQ”.

14.20.3 Setting Up the InnoDB memcached Plugin

This section describes how to set up the `daemon_memcached` plugin on a MySQL server. Because the `memcached` daemon is tightly integrated with the MySQL server to avoid network traffic and minimize latency, you perform this process on each MySQL instance that uses this feature.

Note

Before setting up the `daemon_memcached` plugin, consult Section 14.20.4, “Security Considerations for the InnoDB memcached Plugin” to understand the security procedures required to prevent unauthorized access.

Prerequisites

- The `daemon_memcached` plugin is only supported on Linux, Solaris, and macOS platforms. Other operating systems are not supported.
- When building MySQL from source, you must build with `-DWITH_INNODB_MEMCACHED=ON`. This build option generates two shared libraries in the MySQL plugin directory (`plugin_dir`) that are required to run the `daemon_memcached` plugin:
  - `libmemcached.so`: the `memcached` daemon plugin to MySQL.
  - `innodb_engine.so`: an `InnoDB` API plugin to `memcached`.
- `libevent` must be installed.
  - If you did not build MySQL from source, the `libevent` library is not included in your installation. Use the installation method for your operating system to install `libevent` 1.4.12 or later. For example, depending on the operating system, you might use `apt-get`, `yum`, or `port install`. For example, on Ubuntu Linux, use:

    ```
    sudo apt-get install libevent-dev
    ```

  - If you installed MySQL from a source code release, `libevent` 1.4.12 is bundled with the package and is located at the top level of the MySQL source code directory. If you use the bundled version of `libevent`, no action is required. If you want to use a local system version of `libevent`, you must build MySQL with the `-DWITH_LIBEVENT` build option set to `system` or `yes`.

Installing and Configuring the InnoDB memcached Plugin

1. Configure the `daemon_memcached` plugin so it can interact with `InnoDB` tables by running the `innodb_memcached_config.sql` configuration script, which is located in *`MYSQL_HOME`*`/share`. This script installs the `innodb_memcache` database with three required tables (`cache_policies`, `config_options`, and `containers`). It also installs the `demo_test` sample table in the `test` database.

```
mysql> source MYSQL_HOME/share/innodb_memcached_config.sql
```

Running the `innodb_memcached_config.sql` script is a one-time operation. The tables remain in place if you later uninstall and re-install the `daemon_memcached` plugin.

```
mysql> USE innodb_memcache;
mysql> SHOW TABLES;
+---------------------------+
| Tables_in_innodb_memcache |
+---------------------------+
| cache_policies            |
| config_options            |
| containers                |
+---------------------------+

mysql> USE test;
mysql> SHOW TABLES;
+----------------+
| Tables_in_test |
+----------------+
| demo_test      |
+----------------+
```

Of these tables, the `innodb_memcache.containers` table is the most important. Entries in the `containers` table provide a mapping to `InnoDB` table columns. Each `InnoDB` table used with the `daemon_memcached` plugin requires an entry in the `containers` table.

The `innodb_memcached_config.sql` script inserts a single entry in the `containers` table that provides a mapping for the `demo_test` table. It also inserts a single row of data into the `demo_test` table. This data allows you to immediately verify the installation after the setup is completed.

```
mysql> SELECT * FROM innodb_memcache.containers\G
*************************** 1. row ***************************
                  name: aaa
             db_schema: test
              db_table: demo_test
           key_columns: c1
         value_columns: c2
                 flags: c3
            cas_column: c4
    expire_time_column: c5
unique_idx_name_on_key: PRIMARY

mysql> SELECT * FROM test.demo_test;
+----+------------------+------+------+------+
| c1 | c2               | c3   | c4   | c5   |
+----+------------------+------+------+------+
| AA | HELLO, HELLO     |    8 |    0 |    0 |
+----+------------------+------+------+------+
```

For more information about `innodb_memcache` tables and the `demo_test` sample table, see Section 14.20.7, “InnoDB memcached Plugin Internals”.

2. Activate the `daemon_memcached` plugin by running the `INSTALL PLUGIN` statement:

```
mysql> INSTALL PLUGIN daemon_memcached soname "libmemcached.so";
```

Once the plugin is installed, it is automatically activated each time the MySQL server is restarted.

Verifying the InnoDB and memcached Setup

To verify the `daemon_memcached` plugin setup, use a `telnet` session to issue `memcached` commands. By default, the `memcached` daemon listens on port 11211.

1. Retrieve data from the `test.demo_test` table. The single row of data in the `demo_test` table has a key value of `AA`.

```
telnet localhost 11211
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
get AA
VALUE AA 8 12
HELLO, HELLO
END
```

2. Insert data using a `set` command.

```
set BB 10 0 16
GOODBYE, GOODBYE
STORED
```

   where:

   - `set` is the command to store a value
   - `BB` is the key
   - `10` is a flag for the operation; ignored by `memcached` but may be used by the client to indicate any type of information; specify `0` if unused
   - `0` is the expiration time (TTL); specify `0` if unused
   - `16` is the length of the supplied value block in bytes
   - `GOODBYE, GOODBYE` is the value that is stored

3. Verify that the data inserted is stored in MySQL by connecting to the MySQL server and querying the `test.demo_test` table.

```
mysql> SELECT * FROM test.demo_test;
+----+------------------+------+------+------+
| c1 | c2               | c3   | c4   | c5   |
+----+------------------+------+------+------+
| AA | HELLO, HELLO     |    8 |    0 |    0 |
| BB | GOODBYE, GOODBYE |   10 |    1 |    0 |
+----+------------------+------+------+------+
```

4. Return to the telnet session and retrieve the data that you inserted earlier using key `BB`.

```
get BB
VALUE BB 10 16
GOODBYE, GOODBYE
END
quit
```

If you shut down the MySQL server, which also shuts off the integrated `memcached` server, further attempts to access the `memcached` data fail with a connection error. Normally, the `memcached` data also disappears at this point, and you would require application logic to load the data back into memory when `memcached` is restarted. However, the `InnoDB memcached` plugin automates this process for you.

When you restart MySQL, `get` operations once again return the key-value pairs you stored in the earlier `memcached` session. When a key is requested and the associated value is not already in the memory cache, the value is automatically queried from the MySQL `test.demo_test` table.

Creating a New Table and Column Mapping

This example shows how to setup your own `InnoDB` table with the `daemon_memcached` plugin.

1. Create an `InnoDB` table. The table must have a key column with a unique index. The key column of the city table is `city_id`, which is defined as the primary key. The table must also include columns for `flags`, `cas`, and `expiry` values. There may be one or more value columns. The `city` table has three value columns (`name`, `state`, `country`).

   Note

   There is no special requirement with respect to column names as along as a valid mapping is added to the `innodb_memcache.containers` table.

   ```
   mysql> CREATE TABLE city (
          city_id VARCHAR(32),
          name VARCHAR(1024),
          state VARCHAR(1024),
          country VARCHAR(1024),
          flags INT,
          cas BIGINT UNSIGNED,
          expiry INT,
          primary key(city_id)
          ) ENGINE=InnoDB;
   ```

2. Add an entry to the `innodb_memcache.containers` table so that the `daemon_memcached` plugin knows how to access the `InnoDB` table. The entry must satisfy the `innodb_memcache.containers` table definition. For a description of each field, see Section 14.20.7, “InnoDB memcached Plugin Internals”.

   ```
   mysql> DESCRIBE innodb_memcache.containers;
   +-----------------------+-------------+------+-----+---------+-------+
   | Field                 | Type        | Null | Key | Default | Extra |
   +-----------------------+-------------+------+-----+---------+-------+
   | name                  | varchar(50)  | NO   | PRI | NULL    |       |
   | db_schema             | varchar(250) | NO   |     | NULL    |       |
   | db_table              | varchar(250) | NO   |     | NULL    |       |
   | key_columns           | varchar(250) | NO   |     | NULL    |       |
   | value_columns         | varchar(250) | YES  |     | NULL    |       |
   | flags                 | varchar(250) | NO   |     | 0       |       |
   | cas_column            | varchar(250) | YES  |     | NULL    |       |
   | expire_time_column    | varchar(250) | YES  |     | NULL    |       |
   | unique_idx_name_on_key | varchar(250) | NO   |     | NULL    |       |
   +-----------------------+-------------+------+-----+---------+-------+
   ```

   The `innodb_memcache.containers` table entry for the city table is defined as:

   ```
   mysql> INSERT INTO `innodb_memcache`.`containers` (
          `name`, `db_schema`, `db_table`, `key_columns`, `value_columns`,
          `flags`, `cas_column`, `expire_time_column`, `unique_idx_name_on_key`)
          VALUES ('default', 'test', 'city', 'city_id', 'name|state|country',
          'flags','cas','expiry','PRIMARY');
   ```

   - `default` is specified for the `containers.name` column to configure the `city` table as the default `InnoDB` table to be used with the `daemon_memcached` plugin.
   - Multiple `InnoDB` table columns (`name`, `state`, `country`) are mapped to `containers.value_columns` using a “|” delimiter.
   - The `flags`, `cas_column`, and `expire_time_column` fields of the `innodb_memcache.containers` table are typically not significant in applications using the `daemon_memcached` plugin. However, a designated `InnoDB` table column is required for each. When inserting data, specify `0` for these columns if they are unused.

3. After updating the `innodb_memcache.containers` table, restart the `daemon_memcache` plugin to apply the changes.

```
mysql> UNINSTALL PLUGIN daemon_memcached;

mysql> INSTALL PLUGIN daemon_memcached soname "libmemcached.so";
```

4. Using telnet, insert data into the `city` table using a `memcached set` command.

```
telnet localhost 11211
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
set B 0 0 22
BANGALORE|BANGALORE|IN
STORED
```

5. Using MySQL, query the `test.city` table to verify that the data you inserted was stored.

```
mysql> SELECT * FROM test.city;
+---------+-----------+-----------+---------+-------+------+--------+
| city_id | name      | state     | country | flags | cas  | expiry |
+---------+-----------+-----------+---------+-------+------+--------+
| B       | BANGALORE | BANGALORE | IN      |     0 |    3 |      0 |
+---------+-----------+-----------+---------+-------+------+--------+
```

6. Using MySQL, insert additional data into the `test.city` table.

```
mysql> INSERT INTO city VALUES ('C','CHENNAI','TAMIL NADU','IN', 0, 0 ,0);
mysql> INSERT INTO city VALUES ('D','DELHI','DELHI','IN', 0, 0, 0);
mysql> INSERT INTO city VALUES ('H','HYDERABAD','TELANGANA','IN', 0, 0, 0);
mysql> INSERT INTO city VALUES ('M','MUMBAI','MAHARASHTRA','IN', 0, 0, 0);
```

Note

It is recommended that you specify a value of `0` for the `flags`, `cas_column`, and `expire_time_column` fields if they are unused.

7. Using telnet, issue a `memcached get` command to retrieve data you inserted using MySQL.

```
get H
VALUE H 0 22
HYDERABAD|TELANGANA|IN
END
```

Configuring the InnoDB memcached Plugin

Traditional `memcached` configuration options may be specified in a MySQL configuration file or a `mysqld` startup string, encoded in the argument of the `daemon_memcached_option` configuration parameter. `memcached` configuration options take effect when the plugin is loaded, which occurs each time the MySQL server is started.

For example, to make `memcached` listen on port 11222 instead of the default port 11211, specify `-p11222` as an argument of the `daemon_memcached_option` configuration option:

```
mysqld .... --daemon_memcached_option="-p11222"
```

Other `memcached` options can be encoded in the `daemon_memcached_option` string. For example, you can specify options to reduce the maximum number of simultaneous connections, change the maximum memory size for a key-value pair, or enable debugging messages for the error log, and so on.

There are also configuration options specific to the `daemon_memcached` plugin. These include:

- `daemon_memcached_engine_lib_name`: Specifies the shared library that implements the `InnoDB memcached` plugin. The default setting is `innodb_engine.so`.
- `daemon_memcached_engine_lib_path`: The path of the directory containing the shared library that implements the `InnoDB memcached` plugin. The default is NULL, representing the plugin directory.

- `daemon_memcached_r_batch_size`: Defines the batch commit size for read operations (`get`). It specifies the number of `memcached` read operations after which a commit occurs. `daemon_memcached_r_batch_size` is set to 1 by default so that every `get` request accesses the most recently committed data in the `InnoDB` table, whether the data was updated through `memcached` or by SQL. When the value is greater than 1, the counter for read operations is incremented with each `get` call. A `flush_all` call resets both read and write counters.

- `daemon_memcached_w_batch_size`: Defines the batch commit size for write operations (`set`, `replace`, `append`, `prepend`, `incr`, `decr`, and so on). `daemon_memcached_w_batch_size` is set to 1 by default so that no uncommitted data is lost in case of an outage, and so that SQL queries on the underlying table access the most recent data. When the value is greater than 1, the counter for write operations is incremented for each `add`, `set`, `incr`, `decr`, and `delete` call. A `flush_all` call resets both read and write counters.

By default, you do not need to modify `daemon_memcached_engine_lib_name` or `daemon_memcached_engine_lib_path`. You might configure these options if, for example, you want to use a different storage engine for `memcached` (such as the NDB `memcached` engine).

`daemon_memcached` plugin configuration parameters may be specified in the MySQL configuration file or in a `mysqld` startup string. They take effect when you load the `daemon_memcached` plugin.

When making changes to `daemon_memcached` plugin configuration, reload the plugin to apply the changes. To do so, issue the following statements:

```
mysql> UNINSTALL PLUGIN daemon_memcached;

mysql> INSTALL PLUGIN daemon_memcached soname "libmemcached.so";
```

Configuration settings, required tables, and data are preserved when the plugin is restarted.

For additional information about enabling and disabling plugins, see Section 5.5.1, “Installing and Uninstalling Plugins”.

14.20.4 Security Considerations for the InnoDB memcached Plugin

Caution

Consult this section before deploying the `daemon_memcached` plugin on a production server, or even on a test server if the MySQL instance contains sensitive data.

Because `memcached` does not use an authentication mechanism by default, and the optional SASL authentication is not as strong as traditional DBMS security measures, only keep non-sensitive data in the MySQL instance that uses the `daemon_memcached` plugin, and wall off any servers that use this configuration from potential intruders. Do not allow `memcached` access to these servers from the Internet; only allow access from within a firewalled intranet, ideally from a subnet whose membership you can restrict.

Password-Protecting memcached Using SASL

SASL support provides the capability to protect your MySQL database from unauthenticated access through `memcached` clients. This section explains how to enable SASL with the `daemon_memcached` plugin. The steps are almost identical to those performed to enabled SASL for a traditional `memcached` server.

SASL stands for “Simple Authentication and Security Layer”, a standard for adding authentication support to connection-based protocols. `memcached` added SASL support in version 1.4.3.

SASL authentication is only supported with the binary protocol.

`memcached` clients are only able to access `InnoDB` tables that are registered in the `innodb_memcache.containers` table. Even though a DBA can place access restrictions on such tables, access through `memcached` applications cannot be controlled. For this reason, SASL support is provided to control access to `InnoDB` tables associated with the `daemon_memcached` plugin.

The following section shows how to build, enable, and test an SASL-enabled `daemon_memcached` plugin.

Building and Enabling SASL with the InnoDB memcached Plugin

By default, an SASL-enabled `daemon_memcached` plugin is not included in MySQL release packages, since an SASL-enabled `daemon_memcached` plugin requires building `memcached` with SASL libraries. To enable SASL support, download the MySQL source and rebuild the `daemon_memcached` plugin after downloading the SASL libraries:

1. Install the SASL development and utility libraries. For example, on Ubuntu, use `apt-get` to obtain the libraries:

   ```
   sudo apt-get -f install libsasl2-2 sasl2-bin libsasl2-2 libsasl2-dev libsasl2-modules
   ```

2. Build the `daemon_memcached` plugin shared libraries with SASL capability by adding `ENABLE_MEMCACHED_SASL=1` to your `cmake` options. `memcached` also provides *simple cleartext password support*, which facilitates testing. To enable simple cleartext password support, specify the `ENABLE_MEMCACHED_SASL_PWDB=1` `cmake` option.

   In summary, add following three `cmake` options:

   ```
   cmake ... -DWITH_INNODB_MEMCACHED=1 -DENABLE_MEMCACHED_SASL=1 -DENABLE_MEMCACHED_SASL_PWDB=1
   ```

3. Install the `daemon_memcached` plugin, as described in Section 14.20.3, “Setting Up the InnoDB memcached Plugin”.

4. Configure a user name and password file. (This example uses `memcached` simple cleartext password support.)

   a. In a file, create a user named `testname` and define the password as `testpasswd`:

      ```
      echo "testname:testpasswd:::::::" >/home/jy/memcached-sasl-db
      ```

   b. Configure the `MEMCACHED_SASL_PWDB` environment variable to inform `memcached` of the user name and password file:

      ```
      export MEMCACHED_SASL_PWDB=/home/jy/memcached-sasl-db
      ```

   c. Inform `memcached` that a cleartext password is used:

      ```
      echo "mech_list: plain" > /home/jy/work2/msasl/clients/memcached.conf
      export SASL_CONF_PATH=/home/jy/work2/msasl/clients
      ```

5. Enable SASL by restarting the MySQL server with the `memcached -S` option encoded in the `daemon_memcached_option` configuration parameter:

   ```
   mysqld ... --daemon_memcached_option="-S"
   ```

6. To test the setup, use an SASL-enabled client such as SASL-enabled libmemcached.

   ```
   memcp --servers=localhost:11211 --binary  --username=testname
     --password=password myfile.txt

   memcat --servers=localhost:11211 --binary --username=testname
     --password=password myfile.txt
   ```

If you specify an incorrect user name or password, the operation is rejected with a `memcache error AUTHENTICATION FAILURE` message. In this case, examine the cleartext password set in the `memcached-sasl-db` file to verify that the credentials you supplied are correct.

There are other methods to test SASL authentication with `memcached`, but the method described above is the most straightforward.

14.20.5 Writing Applications for the InnoDB memcached Plugin

Typically, writing an application for the `InnoDB memcached` plugin involves some degree of rewriting or adapting existing code that uses MySQL or the `memcached` API.

- With the `daemon_memcached` plugin, instead of many traditional `memcached` servers running on low-powered machines, you have the same number of `memcached` servers as MySQL servers, running on relatively high-powered machines with substantial disk storage and memory. You might reuse some existing code that works with the `memcached` API, but adaptation is likely required due to the different server configuration.
- The data stored through the `daemon_memcached` plugin goes into `VARCHAR`, `TEXT`, or `BLOB` columns, and must be converted to do numeric operations. You can perform the conversion on the application side, or by using the `CAST()` function in queries.
- Coming from a database background, you might be used to general-purpose SQL tables with many columns. The tables accessed by `memcached` code likely have only a few or even a single column holding data values.
- You might adapt parts of your application that perform single-row queries, inserts, updates, or deletes, to improve performance in critical sections of code. Both queries (read) and DML (write) operations can be substantially faster when performed through the `InnoDB memcached` interface. The performance improvement for writes is typically greater than the performance improvement for reads, so you might focus on adapting code that performs logging or records interactive choices on a website.

The following sections explore these points in more detail.

14.20.5.1 Adapting an Existing MySQL Schema for the InnoDB memcached Plugin

Consider these aspects of `memcached` applications when adapting an existing MySQL schema or application to use the `daemon_memcached` plugin:

- `memcached` keys cannot contain spaces or newlines, because these characters are used as separators in the ASCII protocol. If you are using lookup values that contain spaces, transform or hash them into values without spaces before using them as keys in calls to `add()`, `set()`, `get()`, and so on. Although theoretically these characters are allowed in keys in programs that use the binary protocol, you should restrict the characters used in keys to ensure compatibility with a broad range of clients.
- If there is a short numeric primary key column in an `InnoDB` table, use it as the unique lookup key for `memcached` by converting the integer to a string value. If the `memcached` server is used for multiple applications, or with more than one `InnoDB` table, consider modifying the name to ensure that it is unique. For example, prepend the table name, or the database name and the table name, before the numeric value.

Note

The `daemon_memcached` plugin supports inserts and reads on mapped `InnoDB` tables that have an `INTEGER` defined as the primary key.

- You cannot use a partitioned table for data queried or stored using `memcached`.

- The `memcached` protocol passes numeric values around as strings. To store numeric values in the underlying `InnoDB` table, to implement counters that can be used in SQL functions such as `SUM()` or `AVG()`, for example:
  - Use `VARCHAR` columns with enough characters to hold all the digits of the largest expected number (and additional characters if appropriate for the negative sign, decimal point, or both).
  - In any query that performs arithmetic using column values, use the `CAST()` function to convert the values from string to integer, or to some other numeric type. For example:

```
# Alphabetic entries are returned as zero.

SELECT CAST(c2 as unsigned integer) FROM demo_test;

# Since there could be numeric values of 0, can't disqualify them.
# Test the string values to find the ones that are integers, and average only those.

SELECT AVG(cast(c2 as unsigned integer)) FROM demo_test
  WHERE c2 BETWEEN '0' and '9999999999';

# Views let you hide the complexity of queries. The results are already converted;
# no need to repeat conversion functions and WHERE clauses each time.

CREATE VIEW numbers AS SELECT c1 KEY, CAST(c2 AS UNSIGNED INTEGER) val
  FROM demo_test WHERE c2 BETWEEN '0' and '9999999999';
SELECT SUM(val) FROM numbers;
```

Note

Any alphabetic values in the result set are converted into 0 by the call to `CAST()`. When using functions such as `AVG()`, which depend on the number of rows in the result set, include `WHERE` clauses to filter out non-numeric values.

- If the `InnoDB` column used as a key could have values longer than 250 bytes, hash the value to less than 250 bytes.
- To use an existing table with the `daemon_memcached` plugin, define an entry for it in the `innodb_memcache.containers` table. To make that table the default for all `memcached` requests, specify a value of `default` in the `name` column, then restart the MySQL server to make the change take effect. If you use multiple tables for different classes of `memcached` data, set up multiple entries in the `innodb_memcache.containers` table with `name` values of your choice, then issue a `memcached` request in the form of `get @@name` or `set @@name` within the application to specify the table to be used for subsequent `memcached` requests.

  For an example of using a table other than the predefined `test.demo_test` table, see Example 14.12, “Using Your Own Table with an InnoDB memcached Application”. For the required table layout, see Section 14.20.7, “InnoDB memcached Plugin Internals”.
- To use multiple `InnoDB` table column values with `memcached` key-value pairs, specify column names separated by comma, semicolon, space, or pipe characters in the `value_columns` field of the `innodb_memcache.containers` entry for the `InnoDB` table. For example, specify `col1,col2,col3` or `col1|col2|col3` in the `value_columns` field.

  Concatenate the column values into a single string using the pipe character as a separator before passing the string to `memcached add` or `set` calls. The string is unpacked automatically into the correct column. Each `get` call returns a single string containing the column values that is also delimited by the pipe character. You can unpack the values using the appropriate application language syntax.

Example 14.12 Using Your Own Table with an InnoDB memcached Application

This example shows how to use your own table with a sample Python application that uses `memcached` for data manipulation.

The example assumes that the daemon_memcached plugin is installed as described in Section 14.20.3, "Setting Up the InnoDB memcached Plugin". It also assumes that your system is configured to run a Python script that uses the python-memcache module.

1. Create the multicol table which stores country information including population, area, and driver side data ('R' for right and 'L' for left).

```
mysql> USE test;

mysql> CREATE TABLE `multicol` (
        `country` varchar(128) NOT NULL DEFAULT '',
        `population` varchar(10) DEFAULT NULL,
        `area_sq_km` varchar(9) DEFAULT NULL,
        `drive_side` varchar(1) DEFAULT NULL,
        `c3` int(11) DEFAULT NULL,
        `c4` bigint(20) unsigned DEFAULT NULL,
        `c5` int(11) DEFAULT NULL,
       PRIMARY KEY (`country`)
       ) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

2. Insert a record into the innodb_memcache.containers table so that the daemon_memcached plugin can access the multicol table.

```
mysql> INSERT INTO innodb_memcache.containers
       (name,db_schema,db_table,key_columns,value_columns,flags,cas_column,
       expire_time_column,unique_idx_name_on_key)
       VALUES
       ('bbb','test','multicol','country','population,area_sq_km,drive_side',
       'c3','c4','c5','PRIMARY');

mysql> COMMIT;
```

   - The innodb_memcache.containers record for the multicol table specifies a name value of 'bbb', which is the table identifier.

     Note

     If a single InnoDB table is used for all memcached applications, the name value can be set to default to avoid using @@ notation to switch tables.

   - The db_schema column is set to test, which is the name of the database where the multicol table resides.
   - The db_table column is set to multicol, which is the name of the InnoDB table.
   - key_columns is set to the unique country column. The country column is defined as the primary key in the multicol table definition.
   - Rather than a single InnoDB table column to hold a composite data value, data is divided among three table columns (population, area_sq_km, and drive_side). To accommodate multiple value columns, a comma-separated list of columns is specified in the value_columns field. The columns defined in the value_columns field are the columns used when storing or retrieving values.
   - Values for the flags, expire_time, and cas_column fields are based on values used in the demo.test sample table. These fields are typically not significant in applications that use the daemon_memcached plugin because MySQL keeps data synchronized, and there is no need to worry about data expiring or becoming stale.
   - The unique_idx_name_on_key field is set to PRIMARY, which refers to the primary index defined on the unique country column in the multicol table.

3. Copy the sample Python application into a file. In this example, the sample script is copied to a file named `multicol.py`.

   The sample Python application inserts data into the `multicol` table and retrieves data for all keys, demonstrating how to access an `InnoDB` table through the `daemon_memcached` plugin.

```
import sys, os
import memcache

def connect_to_memcached():
  memc = memcache.Client(['127.0.0.1:11211'], debug=0);
  print "Connected to memcached."
  return memc

def banner(message):
  print
  print "=" * len(message)
  print message
  print "=" * len(message)

country_data = [
("Canada","34820000","9984670","R"),
("USA","314242000","9826675","R"),
("Ireland","6399152","84421","L"),
("UK","62262000","243610","L"),
("Mexico","113910608","1972550","R"),
("Denmark","5543453","43094","R"),
("Norway","5002942","385252","R"),
("UAE","8264070","83600","R"),
("India","1210193422","3287263","L"),
("China","1347350000","9640821","R"),
]

def switch_table(memc,table):
  key = "@@" + table
  print "Switching default table to '" + table + "' by issuing GET for '" + key + "'."
  result = memc.get(key)

def insert_country_data(memc):
  banner("Inserting initial data via memcached interface")
  for item in country_data:
    country = item[0]
    population = item[1]
    area = item[2]
    drive_side = item[3]

    key = country
    value = "|".join([population,area,drive_side])
    print "Key = " + key
    print "Value = " + value

    if memc.add(key,value):
      print "Added new key, value pair."
    else:
      print "Updating value for existing key."
      memc.set(key,value)

def query_country_data(memc):
  banner("Retrieving data for all keys (country names)")
  for item in country_data:
    key = item[0]
    result = memc.get(key)
    print "Here is the result retrieved from the database for key " + key + ":"
    print result
    (m_population, m_area, m_drive_side) = result.split("|")
    print "Unpacked population value: " + m_population
    print "Unpacked area value      : " + m_area
    print "Unpacked drive side value: " + m_drive_side

if __name__ == '__main__':
```

```
  memc = connect_to_memcached()
  switch_table(memc,"bbb")
  insert_country_data(memc)
  query_country_data(memc)

  sys.exit(0)
```

Sample Python application notes:

- No database authorization is required to run the application, since data manipulation is performed through the `memcached` interface. The only required information is the port number on the local system where the `memcached` daemon listens.
- To make sure the application uses the `multicol` table, the `switch_table()` function is called, which performs a dummy `get` or `set` request using `@@` notation. The `name` value in the request is `bbb`, which is the `multicol` table identifier defined in the `innodb_memcache.containers.name` field.

  A more descriptive `name` value might be used in a real-world application. This example simply illustrates that a table identifier is specified rather than the table name in `get @@...` requests.
- The utility functions used to insert and query data demonstrate how to turn a Python data structure into pipe-separated values for sending data to MySQL with `add` or `set` requests, and how to unpack the pipe-separated values returned by `get` requests. This extra processing is only required when mapping a single `memcached` value to multiple MySQL table columns.

4. Run the sample Python application.

```
shell> python multicol.py
```

If successful, the sample application returns this output:

```
Connected to memcached.
Switching default table to 'bbb' by issuing GET for '@@bbb'.

============================================
Inserting initial data via memcached interface
============================================
Key = Canada
Value = 34820000|9984670|R
Added new key, value pair.
Key = USA
Value = 314242000|9826675|R
Added new key, value pair.
Key = Ireland
Value = 6399152|84421|L
Added new key, value pair.
Key = UK
Value = 62262000|243610|L
Added new key, value pair.
Key = Mexico
Value = 113910608|1972550|R
Added new key, value pair.
Key = Denmark
Value = 5543453|43094|R
Added new key, value pair.
Key = Norway
Value = 5002942|385252|R
Added new key, value pair.
Key = UAE
Value = 8264070|83600|R
Added new key, value pair.
Key = India
Value = 1210193422|3287263|L
Added new key, value pair.
Key = China
Value = 1347350000|9640821|R
Added new key, value pair.
```

```
==========================================
Retrieving data for all keys (country names)
==========================================
Here is the result retrieved from the database for key Canada:
34820000|9984670|R
Unpacked population value: 34820000
Unpacked area value      : 9984670
Unpacked drive side value: R
Here is the result retrieved from the database for key USA:
314242000|9826675|R
Unpacked population value: 314242000
Unpacked area value      : 9826675
Unpacked drive side value: R
Here is the result retrieved from the database for key Ireland:
6399152|84421|L
Unpacked population value: 6399152
Unpacked area value      : 84421
Unpacked drive side value: L
Here is the result retrieved from the database for key UK:
62262000|243610|L
Unpacked population value: 62262000
Unpacked area value      : 243610
Unpacked drive side value: L
Here is the result retrieved from the database for key Mexico:
113910608|1972550|R
Unpacked population value: 113910608
Unpacked area value      : 1972550
Unpacked drive side value: R
Here is the result retrieved from the database for key Denmark:
5543453|43094|R
Unpacked population value: 5543453
Unpacked area value      : 43094
Unpacked drive side value: R
Here is the result retrieved from the database for key Norway:
5002942|385252|R
Unpacked population value: 5002942
Unpacked area value      : 385252
Unpacked drive side value: R
Here is the result retrieved from the database for key UAE:
8264070|83600|R
Unpacked population value: 8264070
Unpacked area value      : 83600
Unpacked drive side value: R
Here is the result retrieved from the database for key India:
1210193422|3287263|L
Unpacked population value: 1210193422
Unpacked area value      : 3287263
Unpacked drive side value: L
Here is the result retrieved from the database for key China:
1347350000|9640821|R
Unpacked population value: 1347350000
Unpacked area value      : 9640821
Unpacked drive side value: R
```

5. Query the `innodb_memcache.containers` table to view the record you inserted earlier for the `multicol` table. The first record is the sample entry for the `demo_test` table that is created during the initial `daemon_memcached` plugin setup. The second record is the entry you inserted for the `multicol` table.

```
mysql> SELECT * FROM innodb_memcache.containers\G
*************************** 1. row ***************************
                name: aaa
           db_schema: test
            db_table: demo_test
         key_columns: c1
       value_columns: c2
               flags: c3
          cas_column: c4
   expire_time_column: c5
unique_idx_name_on_key: PRIMARY
```

```
*************************** 2. row ***************************
                  name: bbb
             db_schema: test
              db_table: multicol
           key_columns: country
         value_columns: population,area_sq_km,drive_side
                 flags: c3
            cas_column: c4
    expire_time_column: c5
unique_idx_name_on_key: PRIMARY
```

6. Query the `multicol` table to view data inserted by the sample Python application. The data is available for MySQL queries, which demonstrates how the same data can be accessed using SQL or through applications (using the appropriate MySQL Connector or API).

```
mysql> SELECT * FROM test.multicol;
+---------+------------+------------+------------+------+------+------+
| country | population | area_sq_km | drive_side | c3   | c4   | c5   |
+---------+------------+------------+------------+------+------+------+
| Canada  | 34820000   | 9984670    | R          |    0 |   11 |    0 |
| China   | 1347350000 | 9640821    | R          |    0 |   20 |    0 |
| Denmark | 5543453    | 43094      | R          |    0 |   16 |    0 |
| India   | 1210193422 | 3287263    | L          |    0 |   19 |    0 |
| Ireland | 6399152    | 84421      | L          |    0 |   13 |    0 |
| Mexico  | 113910608  | 1972550    | R          |    0 |   15 |    0 |
| Norway  | 5002942    | 385252     | R          |    0 |   17 |    0 |
| UAE     | 8264070    | 83600      | R          |    0 |   18 |    0 |
| UK      | 62262000   | 243610     | L          |    0 |   14 |    0 |
| USA     | 314242000  | 9826675    | R          |    0 |   12 |    0 |
+---------+------------+------------+------------+------+------+------+
```

Note

Always allow sufficient size to hold necessary digits, decimal points, sign characters, leading zeros, and so on when defining the length for columns that are treated as numbers. Too-long values in a string column such as a `VARCHAR` are truncated by removing some characters, which could produce nonsensical numeric values.

7. Optionally, run report-type queries on the `InnoDB` table that stores the `memcached` data.

   You can produce reports through SQL queries, performing calculations and tests across any columns, not just the `country` key column. (Because the following examples use data from only a few countries, the numbers are for illustration purposes only.) The following queries return the average population of countries where people drive on the right, and the average size of countries whose names start with “U”:

```
mysql> SELECT AVG(population) FROM multicol WHERE drive_side = 'R';
+-------------------+
| avg(population)   |
+-------------------+
| 261304724.7142857 |
+-------------------+

mysql> SELECT SUM(area_sq_km) FROM multicol WHERE country LIKE 'U%';
+-----------------+
| sum(area_sq_km) |
+-----------------+
|        10153885 |
+-----------------+
```

   Because the `population` and `area_sq_km` columns store character data rather than strongly typed numeric data, functions such as `AVG()` and `SUM()` work by converting each value to a number first. This approach *does not work* for operators such as `<` or `>`, for example, when comparing character-based values, `9 > 1000`, which is not expected from a clause such as `ORDER BY population DESC`. For the most accurate type treatment, perform queries against views that cast numeric columns to the appropriate types. This technique lets you issue simple

`SELECT *` queries from database applications, while ensuring that casting, filtering, and ordering is correct. The following example shows a view that can be queried to find the top three countries in descending order of population, with the results reflecting the latest data in the `multicol` table, and with population and area figures treated as numbers:

```
mysql> CREATE VIEW populous_countries AS
       SELECT
       country,
       cast(population as unsigned integer) population,
       cast(area_sq_km as unsigned integer) area_sq_km,
       drive_side FROM multicol
       ORDER BY CAST(population as unsigned integer) DESC
       LIMIT 3;

mysql> SELECT * FROM populous_countries;
+---------+------------+------------+------------+
| country | population | area_sq_km | drive_side |
+---------+------------+------------+------------+
| China   | 1347350000 |    9640821 | R          |
| India   | 1210193422 |    3287263 | L          |
| USA     |  314242000 |    9826675 | R          |
+---------+------------+------------+------------+

mysql> DESC populous_countries;
+------------+---------------------+------+-----+---------+-------+
| Field      | Type                | Null | Key | Default | Extra |
+------------+---------------------+------+-----+---------+-------+
| country    | varchar(128)        | NO   |     |         |       |
| population | bigint(10) unsigned | YES  |     | NULL    |       |
| area_sq_km | int(9) unsigned     | YES  |     | NULL    |       |
| drive_side | varchar(1)          | YES  |     | NULL    |       |
+------------+---------------------+------+-----+---------+-------+
```

14.20.5.2 Adapting a memcached Application for the InnoDB memcached Plugin

Consider these aspects of MySQL and `InnoDB` tables when adapting existing `memcached` applications to use the `daemon_memcached` plugin:

- If there are key values longer than a few bytes, it may be more efficient to use a numeric auto-increment column as the primary key of the `InnoDB` table, and to create a unique secondary index on the column that contains the `memcached` key values. This is because `InnoDB` performs best for large-scale insertions if primary key values are added in sorted order (as they are with auto-increment values). Primary key values are included in secondary indexes, which takes up unnecessary space if the primary key is a long string value.

- If you store several different classes of information using `memcached`, consider setting up a separate `InnoDB` table for each type of data. Define additional table identifiers in the `innodb_memcache.containers` table, and use the *`@@table_id.key`* notation to store and retrieve items from different tables. Physically dividing different types of information allows you tune the characteristics of each table for optimum space utilization, performance, and reliability. For example, you might enable compression for a table that holds blog posts, but not for a table that holds thumbnail images. You might back up one table more frequently than another because it holds critical data. You might create additional secondary indexes on tables that are frequently used to generate reports using SQL.

- Preferably, configure a stable set of table definitions for use with the `daemon_memcached` plugin, and leave the tables in place permanently. Changes to the `innodb_memcache.containers` table take effect the next time the `innodb_memcache.containers` table is queried. Entries in the containers table are processed at startup, and are consulted whenever an unrecognized table identifier (as defined by `containers.name`) is requested using `@@` notation. Thus, new entries are visible as soon as you use the associated table identifier, but changes to existing entries require a server restart before they take effect.

- When you use the default `innodb_only` caching policy, calls to `add()`, `set()`, `incr()`, and so on can succeed but still trigger debugging messages such as `while expecting 'STORED',`

`got unexpected response 'NOT_STORED`. Debug messages occur because new and updated values are sent directly to the `InnoDB` table without being saved in the memory cache, due to the `innodb_only` caching policy.

14.20.5.3 Tuning InnoDB memcached Plugin Performance

Because using `InnoDB` in combination with `memcached` involves writing all data to disk, whether immediately or sometime later, raw performance is expected to be somewhat slower than using `memcached` by itself. When using the `InnoDB` `memcached` plugin, focus tuning goals for `memcached` operations on achieving better performance than equivalent SQL operations.

Benchmarks suggest that queries and DML operations (inserts, updates, and deletes) that use the `memcached` interface are faster than traditional SQL. DML operations typically see a larger improvements. Therefore, consider adapting write-intensive applications to use the `memcached` interface first. Also consider prioritizing adaptation of write-intensive applications that use fast, lightweight mechanisms that lack reliability.

Adapting SQL Queries

The types of queries that are most suited to simple `GET` requests are those with a single clause or a set of `AND` conditions in the `WHERE` clause:

```
SQL:
SELECT col FROM tbl WHERE key = 'key_value';

memcached:
get key_value

SQL:
SELECT col FROM tbl WHERE col1 = val1 and col2 = val2 and col3 = val3;

memcached:
# Since you must always know these 3 values to look up the key,
# combine them into a unique string and use that as the key
# for all ADD, SET, and GET operations.
key_value = val1 + ":" + val2 + ":" + val3
get key_value

SQL:
SELECT 'key exists!' FROM tbl
  WHERE EXISTS (SELECT col1 FROM tbl WHERE KEY = 'key_value') LIMIT 1;

memcached:
# Test for existence of key by asking for its value and checking if the call succeeds,
# ignoring the value itself. For existence checking, you typically only store a very
# short value such as "1".
get key_value
```

Using System Memory

For best performance, deploy the `daemon_memcached` plugin on machines that are configured as typical database servers, where the majority of system RAM is devoted to the `InnoDB` buffer pool, through the `innodb_buffer_pool_size` configuration option. For systems with multi-gigabyte buffer pools, consider raising the value of `innodb_buffer_pool_instances` for maximum throughput when most operations involve data that is already cached in memory.

Reducing Redundant I/O

`InnoDB` has a number of settings that let you choose the balance between high reliability, in case of a crash, and the amount of I/O overhead during high write workloads. For example, consider setting the `innodb_doublewrite` to `0` and `innodb_flush_log_at_trx_commit` to `2`. Measure performance with different `innodb_flush_method` settings.

For other ways to reduce or tune I/O for table operations, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

Reducing Transactional Overhead

A default value of 1 for `daemon_memcached_r_batch_size` and `daemon_memcached_w_batch_size` is intended for maximum reliability of results and safety of stored or updated data.

Depending on the type of application, you might increase one or both of these settings to reduce the overhead of frequent commit operations. On a busy system, you might increase `daemon_memcached_r_batch_size`, knowing that changes to data made through SQL may not become visible to `memcached` immediately (that is, until *`N`* more `get` operations are processed). When processing data where every write operation must be reliably stored, leave `daemon_memcached_w_batch_size` set to `1`. Increase the setting when processing large numbers of updates intended only for statistical analysis, where losing the last *`N`* updates in an unexpected exit is an acceptable risk.

For example, imagine a system that monitors traffic crossing a busy bridge, recording data for approximately 100,000 vehicles each day. If the application counts different types of vehicles to analyze traffic patterns, changing `daemon_memcached_w_batch_size` from `1` to `100` reduces I/O overhead for commit operations by 99%. In case of an outage, a maximum of 100 records are lost, which may be an acceptable margin of error. If instead the application performed automated toll collection for each car, you would set `daemon_memcached_w_batch_size` to `1` to ensure that each toll record is immediately saved to disk.

Because of the way `InnoDB` organizes `memcached` key values on disk, if you have a large number of keys to create, it may be faster to sort the data items by key value in the application and `add` them in sorted order, rather than create keys in arbitrary order.

The `memslap` command, which is part of the regular `memcached` distribution but not included with the `daemon_memcached` plugin, can be useful for benchmarking different configurations. It can also be used to generate sample key-value pairs to use in your own benchmarks. See `libmemcached` Command-Line Utilities for details.

14.20.5.4 Controlling Transactional Behavior of the InnoDB memcached Plugin

Unlike traditional `memcached`, the `daemon_memcached` plugin allows you to control durability of data values produced through calls to `add`, `set`, `incr`, and so on. By default, data written through the `memcached` interface is stored to disk, and calls to `get` return the most recent value from disk. Although the default behavior does not offer the best possible raw performance, it is still fast compared to the SQL interface for `InnoDB` tables.

As you gain experience using the `daemon_memcached` plugin, you can consider relaxing durability settings for non-critical classes of data, at the risk of losing some updated values in the event of an outage, or returning data that is slightly out-of-date.

Frequency of Commits

One tradeoff between durability and raw performance is how frequently new and changed data is committed. If data is critical, is should be committed immediately so that it is safe in case of an unexpected exit or outage. If data is less critical, such as counters that are reset after an unexpected exit or logging data that you can afford to lose, you might prefer higher raw throughput that is available with less frequent commits.

When a `memcached` operation inserts, updates, or deletes data in the underlying `InnoDB` table, the change might be committed to the `InnoDB` table instantly (if `daemon_memcached_w_batch_size=1`) or some time later (if the `daemon_memcached_w_batch_size` value is greater than 1). In either case, the change cannot be rolled back. If you increase the value of `daemon_memcached_w_batch_size` to avoid high I/O overhead during busy times, commits could become infrequent when the workload decreases. As a safety measure, a background thread automatically commits changes made through the `memcached` API at regular intervals. The interval

is controlled by the `innodb_api_bk_commit_interval` configuration option, which has a default setting of `5` seconds.

When a `memcached` operation inserts or updates data in the underlying `InnoDB` table, the changed data is immediately visible to other `memcached` requests because the new value remains in the memory cache, even if it is not yet committed on the MySQL side.

Transaction Isolation

When a `memcached` operation such as `get` or `incr` causes a query or DML operation on the underlying `InnoDB` table, you can control whether the operation sees the very latest data written to the table, only data that has been committed, or other variations of transaction isolation level. Use the `innodb_api_trx_level` configuration option to control this feature. The numeric values specified for this option correspond to isolation levels such as `REPEATABLE READ`. See the description of the `innodb_api_trx_level` option for information about other settings.

A strict isolation level ensures that data you retrieve is not rolled back or changed suddenly causing subsequent queries to return different values. However, strict isolation levels require greater locking overhead, which can cause waits. For a NoSQL-style application that does not use long-running transactions, you can typically use the default isolation level or switch to a less strict isolation level.

Disabling Row Locks for memcached DML Operations

The `innodb_api_disable_rowlock` option can be used to disable row locks when `memcached` requests through the `daemon_memcached` plugin cause DML operations. By default, `innodb_api_disable_rowlock` is set to `OFF` which means that `memcached` requests row locks for `get` and `set` operations. When `innodb_api_disable_rowlock` is set to `ON`, `memcached` requests a table lock instead of row locks.

The `innodb_api_disable_rowlock` option is not dynamic. It must be specified at startup on the `mysqld` command line or entered in a MySQL configuration file.

Allowing or Disallowing DDL

By default, you can perform DDL operations such as `ALTER TABLE` on tables used by the `daemon_memcached` plugin. To avoid potential slowdowns when these tables are used for high-throughput applications, disable DDL operations on these tables by enabling `innodb_api_enable_mdl` at startup. This option is less appropriate when accessing the same tables through both `memcached` and SQL, because it blocks `CREATE INDEX` statements on the tables, which could be important for running reporting queries.

Storing Data on Disk, in Memory, or Both

The `innodb_memcache.cache_policies` table specifies whether to store data written through the `memcached` interface to disk (`innodb_only`, the default); in memory only, as with traditional `memcached` (`cache_only`); or both (`caching`).

With the `caching` setting, if `memcached` cannot find a key in memory, it searches for the value in an `InnoDB` table. Values returned from `get` calls under the `caching` setting could be out-of-date if the values were updated on disk in the `InnoDB` table but are not yet expired from the memory cache.

The caching policy can be set independently for `get`, `set` (including `incr` and `decr`), `delete`, and `flush` operations.

For example, you might allow `get` and `set` operations to query or update a table and the `memcached` memory cache at the same time (using the `caching` setting), while making `delete`, `flush`, or both operate only on the in-memory copy (using the `cache_only` setting). That way, deleting or flushing an item only expires the item from the cache, and the latest value is returned from the `InnoDB` table the next time the item is requested.

```
mysql> SELECT * FROM innodb_memcache.cache_policies;
+--------------+-------------+-------------+---------------+--------------+
| policy_name  | get_policy  | set_policy  | delete_policy | flush_policy |
+--------------+-------------+-------------+---------------+--------------+
| cache_policy | innodb_only | innodb_only | innodb_only   | innodb_only  |
+--------------+-------------+-------------+---------------+--------------+

mysql> UPDATE innodb_memcache.cache_policies SET set_policy = 'caching'
       WHERE policy_name = 'cache_policy';
```

`innodb_memcache.cache_policies` values are only read at startup. After changing values in this table, uninstall and reinstall the `daemon_memcached` plugin to ensure that changes take effect.

```
mysql> UNINSTALL PLUGIN daemon_memcached;

mysql> INSTALL PLUGIN daemon_memcached soname "libmemcached.so";
```

14.20.5.5 Adapting DML Statements to memcached Operations

Benchmarks suggest that the `daemon_memcached` plugin speeds up DML operations (inserts, updates, and deletes) more than it speeds up queries. Therefore, consider focussing initial development efforts on write-intensive applications that are I/O-bound, and look for opportunities to use MySQL with the `daemon_memcached` plugin for new write-intensive applications.

Single-row DML statements are the easiest types of statements to turn into `memcached` operations. `INSERT` becomes `add`, `UPDATE` becomes `set`, `incr` or `decr`, and `DELETE` becomes `delete`. These operations are guaranteed to only affect one row when issued through the `memcached` interface, because the *`key`* is unique within the table.

In the following SQL examples, `t1` refers to the table used for `memcached` operations, based on the configuration in the `innodb_memcache.containers` table. `key` refers to the column listed under `key_columns`, and `val` refers to the column listed under `value_columns`.

```
INSERT INTO t1 (key,val) VALUES (some_key,some_value);
SELECT val FROM t1 WHERE key = some_key;
UPDATE t1 SET val = new_value WHERE key = some_key;
UPDATE t1 SET val = val + x WHERE key = some_key;
DELETE FROM t1 WHERE key = some_key;
```

The following `TRUNCATE TABLE` and `DELETE` statements, which remove all rows from the table, correspond to the `flush_all` operation, where `t1` is configured as the table for `memcached` operations, as in the previous example.

```
TRUNCATE TABLE t1;
DELETE FROM t1;
```

14.20.5.6 Performing DML and DDL Statements on the Underlying InnoDB Table

You can access the underlying `InnoDB` table (which is `test.demo_test` by default) through standard SQL interfaces. However, there are some restrictions:

- When querying a table that is also accessed through the `memcached` interface, remember that `memcached` operations can be configured to be committed periodically rather than after every write operation. This behavior is controlled by the `daemon_memcached_w_batch_size` option. If this option is set to a value greater than `1`, use `READ UNCOMMITTED` queries to find rows that were just inserted.

  ```
  mysql> SET SESSSION TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;

  mysql> SELECT * FROM demo_test;
  +------+------+------+------+-----------+------+------+------+------+------+------+
  | cx   | cy   | c1   | cz   | c2        | ca   | CB   | c3   | cu   | c4   | C5   |
  +------+------+------+------+-----------+------+------+------+------+------+------+
  | NULL | NULL | a11  | NULL | 123456789 | NULL | NULL |   10 | NULL |    3 | NULL |
  ```

```
+------+------+------+------+-----------+------+------+------+------+------+------+
```

- When modifying a table using SQL that is also accessed through the `memcached` interface, you can configure `memcached` operations to start a new transaction periodically rather than for every read operation. This behavior is controlled by the `daemon_memcached_r_batch_size` option. If this option is set to a value greater than `1`, changes made to the table using SQL are not immediately visible to `memcached` operations.

- The `InnoDB` table is either IS (intention shared) or IX (intention exclusive) locked for all operations in a transaction. If you increase `daemon_memcached_r_batch_size` and `daemon_memcached_w_batch_size` substantially from their default value of `1`, the table is most likely locked between each operation, preventing DDL statements on the table.

14.20.6 The InnoDB memcached Plugin and Replication

Because the `daemon_memcached` plugin supports the MySQL binary log, updates made on a source server through the `memcached` interface can be replicated for backup, balancing intensive read workloads, and high availability. All `memcached` commands are supported with binary logging.

You do not need to set up the `daemon_memcached` plugin on replica servers. The primary advantage of this configuration is increased write throughput on the source. The speed of the replication mechanism is not affected.

The following sections show how to use the binary log capability when using the `daemon_memcached` plugin with MySQL replication. It is assumed that you have completed the setup described in Section 14.20.3, “Setting Up the InnoDB memcached Plugin”.

Enabling the InnoDB memcached Binary Log

1. To use the `daemon_memcached` plugin with the MySQL binary log, enable the `innodb_api_enable_binlog` configuration option on the source server. This option can only be set at server startup. You must also enable the MySQL binary log on the source server using the `--log-bin` option. You can add these options to the MySQL configuration file, or on the `mysqld` command line.

   ```
   mysqld ... --log-bin --innodb_api_enable_binlog=1
   ```

2. Configure the source and replica server, as described in Section 17.1.1, “How to Set Up Replication”.

3. Use `mysqldump` to create a source data snapshot, and sync the snapshot to the replica server.

   ```
   source shell> mysqldump --all-databases --lock-all-tables > dbdump.db
   replica shell> mysql < dbdump.db
   ```

4. On the source server, issue `SHOW MASTER STATUS` to obtain the source binary log coordinates.

   ```
   mysql> SHOW MASTER STATUS;
   ```

5. On the replica server, use a `CHANGE MASTER TO` statement to set up a replica server using the source binary log coordinates.

   ```
   mysql> CHANGE MASTER TO
          MASTER_HOST='localhost',
          MASTER_USER='root',
          MASTER_PASSWORD='',
          MASTER_PORT = 13000,
          MASTER_LOG_FILE='0.000001,
          MASTER_LOG_POS=114;
   ```

6. Start the replica.

   ```
   mysql> START SLAVE;
   ```

If the error log prints output similar to the following, the replica is ready for replication.

```
2013-09-24T13:04:38.639684Z 49 [Note] Slave I/O thread: connected to
master 'root@localhost:13000', replication started in log '0.000001'
at position 114
```

Testing the InnoDB memcached Replication Configuration

This example demonstrates how to test the `InnoDB memcached` replication configuration using the `memcached` and telnet to insert, update, and delete data. A MySQL client is used to verify results on the source and replica servers.

The example uses the `demo_test` table, which was created by the `innodb_memcached_config.sql` configuration script during the initial setup of the `daemon_memcached` plugin. The `demo_test` table contains a single example record.

1. Use the `set` command to insert a record with a key of `test1`, a flag value of `10`, an expiration value of `0`, a cas value of 1, and a value of `t1`.

```
telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
set test1 10 0 1
t1
STORED
```

2. On the source server, check that the record was inserted into the `demo_test` table. Assuming the `demo_test` table was not previously modified, there should be two records. The example record with a key of `AA`, and the record you just inserted, with a key of `test1`. The `c1` column maps to the key, the `c2` column to the value, the `c3` column to the flag value, the `c4` column to the cas value, and the `c5` column to the expiration time. The expiration time was set to 0, since it is unused.

```
mysql> SELECT * FROM test.demo_test;
+-------+--------------+------+------+------+
| c1    | c2           | c3   | c4   | c5   |
+-------+--------------+------+------+------+
| AA    | HELLO, HELLO |    8 |    0 |    0 |
| test1 | t1           |   10 |    1 |    0 |
+-------+--------------+------+------+------+
```

3. Check to verify that the same record was replicated to the replica server.

```
mysql> SELECT * FROM test.demo_test;
+-------+--------------+------+------+------+
| c1    | c2           | c3   | c4   | c5   |
+-------+--------------+------+------+------+
| AA    | HELLO, HELLO |    8 |    0 |    0 |
| test1 | t1           |   10 |    1 |    0 |
+-------+--------------+------+------+------+
```

4. Use the `set` command to update the key to a value of `new`.

```
telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
set test1 10 0 2
new
STORED
```

The update is replicated to the replica server (notice that the `cas` value is also updated).

```
mysql> SELECT * FROM test.demo_test;
+-------+--------------+------+------+------+
| c1    | c2           | c3   | c4   | c5   |
```

```
+-------+--------------+------+------+------+
| AA    | HELLO, HELLO |    8 |    0 |    0 |
| test1 | new          |   10 |    2 |    0 |
+-------+--------------+------+------+------+
```

5. Delete the `test1` record using a `delete` command.

```
telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
delete test1
DELETED
```

When the `delete` operation is replicated to the replica, the `test1` record on the replica is also deleted.

```
mysql> SELECT * FROM test.demo_test;
+----+--------------+------+------+------+
| c1 | c2           | c3   | c4   | c5   |
+----+--------------+------+------+------+
| AA | HELLO, HELLO |    8 |    0 |    0 |
+----+--------------+------+------+------+
```

6. Remove all rows from the table using the `flush_all` command.

```
telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
flush_all
OK
```

```
mysql> SELECT * FROM test.demo_test;
Empty set (0.00 sec)
```

7. Telnet to the source server and enter two new records.

```
telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'
set test2 10 0 4
again
STORED
set test3 10 0 5
again1
STORED
```

8. Confirm that the two records were replicated to the replica server.

```
mysql> SELECT * FROM test.demo_test;
+-------+--------------+------+------+------+
| c1    | c2           | c3   | c4   | c5   |
+-------+--------------+------+------+------+
| test2 | again        |   10 |    4 |    0 |
| test3 | again1       |   10 |    5 |    0 |
+-------+--------------+------+------+------+
```

9. Remove all rows from the table using the `flush_all` command.

```
telnet 127.0.0.1 11211
Trying 127.0.0.1...
Connected to 127.0.0.1.
Escape character is '^]'.
flush_all
OK
```

10. Check to ensure that the `flush_all` operation was replicated on the replica server.

```
mysql> SELECT * FROM test.demo_test;
```

```
Empty set (0.00 sec)
```

InnoDB memcached Binary Log Notes

Binary Log Format:

- Most `memcached` operations are mapped to DML statements (analogous to insert, delete, update). Since there is no actual SQL statement being processed by the MySQL server, all `memcached` commands (except for `flush_all`) use Row-Based Replication (RBR) logging, which is independent of any server `binlog_format` setting.
- The `memcached flush_all` command is mapped to the `TRUNCATE TABLE` command. Since DDL commands can only use statement-based logging, the `flush_all` command is replicated by sending a `TRUNCATE TABLE` statement.

Transactions:

- The concept of transactions has not typically been part of `memcached` applications. For performance considerations, `daemon_memcached_r_batch_size` and `daemon_memcached_w_batch_size` are used to control the batch size for read and write transactions. These settings do not affect replication. Each SQL operation on the underlying `InnoDB` table is replicated after successful completion.
- The default value of `daemon_memcached_w_batch_size` is `1`, which means that each `memcached` write operation is committed immediately. This default setting incurs a certain amount of performance overhead to avoid inconsistencies in the data that is visible on the source and replica servers. The replicated records are always available immediately on the replica server. If you set `daemon_memcached_w_batch_size` to a value greater than `1`, records inserted or updated through `memcached` are not immediately visible on the source server; to view the records on the source server before they are committed, issue `SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED`.

14.20.7 InnoDB memcached Plugin Internals

InnoDB API for the InnoDB memcached Plugin

The `InnoDB memcached` engine accesses `InnoDB` through `InnoDB` APIs, most of which are directly adopted from embedded `InnoDB`. `InnoDB` API functions are passed to the `InnoDB memcached` engine as callback functions. `InnoDB` API functions access the `InnoDB` tables directly, and are mostly DML operations with the exception of `TRUNCATE TABLE`.

`memcached` commands are implemented through the `InnoDB memcached` API. The following table outlines how `memcached` commands are mapped to DML or DDL operations.

Table 14.16 memcached Commands and Associated DML or DDL Operations

memcached Command	DML or DDL Operations
`get`	a read/fetch command
`set`	a search followed by an `INSERT` or `UPDATE` (depending on whether or not a key exists)
`add`	a search followed by an `INSERT` or `UPDATE`
`replace`	a search followed by an `UPDATE`
`append`	a search followed by an `UPDATE` (appends data to the result before `UPDATE`)
`prepend`	a search followed by an `UPDATE` (prepends data to the result before `UPDATE`)
`incr`	a search followed by an `UPDATE`
`decr`	a search followed by an `UPDATE`

memcached Command	DML or DDL Operations
delete	a search followed by a DELETE
flush_all	TRUNCATE TABLE (DDL)

InnoDB memcached Plugin Configuration Tables

This section describes configuration tables used by the daemon_memcached plugin. The cache_policies table, config_options table, and containers table are created by the innodb_memcached_config.sql configuration script in the innodb_memcache database.

```
mysql> USE innodb_memcache;
Database changed
mysql> SHOW TABLES;
+---------------------------+
| Tables_in_innodb_memcache |
+---------------------------+
| cache_policies            |
| config_options            |
| containers                |
+---------------------------+
```

cache_policies Table

The cache_policies table defines a cache policy for the InnoDB memcached installation. You can specify individual policies for get, set, delete, and flush operations, within a single cache policy. The default setting for all operations is innodb_only.

- innodb_only: Use InnoDB as the data store.
- cache_only: Use the memcached engine as the data store.
- caching: Use both InnoDB and the memcached engine as data stores. In this case, if memcached cannot find a key in memory, it searches for the value in an InnoDB table.
- disable: Disable caching.

Table 14.17 cache_policies Columns

Column	Description
policy_name	Name of the cache policy. The default cache policy name is cache_policy.
get_policy	The cache policy for get operations. Valid values are innodb_only, cache_only, caching, or disabled. The default setting is innodb_only.
set_policy	The cache policy for set operations. Valid values are innodb_only, cache_only, caching, or disabled. The default setting is innodb_only.
delete_policy	The cache policy for delete operations. Valid values are innodb_only, cache_only, caching, or disabled. The default setting is innodb_only.
flush_policy	The cache policy for flush operations. Valid values are innodb_only, cache_only, caching, or disabled. The default setting is innodb_only.

config_options Table

The config_options table stores memcached-related settings that can be changed at runtime using SQL. Supported configuration options are separator and table_map_delimiter.

Table 14.18 config_options Columns

Column	Description

Name	Name of the memcached-related configuration option. The following configuration options are supported by the config_options table:
	
	• separator: Used to separate values of a long string into separate values when there are multiple value_columns defined. By default, the separator is a | character. For example, if you define col1, col2 as value columns, and you define | as the separator, you can issue the following memcached command to insert values into col1 and col2, respectively:
	
	`set keyx 10 0 19`
	`valuecolx|valuecoly`
	
	valuecol1x is stored in col1 and valuecoly is stored in col2.
	
	• table_map_delimiter: The character separating the schema name and the table name when you use the @@ notation in a key name to access a key in a specific table. For example, @@t1.some_key and @@t2.some_key have the same key value, but are stored in different tables.

Value	The value assigned to the memcached-related configuration option.

containers Table

The containers table is the most important of the three configuration tables. Each InnoDB table that is used to store memcached values must have an entry in the containers table. The entry provides a mapping between InnoDB table columns and container table columns, which is required for memcached to work with InnoDB tables.

The containers table contains a default entry for the test.demo_test table, which is created by the innodb_memcached_config.sql configuration script. To use the daemon_memcached plugin with your own InnoDB table, you must create an entry in the containers table.

Table 14.19 containers Columns

Column	Description
name	The name given to the container. If an InnoDB table is not requested by name using @@ notation, the daemon_memcached plugin uses the InnoDB table with a containers.name value of default. If there is no such entry, the first entry in the containers table, ordered alphabetically by name (ascending), determines the default InnoDB table.
db_schema	The name of the database where the InnoDB table resides. This is a required value.
db_table	The name of the InnoDB table that stores memcached values. This is a required value.
key_columns	The column in the InnoDB table that contains lookup key values for memcached operations. This is a required value.
value_columns	The InnoDB table columns (one or more) that store memcached data. Multiple columns can be specified using the separator character specified in the innodb_memcached.config_options table. By default, the separator is a pipe character ("|"). To specify multiple columns, separate them with the defined separator character. For example: col1|col2|col3. This is a required value.
flags	The InnoDB table columns that are used as flags (a user-defined numeric value that is stored and retrieved along with the main value)

Column	Description
	for `memcached`. A flag value can be used as a column specifier for some operations (such as `incr`, `prepend`) if a `memcached` value is mapped to multiple columns, so that an operation is performed on a specified column. For example, if you have mapped a `value_columns` to three `InnoDB` table columns, and only want the increment operation performed on one columns, use the `flags` column to specify the column. If you do not use the `flags` column, set a value of `0` to indicate that it is unused.
`cas_column`	The `InnoDB` table column that stores compare-and-swap (cas) values. The `cas_column` value is related to the way `memcached` hashes requests to different servers and caches data in memory. Because the `InnoDB` `memcached` plugin is tightly integrated with a single `memcached` daemon, and the in-memory caching mechanism is handled by MySQL and the InnoDB buffer pool, this column is rarely needed. If you do not use this column, set a value of `0` to indicate that it is unused.
`expire_time_column`	The `InnoDB` table column that stores expiration values. The `expire_time_column` value is related to the way `memcached` hashes requests to different servers and caches data in memory. Because the `InnoDB` `memcached` plugin is tightly integrated with a single `memcached` daemon, and the in-memory caching mechanism is handled by MySQL and the buffer pool, this columns is rarely needed. If you do not use this column, set a value of `0` to indicate that it is unused. As of MySQL 5.6.25, maximum expire time is defined as `INT_MAX32` or 2147483647 seconds (approximately 68 years).
`unique_idx_name_on_key`	The name of the index on the key column. It must be a unique index. It can be the primary key or a secondary index. Preferably, use the primary key of the `InnoDB` table. Using the primary key avoids a lookup that is performed when using a secondary index. You cannot make a covering index for `memcached` lookups; `InnoDB` returns an error if you try to define a composite secondary index over both the key and value columns.

containers Table Column Constraints

- You must supply a value for `db_schema`, `db_name`, `key_columns`, `value_columns` and `unique_idx_name_on_key`. Specify `0` for `flags`, `cas_column`, and `expire_time_column` if they are unused. Failing to do so could cause your setup to fail.
- `key_columns`: The maximum limit for a `memcached` key is 250 characters, which is enforced by `memcached`. The mapped key must be a non-Null `CHAR` or `VARCHAR` type.
- `value_columns`: Must be mapped to a `CHAR`, `VARCHAR`, or `BLOB` column. There is no length restriction and the value can be NULL.
- `cas_column`: The `cas` value is a 64 bit integer. It must be mapped to a `BIGINT` of at least 8 bytes. If you do not use this column, set a value of `0` to indicate that it is unused.
- `expiration_time_column`: Must mapped to an `INTEGER` of at least 4 bytes. Expiration time is defined as a 32-bit integer for Unix time (the number of seconds since January 1, 1970, as a 32-bit value), or the number of seconds starting from the current time. For the latter, the number of seconds may not exceed 60*60*24*30 (the number of seconds in 30 days). If the number sent by a client is larger, the server considers it to be a real Unix time value rather than an offset from the current time. If you do not use this column, set a value of `0` to indicate that it is unused.
- `flags`: Must be mapped to an `INTEGER` of at least 32-bits and can be NULL. If you do not use this column, set a value of `0` to indicate that it is unused.

A pre-check is performed at plugin load time to enforce column constraints. If mismatches are found, the plugin is not loaded.

Multiple Value Column Mapping

- During plugin initialization, when `InnoDB memcached` is configured with information defined in the `containers` table, each mapped column defined in `containers.value_columns` is verified against the mapped `InnoDB` table. If multiple `InnoDB` table columns are mapped, there is a check to ensure that each column exists and is the right type.
- At run-time, for `memcached` insert operations, if there are more delimited values than the number of mapped columns, only the number of mapped values are taken. For example, if there are six mapped columns, and seven delimited values are provided, only the first six delimited values are taken. The seventh delimited value is ignored.
- If there are fewer delimited values than mapped columns, unfilled columns are set to NULL. If an unfilled column cannot be set to NULL, insert operations fail.
- If a table has more columns than mapped values, the extra columns do not affect results.

The demo_test Example Table

The `innodb_memcached_config.sql` configuration script creates a `demo_test` table in the `test` database, which can be used to verify `InnoDB memcached` plugin installation immediately after setup.

The `innodb_memcached_config.sql` configuration script also creates an entry for the `demo_test` table in the `innodb_memcache.containers` table.

```
mysql> SELECT * FROM innodb_memcache.containers\G
*************************** 1. row ***************************
                name: aaa
           db_schema: test
            db_table: demo_test
         key_columns: c1
       value_columns: c2
               flags: c3
          cas_column: c4
  expire_time_column: c5
unique_idx_name_on_key: PRIMARY

mysql> SELECT * FROM test.demo_test;
+----+------------------+------+------+------+
| c1 | c2               | c3   | c4   | c5   |
+----+------------------+------+------+------+
| AA | HELLO, HELLO     |    8 |    0 |    0 |
+----+------------------+------+------+------+
```

14.20.8 Troubleshooting the InnoDB memcached Plugin

This section describes issues that you may encounter when using the `InnoDB memcached` plugin.

- If you encounter the following error in the MySQL error log, the server might fail to start:

  ```
  failed to set rlimit for open files. Try running as root or requesting
  smaller maxconns value.
  ```

  The error message is from the `memcached` daemon. One solution is to raise the OS limit for the number of open files. The commands for checking and increasing the open file limit varies by operating system. This example shows commands for Linux and macOS:

  ```
  # Linux
  shell> ulimit -n
  1024
  shell> ulimit -n 4096
  shell> ulimit -n
  ```

```
4096

# macOS
shell> ulimit -n
256
shell> ulimit -n 4096
shell> ulimit -n
4096
```

The other solution is to reduce the number of concurrent connections permitted for the `memcached` daemon. To do so, encode the `-c memcached` option in the `daemon_memcached_option` configuration parameter in the MySQL configuration file. The `-c` option has a default value of 1024.

```
[mysqld]
...
loose-daemon_memcached_option='-c 64'
```

- To troubleshoot problems where the `memcached` daemon is unable to store or retrieve `InnoDB` table data, encode the `-vvv memcached` option in the `daemon_memcached_option` configuration parameter in the MySQL configuration file. Examine the MySQL error log for debug output related to `memcached` operations.

  ```
  [mysqld]
  ...
  loose-daemon_memcached_option='-vvv'
  ```

- If columns specified to hold `memcached` values are the wrong data type, such as a numeric type instead of a string type, attempts to store key-value pairs fail with no specific error code or message.

- If the `daemon_memcached` plugin causes MySQL server startup issues, you can temporarily disable the `daemon_memcached` plugin while troubleshooting by adding this line under the `[mysqld]` group in the MySQL configuration file:

  ```
  daemon_memcached=OFF
  ```

  For example, if you run the `INSTALL PLUGIN` statement before running the `innodb_memcached_config.sql` configuration script to set up the necessary database and tables, the server might unexpectedly exit and fail to start. The server could also fail to start if you incorrectly configure an entry in the `innodb_memcache.containers` table.

  To uninstall the `memcached` plugin for a MySQL instance, issue the following statement:

  ```
  mysql> UNINSTALL PLUGIN daemon_memcached;
  ```

- If you run more than one instance of MySQL on the same machine with the `daemon_memcached` plugin enabled in each instance, use the `daemon_memcached_option` configuration parameter to specify a unique `memcached` port for each `daemon_memcached` plugin.

- If an SQL statement cannot find the `InnoDB` table or finds no data in the table, but `memcached` API calls retrieve the expected data, you may be missing an entry for the `InnoDB` table in the `innodb_memcache.containers` table, or you may have not switched to the correct `InnoDB` table by issuing a `get` or `set` request using `@@table_id` notation. This problem could also occur if you change an existing entry in the `innodb_memcache.containers` table without restarting the MySQL server afterward. The free-form storage mechanism is flexible enough that your requests to store or retrieve a multi-column value such as `col1|col2|col3` may still work, even if the daemon is using the `test.demo_test` table which stores values in a single column.

- When defining your own `InnoDB` table for use with the `daemon_memcached` plugin, and columns in the table are defined as `NOT NULL`, ensure that values are supplied for the `NOT NULL` columns when inserting a record for the table into the `innodb_memcache.containers` table. If the `INSERT` statement for the `innodb_memcache.containers` record contains fewer delimited values than there are mapped columns, unfilled columns are set to `NULL`. Attempting to insert a `NULL` value into a `NOT NULL` column causes the `INSERT` to fail, which may only become evident after you reinitialize the `daemon_memcached` plugin to apply changes to the `innodb_memcache.containers` table.

- If `cas_column` and `expire_time_column` fields of the `innodb_memcached.containers` table are set to `NULL`, the following error is returned when attempting to load the `memcached` plugin:

```
InnoDB_Memcached: column 6 in the entry for config table 'containers' in
database 'innodb_memcache' has an invalid NULL value.
```

  The `memcached` plugin rejects usage of `NULL` in the `cas_column` and `expire_time_column` columns. Set the value of these columns to `0` when the columns are unused.

- As the length of the `memcached` key and values increase, you might encounter size and length limits.
  - When the key exceeds 250 bytes, `memcached` operations return an error. This is currently a fixed limit within `memcached`.
  - `InnoDB` table limits may be encountered if values exceed 768 bytes in size, 3072 bytes in size, or half of the `innodb_page_size` value. These limits primarily apply if you intend to create an index on a value column to run report-generating queries on that column using SQL. See Section 14.22, “InnoDB Limits” for details.
  - The maximum size for the key-value combination is 1 MB.
- If you share configuration files across MySQL servers of different versions, using the latest configuration options for the `daemon_memcached` plugin could cause startup errors on older MySQL versions. To avoid compatibility problems, use the `loose` prefix with option names. For example, use `loose-daemon_memcached_option='-c 64'` instead of `daemon_memcached_option='-c 64'`.
- There is no restriction or check in place to validate character set settings. `memcached` stores and retrieves keys and values in bytes and is therefore not character set-sensitive. However, you must ensure that the `memcached` client and the MySQL table use the same character set.

14.21 InnoDB Troubleshooting

The following general guidelines apply to troubleshooting `InnoDB` problems:

- When an operation fails or you suspect a bug, look at the MySQL server error log (see Section 5.4.2, “The Error Log”). Server Error Message Reference provides troubleshooting information for some of the common `InnoDB`-specific errors that you may encounter.
- If the failure is related to a deadlock, run with the `innodb_print_all_deadlocks` option enabled so that details about each deadlock are printed to the MySQL server error log. For information about deadlocks, see Section 14.7.5, “Deadlocks in InnoDB”.
- Issues relating to the `InnoDB` data dictionary include failed `CREATE TABLE` statements (orphan table files), inability to open `InnoDB` files, and `system cannot find the path specified` errors. For information about these sorts of problems and errors, see Section 14.21.3, “Troubleshooting InnoDB Data Dictionary Operations”.
- When troubleshooting, it is usually best to run the MySQL server from the command prompt, rather than through `mysqld_safe` or as a Windows service. You can then see what `mysqld` prints to the console, and so have a better grasp of what is going on. On Windows, start `mysqld` with the `--console` option to direct the output to the console window.
- Enable the `InnoDB` Monitors to obtain information about a problem (see Section 14.17, “InnoDB Monitors”). If the problem is performance-related, or your server appears to be hung, you should enable the standard Monitor to print information about the internal state of `InnoDB`. If the problem is with locks, enable the Lock Monitor. If the problem is in creation of tables or other data dictionary operations, enable the Table Monitor to print the contents of the `InnoDB` internal data dictionary. To see tablespace information enable the Tablespace Monitor.

  `InnoDB` temporarily enables standard `InnoDB` Monitor output under the following conditions:

- A long semaphore wait
- `InnoDB` cannot find free blocks in the buffer pool
- Over 67% of the buffer pool is occupied by lock heaps or the adaptive hash index

- If you suspect that a table is corrupt, run `CHECK TABLE` on that table.

14.21.1 Troubleshooting InnoDB I/O Problems

The troubleshooting steps for `InnoDB` I/O problems depend on when the problem occurs: during startup of the MySQL server, or during normal operations when a DML or DDL statement fails due to problems at the file system level.

Initialization Problems

If something goes wrong when `InnoDB` attempts to initialize its tablespace or its log files, delete all files created by `InnoDB`: all `ibdata` files and all `ib_logfile` files. If you already created some `InnoDB` tables, also delete the corresponding `.frm` files for these tables, and any `.ibd` files if you are using multiple tablespaces, from the MySQL database directories. Then try the `InnoDB` database creation again. For easiest troubleshooting, start the MySQL server from a command prompt so that you see what is happening.

Runtime Problems

If `InnoDB` prints an operating system error during a file operation, usually the problem has one of the following solutions:

- Make sure the `InnoDB` data file directory and the `InnoDB` log directory exist.
- Make sure `mysqld` has access rights to create files in those directories.
- Make sure `mysqld` can read the proper `my.cnf` or `my.ini` option file, so that it starts with the options that you specified.
- Make sure the disk is not full and you are not exceeding any disk quota.
- Make sure that the names you specify for subdirectories and data files do not clash.
- Doublecheck the syntax of the `innodb_data_home_dir` and `innodb_data_file_path` values. In particular, any `MAX` value in the `innodb_data_file_path` option is a hard limit, and exceeding that limit causes a fatal error.

14.21.2 Forcing InnoDB Recovery

To investigate database page corruption, you might dump your tables from the database with `SELECT ... INTO OUTFILE`. Usually, most of the data obtained in this way is intact. Serious corruption might cause `SELECT * FROM tbl_name` statements or `InnoDB` background operations to unexpectedly exit or assert, or even cause `InnoDB` roll-forward recovery to crash. In such cases, you can use the `innodb_force_recovery` option to force the `InnoDB` storage engine to start up while preventing background operations from running, so that you can dump your tables. For example, you can add the following line to the `[mysqld]` section of your option file before restarting the server:

```
[mysqld]
innodb_force_recovery = 1
```

For information about using option files, see Section 4.2.2.2, “Using Option Files”.

Warning

Only set `innodb_force_recovery` to a value greater than 0 in an emergency situation, so that you can start `InnoDB` and dump your tables. Before doing

> so, ensure that you have a backup copy of your database in case you need to recreate it. Values of 4 or greater can permanently corrupt data files. Only use an `innodb_force_recovery` setting of 4 or greater on a production server instance after you have successfully tested the setting on a separate physical copy of your database. When forcing `InnoDB` recovery, you should always start with `innodb_force_recovery=1` and only increase the value incrementally, as necessary.

`innodb_force_recovery` is 0 by default (normal startup without forced recovery). The permissible nonzero values for `innodb_force_recovery` are 1 to 6. A larger value includes the functionality of lesser values. For example, a value of 3 includes all of the functionality of values 1 and 2.

If you are able to dump your tables with an `innodb_force_recovery` value of 3 or less, then you are relatively safe that only some data on corrupt individual pages is lost. A value of 4 or greater is considered dangerous because data files can be permanently corrupted. A value of 6 is considered drastic because database pages are left in an obsolete state, which in turn may introduce more corruption into B-trees and other database structures.

As a safety measure, `InnoDB` prevents `INSERT`, `UPDATE`, or `DELETE` operations when `innodb_force_recovery` is greater than 0. As of MySQL 5.6.15, an `innodb_force_recovery` setting of 4 or greater places `InnoDB` in read-only mode.

- `1` (`SRV_FORCE_IGNORE_CORRUPT`)

  Lets the server run even if it detects a corrupt page. Tries to make `SELECT * FROM tbl_name` jump over corrupt index records and pages, which helps in dumping tables.

- `2` (`SRV_FORCE_NO_BACKGROUND`)

  Prevents the master thread and any purge threads from running. If an unexpected exit would occur during the purge operation, this recovery value prevents it.

- `3` (`SRV_FORCE_NO_TRX_UNDO`)

  Does not run transaction rollbacks after crash recovery.

- `4` (`SRV_FORCE_NO_IBUF_MERGE`)

  Prevents insert buffer merge operations. If they would cause a crash, does not do them. Does not calculate table statistics. This value can permanently corrupt data files. After using this value, be prepared to drop and recreate all secondary indexes. As of MySQL 5.6.15, sets `InnoDB` to read-only.

- `5` (`SRV_FORCE_NO_UNDO_LOG_SCAN`)

  Does not look at undo logs when starting the database: `InnoDB` treats even incomplete transactions as committed. This value can permanently corrupt data files. As of MySQL 5.6.15, sets `InnoDB` to read-only.

- `6` (`SRV_FORCE_NO_LOG_REDO`)

  Does not do the redo log roll-forward in connection with recovery. This value can permanently corrupt data files. Leaves database pages in an obsolete state, which in turn may introduce more corruption into B-trees and other database structures. As of MySQL 5.6.15, sets `InnoDB` to read-only.

You can `SELECT` from tables to dump them. With an `innodb_force_recovery` value of 3 or less you can `DROP` or `CREATE` tables. As of MySQL 5.6.27, `DROP TABLE` is also supported with an `innodb_force_recovery` value greater than 3.

If you know that a given table is causing an unexpected exit on rollback, you can drop it. If you encounter a runaway rollback caused by a failing mass import or `ALTER TABLE`, you can kill the

`mysqld` process and set `innodb_force_recovery` to `3` to bring the database up without the rollback, and then `DROP` the table that is causing the runaway rollback.

If corruption within the table data prevents you from dumping the entire table contents, a query with an `ORDER BY primary_key DESC` clause might be able to dump the portion of the table after the corrupted part.

If a high `innodb_force_recovery` value is required to start `InnoDB`, there may be corrupted data structures that could cause complex queries (queries containing `WHERE`, `ORDER BY`, or other clauses) to fail. In this case, you may only be able to run basic `SELECT * FROM t` queries.

14.21.3 Troubleshooting InnoDB Data Dictionary Operations

Information about table definitions is stored both in the `.frm` files, and in the InnoDB data dictionary. If you move `.frm` files around, or if the server crashes in the middle of a data dictionary operation, these sources of information can become inconsistent.

If a data dictionary corruption or consistency issue prevents you from starting `InnoDB`, see Section 14.21.2, “Forcing InnoDB Recovery” for information about manual recovery.

CREATE TABLE Failure Due to Orphan Table

A symptom of an out-of-sync data dictionary is that a `CREATE TABLE` statement fails. If this occurs, look in the server's error log. If the log says that the table already exists inside the `InnoDB` internal data dictionary, you have an orphan table inside the `InnoDB` tablespace files that has no corresponding `.frm` file. The error message looks like this:

```
InnoDB: Error: table test/parent already exists in InnoDB internal
InnoDB: data dictionary. Have you deleted the .frm file
InnoDB: and not used DROP TABLE? Have you used DROP DATABASE
InnoDB: for InnoDB tables in MySQL version <= 3.23.43?
InnoDB: See the Restrictions section of the InnoDB manual.
InnoDB: You can drop the orphaned table inside InnoDB by
InnoDB: creating an InnoDB table with the same name in another
InnoDB: database and moving the .frm file to the current database.
InnoDB: Then MySQL thinks the table exists, and DROP TABLE will
InnoDB: succeed.
```

You can drop the orphan table by following the instructions given in the error message. If you are still unable to use `DROP TABLE` successfully, the problem may be due to name completion in the `mysql` client. To work around this problem, start the `mysql` client with the `--skip-auto-rehash` option and try `DROP TABLE` again. (With name completion on, `mysql` tries to construct a list of table names, which fails when a problem such as just described exists.)

Cannot Open File Error

Another symptom of an out-of-sync data dictionary is that MySQL prints an error that it cannot open an `InnoDB` file:

```
ERROR 1016: Can't open file: 'child2.ibd'. (errno: 1)
```

In the error log you can find a message like this:

```
InnoDB: Cannot find table test/child2 from the internal data dictionary
InnoDB: of InnoDB though the .frm file for the table exists. Maybe you
InnoDB: have deleted and recreated InnoDB data files but have forgotten
InnoDB: to delete the corresponding .frm files of InnoDB tables?
```

This means that there is an orphan `.frm` file without a corresponding table inside `InnoDB`. You can drop the orphan `.frm` file by deleting it manually.

Orphan Intermediate Tables

If MySQL exits in the middle of an in-place `ALTER TABLE` operation (`ALGORITHM=INPLACE`), you may be left with an orphan intermediate table that takes up space on your system. This section describes how to identify and remove orphan intermediate tables.

Intermediate table names begin with an `#sql-ib` prefix (e.g., `#sql-ib87-856498050`). The accompanying `.frm` file has an `#sql-*` prefix and is named differently (e.g., `#sql-36ab_2.frm`).

To identify orphan intermediate tables on your system, you can view Table Monitor output or query `INFORMATION_SCHEMA.INNODB_SYS_TABLES`. Look for table names that begin with `#sql`. If the original table resides in a file-per-table tablespace, the tablespace file (the `#sql-*.ibd` file) for the orphan intermediate table should be visible in the database directory.

```
SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_TABLES WHERE NAME LIKE '%#sql%';
```

To remove an orphan intermediate table, perform the following steps:

1. In the database directory, rename the `#sql-*.frm` file to match the base name of the orphan intermediate table:

   ```
   shell> mv #sql-36ab_2.frm #sql-ib87-856498050.frm
   ```

   Note

   If there is no `.frm` file, you can recreate it. The `.frm` file must have the same table schema as the orphan intermediate table (it must have the same columns and indexes) and must be placed in the database directory of the orphan intermediate table.

2. Drop the orphan intermediate table by issuing a `DROP TABLE` statement, prefixing the name of the table with `#mysql50#` and enclosing the table name in backticks. For example:

   ```
   mysql> DROP TABLE `#mysql50##sql-ib87-856498050`;
   ```

   The `#mysql50#` prefix tells MySQL to ignore `file name safe encoding` introduced in MySQL 5.1. Enclosing the table name in backticks is required to perform SQL statements on table names with special characters such as “#”.

Note

If MySQL exits in the middle of an in-place `ALTER TABLE` operation on a partitioned table, you may be left with multiple orphan intermediate tables, one per partition. In this case, use the following procedure to remove the orphan intermediate tables:

1. In a separate instance of the same MySQL version, create a non-partitioned table with the same schema name and columns as the partitioned table.
2. Copy the `.frm` file of the non-partitioned table to the database directory with the orphan intermediate tables.
3. Make a copy of the `.frm` file for each table, and rename the `.frm` files to match names of the orphan intermediate tables (as described above).
4. Perform a `DROP TABLE` operation (as described above) for each table.

Orphan Temporary Tables

If MySQL exits in the middle of a table-copying `ALTER TABLE` operation (`ALGORITHM=COPY`), you may be left with an orphan temporary table that takes up space on your system. This section describes how to identify and remove orphan temporary tables.

Orphan temporary table names begin with an `#sql-` prefix (e.g., `#sql-540_3`). The accompanying `.frm` file has the same base name as the orphan temporary table.

Note

If there is no `.frm` file, you can recreate it. The `.frm` file must have the same table schema as the orphan temporary table (it must have the same columns and indexes) and must be placed in the database directory of the orphan temporary table.

To identify orphan temporary tables on your system, you can view Table Monitor output or query `INFORMATION_SCHEMA.INNODB_SYS_TABLES`. Look for table names that begin with `#sql`. If the original table resides in a file-per-table tablespace, the tablespace file (the `#sql-*.ibd` file) for the orphan temporary table should be visible in the database directory.

```
SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_TABLES WHERE NAME LIKE '%#sql%';
```

To remove an orphan temporary table, drop the table by issuing a `DROP TABLE` statement, prefixing the name of the table with `#mysql50#` and enclosing the table name in backticks. For example:

```
mysql> DROP TABLE `#mysql50##sql-540_3`;
```

The `#mysql50#` prefix tells MySQL to ignore `file name safe encoding` introduced in MySQL 5.1. Enclosing the table name in backticks is required to perform SQL statements on table names with special characters such as “#”.

Note

If MySQL exits in the middle of an table-copying `ALTER TABLE` operation on a partitioned table, you may be left with multiple orphan temporary tables, one per partition. In this case, use the following procedure to remove the orphan temporary tables:

1. In a separate instance of the same MySQL version, create a non-partitioned table with the same schema name and columns as the partitioned table.
2. Copy the `.frm` file of the non-partitioned table to the database directory with the orphan temporary tables.
3. Make a copy of the `.frm` file for each table, and rename the `.frm` files to match the names of the orphan temporary tables (as described above).
4. Perform a `DROP TABLE` operation (as described above) for each table.

Tablespace Does Not Exist

With `innodb_file_per_table` enabled, the following message might occur if the `.frm` or `.ibd` files (or both) are missing:

```
InnoDB: in InnoDB data dictionary has tablespace id N,
InnoDB: but tablespace with that id or name does not exist. Have
InnoDB: you deleted or moved .ibd files?
InnoDB: This may also be a table created with CREATE TEMPORARY TABLE
InnoDB: whose .ibd and .frm files MySQL automatically removed, but the
InnoDB: table still exists in the InnoDB internal data dictionary.
```

If this occurs, try the following procedure to resolve the problem:

1. Create a matching `.frm` file in some other database directory and copy it to the database directory where the orphan table is located.
2. Issue `DROP TABLE` for the original table. That should successfully drop the table and `InnoDB` should print a warning to the error log that the `.ibd` file was missing.

Restoring Orphaned File-Per-Table ibd Files

This procedure describes how to restore orphan file-per-table `.ibd` files to another MySQL instance. You might use this procedure if the system tablespace is lost or unrecoverable and you want to restore `.ibd` file backups on a new MySQL instance.

The procedure assumes that you only have `.ibd` file backups, you are recovering to the same version of MySQL that initially created the orphan `.ibd` files, and that `.ibd` file backups are clean. See Section 14.6.1.4, “Moving or Copying InnoDB Tables” for information about creating clean backups.

Table import limitations outlined in Section 14.6.1.3, “Importing InnoDB Tables” are applicable to this procedure.

1. On the new MySQL instance, recreate the table in a database of the same name.

```
mysql> CREATE DATABASE sakila;

mysql> USE sakila;

mysql> CREATE TABLE actor (
         actor_id SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT,
         first_name VARCHAR(45) NOT NULL,
         last_name VARCHAR(45) NOT NULL,
         last_update TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP,
         PRIMARY KEY  (actor_id),
         KEY idx_actor_last_name (last_name)
       )ENGINE=InnoDB DEFAULT CHARSET=utf8;
```

2. Discard the tablespace of the newly created table.

```
mysql> ALTER TABLE sakila.actor DISCARD TABLESPACE;
```

3. Copy the orphan `.ibd` file from your backup directory to the new database directory.

```
shell> cp /backup_directory/actor.ibd path/to/mysql-5.6/data/sakila/
```

4. Ensure that the `.ibd` file has the necessary file permissions.

5. Import the orphan `.ibd` file. A warning is issued indicating that `InnoDB` attempts to import the file without schema verification.

```
mysql> ALTER TABLE sakila.actor IMPORT TABLESPACE; SHOW WARNINGS;
Query OK, 0 rows affected, 1 warning (0.15 sec)

Warning | 1810 | InnoDB: IO Read error: (2, No such file or directory)
Error opening './sakila/actor.cfg', will attempt to import
without schema verification
```

6. Query the table to verify that the `.ibd` file was successfully restored.

```
mysql> SELECT COUNT(*) FROM sakila.actor;
+----------+
| count(*) |
+----------+
|      200 |
+----------+
```

14.21.4 InnoDB Error Handling

The following items describe how `InnoDB` performs error handling. `InnoDB` sometimes rolls back only the statement that failed, other times it rolls back the entire transaction.

- If you run out of file space in a tablespace, a MySQL `Table is full` error occurs and `InnoDB` rolls back the SQL statement.
- A transaction deadlock causes `InnoDB` to roll back the entire transaction. Retry the entire transaction when this happens.

A lock wait timeout causes `InnoDB` to roll back the current statement (the statement that was waiting for the lock and encountered the timeout). To have the entire transaction roll back, start the server with `--innodb-rollback-on-timeout` enabled. Retry the statement if using the default behavior, or the entire transaction if `--innodb-rollback-on-timeout` is enabled.

Both deadlocks and lock wait timeouts are normal on busy servers and it is necessary for applications to be aware that they may happen and handle them by retrying. You can make them less likely by doing as little work as possible between the first change to data during a transaction and the commit, so the locks are held for the shortest possible time and for the smallest possible number of rows. Sometimes splitting work between different transactions may be practical and helpful.

- A duplicate-key error rolls back the SQL statement, if you have not specified the `IGNORE` option in your statement.
- A `row too long error` rolls back the SQL statement.
- Other errors are mostly detected by the MySQL layer of code (above the `InnoDB` storage engine level), and they roll back the corresponding SQL statement. Locks are not released in a rollback of a single SQL statement.

During implicit rollbacks, as well as during the execution of an explicit `ROLLBACK` SQL statement, `SHOW PROCESSLIST` displays `Rolling back` in the `State` column for the relevant connection.

14.22 InnoDB Limits

This section describes limits for `InnoDB` tables, indexes, tablespaces, and other aspects of the `InnoDB` storage engine.

- A table can contain a maximum of 1017 columns (raised in MySQL 5.6.9 from the earlier limit of 1000).
- A table can contain a maximum of 64 secondary indexes.
- By default, the index key prefix length limit is 767 bytes. See Section 13.1.13, “CREATE INDEX Statement”. For example, you might hit this limit with a column prefix index of more than 255 characters on a `TEXT` or `VARCHAR` column, assuming a `utf8mb3` character set and the maximum of 3 bytes for each character. When the `innodb_large_prefix` configuration option is enabled, the index key prefix length limit is raised to 3072 bytes for `InnoDB` tables that use the `DYNAMIC` or `COMPRESSED` row format.

  Attempting to use an index key prefix length that exceeds the limit returns an error. To avoid such errors in replication configurations, avoid enabling `innodb_large_prefix` on the source if it cannot also be enabled on replicas.

  If you reduce the `InnoDB` page size to 8KB or 4KB by specifying the `innodb_page_size` option when creating the MySQL instance, the maximum length of the index key is lowered proportionally, based on the limit of 3072 bytes for a 16KB page size. That is, the maximum index key length is 1536 bytes when the page size is 8KB, and 768 bytes when the page size is 4KB.

  The limits that apply to index key prefixes also apply to full-column index keys.

- A maximum of 16 columns is permitted for multicolumn indexes. Exceeding the limit returns an error.

```
ERROR 1070 (42000): Too many key parts specified; max 16 parts allowed
```

- The maximum row size, excluding any variable-length columns that are stored off-page, is slightly less than half of a page. That is, the maximum row size is about 8000 bytes for the default page size of 16KB. If you reduce the page size by specifying the `innodb_page_size` option when creating the MySQL instance, the maximum row size is 4000 bytes for 8KB pages, or 2000 bytes for 4KB

pages. LONGBLOB and LONGTEXT columns must be less than 4GB, and the total row size, including BLOB and TEXT columns must be less than 4GB.

If a row is less than half a page long, all of it is stored locally within the page. If it exceeds half a page, variable-length columns are chosen for external off-page storage until the row fits within half a page, as described in Section 14.12.2, “File Space Management”.

- Although InnoDB supports row sizes larger than 65,535 bytes internally, MySQL itself imposes a row-size limit of 65,535 for the combined size of all columns. See Section 8.4.7, “Limits on Table Column Count and Row Size”.
- On some older operating systems, files must be less than 2GB. This is not an InnoDB limitation. If you require a large system tablespace, configure it using several smaller data files rather than one large data file, or distribute table data across file-per-table data files.
- The combined maximum size for InnoDB log files is 512GB.
- The minimum tablespace size is slightly larger than 10MB. The maximum tablespace size depends on the InnoDB page size.

  Table 14.20 InnoDB Maximum Tablespace Size

InnoDB Page Size	Maximum Tablespace Size
4KB	16TB
8KB	32TB
16KB	64TB

  The maximum tablespace size is also the maximum size for a table.
- Tablespace files cannot exceed 4GB on Windows 32-bit systems (Bug #80149).
- The path of a tablespace file, including the file name, cannot exceed the MAX_PATH limit on Windows. Prior to Windows 10, the MAX_PATH limit is 260 characters. As of Windows 10, version 1607, MAX_PATH limitations are removed from common Win32 file and directory functions, but you must enable the new behavior.
- For limits associated with concurrent read-write transactions, see Section 14.6.7, “Undo Logs”.

14.23 InnoDB Restrictions and Limitations

This section describes restrictions and limitations of the InnoDB storage engine.

- You cannot create a table with a column name that matches the name of an internal InnoDB column (including DB_ROW_ID, DB_TRX_ID, and DB_ROLL_PTR. This restriction applies to use of the names in any lettercase.

```
mysql> CREATE TABLE t1 (c1 INT, db_row_id INT) ENGINE=INNODB;
ERROR 1166 (42000): Incorrect column name 'db_row_id'
```

- SHOW TABLE STATUS does not provide accurate statistics for InnoDB tables except for the physical size reserved by the table. The row count is only a rough estimate used in SQL optimization.
- InnoDB does not keep an internal count of rows in a table because concurrent transactions might “see” different numbers of rows at the same time. Consequently, SELECT COUNT(*) statements only count rows visible to the current transaction.

  For information about how InnoDB processes SELECT COUNT(*) statements, refer to the COUNT() description in Section 12.19.1, “Aggregate Function Descriptions”.
- There is no guarantee that InnoDB functions normally with a page size greater than 16KB. Problems compiling or running InnoDB may occur. In particular, ROW_FORMAT=COMPRESSED in the Barracuda file format assumes that the page size is at most 16KB and uses 14-bit pointers.

- A MySQL instance using a particular `InnoDB` page size (`innodb_page_size`) cannot use data files or log files from an instance that uses a different page size.
- `InnoDB` does not support indexes on spatial data type columns.
- For limitations associated with importing tables using the *Transportable Tablespaces* feature, see Table Import Limitations.
- For limitations associated with online DDL, see Section 14.13.6, “Online DDL Limitations”.

Chapter 15 Alternative Storage Engines

Table of Contents

15.1 Setting the Storage Engine 2127
15.2 The MyISAM Storage Engine 2127
15.2.1 MyISAM Startup Options 2130
15.2.2 Space Needed for Keys 2131
15.2.3 MyISAM Table Storage Formats 2132
15.2.4 MyISAM Table Problems 2134
15.3 The MEMORY Storage Engine 2136
15.4 The CSV Storage Engine 2140
15.4.1 Repairing and Checking CSV Tables 2141
15.4.2 CSV Limitations 2141
15.5 The ARCHIVE Storage Engine 2141
15.6 The BLACKHOLE Storage Engine 2143
15.7 The MERGE Storage Engine 2145
15.7.1 MERGE Table Advantages and Disadvantages 2148
15.7.2 MERGE Table Problems 2148
15.8 The FEDERATED Storage Engine 2150
15.8.1 FEDERATED Storage Engine Overview 2150
15.8.2 How to Create FEDERATED Tables 2151
15.8.3 FEDERATED Storage Engine Notes and Tips 2154
15.8.4 FEDERATED Storage Engine Resources 2155
15.9 The EXAMPLE Storage Engine 2155
15.10 Other Storage Engines 2156
15.11 Overview of MySQL Storage Engine Architecture 2156
15.11.1 Pluggable Storage Engine Architecture 2156
15.11.2 The Common Database Server Layer 2157

Storage engines are MySQL components that handle the SQL operations for different table types. `InnoDB` is the default and most general-purpose storage engine, and Oracle recommends using it for tables except for specialized use cases. (The `CREATE TABLE` statement in MySQL 5.6 creates `InnoDB` tables by default.)

MySQL Server uses a pluggable storage engine architecture that enables storage engines to be loaded into and unloaded from a running MySQL server.

To determine which storage engines your server supports, use the `SHOW ENGINES` statement. The value in the `Support` column indicates whether an engine can be used. A value of `YES`, `NO`, or `DEFAULT` indicates that an engine is available, not available, or available and currently set as the default storage engine.

```
mysql> SHOW ENGINES\G
*************************** 1. row ***************************
      Engine: PERFORMANCE_SCHEMA
     Support: YES
     Comment: Performance Schema
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 2. row ***************************
      Engine: InnoDB
     Support: DEFAULT
     Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
          XA: YES
  Savepoints: YES
*************************** 3. row ***************************
      Engine: MRG_MYISAM
```

```
     Support: YES
     Comment: Collection of identical MyISAM tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 4. row ***************************
      Engine: BLACKHOLE
     Support: YES
     Comment: /dev/null storage engine (anything you write to it disappears)
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 5. row ***************************
      Engine: MyISAM
     Support: YES
     Comment: MyISAM storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
...
```

This chapter covers use cases for special-purpose MySQL storage engines. It does not cover the default `InnoDB` storage engine or the `NDB` storage engine which are covered in Chapter 14, *The InnoDB Storage Engine* and Chapter 18, *MySQL NDB Cluster 7.3 and NDB Cluster 7.4*. For advanced users, this chapter also contains a description of the pluggable storage engine architecture (see Section 15.11, “Overview of MySQL Storage Engine Architecture”).

For information about features offered in commercial MySQL Server binaries, see *MySQL Editions*, on the MySQL website. The storage engines available might depend on which edition of MySQL you are using.

For answers to commonly asked questions about MySQL storage engines, see Section A.2, “MySQL 5.6 FAQ: Storage Engines”.

MySQL 5.6 Supported Storage Engines

- `InnoDB`: The default storage engine in MySQL 5.6. `InnoDB` is a transaction-safe (ACID compliant) storage engine for MySQL that has commit, rollback, and crash-recovery capabilities to protect user data. `InnoDB` row-level locking (without escalation to coarser granularity locks) and Oracle-style consistent nonlocking reads increase multi-user concurrency and performance. `InnoDB` stores user data in clustered indexes to reduce I/O for common queries based on primary keys. To maintain data integrity, `InnoDB` also supports `FOREIGN KEY` referential-integrity constraints. For more information about `InnoDB`, see Chapter 14, *The InnoDB Storage Engine*.

- `MyISAM`: These tables have a small footprint. Table-level locking limits the performance in read/write workloads, so it is often used in read-only or read-mostly workloads in Web and data warehousing configurations.

- `Memory`: Stores all data in RAM, for fast access in environments that require quick lookups of non-critical data. This engine was formerly known as the `HEAP` engine. Its use cases are decreasing; `InnoDB` with its buffer pool memory area provides a general-purpose and durable way to keep most or all data in memory, and `NDBCLUSTER` provides fast key-value lookups for huge distributed data sets.

- `CSV`: Its tables are really text files with comma-separated values. CSV tables let you import or dump data in CSV format, to exchange data with scripts and applications that read and write that same format. Because CSV tables are not indexed, you typically keep the data in `InnoDB` tables during normal operation, and only use CSV tables during the import or export stage.

- `Archive`: These compact, unindexed tables are intended for storing and retrieving large amounts of seldom-referenced historical, archived, or security audit information.

- `Blackhole`: The Blackhole storage engine accepts but does not store data, similar to the Unix `/dev/null` device. Queries always return an empty set. These tables can be used in replication

configurations where DML statements are sent to replicas, but the source server does not keep its own copy of the data.

- `NDB` (also known as `NDBCLUSTER`)—This clustered database engine is particularly suited for applications that require the highest possible degree of uptime and availability.

Note

The `NDB` storage engine is not supported in standard MySQL 5.6 releases. Currently supported NDB Cluster releases include MySQL NDB Cluster 7.1, which is based on MySQL 5.1; MySQL NDB Cluster 7.2, which is based on MySQL 5.5; and MySQL NDB Cluster 7.3, which is based on MySQL 5.6. While based on MySQL Server, these releases also contain support for `NDB`. MySQL NDB Cluster 7.4, currently in development and also based on MySQL 5.6, is now also available in a Developer Milestone release.

- `Merge`: Enables a MySQL DBA or developer to logically group a series of identical `MyISAM` tables and reference them as one object. Good for VLDB environments such as data warehousing.
- `Federated`: Offers the ability to link separate MySQL servers to create one logical database from many physical servers. Very good for distributed or data mart environments.
- `Example`: This engine serves as an example in the MySQL source code that illustrates how to begin writing new storage engines. It is primarily of interest to developers. The storage engine is a “stub” that does nothing. You can create tables with this engine, but no data can be stored in them or retrieved from them.

You are not restricted to using the same storage engine for an entire server or schema. You can specify the storage engine for any table. For example, an application might use mostly `InnoDB` tables, with one `CSV` table for exporting data to a spreadsheet and a few `MEMORY` tables for temporary workspaces.

Choosing a Storage Engine

The various storage engines provided with MySQL are designed with different use cases in mind. The following table provides an overview of some storage engines provided with MySQL, with clarifying notes following the table.

Table 15.1 Storage Engines Feature Summary

Feature	MyISAM	Memory	InnoDB	Archive	NDB
B-tree indexes	Yes	Yes	Yes	No	No
Backup/ point-in-time recovery (note 1)	Yes	Yes	Yes	Yes	Yes
Cluster database support	No	No	No	No	Yes
Clustered indexes	No	No	Yes	No	No
Compressed data	Yes (note 2)	No	Yes	Yes	No
Data caches	No	N/A	Yes	No	Yes

Feature	MyISAM	Memory	InnoDB	Archive	NDB
Encrypted data	Yes (note 3)	Yes (note 3)	Yes (note 4)	Yes (note 3)	Yes (note 3)
Foreign key support	No	No	Yes	No	Yes (note 5)
Full-text search indexes	Yes	No	Yes (note 6)	No	No
Geospatial data type support	Yes	No	Yes	Yes	Yes
Geospatial indexing support	Yes	No	Yes (note 7)	No	No
Hash indexes	No	Yes	No (note 8)	No	Yes
Index caches	Yes	N/A	Yes	No	Yes
Locking granularity	Table	Table	Row	Row	Row
MVCC	No	No	Yes	No	No
Replication support (note 1)	Yes	Limited (note 9)	Yes	Yes	Yes
Storage limits	256TB	RAM	64TB	None	384EB
T-tree indexes	No	No	No	No	Yes
Transactions	No	No	Yes	No	Yes
Update statistics for data dictionary	Yes	Yes	Yes	Yes	Yes

Notes:

1. Implemented in the server, rather than in the storage engine.

2. Compressed MyISAM tables are supported only when using the compressed row format. Tables using the compressed row format with MyISAM are read only.

3. Implemented in the server via encryption functions.

4. Implemented in the server via encryption functions; In MySQL 5.7 and later, data-at-rest tablespace encryption is supported.

5. Support for foreign keys is available in MySQL Cluster NDB 7.3 and later.

6. InnoDB support for FULLTEXT indexes is available in MySQL 5.6 and later.

7. InnoDB support for geospatial indexing is available in MySQL 5.7 and later.

8. InnoDB utilizes hash indexes internally for its Adaptive Hash Index feature.

9. See the discussion later in this section.

15.1 Setting the Storage Engine

When you create a new table, you can specify which storage engine to use by adding an `ENGINE` table option to the `CREATE TABLE` statement:

```
-- ENGINE=INNODB not needed unless you have set a different
-- default storage engine.
CREATE TABLE t1 (i INT) ENGINE = INNODB;
-- Simple table definitions can be switched from one to another.
CREATE TABLE t2 (i INT) ENGINE = CSV;
CREATE TABLE t3 (i INT) ENGINE = MEMORY;
```

When you omit the `ENGINE` option, the default storage engine is used. The default engine is `InnoDB` in MySQL 5.6. You can specify the default engine by using the `--default-storage-engine` server startup option, or by setting the `default-storage-engine` option in the `my.cnf` configuration file.

You can set the default storage engine for the current session by setting the `default_storage_engine` variable:

```
SET default_storage_engine=NDBCLUSTER;
```

As of MySQL 5.6.3, the storage engine for `TEMPORARY` tables created with `CREATE TEMPORARY TABLE` can be set separately from the engine for permanent tables by setting the `default_tmp_storage_engine`, either at startup or at runtime. Before MySQL 5.6.3, `default_storage_engine` sets the engine for both permanent and `TEMPORARY` tables.

To convert a table from one storage engine to another, use an `ALTER TABLE` statement that indicates the new engine:

```
ALTER TABLE t ENGINE = InnoDB;
```

See Section 13.1.17, “CREATE TABLE Statement”, and Section 13.1.7, “ALTER TABLE Statement”.

If you try to use a storage engine that is not compiled in or that is compiled in but deactivated, MySQL instead creates a table using the default storage engine. For example, in a replication setup, perhaps your source server uses `InnoDB` tables for maximum safety, but the replica servers use alternative storage engines for speed at the expense of durability or concurrency.

By default, a warning is generated whenever `CREATE TABLE` or `ALTER TABLE` cannot use the default storage engine. To prevent confusing, unintended behavior if the desired engine is unavailable, enable the `NO_ENGINE_SUBSTITUTION` SQL mode. If the desired engine is unavailable, this setting produces an error instead of a warning, and the table is not created or altered. See Section 5.1.10, “Server SQL Modes”.

For new tables, MySQL always creates an `.frm` file to hold the table and column definitions. The table's index and data may be stored in one or more other files, depending on the storage engine. The server creates the `.frm` file above the storage engine level. Individual storage engines create any additional files required for the tables that they manage. If a table name contains special characters, the names for the table files contain encoded versions of those characters as described in Section 9.2.4, “Mapping of Identifiers to File Names”.

15.2 The MyISAM Storage Engine

`MyISAM` is based on the older (and no longer available) `ISAM` storage engine but has many useful extensions.

Table 15.2 MyISAM Storage Engine Features

Feature	Support
B-tree indexes	Yes

Feature	Support
Backup/point-in-time recovery (Implemented in the server, rather than in the storage engine.)	Yes
Cluster database support	No
Clustered indexes	No
Compressed data	Yes (Compressed MyISAM tables are supported only when using the compressed row format. Tables using the compressed row format with MyISAM are read only.)
Data caches	No
Encrypted data	Yes (Implemented in the server via encryption functions.)
Foreign key support	No
Full-text search indexes	Yes
Geospatial data type support	Yes
Geospatial indexing support	Yes
Hash indexes	No
Index caches	Yes
Locking granularity	Table
MVCC	No
Replication support (Implemented in the server, rather than in the storage engine.)	Yes
Storage limits	256TB
T-tree indexes	No
Transactions	No
Update statistics for data dictionary	Yes

Each `MyISAM` table is stored on disk in three files. The files have names that begin with the table name and have an extension to indicate the file type. An `.frm` file stores the table format. The data file has an `.MYD` (`MYData`) extension. The index file has an `.MYI` (`MYIndex`) extension.

To specify explicitly that you want a `MyISAM` table, indicate that with an `ENGINE` table option:

```
CREATE TABLE t (i INT) ENGINE = MYISAM;
```

In MySQL 5.6, it is normally necessary to use `ENGINE` to specify the `MyISAM` storage engine because `InnoDB` is the default engine.

You can check or repair `MyISAM` tables with the `mysqlcheck` client or `myisamchk` utility. You can also compress `MyISAM` tables with `myisampack` to take up much less space. See Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”, Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”, and Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”.

`MyISAM` tables have the following characteristics:

- All data values are stored with the low byte first. This makes the data machine and operating system independent. The only requirements for binary portability are that the machine uses two's-complement signed integers and IEEE floating-point format. These requirements are widely used among mainstream machines. Binary compatibility might not be applicable to embedded systems, which sometimes have peculiar processors.

There is no significant speed penalty for storing data low byte first; the bytes in a table row normally are unaligned and it takes little more processing to read an unaligned byte in order than in reverse order. Also, the code in the server that fetches column values is not time critical compared to other code.

- All numeric key values are stored with the high byte first to permit better index compression.
- Large files (up to 63-bit file length) are supported on file systems and operating systems that support large files.
- There is a limit of $(2^{32})^2$ (1.844E+19) rows in a `MyISAM` table.
- The maximum number of indexes per `MyISAM` table is 64.

  The maximum number of columns per index is 16.
- The maximum key length is 1000 bytes. This can also be changed by changing the source and recompiling. For the case of a key longer than 250 bytes, a larger key block size than the default of 1024 bytes is used.
- When rows are inserted in sorted order (as when you are using an `AUTO_INCREMENT` column), the index tree is split so that the high node only contains one key. This improves space utilization in the index tree.
- Internal handling of one `AUTO_INCREMENT` column per table is supported. `MyISAM` automatically updates this column for `INSERT` and `UPDATE` operations. This makes `AUTO_INCREMENT` columns faster (at least 10%). Values at the top of the sequence are not reused after being deleted. (When an `AUTO_INCREMENT` column is defined as the last column of a multiple-column index, reuse of values deleted from the top of a sequence does occur.) The `AUTO_INCREMENT` value can be reset with `ALTER TABLE` or `myisamchk`.
- Dynamic-sized rows are much less fragmented when mixing deletes with updates and inserts. This is done by automatically combining adjacent deleted blocks and by extending blocks if the next block is deleted.
- `MyISAM` supports concurrent inserts: If a table has no free blocks in the middle of the data file, you can `INSERT` new rows into it at the same time that other threads are reading from the table. A free block can occur as a result of deleting rows or an update of a dynamic length row with more data than its current contents. When all free blocks are used up (filled in), future inserts become concurrent again. See Section 8.11.3, “Concurrent Inserts”.
- You can put the data file and index file in different directories on different physical devices to get more speed with the `DATA DIRECTORY` and `INDEX DIRECTORY` table options to `CREATE TABLE`. See Section 13.1.17, “CREATE TABLE Statement”.
- `BLOB` and `TEXT` columns can be indexed.
- `NULL` values are permitted in indexed columns. This takes 0 to 1 bytes per key.
- Each character column can have a different character set. See Chapter 10, *Character Sets, Collations, Unicode*.
- There is a flag in the `MyISAM` index file that indicates whether the table was closed correctly. If `mysqld` is started with the `myisam_recover_options` system variable set, `MyISAM` tables are automatically checked when opened, and are repaired if the table wasn't closed properly.
- `myisamchk` marks tables as checked if you run it with the `--update-state` option. `myisamchk --fast` checks only those tables that don't have this mark.
- `myisamchk --analyze` stores statistics for portions of keys, as well as for entire keys.
- `myisampack` can pack `BLOB` and `VARCHAR` columns.

`MyISAM` also supports the following features:

- Support for a true `VARCHAR` type; a `VARCHAR` column starts with a length stored in one or two bytes.
- Tables with `VARCHAR` columns may have fixed or dynamic row length.
- The sum of the lengths of the `VARCHAR` and `CHAR` columns in a table may be up to 64KB.
- Arbitrary length `UNIQUE` constraints.

Additional Resources

- A forum dedicated to the `MyISAM` storage engine is available at https://forums.mysql.com/list.php?21.

15.2.1 MyISAM Startup Options

The following options to `mysqld` can be used to change the behavior of `MyISAM` tables. For additional information, see Section 5.1.6, “Server Command Options”.

Table 15.3 MyISAM Option and Variable Reference

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
bulk_insert_buffer_size	Yes	Yes	Yes		Both	Yes
concurrent_insert	Yes	Yes	Yes		Global	Yes
delay_key_write	Yes	Yes	Yes		Global	Yes
have_rtree_keys			Yes		Global	No
key_buffer_size	Yes	Yes	Yes		Global	Yes
log-isam	Yes	Yes				
myisam-block-size	Yes	Yes				
myisam_data_pointer_size	Yes	Yes	Yes		Global	Yes
myisam_max_sort_file_size	Yes	Yes	Yes		Global	Yes
myisam_mmap_size	Yes	Yes	Yes		Global	No
myisam_recover_options	Yes	Yes	Yes		Global	No
myisam_repair_threads	Yes	Yes	Yes		Both	Yes
myisam_sort_buffer_size	Yes	Yes	Yes		Both	Yes
myisam_stats_method	Yes	Yes	Yes		Both	Yes
myisam_use_mmap	Yes	Yes	Yes		Global	Yes
tmp_table_size	Yes	Yes	Yes		Both	Yes

The following system variables affect the behavior of `MyISAM` tables. For additional information, see Section 5.1.7, “Server System Variables”.

- `bulk_insert_buffer_size`

  The size of the tree cache used in bulk insert optimization.

  Note

  This is a limit *per thread*!

- `delay_key_write=ALL`

  Don't flush key buffers between writes for any `MyISAM` table.

Note

If you do this, you should not access `MyISAM` tables from another program (such as from another MySQL server or with `myisamchk`) when the tables are in use. Doing so risks index corruption. Using `--external-locking` does not eliminate this risk.

- `myisam_max_sort_file_size`

  The maximum size of the temporary file that MySQL is permitted to use while re-creating a `MyISAM` index (during `REPAIR TABLE`, `ALTER TABLE`, or `LOAD DATA`). If the file size would be larger than this value, the index is created using the key cache instead, which is slower. The value is given in bytes.

- `myisam_recover_options=`*`mode`*

  Set the mode for automatic recovery of crashed `MyISAM` tables.

- `myisam_sort_buffer_size`

  Set the size of the buffer used when recovering tables.

Automatic recovery is activated if you start `mysqld` with the `myisam_recover_options` system variable set. In this case, when the server opens a `MyISAM` table, it checks whether the table is marked as crashed or whether the open count variable for the table is not 0 and you are running the server with external locking disabled. If either of these conditions is true, the following happens:

- The server checks the table for errors.
- If the server finds an error, it tries to do a fast table repair (with sorting and without re-creating the data file).
- If the repair fails because of an error in the data file (for example, a duplicate-key error), the server tries again, this time re-creating the data file.
- If the repair still fails, the server tries once more with the old repair option method (write row by row without sorting). This method should be able to repair any type of error and has low disk space requirements.

If the recovery wouldn't be able to recover all rows from previously completed statements and you didn't specify `FORCE` in the value of the `myisam_recover_options` system variable, automatic repair aborts with an error message in the error log:

```
Error: Couldn't repair table: test.g00pages
```

If you specify `FORCE`, a warning like this is written instead:

```
Warning: Found 344 of 354 rows when repairing ./test/g00pages
```

If the automatic recovery value includes `BACKUP`, the recovery process creates files with names of the form *`tbl_name-datetime`*`.BAK`. You should have a `cron` script that automatically moves these files from the database directories to backup media.

15.2.2 Space Needed for Keys

`MyISAM` tables use B-tree indexes. You can roughly calculate the size for the index file as `(key_length+4)/0.67`, summed over all keys. This is for the worst case when all keys are inserted in sorted order and the table doesn't have any compressed keys.

String indexes are space compressed. If the first index part is a string, it is also prefix compressed. Space compression makes the index file smaller than the worst-case figure if a string column has a lot

of trailing space or is a VARCHAR column that is not always used to the full length. Prefix compression is used on keys that start with a string. Prefix compression helps if there are many strings with an identical prefix.

In MyISAM tables, you can also prefix compress numbers by specifying the PACK_KEYS=1 table option when you create the table. Numbers are stored with the high byte first, so this helps when you have many integer keys that have an identical prefix.

15.2.3 MyISAM Table Storage Formats

MyISAM supports three different storage formats. Two of them, fixed and dynamic format, are chosen automatically depending on the type of columns you are using. The third, compressed format, can be created only with the myisampack utility (see Section 4.6.5, "myisampack — Generate Compressed, Read-Only MyISAM Tables").

When you use CREATE TABLE or ALTER TABLE for a table that has no BLOB or TEXT columns, you can force the table format to FIXED or DYNAMIC with the ROW_FORMAT table option.

See Section 13.1.17, "CREATE TABLE Statement", for information about ROW_FORMAT.

You can decompress (unpack) compressed MyISAM tables using myisamchk --unpack; see Section 4.6.3, "myisamchk — MyISAM Table-Maintenance Utility", for more information.

15.2.3.1 Static (Fixed-Length) Table Characteristics

Static format is the default for MyISAM tables. It is used when the table contains no variable-length columns (VARCHAR, VARBINARY, BLOB, or TEXT). Each row is stored using a fixed number of bytes.

Of the three MyISAM storage formats, static format is the simplest and most secure (least subject to corruption). It is also the fastest of the on-disk formats due to the ease with which rows in the data file can be found on disk: To look up a row based on a row number in the index, multiply the row number by the row length to calculate the row position. Also, when scanning a table, it is very easy to read a constant number of rows with each disk read operation.

The security is evidenced if your computer crashes while the MySQL server is writing to a fixed-format MyISAM file. In this case, myisamchk can easily determine where each row starts and ends, so it can usually reclaim all rows except the partially written one. MyISAM table indexes can always be reconstructed based on the data rows.

Note

Fixed-length row format is only available for tables without BLOB or TEXT columns. Creating a table with these columns with an explicit ROW_FORMAT clause does not raise an error or warning; the format specification is ignored.

Static-format tables have these characteristics:

- CHAR and VARCHAR columns are space-padded to the specified column width, although the column type is not altered. BINARY and VARBINARY columns are padded with 0x00 bytes to the column width.
- NULL columns require additional space in the row to record whether their values are NULL. Each NULL column takes one bit extra, rounded up to the nearest byte.
- Very quick.
- Easy to cache.
- Easy to reconstruct after a crash, because rows are located in fixed positions.
- Reorganization is unnecessary unless you delete a huge number of rows and want to return free disk space to the operating system. To do this, use OPTIMIZE TABLE or myisamchk -r.

- Usually require more disk space than dynamic-format tables.
- The expected row length in bytes for static-sized rows is calculated using the following expression:

```
row length = 1
             + (sum of column lengths)
             + (number of NULL columns + delete_flag + 7)/8
             + (number of variable-length columns)
```

  delete_flag is 1 for tables with static row format. Static tables use a bit in the row record for a flag that indicates whether the row has been deleted. *delete_flag* is 0 for dynamic tables because the flag is stored in the dynamic row header.

15.2.3.2 Dynamic Table Characteristics

Dynamic storage format is used if a MyISAM table contains any variable-length columns (VARCHAR, VARBINARY, BLOB, or TEXT), or if the table was created with the ROW_FORMAT=DYNAMIC table option.

Dynamic format is a little more complex than static format because each row has a header that indicates how long it is. A row can become fragmented (stored in noncontiguous pieces) when it is made longer as a result of an update.

You can use OPTIMIZE TABLE or myisamchk -r to defragment a table. If you have fixed-length columns that you access or change frequently in a table that also contains some variable-length columns, it might be a good idea to move the variable-length columns to other tables just to avoid fragmentation.

Dynamic-format tables have these characteristics:

- All string columns are dynamic except those with a length less than four.
- Each row is preceded by a bitmap that indicates which columns contain the empty string (for string columns) or zero (for numeric columns). This does not include columns that contain NULL values. If a string column has a length of zero after trailing space removal, or a numeric column has a value of zero, it is marked in the bitmap and not saved to disk. Nonempty strings are saved as a length byte plus the string contents.
- NULL columns require additional space in the row to record whether their values are NULL. Each NULL column takes one bit extra, rounded up to the nearest byte.
- Much less disk space usually is required than for fixed-length tables.
- Each row uses only as much space as is required. However, if a row becomes larger, it is split into as many pieces as are required, resulting in row fragmentation. For example, if you update a row with information that extends the row length, the row becomes fragmented. In this case, you may have to run OPTIMIZE TABLE or myisamchk -r from time to time to improve performance. Use myisamchk -ei to obtain table statistics.
- More difficult than static-format tables to reconstruct after a crash, because rows may be fragmented into many pieces and links (fragments) may be missing.
- The expected row length for dynamic-sized rows is calculated using the following expression:

```
3
+ (number of columns + 7) / 8
+ (number of char columns)
+ (packed size of numeric columns)
+ (length of strings)
+ (number of NULL columns + 7) / 8
```

  There is a penalty of 6 bytes for each link. A dynamic row is linked whenever an update causes an enlargement of the row. Each new link is at least 20 bytes, so the next enlargement probably goes in the same link. If not, another link is created. You can find the number of links using myisamchk -ed. All links may be removed with OPTIMIZE TABLE or myisamchk -r.

15.2.3.3 Compressed Table Characteristics

Compressed storage format is a read-only format that is generated with the `myisampack` tool. Compressed tables can be uncompressed with `myisamchk`.

Compressed tables have the following characteristics:

- Compressed tables take very little disk space. This minimizes disk usage, which is helpful when using slow disks (such as CD-ROMs).
- Each row is compressed separately, so there is very little access overhead. The header for a row takes up one to three bytes depending on the biggest row in the table. Each column is compressed differently. There is usually a different Huffman tree for each column. Some of the compression types are:
  - Suffix space compression.
  - Prefix space compression.
  - Numbers with a value of zero are stored using one bit.
  - If values in an integer column have a small range, the column is stored using the smallest possible type. For example, a `BIGINT` column (eight bytes) can be stored as a `TINYINT` column (one byte) if all its values are in the range from `-128` to `127`.
  - If a column has only a small set of possible values, the data type is converted to `ENUM`.
  - A column may use any combination of the preceding compression types.
- Can be used for fixed-length or dynamic-length rows.

Note

While a compressed table is read only, and you cannot therefore update or add rows in the table, DDL (Data Definition Language) operations are still valid. For example, you may still use `DROP` to drop the table, and `TRUNCATE TABLE` to empty the table.

15.2.4 MyISAM Table Problems

The file format that MySQL uses to store data has been extensively tested, but there are always circumstances that may cause database tables to become corrupted. The following discussion describes how this can happen and how to handle it.

15.2.4.1 Corrupted MyISAM Tables

Even though the `MyISAM` table format is very reliable (all changes to a table made by an SQL statement are written before the statement returns), you can still get corrupted tables if any of the following events occur:

- The `mysqld` process is killed in the middle of a write.
- An unexpected computer shutdown occurs (for example, the computer is turned off).
- Hardware failures.
- You are using an external program (such as `myisamchk`) to modify a table that is being modified by the server at the same time.
- A software bug in the MySQL or `MyISAM` code.

Typical symptoms of a corrupt table are:

- You get the following error while selecting data from the table:

```
Incorrect key file for table: '...'. Try to repair it
```

- Queries don't find rows in the table or return incomplete results.

You can check the health of a `MyISAM` table using the `CHECK TABLE` statement, and repair a corrupted `MyISAM` table with `REPAIR TABLE`. When `mysqld` is not running, you can also check or repair a table with the `myisamchk` command. See Section 13.7.2.2, “CHECK TABLE Statement”, Section 13.7.2.5, “REPAIR TABLE Statement”, and Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”.

If your tables become corrupted frequently, you should try to determine why this is happening. The most important thing to know is whether the table became corrupted as a result of an unexpected server exit. You can verify this easily by looking for a recent `restarted mysqld` message in the error log. If there is such a message, it is likely that table corruption is a result of the server dying. Otherwise, corruption may have occurred during normal operation. This is a bug. You should try to create a reproducible test case that demonstrates the problem. See Section B.3.3.3, “What to Do If MySQL Keeps Crashing”, and Section 5.8, “Debugging MySQL”.

15.2.4.2 Problems from Tables Not Being Closed Properly

Each `MyISAM` index file (`.MYI` file) has a counter in the header that can be used to check whether a table has been closed properly. If you get the following warning from `CHECK TABLE` or `myisamchk`, it means that this counter has gone out of sync:

```
clients are using or haven't closed the table properly
```

This warning doesn't necessarily mean that the table is corrupted, but you should at least check the table.

The counter works as follows:

- The first time a table is updated in MySQL, a counter in the header of the index files is incremented.
- The counter is not changed during further updates.
- When the last instance of a table is closed (because a `FLUSH TABLES` operation was performed or because there is no room in the table cache), the counter is decremented if the table has been updated at any point.
- When you repair the table or check the table and it is found to be okay, the counter is reset to zero.
- To avoid problems with interaction with other processes that might check the table, the counter is not decremented on close if it was zero.

In other words, the counter can become incorrect only under these conditions:

- A `MyISAM` table is copied without first issuing `LOCK TABLES` and `FLUSH TABLES`.
- MySQL has crashed between an update and the final close. (The table may still be okay because MySQL always issues writes for everything between each statement.)
- A table was modified by `myisamchk --recover` or `myisamchk --update-state` at the same time that it was in use by `mysqld`.
- Multiple `mysqld` servers are using the table and one server performed a `REPAIR TABLE` or `CHECK TABLE` on the table while it was in use by another server. In this setup, it is safe to use `CHECK TABLE`, although you might get the warning from other servers. However, `REPAIR TABLE` should be avoided because when one server replaces the data file with a new one, this is not known to the other servers.

In general, it is a bad idea to share a data directory among multiple servers. See Section 5.7, “Running Multiple MySQL Instances on One Machine”, for additional discussion.

15.3 The MEMORY Storage Engine

The `MEMORY` storage engine (formerly known as `HEAP`) creates special-purpose tables with contents that are stored in memory. Because the data is vulnerable to crashes, hardware issues, or power outages, only use these tables as temporary work areas or read-only caches for data pulled from other tables.

Table 15.4 MEMORY Storage Engine Features

Feature	Support
B-tree indexes	Yes
Backup/point-in-time recovery (Implemented in the server, rather than in the storage engine.)	Yes
Cluster database support	No
Clustered indexes	No
Compressed data	No
Data caches	N/A
Encrypted data	Yes (Implemented in the server via encryption functions.)
Foreign key support	No
Full-text search indexes	No
Geospatial data type support	No
Geospatial indexing support	No
Hash indexes	Yes
Index caches	N/A
Locking granularity	Table
MVCC	No
Replication support (Implemented in the server, rather than in the storage engine.)	Limited (See the discussion later in this section.)
Storage limits	RAM
T-tree indexes	No
Transactions	No
Update statistics for data dictionary	Yes

- When to Use MEMORY or NDB Cluster
- Performance Characteristics
- Characteristics of MEMORY Tables
- DDL Operations for MEMORY Tables
- Indexes
- User-Created and Temporary Tables
- Loading Data
- MEMORY Tables and Replication

- Managing Memory Use
- Additional Resources

When to Use MEMORY or NDB Cluster

Developers looking to deploy applications that use the `MEMORY` storage engine for important, highly available, or frequently updated data should consider whether NDB Cluster is a better choice. A typical use case for the `MEMORY` engine involves these characteristics:

- Operations involving transient, non-critical data such as session management or caching. When the MySQL server halts or restarts, the data in `MEMORY` tables is lost.
- In-memory storage for fast access and low latency. Data volume can fit entirely in memory without causing the operating system to swap out virtual memory pages.
- A read-only or read-mostly data access pattern (limited updates).

NDB Cluster offers the same features as the `MEMORY` engine with higher performance levels, and provides additional features not available with `MEMORY`:

- Row-level locking and multiple-thread operation for low contention between clients.
- Scalability even with statement mixes that include writes.
- Optional disk-backed operation for data durability.
- Shared-nothing architecture and multiple-host operation with no single point of failure, enabling 99.999% availability.
- Automatic data distribution across nodes; application developers need not craft custom sharding or partitioning solutions.
- Support for variable-length data types (including `BLOB` and `TEXT`) not supported by `MEMORY`.

Performance Characteristics

`MEMORY` performance is constrained by contention resulting from single-thread execution and table lock overhead when processing updates. This limits scalability when load increases, particularly for statement mixes that include writes.

Despite the in-memory processing for `MEMORY` tables, they are not necessarily faster than `InnoDB` tables on a busy server, for general-purpose queries, or under a read/write workload. In particular, the table locking involved with performing updates can slow down concurrent usage of `MEMORY` tables from multiple sessions.

Depending on the kinds of queries performed on a `MEMORY` table, you might create indexes as either the default hash data structure (for looking up single values based on a unique key), or a general-purpose B-tree data structure (for all kinds of queries involving equality, inequality, or range operators such as less than or greater than). The following sections illustrate the syntax for creating both kinds of indexes. A common performance issue is using the default hash indexes in workloads where B-tree indexes are more efficient.

Characteristics of MEMORY Tables

The `MEMORY` storage engine associates each table with one disk file, which stores the table definition (not the data). The file name begins with the table name and has an extension of `.frm`.

`MEMORY` tables have the following characteristics:

- Space for `MEMORY` tables is allocated in small blocks. Tables use 100% dynamic hashing for inserts. No overflow area or extra key space is needed. No extra space is needed for free lists. Deleted rows

are put in a linked list and are reused when you insert new data into the table. MEMORY tables also have none of the problems commonly associated with deletes plus inserts in hashed tables.

- MEMORY tables use a fixed-length row-storage format. Variable-length types such as VARCHAR are stored using a fixed length.
- MEMORY tables cannot contain BLOB or TEXT columns.
- MEMORY includes support for AUTO_INCREMENT columns.
- Non-TEMPORARY MEMORY tables are shared among all clients, just like any other non-TEMPORARY table.

DDL Operations for MEMORY Tables

To create a MEMORY table, specify the clause ENGINE=MEMORY on the CREATE TABLE statement.

```
CREATE TABLE t (i INT) ENGINE = MEMORY;
```

As indicated by the engine name, MEMORY tables are stored in memory. They use hash indexes by default, which makes them very fast for single-value lookups, and very useful for creating temporary tables. However, when the server shuts down, all rows stored in MEMORY tables are lost. The tables themselves continue to exist because their definitions are stored in .frm files on disk, but they are empty when the server restarts.

This example shows how you might create, use, and remove a MEMORY table:

```
mysql> CREATE TABLE test ENGINE=MEMORY
           SELECT ip,SUM(downloads) AS down
           FROM log_table GROUP BY ip;
mysql> SELECT COUNT(ip),AVG(down) FROM test;
mysql> DROP TABLE test;
```

The maximum size of MEMORY tables is limited by the max_heap_table_size system variable, which has a default value of 16MB. To enforce different size limits for MEMORY tables, change the value of this variable. The value in effect for CREATE TABLE, or a subsequent ALTER TABLE or TRUNCATE TABLE, is the value used for the life of the table. A server restart also sets the maximum size of existing MEMORY tables to the global max_heap_table_size value. You can set the size for individual tables as described later in this section.

Indexes

The MEMORY storage engine supports both HASH and BTREE indexes. You can specify one or the other for a given index by adding a USING clause as shown here:

```
CREATE TABLE lookup
    (id INT, INDEX USING HASH (id))
    ENGINE = MEMORY;
CREATE TABLE lookup
    (id INT, INDEX USING BTREE (id))
    ENGINE = MEMORY;
```

For general characteristics of B-tree and hash indexes, see Section 8.3.1, “How MySQL Uses Indexes”.

MEMORY tables can have up to 64 indexes per table, 16 columns per index and a maximum key length of 3072 bytes.

If a MEMORY table hash index has a high degree of key duplication (many index entries containing the same value), updates to the table that affect key values and all deletes are significantly slower. The degree of this slowdown is proportional to the degree of duplication (or, inversely proportional to the index cardinality). You can use a BTREE index to avoid this problem.

MEMORY tables can have nonunique keys. (This is an uncommon feature for implementations of hash indexes.)

Columns that are indexed can contain NULL values.

User-Created and Temporary Tables

MEMORY table contents are stored in memory, which is a property that MEMORY tables share with internal temporary tables that the server creates on the fly while processing queries. However, the two types of tables differ in that MEMORY tables are not subject to storage conversion, whereas internal temporary tables are:

- If an internal temporary table becomes too large, the server automatically converts it to on-disk storage, as described in Section 8.4.4, “Internal Temporary Table Use in MySQL”.
- User-created MEMORY tables are never converted to disk tables.

Loading Data

To populate a MEMORY table when the MySQL server starts, you can use the init_file system variable. For example, you can put statements such as INSERT INTO ... SELECT or LOAD DATA into a file to load the table from a persistent data source, and use init_file to name the file. See Section 5.1.7, “Server System Variables”, and Section 13.2.6, “LOAD DATA Statement”.

For loading data into MEMORY tables accessed by other sessions concurrently, MEMORY supports INSERT DELAYED. See Section 13.2.5.3, “INSERT DELAYED Statement”.

MEMORY Tables and Replication

When a replication source server shuts down and restarts, its MEMORY tables become empty. To replicate this effect to replicas, the first time that the source uses a given MEMORY table after startup, it logs an event that notifies replicas that the table must be emptied by writing a DELETE statement for that table to the binary log. When a replica server shuts down and restarts, its MEMORY tables also become empty, and it writes a DELETE statement to its own binary log, which is passed on to any downstream replicas.

When you use MEMORY tables in a replication topology, in some situations, the table on the source and the table on the replica may differ. For information on handling each of these situations to prevent stale reads or errors, see Section 17.4.1.20, “Replication and MEMORY Tables”.

Managing Memory Use

The server needs sufficient memory to maintain all MEMORY tables that are in use at the same time.

Memory is not reclaimed if you delete individual rows from a MEMORY table. Memory is reclaimed only when the entire table is deleted. Memory that was previously used for deleted rows is re-used for new rows within the same table. To free all the memory used by a MEMORY table when you no longer require its contents, execute DELETE or TRUNCATE TABLE to remove all rows, or remove the table altogether using DROP TABLE. To free up the memory used by deleted rows, use ALTER TABLE ENGINE=MEMORY to force a table rebuild.

The memory needed for one row in a MEMORY table is calculated using the following expression:

```
SUM_OVER_ALL_BTREE_KEYS(max_length_of_key + sizeof(char*) * 4)
+ SUM_OVER_ALL_HASH_KEYS(sizeof(char*) * 2)
+ ALIGN(length_of_row+1, sizeof(char*))
```

ALIGN() represents a round-up factor to cause the row length to be an exact multiple of the char pointer size. sizeof(char*) is 4 on 32-bit machines and 8 on 64-bit machines.

As mentioned earlier, the `max_heap_table_size` system variable sets the limit on the maximum size of `MEMORY` tables. To control the maximum size for individual tables, set the session value of this variable before creating each table. (Do not change the global `max_heap_table_size` value unless you intend the value to be used for `MEMORY` tables created by all clients.) The following example creates two `MEMORY` tables, with a maximum size of 1MB and 2MB, respectively:

```
mysql> SET max_heap_table_size = 1024*1024;
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t1 (id INT, UNIQUE(id)) ENGINE = MEMORY;
Query OK, 0 rows affected (0.01 sec)

mysql> SET max_heap_table_size = 1024*1024*2;
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE TABLE t2 (id INT, UNIQUE(id)) ENGINE = MEMORY;
Query OK, 0 rows affected (0.00 sec)
```

Both tables revert to the server's global `max_heap_table_size` value if the server restarts.

You can also specify a `MAX_ROWS` table option in `CREATE TABLE` statements for `MEMORY` tables to provide a hint about the number of rows you plan to store in them. This does not enable the table to grow beyond the `max_heap_table_size` value, which still acts as a constraint on maximum table size. For maximum flexibility in being able to use `MAX_ROWS`, set `max_heap_table_size` at least as high as the value to which you want each `MEMORY` table to be able to grow.

Additional Resources

A forum dedicated to the `MEMORY` storage engine is available at https://forums.mysql.com/list.php?92.

15.4 The CSV Storage Engine

The `CSV` storage engine stores data in text files using comma-separated values format.

The `CSV` storage engine is always compiled into the MySQL server.

To examine the source for the `CSV` engine, look in the `storage/csv` directory of a MySQL source distribution.

When you create a `CSV` table, the server creates a table format file in the database directory. The file begins with the table name and has an `.frm` extension. The storage engine also creates plain text data file having a name that begins with the table name and has a `.CSV` extension. When you store data into the table, the storage engine saves it into the data file in comma-separated values format.

```
mysql> CREATE TABLE test (i INT NOT NULL, c CHAR(10) NOT NULL)
       ENGINE = CSV;
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO test VALUES(1,'record one'),(2,'record two');
Query OK, 2 rows affected (0.05 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM test;
+---+------------+
| i | c          |
+---+------------+
| 1 | record one |
| 2 | record two |
+---+------------+
2 rows in set (0.00 sec)
```

Creating a `CSV` table also creates a corresponding metafile that stores the state of the table and the number of rows that exist in the table. The name of this file is the same as the name of the table with the extension `CSM`.

If you examine the `test.CSV` file in the database directory created by executing the preceding statements, its contents should look like this:

```
"1","record one"
"2","record two"
```

This format can be read, and even written, by spreadsheet applications such as Microsoft Excel.

15.4.1 Repairing and Checking CSV Tables

The `CSV` storage engine supports the `CHECK TABLE` and `REPAIR TABLE` statements to verify and, if possible, repair a damaged `CSV` table.

When running the `CHECK TABLE` statement, the `CSV` file is checked for validity by looking for the correct field separators, escaped fields (matching or missing quotation marks), the correct number of fields compared to the table definition and the existence of a corresponding `CSV` metafile. The first invalid row discovered raises an error. Checking a valid table produces output like that shown below:

```
mysql> CHECK TABLE csvtest;
+--------------+-------+----------+----------+
| Table        | Op    | Msg_type | Msg_text |
+--------------+-------+----------+----------+
| test.csvtest | check | status   | OK       |
+--------------+-------+----------+----------+
```

A check on a corrupted table returns a fault such as

```
mysql> CHECK TABLE csvtest;
+--------------+-------+----------+----------+
| Table        | Op    | Msg_type | Msg_text |
+--------------+-------+----------+----------+
| test.csvtest | check | error    | Corrupt  |
+--------------+-------+----------+----------+
```

To repair a table, use `REPAIR TABLE`, which copies as many valid rows from the existing `CSV` data as possible, and then replaces the existing `CSV` file with the recovered rows. Any rows beyond the corrupted data are lost.

```
mysql> REPAIR TABLE csvtest;
+--------------+--------+----------+----------+
| Table        | Op     | Msg_type | Msg_text |
+--------------+--------+----------+----------+
| test.csvtest | repair | status   | OK       |
+--------------+--------+----------+----------+
```

Warning

During repair, only the rows from the `CSV` file up to the first damaged row are copied to the new table. All other rows from the first damaged row to the end of the table are removed, even valid rows.

15.4.2 CSV Limitations

The `CSV` storage engine does not support indexing.

Partitioning is not supported for tables using the `CSV` storage engine.

All tables that you create using the `CSV` storage engine must have the `NOT NULL` attribute on all columns.

15.5 The ARCHIVE Storage Engine

The `ARCHIVE` storage engine produces special-purpose tables that store large amounts of unindexed data in a very small footprint.

Table 15.5 ARCHIVE Storage Engine Features

Feature	Support
B-tree indexes	No
Backup/point-in-time recovery (Implemented in the server, rather than in the storage engine.)	Yes
Cluster database support	No
Clustered indexes	No
Compressed data	Yes
Data caches	No
Encrypted data	Yes (Implemented in the server via encryption functions.)
Foreign key support	No
Full-text search indexes	No
Geospatial data type support	Yes
Geospatial indexing support	No
Hash indexes	No
Index caches	No
Locking granularity	Row
MVCC	No
Replication support (Implemented in the server, rather than in the storage engine.)	Yes
Storage limits	None
T-tree indexes	No
Transactions	No
Update statistics for data dictionary	Yes

The `ARCHIVE` storage engine is included in MySQL binary distributions. To enable this storage engine if you build MySQL from source, invoke `CMake` with the `-DWITH_ARCHIVE_STORAGE_ENGINE` option.

To examine the source for the `ARCHIVE` engine, look in the `storage/archive` directory of a MySQL source distribution.

You can check whether the `ARCHIVE` storage engine is available with the `SHOW ENGINES` statement.

When you create an `ARCHIVE` table, the server creates a table format file in the database directory. The file begins with the table name and has an `.frm` extension. The storage engine creates other files, all having names beginning with the table name. The data file has an extension of `.ARZ`. An `.ARN` file may appear during optimization operations.

The `ARCHIVE` engine supports `INSERT`, `REPLACE`, and `SELECT`, but not `DELETE` or `UPDATE`. It does support `ORDER BY` operations, `BLOB` columns, and basically all data types including spatial data types (see Section 11.4.1, “Spatial Data Types”). Geographic spatial reference systems are not supported. The `ARCHIVE` engine uses row-level locking.

The `ARCHIVE` engine supports the `AUTO_INCREMENT` column attribute. The `AUTO_INCREMENT` column can have either a unique or nonunique index. Attempting to create an index on any other column results in an error. The `ARCHIVE` engine also supports the `AUTO_INCREMENT` table option in `CREATE TABLE` statements to specify the initial sequence value for a new table or reset the sequence value for an existing table, respectively.

`ARCHIVE` does not support inserting a value into an `AUTO_INCREMENT` column less than the current maximum column value. Attempts to do so result in an `ER_DUP_KEY` error.

The ARCHIVE engine ignores BLOB columns if they are not requested and scans past them while reading.

Storage: Rows are compressed as they are inserted. The ARCHIVE engine uses zlib lossless data compression (see http://www.zlib.net/). You can use OPTIMIZE TABLE to analyze the table and pack it into a smaller format (for a reason to use OPTIMIZE TABLE, see later in this section). The engine also supports CHECK TABLE. There are several types of insertions that are used:

- An INSERT statement just pushes rows into a compression buffer, and that buffer flushes as necessary. The insertion into the buffer is protected by a lock. A SELECT forces a flush to occur, unless the only insertions that have come in were INSERT DELAYED (those flush as necessary). See Section 13.2.5.3, “INSERT DELAYED Statement”.
- A bulk insert is visible only after it completes, unless other inserts occur at the same time, in which case it can be seen partially. A SELECT never causes a flush of a bulk insert unless a normal insert occurs while it is loading.

Retrieval: On retrieval, rows are uncompressed on demand; there is no row cache. A SELECT operation performs a complete table scan: When a SELECT occurs, it finds out how many rows are currently available and reads that number of rows. SELECT is performed as a consistent read. Note that lots of SELECT statements during insertion can deteriorate the compression, unless only bulk or delayed inserts are used. To achieve better compression, you can use OPTIMIZE TABLE or REPAIR TABLE. The number of rows in ARCHIVE tables reported by SHOW TABLE STATUS is always accurate. See Section 13.7.2.4, “OPTIMIZE TABLE Statement”, Section 13.7.2.5, “REPAIR TABLE Statement”, and Section 13.7.5.37, “SHOW TABLE STATUS Statement”.

Additional Resources

- A forum dedicated to the ARCHIVE storage engine is available at https://forums.mysql.com/list.php?112.

15.6 The BLACKHOLE Storage Engine

The BLACKHOLE storage engine acts as a “black hole” that accepts data but throws it away and does not store it. Retrievals always return an empty result:

```
mysql> CREATE TABLE test(i INT, c CHAR(10)) ENGINE = BLACKHOLE;
Query OK, 0 rows affected (0.03 sec)

mysql> INSERT INTO test VALUES(1,'record one'),(2,'record two');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM test;
Empty set (0.00 sec)
```

To enable the BLACKHOLE storage engine if you build MySQL from source, invoke CMake with the -DWITH_BLACKHOLE_STORAGE_ENGINE option.

To examine the source for the BLACKHOLE engine, look in the sql directory of a MySQL source distribution.

When you create a BLACKHOLE table, the server creates a table format file in the database directory. The file begins with the table name and has an .frm extension. There are no other files associated with the table.

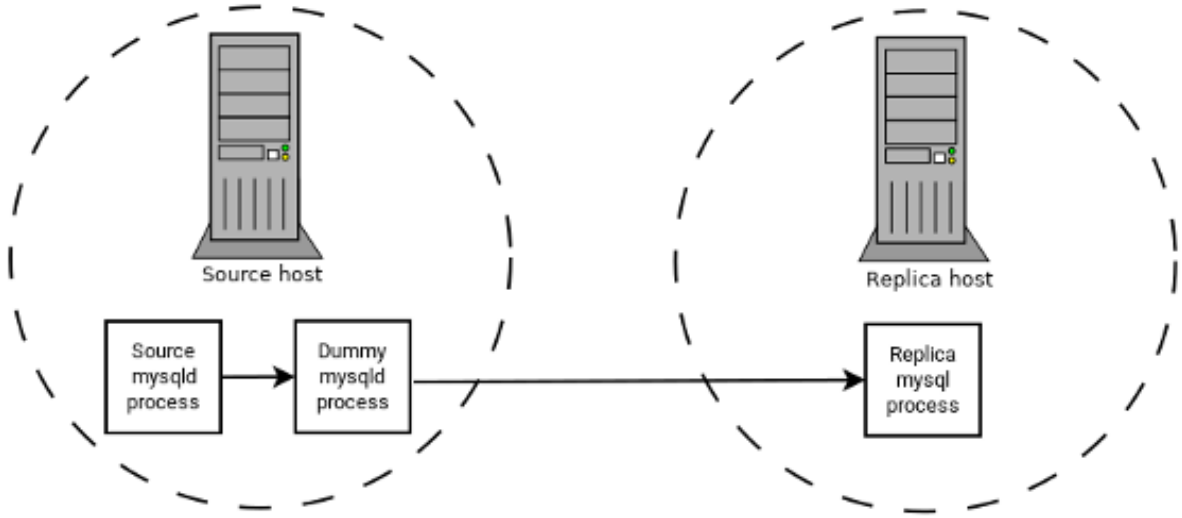
The BLACKHOLE storage engine supports all kinds of indexes. That is, you can include index declarations in the table definition.

You can check whether the BLACKHOLE storage engine is available with the SHOW ENGINES statement.

Inserts into a BLACKHOLE table do not store any data, but if statement based binary logging is enabled, the SQL statements are logged and replicated to replica servers. This can be useful as a repeater or filter mechanism.

Suppose that your application requires replica-side filtering rules, but transferring all binary log data to the replica first results in too much traffic. In such a case, it is possible to set up on the source host a "dummy" replica process whose default storage engine is BLACKHOLE, depicted as follows:

Figure 15.1 Replication using BLACKHOLE for Filtering



The source writes to its binary log. The "dummy" mysqld process acts as a replica, applying the desired combination of replicate-do-* and replicate-ignore-* rules, and writes a new, filtered binary log of its own. (See Section 17.1.4, "Replication and Binary Logging Options and Variables".) This filtered log is provided to the replica.

The dummy process does not actually store any data, so there is little processing overhead incurred by running the additional mysqld process on the replication source host. This type of setup can be repeated with additional replicas.

INSERT triggers for BLACKHOLE tables work as expected. However, because the BLACKHOLE table does not actually store any data, UPDATE and DELETE triggers are not activated: The FOR EACH ROW clause in the trigger definition does not apply because there are no rows.

Other possible uses for the BLACKHOLE storage engine include:

- Verification of dump file syntax.
- Measurement of the overhead from binary logging, by comparing performance using BLACKHOLE with and without binary logging enabled.
- BLACKHOLE is essentially a "no-op" storage engine, so it could be used for finding performance bottlenecks not related to the storage engine itself.

The BLACKHOLE engine is transaction-aware, in the sense that committed transactions are written to the binary log and rolled-back transactions are not.

Blackhole Engine and Auto Increment Columns

The BLACKHOLE engine is a no-op engine. Any operations performed on a table using BLACKHOLE have no effect. This should be born in mind when considering the behavior of primary key columns that

auto increment. The engine does not automatically increment field values, and does not retain auto increment column state. This has important implications in replication.

Consider the following replication scenario where all three of the following conditions apply:

1. On a source server there is a blackhole table with an auto increment field that is a primary key.
2. On a replica the same table exists but using the MyISAM engine.
3. Inserts are performed into the source's table without explicitly setting the auto increment value in the `INSERT` statement itself or through using a `SET INSERT_ID` statement.

In this scenario replication fails with a duplicate entry error on the primary key column.

In statement based replication, the value of `INSERT_ID` in the context event is always the same. Replication therefore fails due to trying insert a row with a duplicate value for a primary key column.

In row based replication, the value that the engine returns for the row always be the same for each insert. This results in the replica attempting to replay two insert log entries using the same value for the primary key column, and so replication fails.

Column Filtering

When using row-based replication, (`binlog_format=ROW`), a replica where the last columns are missing from a table is supported, as described in the section Section 17.4.1.9, “Replication with Differing Table Definitions on Source and Replica”.

This filtering works on the replica side, that is, the columns are copied to the replica before they are filtered out. There are at least two cases where it is not desirable to copy the columns to the replica:

1. If the data is confidential, so the replica server should not have access to it.
2. If the source has many replicas, filtering before sending to the replicas may reduce network traffic.

Source column filtering can be achieved using the `BLACKHOLE` engine. This is carried out in a way similar to how source table filtering is achieved - by using the `BLACKHOLE` engine and the `--replicate-do-table` or `--replicate-ignore-table` option.

The setup for the source is:

```
CREATE TABLE t1 (public_col_1, ..., public_col_N,
                 secret_col_1, ..., secret_col_M) ENGINE=MyISAM;
```

The setup for the trusted replica is:

```
CREATE TABLE t1 (public_col_1, ..., public_col_N) ENGINE=BLACKHOLE;
```

The setup for the untrusted replica is:

```
CREATE TABLE t1 (public_col_1, ..., public_col_N) ENGINE=MyISAM;
```

15.7 The MERGE Storage Engine

The `MERGE` storage engine, also known as the `MRG_MyISAM` engine, is a collection of identical `MyISAM` tables that can be used as one. “Identical” means that all tables have identical column data types and index information. You cannot merge `MyISAM` tables in which the columns are listed in a different order, do not have exactly the same data types in corresponding columns, or have the indexes in different order. However, any or all of the `MyISAM` tables can be compressed with `myisampack`. See Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”. Differences between tables such as these do not matter:

- Names of corresponding columns and indexes can differ.
- Comments for tables, columns, and indexes can differ.

- Table options such as AVG_ROW_LENGTH, MAX_ROWS, or PACK_KEYS can differ.

An alternative to a MERGE table is a partitioned table, which stores partitions of a single table in separate files. Partitioning enables some operations to be performed more efficiently and is not limited to the MyISAM storage engine. For more information, see Chapter 19, *Partitioning*.

When you create a MERGE table, MySQL creates two files on disk. The files have names that begin with the table name and have an extension to indicate the file type. An .frm file stores the table format, and an .MRG file contains the names of the underlying MyISAM tables that should be used as one. The tables do not have to be in the same database as the MERGE table.

You can use SELECT, DELETE, UPDATE, and INSERT on MERGE tables. You must have SELECT, DELETE, and UPDATE privileges on the MyISAM tables that you map to a MERGE table.

Note

The use of MERGE tables entails the following security issue: If a user has access to MyISAM table *t*, that user can create a MERGE table *m* that accesses *t*. However, if the user's privileges on *t* are subsequently revoked, the user can continue to access *t* by doing so through *m*.

Use of DROP TABLE with a MERGE table drops only the MERGE specification. The underlying tables are not affected.

To create a MERGE table, you must specify a UNION=(*list-of-tables*) option that indicates which MyISAM tables to use. You can optionally specify an INSERT_METHOD option to control how inserts into the MERGE table take place. Use a value of FIRST or LAST to cause inserts to be made in the first or last underlying table, respectively. If you specify no INSERT_METHOD option or if you specify it with a value of NO, inserts into the MERGE table are not permitted and attempts to do so result in an error.

The following example shows how to create a MERGE table:

```
mysql> CREATE TABLE t1 (
    ->    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->    message CHAR(20)) ENGINE=MyISAM;
mysql> CREATE TABLE t2 (
    ->    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->    message CHAR(20)) ENGINE=MyISAM;
mysql> INSERT INTO t1 (message) VALUES ('Testing'),('table'),('t1');
mysql> INSERT INTO t2 (message) VALUES ('Testing'),('table'),('t2');
mysql> CREATE TABLE total (
    ->    a INT NOT NULL AUTO_INCREMENT,
    ->    message CHAR(20), INDEX(a))
    ->    ENGINE=MERGE UNION=(t1,t2) INSERT_METHOD=LAST;
```

Column a is indexed as a PRIMARY KEY in the underlying MyISAM tables, but not in the MERGE table. There it is indexed but not as a PRIMARY KEY because a MERGE table cannot enforce uniqueness over the set of underlying tables. (Similarly, a column with a UNIQUE index in the underlying tables should be indexed in the MERGE table but not as a UNIQUE index.)

After creating the MERGE table, you can use it to issue queries that operate on the group of tables as a whole:

```
mysql> SELECT * FROM total;
+---+---------+
| a | message |
+---+---------+
| 1 | Testing |
| 2 | table   |
| 3 | t1      |
| 1 | Testing |
| 2 | table   |
| 3 | t2      |
+---+---------+
```

To remap a MERGE table to a different collection of MyISAM tables, you can use one of the following methods:

- DROP the MERGE table and re-create it.
- Use ALTER TABLE *tbl_name* UNION=(...) to change the list of underlying tables.

  It is also possible to use ALTER TABLE ... UNION=() (that is, with an empty UNION clause) to remove all of the underlying tables. However, in this case, the table is effectively empty and inserts fail because there is no underlying table to take new rows. Such a table might be useful as a template for creating new MERGE tables with CREATE TABLE ... LIKE.

The underlying table definitions and indexes must conform closely to the definition of the MERGE table. Conformance is checked when a table that is part of a MERGE table is opened, not when the MERGE table is created. If any table fails the conformance checks, the operation that triggered the opening of the table fails. This means that changes to the definitions of tables within a MERGE may cause a failure when the MERGE table is accessed. The conformance checks applied to each table are:

- The underlying table and the MERGE table must have the same number of columns.
- The column order in the underlying table and the MERGE table must match.
- Additionally, the specification for each corresponding column in the parent MERGE table and the underlying tables are compared and must satisfy these checks:
  - The column type in the underlying table and the MERGE table must be equal.
  - The column length in the underlying table and the MERGE table must be equal.
  - The column of the underlying table and the MERGE table can be NULL.
- The underlying table must have at least as many indexes as the MERGE table. The underlying table may have more indexes than the MERGE table, but cannot have fewer.

  Note

  A known issue exists where indexes on the same columns must be in identical order, in both the MERGE table and the underlying MyISAM table. See Bug #33653.

  Each index must satisfy these checks:

  - The index type of the underlying table and the MERGE table must be the same.
  - The number of index parts (that is, multiple columns within a compound index) in the index definition for the underlying table and the MERGE table must be the same.
  - For each index part:
    - Index part lengths must be equal.
    - Index part types must be equal.
    - Index part languages must be equal.
    - Check whether index parts can be NULL.

If a MERGE table cannot be opened or used because of a problem with an underlying table, CHECK TABLE displays information about which table caused the problem.

Additional Resources

- A forum dedicated to the MERGE storage engine is available at https://forums.mysql.com/list.php?93.

15.7.1 MERGE Table Advantages and Disadvantages

`MERGE` tables can help you solve the following problems:

- Easily manage a set of log tables. For example, you can put data from different months into separate tables, compress some of them with `myisampack`, and then create a `MERGE` table to use them as one.
- Obtain more speed. You can split a large read-only table based on some criteria, and then put individual tables on different disks. A `MERGE` table structured this way could be much faster than using a single large table.
- Perform more efficient searches. If you know exactly what you are looking for, you can search in just one of the underlying tables for some queries and use a `MERGE` table for others. You can even have many different `MERGE` tables that use overlapping sets of tables.
- Perform more efficient repairs. It is easier to repair individual smaller tables that are mapped to a `MERGE` table than to repair a single large table.
- Instantly map many tables as one. A `MERGE` table need not maintain an index of its own because it uses the indexes of the individual tables. As a result, `MERGE` table collections are *very* fast to create or remap. (You must still specify the index definitions when you create a `MERGE` table, even though no indexes are created.)
- If you have a set of tables from which you create a large table on demand, you can instead create a `MERGE` table from them on demand. This is much faster and saves a lot of disk space.
- Exceed the file size limit for the operating system. Each `MyISAM` table is bound by this limit, but a collection of `MyISAM` tables is not.
- You can create an alias or synonym for a `MyISAM` table by defining a `MERGE` table that maps to that single table. There should be no really notable performance impact from doing this (only a couple of indirect calls and `memcpy()` calls for each read).

The disadvantages of `MERGE` tables are:

- You can use only identical `MyISAM` tables for a `MERGE` table.
- Some `MyISAM` features are unavailable in `MERGE` tables. For example, you cannot create `FULLTEXT` indexes on `MERGE` tables. (You can create `FULLTEXT` indexes on the underlying `MyISAM` tables, but you cannot search the `MERGE` table with a full-text search.)
- If the `MERGE` table is nontemporary, all underlying `MyISAM` tables must be nontemporary. If the `MERGE` table is temporary, the `MyISAM` tables can be any mix of temporary and nontemporary.
- `MERGE` tables use more file descriptors than `MyISAM` tables. If 10 clients are using a `MERGE` table that maps to 10 tables, the server uses (10 × 10) + 10 file descriptors. (10 data file descriptors for each of the 10 clients, and 10 index file descriptors shared among the clients.)
- Index reads are slower. When you read an index, the `MERGE` storage engine needs to issue a read on all underlying tables to check which one most closely matches a given index value. To read the next index value, the `MERGE` storage engine needs to search the read buffers to find the next value. Only when one index buffer is used up does the storage engine need to read the next index block. This makes `MERGE` indexes much slower on `eq_ref` searches, but not much slower on `ref` searches. For more information about `eq_ref` and `ref`, see Section 13.8.2, “EXPLAIN Statement”.

15.7.2 MERGE Table Problems

The following are known problems with `MERGE` tables:

- In versions of MySQL Server prior to 5.1.23, it was possible to create temporary merge tables with nontemporary child MyISAM tables.

From versions 5.1.23, MERGE children were locked through the parent table. If the parent was temporary, it was not locked and so the children were not locked either. Parallel use of the MyISAM tables corrupted them.

- If you use `ALTER TABLE` to change a `MERGE` table to another storage engine, the mapping to the underlying tables is lost. Instead, the rows from the underlying `MyISAM` tables are copied into the altered table, which then uses the specified storage engine.
- The `INSERT_METHOD` table option for a `MERGE` table indicates which underlying `MyISAM` table to use for inserts into the `MERGE` table. However, use of the `AUTO_INCREMENT` table option for that `MyISAM` table has no effect for inserts into the `MERGE` table until at least one row has been inserted directly into the `MyISAM` table.
- A `MERGE` table cannot maintain uniqueness constraints over the entire table. When you perform an `INSERT`, the data goes into the first or last `MyISAM` table (as determined by the `INSERT_METHOD` option). MySQL ensures that unique key values remain unique within that `MyISAM` table, but not over all the underlying tables in the collection.
- Because the `MERGE` engine cannot enforce uniqueness over the set of underlying tables, `REPLACE` does not work as expected. The two key facts are:
  - `REPLACE` can detect unique key violations only in the underlying table to which it is going to write (which is determined by the `INSERT_METHOD` option). This differs from violations in the `MERGE` table itself.
  - If `REPLACE` detects a unique key violation, it changes only the corresponding row in the underlying table it is writing to; that is, the first or last table, as determined by the `INSERT_METHOD` option.

  Similar considerations apply for `INSERT ... ON DUPLICATE KEY UPDATE`.
- `MERGE` tables do not support partitioning. That is, you cannot partition a `MERGE` table, nor can any of a `MERGE` table's underlying `MyISAM` tables be partitioned.
- You should not use `ANALYZE TABLE`, `REPAIR TABLE`, `OPTIMIZE TABLE`, `ALTER TABLE`, `DROP TABLE`, `DELETE` without a `WHERE` clause, or `TRUNCATE TABLE` on any of the tables that are mapped into an open `MERGE` table. If you do so, the `MERGE` table may still refer to the original table and yield unexpected results. To work around this problem, ensure that no `MERGE` tables remain open by issuing a `FLUSH TABLES` statement prior to performing any of the named operations.

  The unexpected results include the possibility that the operation on the `MERGE` table may report table corruption. If this occurs after one of the named operations on the underlying `MyISAM` tables, the corruption message is spurious. To deal with this, issue a `FLUSH TABLES` statement after modifying the `MyISAM` tables.
- `DROP TABLE` on a table that is in use by a `MERGE` table does not work on Windows because the `MERGE` storage engine's table mapping is hidden from the upper layer of MySQL. Windows does not permit open files to be deleted, so you first must flush all `MERGE` tables (with `FLUSH TABLES`) or drop the `MERGE` table before dropping the table.
- The definition of the `MyISAM` tables and the `MERGE` table are checked when the tables are accessed (for example, as part of a `SELECT` or `INSERT` statement). The checks ensure that the definitions of the tables and the parent `MERGE` table definition match by comparing column order, types, sizes and associated indexes. If there is a difference between the tables, an error is returned and the statement fails. Because these checks take place when the tables are opened, any changes to the definition of a single table, including column changes, column ordering, and engine alterations, cause the statement to fail.
- The order of indexes in the `MERGE` table and its underlying tables should be the same. If you use `ALTER TABLE` to add a `UNIQUE` index to a table used in a `MERGE` table, and then use `ALTER TABLE` to add a nonunique index on the `MERGE` table, the index ordering is different for the tables if there

was already a nonunique index in the underlying table. (This happens because `ALTER TABLE` puts `UNIQUE` indexes before nonunique indexes to facilitate rapid detection of duplicate keys.) Consequently, queries on tables with such indexes may return unexpected results.

- If you encounter an error message similar to `ERROR 1017 (HY000): Can't find file: 'tbl_name.MRG' (errno: 2)`, it generally indicates that some of the underlying tables do not use the `MyISAM` storage engine. Confirm that all of these tables are `MyISAM`.
- The maximum number of rows in a `MERGE` table is 2^{64} (~1.844E+19; the same as for a `MyISAM` table). It is not possible to merge multiple `MyISAM` tables into a single `MERGE` table that would have more than this number of rows.
- The `MERGE` storage engine does not support `INSERT DELAYED` statements.
- Use of underlying `MyISAM` tables of differing row formats with a parent `MERGE` table is currently known to fail. See Bug #32364.
- You cannot change the union list of a nontemporary `MERGE` table when `LOCK TABLES` is in effect. The following does *not* work:

```
CREATE TABLE m1 ... ENGINE=MRG_MYISAM ...;
LOCK TABLES t1 WRITE, t2 WRITE, m1 WRITE;
ALTER TABLE m1 ... UNION=(t1,t2) ...;
```

  However, you can do this with a temporary `MERGE` table.

- You cannot create a `MERGE` table with `CREATE ... SELECT`, neither as a temporary `MERGE` table, nor as a nontemporary `MERGE` table. For example:

```
CREATE TABLE m1 ... ENGINE=MRG_MYISAM ... SELECT ...;
```

  Attempts to do this result in an error: *`tbl_name`* is not `BASE TABLE`.

- In some cases, differing `PACK_KEYS` table option values among the `MERGE` and underlying tables cause unexpected results if the underlying tables contain `CHAR` or `BINARY` columns. As a workaround, use `ALTER TABLE` to ensure that all involved tables have the same `PACK_KEYS` value. (Bug #50646)

15.8 The FEDERATED Storage Engine

The `FEDERATED` storage engine lets you access data from a remote MySQL database without using replication or cluster technology. Querying a local `FEDERATED` table automatically pulls the data from the remote (federated) tables. No data is stored on the local tables.

To include the `FEDERATED` storage engine if you build MySQL from source, invoke `CMake` with the `-DWITH_FEDERATED_STORAGE_ENGINE` option.

The `FEDERATED` storage engine is not enabled by default in the running server; to enable `FEDERATED`, you must start the MySQL server binary using the `--federated` option.

To examine the source for the `FEDERATED` engine, look in the `storage/federated` directory of a MySQL source distribution.

15.8.1 FEDERATED Storage Engine Overview

When you create a table using one of the standard storage engines (such as `MyISAM`, `CSV` or `InnoDB`), the table consists of the table definition and the associated data. When you create a `FEDERATED` table, the table definition is the same, but the physical storage of the data is handled on a remote server.

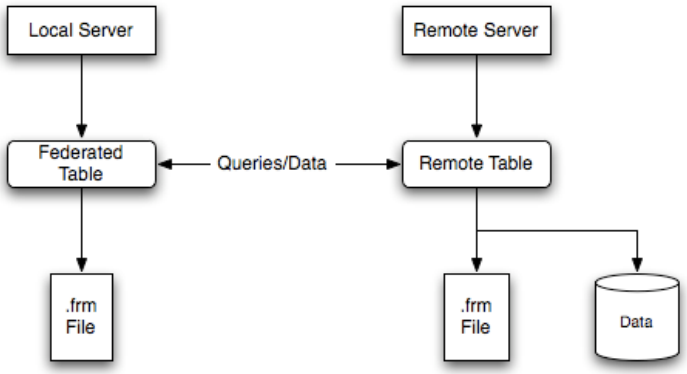
A `FEDERATED` table consists of two elements:

- A *remote server* with a database table, which in turn consists of the table definition (stored in the `.frm` file) and the associated table. The table type of the remote table may be any type supported by the remote `mysqld` server, including `MyISAM` or `InnoDB`.
- A *local server* with a database table, where the table definition matches that of the corresponding table on the remote server. The table definition is stored within the `.frm` file. However, there is no data file on the local server. Instead, the table definition includes a connection string that points to the remote table.

When executing queries and statements on a `FEDERATED` table on the local server, the operations that would normally insert, update or delete information from a local data file are instead sent to the remote server for execution, where they update the data file on the remote server or return matching rows from the remote server.

The basic structure of a `FEDERATED` table setup is shown in Figure 15.2, “FEDERATED Table Structure”.

Figure 15.2 FEDERATED Table Structure



When a client issues an SQL statement that refers to a `FEDERATED` table, the flow of information between the local server (where the SQL statement is executed) and the remote server (where the data is physically stored) is as follows:

1. The storage engine looks through each column that the `FEDERATED` table has and constructs an appropriate SQL statement that refers to the remote table.
2. The statement is sent to the remote server using the MySQL client API.
3. The remote server processes the statement and the local server retrieves any result that the statement produces (an affected-rows count or a result set).
4. If the statement produces a result set, each column is converted to internal storage engine format that the `FEDERATED` engine expects and can use to display the result to the client that issued the original statement.

The local server communicates with the remote server using MySQL client C API functions. It invokes `mysql_real_query()` to send the statement. To read a result set, it uses `mysql_store_result()` and fetches rows one at a time using `mysql_fetch_row()`.

15.8.2 How to Create FEDERATED Tables

To create a `FEDERATED` table you should follow these steps:

1. Create the table on the remote server. Alternatively, make a note of the table definition of an existing table, perhaps using the `SHOW CREATE TABLE` statement.

2. Create the table on the local server with an identical table definition, but adding the connection information that links the local table to the remote table.

For example, you could create the following table on the remote server:

```
CREATE TABLE test_table (
    id     INT(20) NOT NULL AUTO_INCREMENT,
    name   VARCHAR(32) NOT NULL DEFAULT '',
    other  INT(20) NOT NULL DEFAULT '0',
    PRIMARY KEY  (id),
    INDEX name (name),
    INDEX other_key (other)
)
ENGINE=MyISAM
DEFAULT CHARSET=latin1;
```

To create the local table that is federated to the remote table, there are two options available. You can either create the local table and specify the connection string (containing the server name, login, password) to be used to connect to the remote table using the `CONNECTION`, or you can use an existing connection that you have previously created using the `CREATE SERVER` statement.

Important

When you create the local table it *must* have an identical field definition to the remote table.

Note

You can improve the performance of a `FEDERATED` table by adding indexes to the table on the host. The optimization occurs because the query sent to the remote server includes the contents of the `WHERE` clause and is sent to the remote server and subsequently executed locally. This reduces the network traffic that would otherwise request the entire table from the server for local processing.

15.8.2.1 Creating a FEDERATED Table Using CONNECTION

To use the first method, you must specify the `CONNECTION` string after the engine type in a `CREATE TABLE` statement. For example:

```
CREATE TABLE federated_table (
    id     INT(20) NOT NULL AUTO_INCREMENT,
    name   VARCHAR(32) NOT NULL DEFAULT '',
    other  INT(20) NOT NULL DEFAULT '0',
    PRIMARY KEY  (id),
    INDEX name (name),
    INDEX other_key (other)
)
ENGINE=FEDERATED
DEFAULT CHARSET=latin1
CONNECTION='mysql://fed_user@remote_host:9306/federated/test_table';
```

Note

`CONNECTION` replaces the `COMMENT` used in some previous versions of MySQL.

The `CONNECTION` string contains the information required to connect to the remote server containing the table that is used to provide physical storage of the data. The connection string specifies the server name, login credentials, port number and database/table information. In the example, the remote table is on the server `remote_host`, using port 9306. The name and port number should match the host name (or IP address) and port number of the remote MySQL server instance you want to use as your remote table.

The format of the connection string is as follows:

```
scheme://user_name[:password]@host_name[:port_num]/db_name/tbl_name
```

Where:

- *scheme*: A recognized connection protocol. Only mysql is supported as the *scheme* value at this point.
- *user_name*: The user name for the connection. This user must have been created on the remote server, and must have suitable privileges to perform the required actions (SELECT, INSERT, UPDATE, and so forth) on the remote table.
- *password*: (Optional) The corresponding password for *user_name*.
- *host_name*: The host name or IP address of the remote server.
- *port_num*: (Optional) The port number for the remote server. The default is 3306.
- *db_name*: The name of the database holding the remote table.
- *tbl_name*: The name of the remote table. The name of the local and the remote table do not have to match.

Sample connection strings:

```
CONNECTION='mysql://username:password@hostname:port/database/tablename'
CONNECTION='mysql://username@hostname/database/tablename'
CONNECTION='mysql://username:password@hostname/database/tablename'
```

15.8.2.2 Creating a FEDERATED Table Using CREATE SERVER

If you are creating a number of FEDERATED tables on the same server, or if you want to simplify the process of creating FEDERATED tables, you can use the CREATE SERVER statement to define the server connection parameters, just as you would with the CONNECTION string.

The format of the CREATE SERVER statement is:

```
CREATE SERVER
server_name
FOREIGN DATA WRAPPER wrapper_name
OPTIONS (option [, option] ...)
```

The *server_name* is used in the connection string when creating a new FEDERATED table.

For example, to create a server connection identical to the CONNECTION string:

```
CONNECTION='mysql://fed_user@remote_host:9306/federated/test_table';
```

You would use the following statement:

```
CREATE SERVER fedlink
FOREIGN DATA WRAPPER mysql
OPTIONS (USER 'fed_user', HOST 'remote_host', PORT 9306, DATABASE 'federated');
```

To create a FEDERATED table that uses this connection, you still use the CONNECTION keyword, but specify the name you used in the CREATE SERVER statement.

```
CREATE TABLE test_table (
    id     INT(20) NOT NULL AUTO_INCREMENT,
    name   VARCHAR(32) NOT NULL DEFAULT '',
    other  INT(20) NOT NULL DEFAULT '0',
    PRIMARY KEY  (id),
    INDEX name (name),
    INDEX other_key (other)
)
ENGINE=FEDERATED
DEFAULT CHARSET=latin1
CONNECTION='fedlink/test_table';
```

The connection name in this example contains the name of the connection (`fedlink`) and the name of the table (`test_table`) to link to, separated by a slash. If you specify only the connection name without a table name, the table name of the local table is used instead.

For more information on `CREATE SERVER`, see Section 13.1.16, "CREATE SERVER Statement".

The `CREATE SERVER` statement accepts the same arguments as the `CONNECTION` string. The `CREATE SERVER` statement updates the rows in the `mysql.servers` table. See the following table for information on the correspondence between parameters in a connection string, options in the `CREATE SERVER` statement, and the columns in the `mysql.servers` table. For reference, the format of the `CONNECTION` string is as follows:

```
scheme://user_name[:password]@host_name[:port_num]/db_name/tbl_name
```

Description	`CONNECTION` string	`CREATE SERVER` option	`mysql.servers` column
Connection scheme	*`scheme`*	`wrapper_name`	`Wrapper`
Remote user	*`user_name`*	`USER`	`Username`
Remote password	*`password`*	`PASSWORD`	`Password`
Remote host	*`host_name`*	`HOST`	`Host`
Remote port	*`port_num`*	`PORT`	`Port`
Remote database	*`db_name`*	`DATABASE`	`Db`

15.8.3 FEDERATED Storage Engine Notes and Tips

You should be aware of the following points when using the `FEDERATED` storage engine:

- `FEDERATED` tables may be replicated to other replicas, but you must ensure that the replica servers are able to use the user/password combination that is defined in the `CONNECTION` string (or the row in the `mysql.servers` table) to connect to the remote server.

The following items indicate features that the `FEDERATED` storage engine does and does not support:

- The remote server must be a MySQL server.
- The remote table that a `FEDERATED` table points to *must* exist before you try to access the table through the `FEDERATED` table.
- It is possible for one `FEDERATED` table to point to another, but you must be careful not to create a loop.
- A `FEDERATED` table does not support indexes in the usual sense; because access to the table data is handled remotely, it is actually the remote table that makes use of indexes. This means that, for a query that cannot use any indexes and so requires a full table scan, the server fetches all rows from the remote table and filters them locally. This occurs regardless of any `WHERE` or `LIMIT` used with this `SELECT` statement; these clauses are applied locally to the returned rows.

  Queries that fail to use indexes can thus cause poor performance and network overload. In addition, since returned rows must be stored in memory, such a query can also lead to the local server swapping, or even hanging.
- Care should be taken when creating a `FEDERATED` table since the index definition from an equivalent `MyISAM` or other table may not be supported. For example, creating a `FEDERATED` table with an index prefix on `VARCHAR`, `TEXT` or `BLOB` columns fails. The following definition in `MyISAM` is valid:

  ```
  CREATE TABLE `T1`(`A` VARCHAR(100),UNIQUE KEY(`A`(30))) ENGINE=MYISAM;
  ```

  The key prefix in this example is incompatible with the `FEDERATED` engine, and the equivalent statement fails:

```
CREATE TABLE `T1`(`A` VARCHAR(100),UNIQUE KEY(`A`(30))) ENGINE=FEDERATED
  CONNECTION='MYSQL://127.0.0.1:3306/TEST/T1';
```

If possible, you should try to separate the column and index definition when creating tables on both the remote server and the local server to avoid these index issues.

- Internally, the implementation uses `SELECT`, `INSERT`, `UPDATE`, and `DELETE`, but not `HANDLER`.
- The `FEDERATED` storage engine supports `SELECT`, `INSERT`, `UPDATE`, `DELETE`, `TRUNCATE TABLE`, and indexes. It does not support `ALTER TABLE`, or any Data Definition Language statements that directly affect the structure of the table, other than `DROP TABLE`. The current implementation does not use prepared statements.
- `FEDERATED` accepts `INSERT ... ON DUPLICATE KEY UPDATE` statements, but if a duplicate-key violation occurs, the statement fails with an error.
- Transactions are not supported.
- `FEDERATED` performs bulk-insert handling such that multiple rows are sent to the remote table in a batch, which improves performance. Also, if the remote table is transactional, it enables the remote storage engine to perform statement rollback properly should an error occur. This capability has the following limitations:
  - The size of the insert cannot exceed the maximum packet size between servers. If the insert exceeds this size, it is broken into multiple packets and the rollback problem can occur.
  - Bulk-insert handling does not occur for `INSERT ... ON DUPLICATE KEY UPDATE`.
- There is no way for the `FEDERATED` engine to know if the remote table has changed. The reason for this is that this table must work like a data file that would never be written to by anything other than the database system. The integrity of the data in the local table could be breached if there was any change to the remote database.
- When using a `CONNECTION` string, you cannot use an '@' character in the password. You can get round this limitation by using the `CREATE SERVER` statement to create a server connection.
- The `insert_id` and `timestamp` options are not propagated to the data provider.
- Any `DROP TABLE` statement issued against a `FEDERATED` table drops only the local table, not the remote table.
- `FEDERATED` tables do not work with the query cache.
- User-defined partitioning is not supported for `FEDERATED` tables.

15.8.4 FEDERATED Storage Engine Resources

The following additional resources are available for the `FEDERATED` storage engine:

- A forum dedicated to the `FEDERATED` storage engine is available at https://forums.mysql.com/list.php?105.

15.9 The EXAMPLE Storage Engine

The `EXAMPLE` storage engine is a stub engine that does nothing. Its purpose is to serve as an example in the MySQL source code that illustrates how to begin writing new storage engines. As such, it is primarily of interest to developers.

To enable the `EXAMPLE` storage engine if you build MySQL from source, invoke `CMake` with the `-DWITH_EXAMPLE_STORAGE_ENGINE` option.

To examine the source for the EXAMPLE engine, look in the storage/example directory of a MySQL source distribution.

When you create an EXAMPLE table, the server creates a table format file in the database directory. The file begins with the table name and has an .frm extension. No other files are created. No data can be stored into the table. Retrievals return an empty result.

```
mysql> CREATE TABLE test (i INT) ENGINE = EXAMPLE;
Query OK, 0 rows affected (0.78 sec)

mysql> INSERT INTO test VALUES(1),(2),(3);
ERROR 1031 (HY000): Table storage engine for 'test' doesn't »
                    have this option

mysql> SELECT * FROM test;
Empty set (0.31 sec)
```

The EXAMPLE storage engine does not support indexing.

15.10 Other Storage Engines

Other storage engines may be available from third parties and community members that have used the Custom Storage Engine interface.

Third party engines are not supported by MySQL. For further information, documentation, installation guides, bug reporting or for any help or assistance with these engines, please contact the developer of the engine directly.

For more information on developing a customer storage engine that can be used with the Pluggable Storage Engine Architecture, see MySQL Internals: Writing a Custom Storage Engine.

15.11 Overview of MySQL Storage Engine Architecture

The MySQL pluggable storage engine architecture enables a database professional to select a specialized storage engine for a particular application need while being completely shielded from the need to manage any specific application coding requirements. The MySQL server architecture isolates the application programmer and DBA from all of the low-level implementation details at the storage level, providing a consistent and easy application model and API. Thus, although there are different capabilities across different storage engines, the application is shielded from these differences.

The pluggable storage engine architecture provides a standard set of management and support services that are common among all underlying storage engines. The storage engines themselves are the components of the database server that actually perform actions on the underlying data that is maintained at the physical server level.

This efficient and modular architecture provides huge benefits for those wishing to specifically target a particular application need—such as data warehousing, transaction processing, or high availability situations—while enjoying the advantage of utilizing a set of interfaces and services that are independent of any one storage engine.

The application programmer and DBA interact with the MySQL database through Connector APIs and service layers that are above the storage engines. If application changes bring about requirements that demand the underlying storage engine change, or that one or more storage engines be added to support new needs, no significant coding or process changes are required to make things work. The MySQL server architecture shields the application from the underlying complexity of the storage engine by presenting a consistent and easy-to-use API that applies across storage engines.

15.11.1 Pluggable Storage Engine Architecture

MySQL Server uses a pluggable storage engine architecture that enables storage engines to be loaded into and unloaded from a running MySQL server.

Plugging in a Storage Engine

Before a storage engine can be used, the storage engine plugin shared library must be loaded into MySQL using the `INSTALL PLUGIN` statement. For example, if the `EXAMPLE` engine plugin is named `example` and the shared library is named `ha_example.so`, you load it with the following statement:

```
INSTALL PLUGIN example SONAME 'ha_example.so';
```

To install a pluggable storage engine, the plugin file must be located in the MySQL plugin directory, and the user issuing the `INSTALL PLUGIN` statement must have `INSERT` privilege for the `mysql.plugin` table.

The shared library must be located in the MySQL server plugin directory, the location of which is given by the `plugin_dir` system variable.

Unplugging a Storage Engine

To unplug a storage engine, use the `UNINSTALL PLUGIN` statement:

```
UNINSTALL PLUGIN example;
```

If you unplug a storage engine that is needed by existing tables, those tables become inaccessible, but are still present on disk (where applicable). Ensure that there are no tables using a storage engine before you unplug the storage engine.

15.11.2 The Common Database Server Layer

A MySQL pluggable storage engine is the component in the MySQL database server that is responsible for performing the actual data I/O operations for a database as well as enabling and enforcing certain feature sets that target a specific application need. A major benefit of using specific storage engines is that you are only delivered the features needed for a particular application, and therefore you have less system overhead in the database, with the end result being more efficient and higher database performance. This is one of the reasons that MySQL has always been known to have such high performance, matching or beating proprietary monolithic databases in industry standard benchmarks.

From a technical perspective, what are some of the unique supporting infrastructure components that are in a storage engine? Some of the key feature differentiations include:

- *Concurrency*: Some applications have more granular lock requirements (such as row-level locks) than others. Choosing the right locking strategy can reduce overhead and therefore improve overall performance. This area also includes support for capabilities such as multi-version concurrency control or “snapshot” read.
- *Transaction Support*: Not every application needs transactions, but for those that do, there are very well defined requirements such as ACID compliance and more.
- *Referential Integrity*: The need to have the server enforce relational database referential integrity through DDL defined foreign keys.
- *Physical Storage*: This involves everything from the overall page size for tables and indexes as well as the format used for storing data to physical disk.
- *Index Support*: Different application scenarios tend to benefit from different index strategies. Each storage engine generally has its own indexing methods, although some (such as B-tree indexes) are common to nearly all engines.
- *Memory Caches*: Different applications respond better to some memory caching strategies than others, so although some memory caches are common to all storage engines (such as those used for user connections or MySQL's high-speed Query Cache), others are uniquely defined only when a particular storage engine is put in play.

- *Performance Aids*: This includes multiple I/O threads for parallel operations, thread concurrency, database checkpointing, bulk insert handling, and more.
- *Miscellaneous Target Features*: This may include support for geospatial operations, security restrictions for certain data manipulation operations, and other similar features.

Each set of the pluggable storage engine infrastructure components are designed to offer a selective set of benefits for a particular application. Conversely, avoiding a set of component features helps reduce unnecessary overhead. It stands to reason that understanding a particular application's set of requirements and selecting the proper MySQL storage engine can have a dramatic impact on overall system efficiency and performance.

Chapter 16 High Availability and Scalability

Table of Contents

16.1 Using ZFS Replication 2161
16.1.1 Using ZFS for File System Replication 2162
16.1.2 Configuring MySQL for ZFS Replication 2163
16.1.3 Handling MySQL Recovery with ZFS 2163
16.2 Using MySQL with `memcached` 2164
16.2.1 Installing `memcached` 2165
16.2.2 Using `memcached` 2166
16.2.3 Developing a `memcached` Application 2185
16.2.4 Getting `memcached` Statistics 2208
16.2.5 `memcached` FAQ 2216

Data is the currency of today's web, mobile, social, enterprise and cloud applications. Ensuring data is always available is a top priority for any organization. Minutes of downtime can result in significant loss of revenue and reputation.

There is no “one size fits all” approach to delivering High Availability (HA). Unique application attributes, business requirements, operational capabilities and legacy infrastructure can all influence HA technology selection. And technology is only one element in delivering HA: people and processes are just as critical as the technology itself.

MySQL is deployed into many applications demanding availability and scalability. **Availability** refers to the ability to cope with, and if necessary recover from, failures on the host, including failures of MySQL, the operating system, or the hardware and maintenance activity that may otherwise cause downtime. Scalability refers to the ability to spread both the database and the load of your application queries across multiple MySQL servers.

Because each application has different operational and availability requirements, MySQL offers a range of certified and supported solutions, delivering the appropriate levels of High Availability (HA) and scalability to meet service level requirements. Such solutions extend from replication, through virtualization and geographically redundant, multi-data center solutions delivering 99.999% uptime.

Selecting the right high availability solution for an application largely depends on:

- The level of availability required.
- The type of application being deployed.
- Accepted best practices within your own environment.

The primary solutions supported by MySQL include:

- MySQL Replication. Learn more: Chapter 17, *Replication*.
- MySQL Cluster. Learn more: Chapter 18, *MySQL NDB Cluster 7.3 and NDB Cluster 7.4*.
- Oracle MySQL Cloud Service. Learn more about MySQL Cloud Service.
- Oracle Clusterware Agent for MySQL. Learn more about Oracle Clusterware.
- MySQL with Solaris Cluster. Learn more about Solaris Cluster.

Further options are available using third-party solutions.

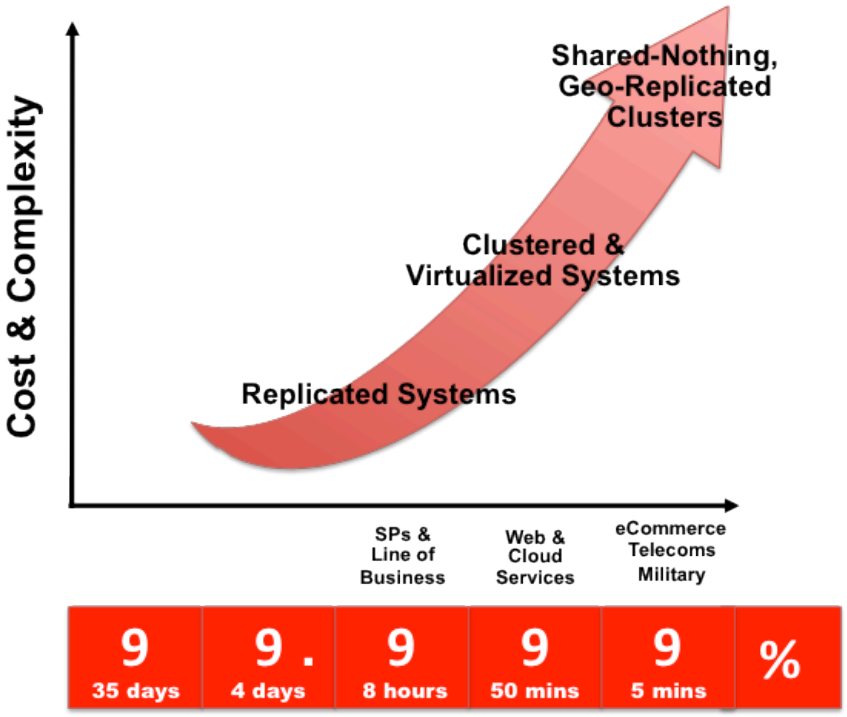
Each architecture used to achieve highly available database services is differentiated by the levels of uptime it offers. These architectures can be grouped into three main categories:

- Data Replication.

- Clustered & Virtualized Systems.
- Shared-Nothing, Geographically-Replicated Clusters.

As illustrated in the following figure, each of these architectures offers progressively higher levels of uptime, which must be balanced against potentially greater levels of cost and complexity that each can incur. Simply deploying a high availability architecture is not a guarantee of actually delivering HA. In fact, a poorly implemented and maintained shared-nothing cluster could easily deliver lower levels of availability than a simple data replication solution.

Figure 16.1 Tradeoffs: Cost and Complexity versus Availability



The following table compares the HA and Scalability capabilities of the various MySQL solutions:

Table 16.1 Feature Comparison of MySQL HA Solutions

Requirement	MySQL Replication	MySQL Cluster
Availability		
Platform Support	All Supported by MySQL Server (https://www.mysql.com/support/supportedplatforms/database.html)	All Supported by MySQL Cluster (https://www.mysql.com/support/supportedplatforms/cluster.html)
Automated IP Failover	No	Depends on Connector and Configuration
Automated Database Failover	No	Yes
Automatic Data Resynchronization	No	Yes
Typical Failover Time	User / Script Dependent	1 Second and Less
Synchronous Replication	No, Asynchronous and Semisynchronous	Yes
Shared Storage	No, Distributed	No, Distributed
Geographic redundancy support	Yes	Yes, via MySQL Replication
Update Schema On-Line	No	Yes

Requirement	MySQL Replication	MySQL Cluster
Scalability		
Number of Nodes	One Master, Multiple Slaves	255
Built-in Load Balancing	Reads, via MySQL Replication	Yes, Reads and Writes
Supports Read-Intensive Workloads	Yes	Yes
Supports Write-Intensive Workloads	Yes, via Application-Level Sharding	Yes, via Auto-Sharding
Scale On-Line (add nodes, repartition, etc.)	No	Yes

16.1 Using ZFS Replication

To support high availability environments, providing an instant copy of the information on both the currently active machine and the hot backup is a critical part of the HA solution. There are many solutions to this problem, such as Chapter 17, *Replication*.

The ZFS file system provides functionality to create a snapshot of the file system contents, transfer the snapshot to another machine, and extract the snapshot to recreate the file system. You can create a snapshot at any time, and you can create as many snapshots as you like. By continually creating, transferring, and restoring snapshots, you can provide synchronization between one or more machines in a fashion similar to DRBD.

The following example shows a simple Solaris system running with a single ZFS pool, mounted at `/scratchpool`:

```
Filesystem             size   used  avail capacity  Mounted on
/dev/dsk/c0d0s0        4.6G   3.7G   886M    82%    /
/devices                 0K     0K     0K     0%    /devices
ctfs                     0K     0K     0K     0%    /system/contract
proc                     0K     0K     0K     0%    /proc
mnttab                   0K     0K     0K     0%    /etc/mnttab
swap                   1.4G   892K   1.4G     1%    /etc/svc/volatile
objfs                    0K     0K     0K     0%    /system/object
/usr/lib/libc/libc_hwcap1.so.1
                       4.6G   3.7G   886M    82%    /lib/libc.so.1
fd                       0K     0K     0K     0%    /dev/fd
swap                   1.4G    40K   1.4G     1%    /tmp
swap                   1.4G    28K   1.4G     1%    /var/run
/dev/dsk/c0d0s7         26G   913M    25G     4%    /export/home
scratchpool             16G    24K    16G     1%    /scratchpool
```

The MySQL data is stored in a directory on `/scratchpool`. To help demonstrate some of the basic replication functionality, there are also other items stored in `/scratchpool` as well:

```
total 17
drwxr-xr-x  31 root     bin           50 Jul 21 07:32 DTT/
drwxr-xr-x   4 root     bin            5 Jul 21 07:32 SUNWmlib/
drwxr-xr-x  14 root     sys           16 Nov  5 09:56 SUNWspro/
drwxrwxrwx  19 1000     1000          40 Nov  6 19:16 emacs-22.1/
```

To create a snapshot of the file system, you use `zfs snapshot`, specifying the pool and the snapshot name:

```
root-shell> zfs snapshot scratchpool@snap1
```

To list the snapshots already taken:

```
root-shell> zfs list -t snapshot
NAME                USED  AVAIL  REFER  MOUNTPOINT
scratchpool@snap1      0      -  24.5K  -
scratchpool@snap2      0      -  24.5K  -
```

The snapshots themselves are stored within the file system metadata, and the space required to keep them varies as time goes on because of the way the snapshots are created. The initial creation of a snapshot is very quick, because instead of taking an entire copy of the data and metadata required to hold the entire snapshot, ZFS records only the point in time and metadata of when the snapshot was created.

As more changes to the original file system are made, the size of the snapshot increases because more space is required to keep the record of the old blocks. If you create lots of snapshots, say one per day, and then delete the snapshots from earlier in the week, the size of the newer snapshots might also increase, as the changes that make up the newer state have to be included in the more recent snapshots, rather than being spread over the seven snapshots that make up the week.

You cannot directly back up the snapshots because they exist within the file system metadata rather than as regular files. To get the snapshot into a format that you can copy to another file system, tape, and so on, you use the `zfs send` command to create a stream version of the snapshot.

For example, to write the snapshot out to a file:

```
root-shell> zfs send scratchpool@snap1 >/backup/scratchpool-snap1
```

Or tape:

```
root-shell> zfs send scratchpool@snap1 >/dev/rmt/0
```

You can also write out the incremental changes between two snapshots using `zfs send`:

```
root-shell> zfs send scratchpool@snap1 scratchpool@snap2 >/backup/scratchpool-changes
```

To recover a snapshot, you use `zfs recv`, which applies the snapshot information either to a new file system, or to an existing one.

16.1.1 Using ZFS for File System Replication

Because `zfs send` and `zfs recv` use streams to exchange data, you can use them to replicate information from one system to another by combining `zfs send`, `ssh`, and `zfs recv`.

For example, to copy a snapshot of the `scratchpool` file system to a new file system called `slavepool` on a new server, you would use the following command. This sequence combines the snapshot of `scratchpool`, the transmission to the replica machine (using `ssh` with login credentials), and the recovery of the snapshot on the replica using `zfs recv`:

```
root-shell> zfs send scratchpool@snap1 |ssh id@host pfexec zfs recv -F slavepool
```

The first part of the pipeline, `zfs send scratchpool@snap1`, streams the snapshot. The `ssh` command, and the command that it executes on the other server, `pfexec zfs recv -F slavepool`, receives the streamed snapshot data and writes it to slavepool. In this instance, I've specified the `-F` option which forces the snapshot data to be applied, and is therefore destructive. This is fine, as I'm creating the first version of my replicated file system.

On the replica machine, the replicated file system contains the exact same content:

```
root-shell> ls -al /slavepool/
total 23
drwxr-xr-x   6 root     root           7 Nov  8 09:13 ./
drwxr-xr-x  29 root     root          34 Nov  9 07:06 ../
drwxr-xr-x  31 root     bin           50 Jul 21 07:32 DTT/
drwxr-xr-x   4 root     bin            5 Jul 21 07:32 SUNWmlib/
drwxr-xr-x  14 root     sys           16 Nov  5 09:56 SUNWspro/
drwxrwxrwx  19 1000     1000          40 Nov  6 19:16 emacs-22.1/
```

Once a snapshot has been created, to synchronize the file system again, you create a new snapshot and then use the incremental snapshot feature of `zfs send` to send the changes between the two snapshots to the replica machine again:

```
root-shell> zfs send -i scratchpool@snapshot1 scratchpool@snapshot2 |ssh id@host pfexec zfs recv slavep
```

This operation only succeeds if the file system on the replica machine has not been modified at all. You cannot apply the incremental changes to a destination file system that has changed. In the example above, the `ls` command would cause problems by changing the metadata, such as the last access time for files or directories.

To prevent changes on the replica file system, set the file system on the replica to be read-only:

```
root-shell> zfs set readonly=on slavepool
```

Setting `readonly` means that you cannot change the file system on the replica by normal means, including the file system metadata. Operations that would normally update metadata (like our `ls`) silently perform their function without attempting to update the file system state.

In essence, the replica file system is nothing but a static copy of the original file system. However, even when configured to be read-only, a file system can have snapshots applied to it. With the file system set to read only, re-run the initial copy:

```
root-shell> zfs send scratchpool@snap1 |ssh id@host pfexec zfs recv -F slavepool
```

Now you can make changes to the original file system and replicate them to the replica.

16.1.2 Configuring MySQL for ZFS Replication

Configuring MySQL on the source file system is a case of creating the data on the file system that you intend to replicate. The configuration file in the example below has been updated to use `/scratchpool/mysql-data` as the data directory, and now you can initialize the tables:

```
root-shell> mysql_install_db --defaults-file=/etc/mysql/5.5/my.cnf --user=mysql
```

To synchronize the initial information, perform a new snapshot and then send an incremental snapshot to the replica using `zfs send`:

```
root-shell> zfs snapshot scratchpool@snap2
root-shell> zfs send -i scratchpool@snap1 scratchpool@snap2|ssh id@host pfexec zfs recv slavepool
```

Doublecheck that the replica has the data by looking at the MySQL data directory on the `slavepool`:

```
root-shell> ls -al /slavepool/mysql-data/
```

Now you can start up MySQL, create some data, and then replicate the changes using `zfs send`/`zfs recv` to the replica to synchronize the changes.

The rate at which you perform the synchronization depends on your application and environment. The limitation is the speed required to perform the snapshot and then to send the changes over the network.

To automate the process, create a script that performs the snapshot, send, and receive operation, and use `cron` to synchronize the changes at set times or intervals.

16.1.3 Handling MySQL Recovery with ZFS

When using ZFS replication to provide a constant copy of your data, ensure that you can recover your tables, either manually or automatically, in the event of a failure of the original system.

In the event of a failure, follow this sequence:

1. Stop the script on the source, if it is still up and running.
2. Set the replica file system to be read/write:

   ```
   root-shell> zfs set readonly=off slavepool
   ```

3. Start up mysqld on the replica. If you are using InnoDB, you get auto-recovery, if it is needed, to make sure the table data is correct, as shown here when I started up from our mid-INSERT snapshot:

```
InnoDB: The log sequence number in ibdata files does not match
InnoDB: the log sequence number in the ib_logfiles!
081109 15:59:59  InnoDB: Database was not shut down normally!
InnoDB: Starting crash recovery.
InnoDB: Reading tablespace information from the .ibd files...
InnoDB: Restoring possible half-written data pages from the doublewrite
InnoDB: buffer...
081109 16:00:03  InnoDB: Started; log sequence number 0 1142807951
081109 16:00:03 [Note] /slavepool/mysql-5.0.67-solaris10-i386/bin/mysqld: ready for connections.
Version: '5.0.67'  socket: '/tmp/mysql.sock'  port: 3306  MySQL Community Server (GPL)
```

Use InnoDB tables and a regular synchronization schedule to reduce the risk for significant data loss. On MyISAM tables, you might need to run REPAIR TABLE, and you might even have lost some information.

16.2 Using MySQL with memcached

memcached is a simple, highly scalable key-based cache that stores data and objects wherever dedicated or spare RAM is available for quick access by applications, without going through layers of parsing or disk I/O. To use, you run the memcached command on one or more hosts and then use the shared cache to store objects. For more usage instructions, see Section 16.2.2, “Using memcached”

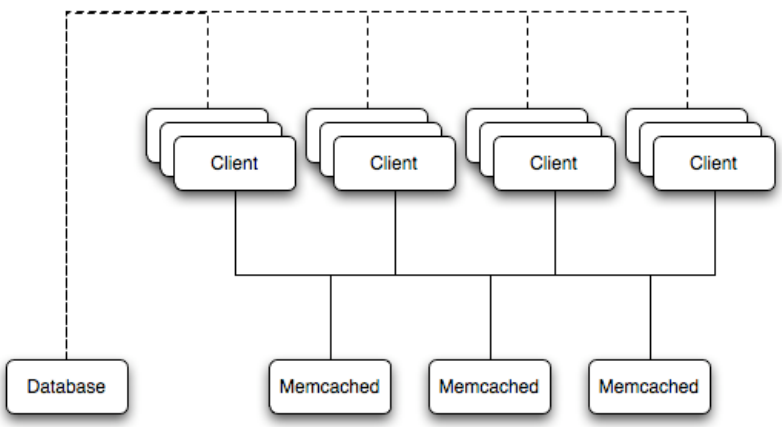
Benefits of using memcached include:

- Because all information is stored in RAM, the access speed is faster than loading the information each time from disk.
- Because the “value” portion of the key-value pair does not have any data type restrictions, you can cache data such as complex structures, documents, images, or a mixture of such things.
- If you use the in-memory cache to hold transient information, or as a read-only cache for information also stored in a database, the failure of any memcached server is not critical. For persistent data, you can fall back to an alternative lookup method using database queries, and reload the data into RAM on a different server.

The typical usage environment is to modify your application so that information is read from the cache provided by memcached. If the information is not in memcached, then the data is loaded from the MySQL database and written into the cache so that future requests for the same object benefit from the cached data.

For a typical deployment layout, see Figure 16.2, “memcached Architecture Overview”.

Figure 16.2 memcached Architecture Overview



In the example structure, any of the clients can contact one of the memcached servers to request a given key. Each client is configured to talk to all of the servers shown in the illustration. Within the client, when the request is made to store the information, the key used to reference the data is hashed and this hash is then used to select one of the memcached servers. The selection of the memcached server takes place on the client before the server is contacted, keeping the process lightweight.

The same algorithm is used again when a client requests the same key. The same key generates the same hash, and the same memcached server is selected as the source for the data. Using this method, the cached data is spread among all of the memcached servers, and the cached information is accessible from any client. The result is a distributed, memory-based, cache that can return information, particularly complex data and structures, much faster than natively reading the information from the database.

The data held within a traditional memcached server is never stored on disk (only in RAM, which means there is no persistence of data), and the RAM cache is always populated from the backing store (a MySQL database). If a memcached server fails, the data can always be recovered from the MySQL database.

16.2.1 Installing memcached

You can build and install memcached from the source code directly, or you can use an existing operating system package or installation.

Installing memcached from a Binary Distribution

To install memcached on a Red Hat, or Fedora host, use yum:

```
root-shell> yum install memcached
```

Note

On CentOS, you may be able to obtain a suitable RPM from another source, or use the source tarball.

To install memcached on a Debian or Ubuntu host, use apt-get:

```
root-shell> apt-get install memcached
```

To install memcached on a Gentoo host, use emerge:

```
root-shell> emerge install memcached
```

Building memcached from Source

On other Unix-based platforms, including Solaris, AIX, HP-UX and macOS, and Linux distributions not mentioned already, you must install from source. For Linux, make sure you have a 2.6-based kernel, which includes the improved epoll interface. For all platforms, ensure that you have libevent 1.1 or higher installed. You can obtain libevent from libevent web page.

You can obtain the source for memcached from memcached website.

To build memcached, follow these steps:

1. Extract the memcached source package:

   ```
   shell> gunzip -c memcached-1.2.5.tar.gz | tar xf -
   ```

2. Change to the memcached-1.2.5 directory:

   ```
   shell> cd memcached-1.2.5
   ```

3. Run configure

```
shell> ./configure
```

Some additional options you might specify to the configure:

- --prefix

  To specify a different installation directory, use the --prefix option:

  ```
  shell> ./configure --prefix=/opt
  ```

  The default is to use the /usr/local directory.

- --with-libevent

  If you have installed libevent and configure cannot find the library, use the --with-libevent option to specify the location of the installed library.

- --enable-64bit

  To build a 64-bit version of memcached (which enables you to use a single instance with a large RAM allocation), use --enable-64bit.

- --enable-threads

  To enable multithreading support in memcached, which improves the response times on servers with a heavy load, use --enable-threads. You must have support for the POSIX threads within your operating system to enable thread support. For more information on the threading support, see Section 16.2.2.8, "memcached Thread Support".

- --enable-dtrace

  memcached includes a range of DTrace threads that can be used to monitor and benchmark a memcached instance. For more information, see Section 16.2.2.6, "Using memcached and DTrace".

4. Run make to build memcached:

   ```
   shell> make
   ```

5. Run make install to install memcached:

   ```
   shell> make install
   ```

16.2.2 Using memcached

To start using memcached, start the memcached service on one or more servers. Running memcached sets up the server, allocates the memory and starts listening for connections from clients.

Note

You do not need to be a privileged user (root) to run memcached except to listen on one of the privileged TCP/IP ports (below 1024). You must, however, use a user that has not had their memory limits restricted using setrlimit or similar.

To start the server, run memcached as a nonprivileged (that is, non-root) user:

```
shell> memcached
```

By default, memcached uses the following settings:

- Memory allocation of 64MB

- Listens for connections on all network interfaces, using port 11211
- Supports a maximum of 1024 simultaneous connections

Typically, you would specify the full combination of options that you want when starting memcached, and normally provide a startup script to handle the initialization of memcached. For example, the following line starts memcached with a maximum of 1024MB RAM for the cache, listening on port 11211 on the IP address 198.51.100.110, running as a background daemon:

```
shell> memcached -d -m 1024 -p 11211 -l 198.51.100.110
```

To ensure that memcached is started up on boot, check the init script and configuration parameters.

16.2.2.1 memcached Command-Line Options

memcached supports the following options:

- `-u user`

  If you start memcached as root, use the -u option to specify the user for executing memcached:

  ```
  shell> memcached -u memcache
  ```

- `-m memory`

  Set the amount of memory allocated to memcached for object storage. Default is 64MB.

  To increase the amount of memory allocated for the cache, use the -m option to specify the amount of RAM to be allocated (in megabytes). The more RAM you allocate, the more data you can store and therefore the more effective your cache is.

Warning

Do not specify a memory allocation larger than your available RAM. If you specify too large a value, then some RAM allocated for memcached uses swap space, and not physical RAM. This may lead to delays when storing and retrieving values, because data is swapped to disk, instead of storing the data directly in RAM.

You can use the output of the vmstat command to get the free memory, as shown in free column:

```
shell> vmstat
kthr      memory            page            disk          faults      cpu
r b w   swap  free  re  mf pi po fr de sr s1 s2 -- --   in   sy   cs us sy id
0 0 0 5170504 3450392 2  7  2  0  0  0  4  0  0  0  0  296   54  199  0  0 100
```

For example, to allocate 3GB of RAM:

```
shell> memcached -m 3072
```

On 32-bit x86 systems where you are using PAE to access memory above the 4GB limit, you cannot allocate RAM beyond the maximum process size. You can get around this by running multiple instances of memcached, each listening on a different port:

```
shell> memcached -m 1024 -p11211
shell> memcached -m 1024 -p11212
shell> memcached -m 1024 -p11213
```

Note

On all systems, particularly 32-bit, ensure that you leave enough room for both memcached application in addition to the memory setting. For example, if you have a dedicated memcached host with 4GB of RAM, do not set

> the memory size above 3500MB. Failure to do this may cause either an unexpected exit or severe performance issues.

- `-l interface`

  Specify a network interface/address to listen for connections. The default is to listen on all available address (`INADDR_ANY`).

  ```
  shell> memcached -l 198.51.100.110
  ```

  Support for IPv6 address support was added in `memcached` 1.2.5.

- `-p port`

  Specify the TCP port to use for connections. Default is 18080.

  ```
  shell> memcached -p 18080
  ```

- `-U port`

  Specify the UDP port to use for connections. Default is 11211, 0 switches UDP off.

  ```
  shell> memcached -U 18080
  ```

- `-s socket`

  Specify a Unix socket to listen on.

  If you are running `memcached` on the same server as the clients, you can disable the network interface and use a local Unix socket using the `-s` option:

  ```
  shell> memcached -s /tmp/memcached
  ```

  Using a Unix socket automatically disables network support, and saves network ports (allowing more ports to be used by your web server or other process).

- `-a mask`

  Specify the access mask to be used for the Unix socket, in octal. Default is 0700.

- `-c connections`

  Specify the maximum number of simultaneous connections to the `memcached` service. The default is 1024.

  ```
  shell> memcached -c 2048
  ```

  Use this option, either to reduce the number of connections (to prevent overloading `memcached` service) or to increase the number to make more effective use of the server running `memcached` server.

- `-t threads`

  Specify the number of threads to use when processing incoming requests.

  By default, `memcached` is configured to use 4 concurrent threads. The threading improves the performance of storing and retrieving data in the cache, using a locking system to prevent different threads overwriting or updating the same values. To increase or decrease the number of threads, use the `-t` option:

  ```
  shell> memcached -t 8
  ```

- `-d`

  Run `memcached` as a daemon (background) process:

```
shell> memcached -d
```

- -r

  Maximize the size of the core file limit. In the event of a failure, this attempts to dump the entire memory space to disk as a core file, up to any limits imposed by setrlimit.

- -M

  Return an error to the client when the memory has been exhausted. This replaces the normal behavior of removing older items from the cache to make way for new items.

- -k

  Lock down all paged memory. This reserves the memory before use, instead of allocating new slabs of memory as new items are stored in the cache.

  Note

  There is a user-level limit on how much memory you can lock. Trying to allocate more than the available memory fails. You can set the limit for the user you started the daemon with (not for the -u user user) within the shell by using ulimit -S -l NUM_KB

- -v

  Verbose mode. Prints errors and warnings while executing the main event loop.

- -vv

  Very verbose mode. In addition to information printed by -v, also prints each client command and the response.

- -vvv

  Extremely verbose mode. In addition to information printed by -vv, also show the internal state transitions.

- -h

  Print the help message and exit.

- -i

  Print the memcached and libevent license.

- -I mem

  Specify the maximum size permitted for storing an object within the memcached instance. The size supports a unit postfix (k for kilobytes, m for megabytes). For example, to increase the maximum supported object size to 32MB:

  ```
  shell> memcached -I 32m
  ```

  The maximum object size you can specify is 128MB, the default remains at 1MB.

  This option was added in 1.4.2.

- -b

  Set the backlog queue limit. The backlog queue configures how many network connections can be waiting to be processed by memcached. Increasing this limit may reduce errors received by the client

that it is not able to connect to the memcached instance, but does not improve the performance of the server. The default is 1024.

- -P pidfile

  Save the process ID of the memcached instance into file.

- -f

  Set the chunk size growth factor. When allocating new memory chunks, the allocated size of new chunks is determined by multiplying the default slab size by this factor.

  To see the effects of this option without extensive testing, use the -vv command-line option to show the calculated slab sizes. For more information, see Section 16.2.2.9, "memcached Logs".

- -n bytes

  The minimum space allocated for the key+value+flags information. The default is 48 bytes.

- -L

  On systems that support large memory pages, enables large memory page use. Using large memory pages enables memcached to allocate the item cache in one large chunk, which can improve the performance by reducing the number misses when accessing memory.

- -C

  Disable the use of compare and swap (CAS) operations.

  This option was added in memcached 1.3.x.

- -D char

  Set the default character to be used as a delimiter between the key prefixes and IDs. This is used for the per-prefix statistics reporting (see Section 16.2.4, "Getting memcached Statistics"). The default is the colon (:). If this option is used, statistics collection is turned on automatically. If not used, you can enable stats collection by sending the stats detail on command to the server.

  This option was added in memcached 1.3.x.

- -R num

  Sets the maximum number of requests per event process. The default is 20.

- -B protocol

  Set the binding protocol, that is, the default memcached protocol support for client connections. Options are ascii, binary or auto. Automatic (auto) is the default.

  This option was added in memcached 1.4.0.

16.2.2.2 memcached Deployment

When using memcached you can use a number of different potential deployment strategies and topologies. The exact strategy to use depends on your application and environment. When developing a system for deploying memcached within your system, keep in mind the following points:

- memcached is only a caching mechanism. It shouldn't be used to store information that you cannot otherwise afford to lose and then load from a different location.

- There is no security built into the memcached protocol. At a minimum, make sure that the servers running memcached are only accessible from inside your network, and that the network ports

being used are blocked (using a firewall or similar). If the information that is being stored on the memcached servers is sensitive, encrypt it before storing it in memcached.

- memcached does not provide any sort of failover. Because there is no communication between different memcached instances. If an instance fails, your application must capable of removing it from the list, reloading the data and then writing data to another memcached instance.
- Latency between the clients and the memcached can be a problem if you are using different physical machines for these tasks. If you find that the latency is a problem, move the memcached instances to be on the clients.
- Key length is determined by the memcached server. The default maximum key size is 250 bytes.
- Try to use at least two memcached instances, especially for multiple clients, to avoid having a single point of failure. Ideally, create as many memcached nodes as possible. When adding and removing memcached instances from a pool, the hashing and distribution of key-value pairs may be affected. For information on how to avoid problems, see Section 16.2.2.5, "memcached Hashing/Distribution Types".

16.2.2.3 Using Namespaces

The memcached cache is a very simple massive key-value storage system, and as such there is no way of compartmentalizing data automatically into different sections. For example, if you are storing information by the unique ID returned from a MySQL database, then storing the data from two different tables could run into issues because the same ID might be valid in both tables.

Some interfaces provide an automated mechanism for creating *namespaces* when storing information into the cache. In practice, these namespaces are merely a prefix before a given ID that is applied every time a value is stored or retrieve from the cache.

You can implement the same basic principle by using keys that describe the object and the unique identifier within the key that you supply when the object is stored. For example, when storing user data, prefix the ID of the user with user: or user-.

Note

Using namespaces or prefixes only controls the keys stored/retrieved. There is no security within memcached, and therefore no way to enforce that a particular client only accesses keys with a particular namespace. Namespaces are only useful as a method of identifying data and preventing corruption of key-value pairs.

16.2.2.4 Data Expiry

There are two types of data expiry within a memcached instance. The first type is applied at the point when you store a new key-value pair into the memcached instance. If there is not enough space within a suitable slab to store the value, then an existing least recently used (LRU) object is removed (evicted) from the cache to make room for the new item.

The LRU algorithm ensures that the object that is removed is one that is either no longer in active use or that was used so long ago that its data is potentially out of date or of little value. However, in a system where the memory allocated to memcached is smaller than the number of regularly used objects required in the cache, a lot of expired items could be removed from the cache even though they are in active use. You use the statistics mechanism to get a better idea of the level of evictions (expired objects). For more information, see Section 16.2.4, "Getting memcached Statistics".

You can change this eviction behavior by setting the -M command-line option when starting memcached. This option forces an error to be returned when the memory has been exhausted, instead of automatically evicting older data.

The second type of expiry system is an explicit mechanism that you can set when a key-value pair is inserted into the cache, or when deleting an item from the cache. Using an expiration time can be a

useful way of ensuring that the data in the cache is up to date and in line with your application needs and requirements.

A typical scenario for explicitly setting the expiry time might include caching session data for a user when accessing a website. memcached uses a lazy expiry mechanism where the explicit expiry time that has been set is compared with the current time when the object is requested. Only objects that have not expired are returned.

You can also set the expiry time when explicitly deleting an object from the cache. In this case, the expiry time is really a timeout and indicates the period when any attempts to set the value for a given key are rejected.

16.2.2.5 memcached Hashing/Distribution Types

The memcached client interface supports a number of different distribution algorithms that are used in multi-server configurations to determine which host should be used when setting or getting data from a given memcached instance. When you get or set a value, a hash is constructed from the supplied key and then used to select a host from the list of configured servers. Because the hashing mechanism uses the supplied key as the basis for the hash, the same server is selected during both set and get operations.

You can think of this process as follows. Given an array of servers (a, b, and c), the client uses a hashing algorithm that returns an integer based on the key being stored or retrieved. The resulting value is then used to select a server from the list of servers configured in the client. Most standard client hashing within memcache clients uses a simple modulus calculation on the value against the number of configured memcached servers. You can summarize the process in pseudocode as:

```
@memcservers = ['a.memc','b.memc','c.memc'];
$value = hash($key);
$chosen = $value % length(@memcservers);
```

Replacing the above with values:

```
@memcservers = ['a.memc','b.memc','c.memc'];
$value = hash('myid');
$chosen = 7009 % 3;
```

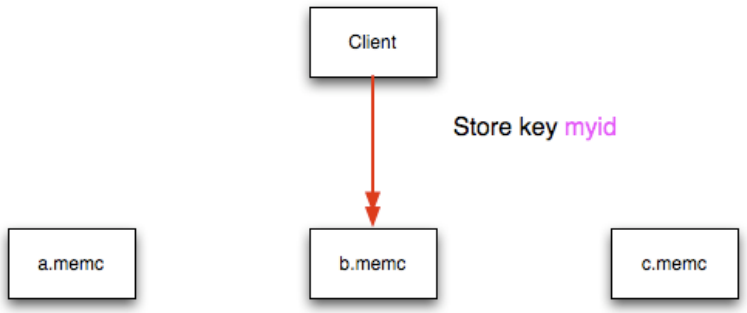
In the above example, the client hashing algorithm chooses the server at index 1 (`7009 % 3 = 1`), and stores or retrieves the key and value with that server.

Note

This selection and hashing process is handled automatically by the memcached client you are using; you need only provide the list of memcached servers to use.

You can see a graphical representation of this below in Figure 16.3, “memcached Hash Selection”.

Figure 16.3 memcached Hash Selection



The same hashing and selection process takes place during any operation on the specified key within the memcached client.

Using this method provides a number of advantages:

- The hashing and selection of the server to contact is handled entirely within the client. This eliminates the need to perform network communication to determine the right machine to contact.
- Because the determination of the memcached server occurs entirely within the client, the server can be selected automatically regardless of the operation being executed (set, get, increment, etc.).
- Because the determination is handled within the client, the hashing algorithm returns the same value for a given key; values are not affected or reset by differences in the server environment.
- Selection is very fast. The hashing algorithm on the key value is quick and the resulting selection of the server is from a simple array of available machines.
- Using client-side hashing simplifies the distribution of data over each memcached server. Natural distribution of the values returned by the hashing algorithm means that keys are automatically spread over the available servers.

Providing that the list of servers configured within the client remains the same, the same stored key returns the same value, and therefore selects the same server.

However, if you do not use the same hashing mechanism then the same data may be recorded on different servers by different interfaces, both wasting space on your memcached and leading to potential differences in the information.

Note

One way to use a multi-interface compatible hashing mechanism is to use the libmemcached library and the associated interfaces. Because the interfaces for the different languages (including C, Ruby, Perl and Python) use the same client library interface, they always generate the same hash code from the ID.

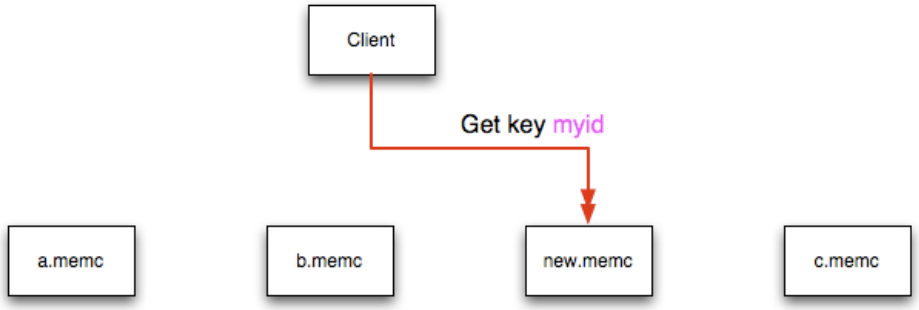
The problem with client-side selection of the server is that the list of the servers (including their sequential order) *must* remain consistent on each client using the memcached servers, and the servers must be available. If you try to perform an operation on a key when:

- A new memcached instance has been added to the list of available instances
- A memcached instance has been removed from the list of available instances
- The order of the memcached instances has changed

When the hashing algorithm is used on the given key, but with a different list of servers, the hash calculation may choose a different server from the list.

If a new memcached instance is added into the list of servers, as new.memc is in the example below, then a GET operation using the same key, myid, can result in a cache-miss. This is because the same value is computed from the key, which selects the same index from the array of servers, but index 2 now points to the new server, not the server c.memc where the data was originally stored. This would result in a cache miss, even though the key exists within the cache on another memcached instance.

Figure 16.4 memcached Hash Selection with New memcached instance



This means that servers c.memc and new.memc both contain the information for key myid, but the information stored against the key in each server may be different in each instance. A more significant problem is a much higher number of cache-misses when retrieving data, as the addition of a new server changes the distribution of keys, and this in turn requires rebuilding the cached data on the memcached instances, causing an increase in database reads.

The same effect can occur if you actively manage the list of servers configured in your clients, adding and removing the configured memcached instances as each instance is identified as being available. For example, removing a memcached instance when the client notices that the instance can no longer be contacted can cause the server selection to fail as described here.

To prevent this causing significant problems and invalidating your cache, you can select the hashing algorithm used to select the server. There are two common types of hashing algorithm, *consistent* and *modula*.

With *consistent* hashing algorithms, the same key when applied to a list of servers always uses the same server to store or retrieve the keys, even if the list of configured servers changes. This means that you can add and remove servers from the configure list and always use the same server for a given key. There are two types of consistent hashing algorithms available, Ketama and Wheel. Both types are supported by libmemcached, and implementations are available for PHP and Java.

Any consistent hashing algorithm has some limitations. When you add servers to an existing list of configured servers, keys are distributed to the new servers as part of the normal distribution. When you remove servers from the list, the keys are re-allocated to another server within the list, meaning that the cache needs to be re-populated with the information. Also, a consistent hashing algorithm does not resolve the issue where you want consistent selection of a server across multiple clients, but where each client contains a different list of servers. The consistency is enforced only within a single client.

With a *modula* hashing algorithm, the client selects a server by first computing the hash and then choosing a server from the list of configured servers. As the list of servers changes, so the server selected when using a modula hashing algorithm also changes. The result is the behavior described above; changes to the list of servers mean that different servers are selected when retrieving data, leading to cache misses and increase in database load as the cache is re-seeded with information.

If you use only a single memcached instance for each client, or your list of memcached servers configured for a client never changes, then the selection of a hashing algorithm is irrelevant, as it has no noticeable effect.

If you change your servers regularly, or you use a common set of servers that are shared among a large number of clients, then using a consistent hashing algorithm should help to ensure that your cache data is not duplicated and the data is evenly distributed.

16.2.2.6 Using memcached and DTrace

memcached includes a number of different DTrace probes that can be used to monitor the operation of the server. The probes included can monitor individual connections, slab allocations, and modifications to the hash table when a key-value pair is added, updated, or removed.

For more information on DTrace and writing DTrace scripts, read the DTrace User Guide.

Support for DTrace probes was added to memcached 1.2.6 includes a number of DTrace probes that can be used to help monitor your application. DTrace is supported on Solaris 10, OpenSolaris, OS X 10.5 and FreeBSD. To enable the DTrace probes in memcached, build from source and use the --enable-dtrace option. For more information, see Section 16.2.1, “Installing memcached”.

The probes supported by memcached are:

- conn-allocate(connid)

  Fired when a connection object is allocated from the connection pool.

  - connid: The connection ID.

- conn-release(connid)

  Fired when a connection object is released back to the connection pool.

  Arguments:

  - connid: The connection ID.

- conn-create(ptr)

  Fired when a new connection object is being created (that is, there are no free connection objects in the connection pool).

  Arguments:

  - ptr: A pointer to the connection. object

- conn-destroy(ptr)

  Fired when a connection object is being destroyed.

  Arguments:

  - ptr: A pointer to the connection object.

- conn-dispatch(connid, threadid)

  Fired when a connection is dispatched from the main or connection-management thread to a worker thread.

  Arguments:

  - connid: The connection ID.
  - threadid: The thread ID.

- slabs-allocate(size, slabclass, slabsize, ptr)

  Allocate memory from the slab allocator.

  Arguments:

  - size: The requested size.
  - slabclass: The allocation is fulfilled in this class.
  - slabsize: The size of each item in this class.
  - ptr: A pointer to allocated memory.

- `slabs-allocate-failed(size, slabclass)`

  Failed to allocate memory (out of memory).

  Arguments:

  - `size`: The requested size.
  - `slabclass`: The class that failed to fulfill the request.

- `slabs-slabclass-allocate(slabclass)`

  Fired when a slab class needs more space.

  Arguments:

  - `slabclass`: The class that needs more memory.

- `slabs-slabclass-allocate-failed(slabclass)`

  Failed to allocate memory (out of memory).

  Arguments:

  - `slabclass`: The class that failed to grab more memory.

- `slabs-free(size, slabclass, ptr)`

  Release memory.

  Arguments:

  - `size`: The amount of memory to release, in bytes.
  - `slabclass`: The class the memory belongs to.
  - `ptr`: A pointer to the memory to release.

- `assoc-find(key, depth)`

  Fired when we have searched the hash table for a named key. These two elements provide an insight into how well the hash function operates. Traversals are a sign of a less optimal function, wasting CPU capacity.

  Arguments:

  - `key`: The key searched for.
  - `depth`: The depth in the list of hash table.

- `assoc-insert(key, nokeys)`

  Fired when a new item has been inserted.

  Arguments:

  - `key`: The key just inserted.
  - `nokeys`: The total number of keys currently being stored, including the key for which insert was called.

- `assoc-delete(key, nokeys)`

  Fired when a new item has been removed.

Arguments:

- key: The key just deleted.
- nokeys: The total number of keys currently being stored, excluding the key for which delete was called.

- item-link(key, size)

  Fired when an item is being linked in the cache.

  Arguments:

  - key: The items key.
  - size: The size of the data.

- item-unlink(key, size)

  Fired when an item is being deleted.

  Arguments:

  - key: The items key.
  - size: The size of the data.

- item-remove(key, size)

  Fired when the refcount for an item is reduced.

  Arguments:

  - key: The item's key.
  - size: The size of the data.

- item-update(key, size)

  Fired when the "last referenced" time is updated.

  Arguments:

  - key: The item's key.
  - size: The size of the data.

- item-replace(oldkey, oldsize, newkey, newsize)

  Fired when an item is being replaced with another item.

  Arguments:

  - oldkey: The key of the item to replace.
  - oldsize: The size of the old item.
  - newkey: The key of the new item.
  - newsize: The size of the new item.

- `process-command-start(connid, request, size)`

  Fired when the processing of a command starts.

  Arguments:

  - `connid`: The connection ID.
  - `request`: The incoming request.
  - `size`: The size of the request.

- `process-command-end(connid, response, size)`

  Fired when the processing of a command is done.

  Arguments:

  - `connid`: The connection ID.
  - `response`: The response to send back to the client.
  - `size`: The size of the response.

- `command-get(connid, key, size)`

  Fired for a `get` command.

  Arguments:

  - `connid`: The connection ID.
  - `key`: The requested key.
  - `size`: The size of the key's data (or -1 if not found).

- `command-gets(connid, key, size, casid)`

  Fired for a `gets` command.

  Arguments:

  - `connid`: The connection ID.
  - `key`: The requested key.
  - `size`: The size of the key's data (or -1 if not found).
  - `casid`: The casid for the item.

- `command-add(connid, key, size)`

  Fired for a `add` command.

  Arguments:

  - `connid`: The connection ID.
  - `key`: The requested key.
  - `size`: The new size of the key's data (or -1 if not found).

- `command-set(connid, key, size)`

  Fired for a `set` command.

  Arguments:

  - `connid`: The connection ID.
  - `key`: The requested key.
  - `size`: The new size of the key's data (or -1 if not found).

- `command-replace(connid, key, size)`

  Fired for a `replace` command.

  Arguments:

  - `connid`: The connection ID.
  - `key`: The requested key.
  - `size`: The new size of the key's data (or -1 if not found).

- `command-prepend(connid, key, size)`

  Fired for a `prepend` command.

  Arguments:

  - `connid`: The connection ID.
  - `key`: The requested key.
  - `size`: The new size of the key's data (or -1 if not found).

- `command-append(connid, key, size)`

  Fired for a `append` command.

  Arguments:

  - `connid`: The connection ID.
  - `key`: The requested key.
  - `size`: The new size of the key's data (or -1 if not found).

- `command-cas(connid, key, size, casid)`

  Fired for a `cas` command.

  Arguments:

  - `connid`: The connection ID.
  - `key`: The requested key.
  - `size`: The size of the key's data (or -1 if not found).
  - `casid`: The cas ID requested.

- command-incr(connid, key, val)

  Fired for incr command.

  Arguments:

  - connid: The connection ID.
  - key: The requested key.
  - val: The new value.

- command-decr(connid, key, val)

  Fired for decr command.

  Arguments:

  - connid: The connection ID.
  - key: The requested key.
  - val: The new value.

- command-delete(connid, key, exptime)

  Fired for a delete command.

  Arguments:

  - connid: The connection ID.
  - key: The requested key.
  - exptime: The expiry time.

16.2.2.7 Memory Allocation within memcached

When you first start memcached, the memory that you have configured is not automatically allocated. Instead, memcached only starts allocating and reserving physical memory once you start saving information into the cache.

When you start to store data into the cache, memcached does not allocate the memory for the data on an item by item basis. Instead, a slab allocation is used to optimize memory usage and prevent memory fragmentation when information expires from the cache.

With slab allocation, memory is reserved in blocks of 1MB. The slab is divided up into a number of blocks of equal size. When you try to store a value into the cache, memcached checks the size of the value that you are adding to the cache and determines which slab contains the right size allocation for the item. If a slab with the item size already exists, the item is written to the block within the slab.

If the new item is bigger than the size of any existing blocks, then a new slab is created, divided up into blocks of a suitable size. If an existing slab with the right block size already exists, but there are no free blocks, a new slab is created. If you update an existing item with data that is larger than the existing block allocation for that key, then the key is re-allocated into a suitable slab.

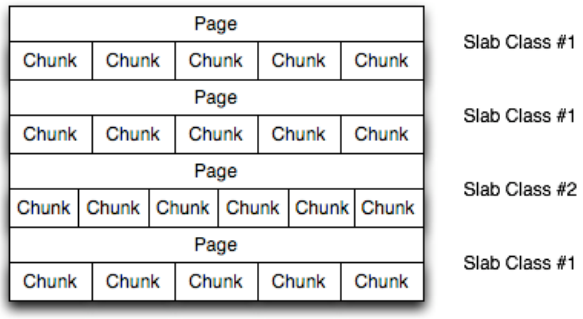
For example, the default size for the smallest block is 88 bytes (40 bytes of value, and the default 48 bytes for the key and flag data). If the size of the first item you store into the cache is less than 40 bytes, then a slab with a block size of 88 bytes is created and the value stored.

If the size of the data that you intend to store is larger than this value, then the block size is increased by the chunk size factor until a block size large enough to hold the value is determined. The block size

is always a function of the scale factor, rounded up to a block size which is exactly divisible into the chunk size.

For a sample of the structure, see Figure 16.5, “Memory Allocation in memcached”.

Figure 16.5 Memory Allocation in memcached



The result is that you have multiple pages allocated within the range of memory allocated to memcached. Each page is 1MB in size (by default), and is split into a different number of chunks, according to the chunk size required to store the key-value pairs. Each instance has multiple pages allocated, and a page is always created when a new item needs to be created requiring a chunk of a particular size. A slab may consist of multiple pages, and each page within a slab contains an equal number of chunks.

The chunk size of a new slab is determined by the base chunk size combined with the chunk size growth factor. For example, if the initial chunks are 104 bytes in size, and the default chunk size growth factor is used (1.25), then the next chunk size allocated would be the best power of 2 fit for 104*1.25, or 136 bytes.

Allocating the pages in this way ensures that memory does not get fragmented. However, depending on the distribution of the objects that you store, it may lead to an inefficient distribution of the slabs and chunks if you have significantly different sized items. For example, having a relatively small number of items within each chunk size may waste a lot of memory with just few chunks in each allocated page.

You can tune the growth factor to reduce this effect by using the `-f` command line option, which adapts the growth factor applied to make more effective use of the chunks and slabs allocated. For information on how to determine the current slab allocation statistics, see Section 16.2.4.2, “memcached Slabs Statistics”.

If your operating system supports it, you can also start memcached with the `-L` command line option. This option preallocates all the memory during startup using large memory pages. This can improve performance by reducing the number of misses in the CPU memory cache.

16.2.2.8 memcached Thread Support

If you enable the thread implementation within when building memcached from source, then memcached uses multiple threads in addition to the `libevent` system to handle requests.

When enabled, the threading implementation operates as follows:

- Threading is handled by wrapping functions within the code to provide basic protection from updating the same global structures at the same time.
- Each thread uses its own instance of the `libevent` to help improve performance.
- TCP/IP connections are handled with a single thread listening on the TCP/IP socket. Each connection is then distributed to one of the active threads on a simple round-robin basis. Each connection then operates solely within this thread while the connection remains open.
- For UDP connections, all the threads listen to a single UDP socket for incoming requests. Threads that are not currently dealing with another request ignore the incoming packet. One of the remaining,

nonbusy, threads reads the request and sends the response. This implementation can lead to increased CPU load as threads wake from sleep to potentially process the request.

Using threads can increase the performance on servers that have multiple CPU cores available, as the requests to update the hash table can be spread between the individual threads. To minimize overhead from the locking mechanism employed, experiment with different thread values to achieve the best performance based on the number and type of requests within your given workload.

16.2.2.9 memcached Logs

If you enable verbose mode, using the `-v`, `-vv`, or `-vvv` options, then the information output by `memcached` includes details of the operations being performed.

Without the verbose options, `memcached` normally produces no output during normal operating.

- **Output when using `-v`**

  The lowest verbosity level shows you:

  - Errors and warnings
  - Transient errors
  - Protocol and socket errors, including exhausting available connections
  - Each registered client connection, including the socket descriptor number and the protocol used.

    For example:

    ```
    32: Client using the ascii protocol
    33: Client using the ascii protocol
    ```

    The socket descriptor is only valid while the client remains connected. Non-persistent connections may not be effectively represented.

  Examples of the error messages output at this level include:

  ```
  <%d send buffer was %d, now %d
  Can't listen for events on fd %d
  Can't read from libevent pipe
  Catastrophic: event fd doesn't match conn fd!
  Couldn't build response
  Couldn't realloc input buffer
  Couldn't update event
  Failed to build UDP headers
  Failed to read, and not due to blocking
  Too many open connections
  Unexpected state %d
  ```

- **Output when using `-vv`**

  When using the second level of verbosity, you get more detailed information about protocol operations, keys updated, chunk and network operatings and details.

  During the initial start-up of `memcached` with this level of verbosity, you are shown the sizes of the individual slab classes, the chunk sizes, and the number of entries per slab. These do not show the allocation of the slabs, just the slabs that would be created when data is added. You are also given information about the listen queues and buffers used to send information. A sample of the output generated for a TCP/IP based system with the default memory and growth factors is given below:

  ```
  shell> memcached -vv
  slab class   1: chunk size     80 perslab 13107
  slab class   2: chunk size    104 perslab 10082
  slab class   3: chunk size    136 perslab  7710
  slab class   4: chunk size    176 perslab  5957
  slab class   5: chunk size    224 perslab  4681
  ```

```
slab class   6: chunk size    280 perslab  3744
slab class   7: chunk size    352 perslab  2978
slab class   8: chunk size    440 perslab  2383
slab class   9: chunk size    552 perslab  1899
slab class  10: chunk size    696 perslab  1506
slab class  11: chunk size    872 perslab  1202
slab class  12: chunk size   1096 perslab   956
slab class  13: chunk size   1376 perslab   762
slab class  14: chunk size   1720 perslab   609
slab class  15: chunk size   2152 perslab   487
slab class  16: chunk size   2696 perslab   388
slab class  17: chunk size   3376 perslab   310
slab class  18: chunk size   4224 perslab   248
slab class  19: chunk size   5280 perslab   198
slab class  20: chunk size   6600 perslab   158
slab class  21: chunk size   8256 perslab   127
slab class  22: chunk size  10320 perslab   101
slab class  23: chunk size  12904 perslab    81
slab class  24: chunk size  16136 perslab    64
slab class  25: chunk size  20176 perslab    51
slab class  26: chunk size  25224 perslab    41
slab class  27: chunk size  31536 perslab    33
slab class  28: chunk size  39424 perslab    26
slab class  29: chunk size  49280 perslab    21
slab class  30: chunk size  61600 perslab    17
slab class  31: chunk size  77000 perslab    13
slab class  32: chunk size  96256 perslab    10
slab class  33: chunk size 120320 perslab     8
slab class  34: chunk size 150400 perslab     6
slab class  35: chunk size 188000 perslab     5
slab class  36: chunk size 235000 perslab     4
slab class  37: chunk size 293752 perslab     3
slab class  38: chunk size 367192 perslab     2
slab class  39: chunk size 458992 perslab     2
<26 server listening (auto-negotiate)
<29 server listening (auto-negotiate)
<30 send buffer was 57344, now 2097152
<31 send buffer was 57344, now 2097152
<30 server listening (udp)
<30 server listening (udp)
<31 server listening (udp)
<30 server listening (udp)
<30 server listening (udp)
<31 server listening (udp)
<31 server listening (udp)
<31 server listening (udp)
```

Using this verbosity level can be a useful way to check the effects of the growth factor used on slabs with different memory allocations, which in turn can be used to better tune the growth factor to suit the data you are storing in the cache. For example, if you set the growth factor to 4 (quadrupling the size of each slab):

```
shell> memcached -f 4 -m 1g -vv
slab class   1: chunk size     80 perslab 13107
slab class   2: chunk size    320 perslab  3276
slab class   3: chunk size   1280 perslab   819
slab class   4: chunk size   5120 perslab   204
slab class   5: chunk size  20480 perslab    51
slab class   6: chunk size  81920 perslab    12
slab class   7: chunk size 327680 perslab     3
...
```

During use of the cache, this verbosity level also prints out detailed information on the storage and recovery of keys and other information. An example of the output during a typical set/get and increment/decrement operation is shown below.

```
32: Client using the ascii protocol
<32 set my_key 0 0 10
>32 STORED
<32 set object_key 1 0 36
```

```
>32 STORED
<32 get my_key
>32 sending key my_key
>32 END
<32 get object_key
>32 sending key object_key
>32 END
<32 set key 0 0 6
>32 STORED
<32 incr key 1
>32 789544
<32 decr key 1
>32 789543
<32 incr key 2
>32 789545
<32 set my_key 0 0 10
>32 STORED
<32 set object_key 1 0 36
>32 STORED
<32 get my_key
>32 sending key my_key
>32 END
<32 get object_key
>32 sending key object_key1 1 36

>32 END
<32 set key 0 0 6
>32 STORED
<32 incr key 1
>32 789544
<32 decr key 1
>32 789543
<32 incr key 2
>32 789545
```

During client communication, for each line, the initial character shows the direction of flow of the information. The < for communication from the client to the memcached server and > for communication back to the client. The number is the numeric socket descriptor for the connection.

- **Output when using -vvv**

  This level of verbosity includes the transitions of connections between different states in the event library while reading and writing content to/from the clients. It should be used to diagnose and identify issues in client communication. For example, you can use this information to determine if memcached is taking a long time to return information to the client, during the read of the client operation or before returning and completing the operation. An example of the typical sequence for a set operation is provided below:

```
<32 new auto-negotiating client connection
32: going from conn_new_cmd to conn_waiting
32: going from conn_waiting to conn_read
32: going from conn_read to conn_parse_cmd
32: Client using the ascii protocol
<32 set my_key 0 0 10
32: going from conn_parse_cmd to conn_nread
> NOT FOUND my_key
>32 STORED
32: going from conn_nread to conn_write
32: going from conn_write to conn_new_cmd
32: going from conn_new_cmd to conn_waiting
32: going from conn_waiting to conn_read
32: going from conn_read to conn_closing
<32 connection closed.
```

All of the verbosity levels in memcached are designed to be used during debugging or examination of issues. The quantity of information generated, particularly when using -vvv, is significant, particularly on a busy server. Also be aware that writing the error information out, especially to disk, may negate some of the performance gains you achieve by using memcached. Therefore, use in production or deployment environments is not recommended.

16.2.3 Developing a memcached Application

A number of language interfaces let applications store and retrieve information with memcached servers. You can write memcached applications in popular languages such as Perl, PHP, Python, Ruby, C, and Java.

Data stored into a memcached server is referred to by a single string (the key), with storage into the cache and retrieval from the cache using the key as the reference. The cache therefore operates like a large associative array or hash table. It is not possible to structure or otherwise organize the information stored in the cache. To emulate database notions such as multiple tables or composite key values, you must encode the extra information into the strings used as keys. For example, to store or look up the address corresponding to a specific latitude and longitude, you might turn those two numeric values into a single comma-separated string to use as a key.

16.2.3.1 Basic memcached Operations

The interface to memcached supports the following methods for storing and retrieving information in the cache, and these are consistent across all the different APIs, although the language specific mechanics might be different:

- get(*key*): Retrieves information from the cache. Returns the value associated with the key if the specified key exists. Returns NULL, nil, undefined, or the closest equivalent in the corresponding language, if the specified key does not exist.
- set(*key, value [, expiry]*): Sets the item associated with a key in the cache to the specified value. This either updates an existing item if the key already exists, or adds a new key-value pair if the key doesn't exist. If the expiry time is specified, then the item expires (and is deleted) when the expiry time is reached. The time is specified in seconds, and is taken as a relative time if the value is less than 30 days (30*24*60*60), or an absolute time (epoch) if larger than this value.
- add(*key, value [, expiry]*): Adds the key and associated value to the cache, if the specified key does not already exist.
- replace(*key, value [, expiry]*): Replaces the item associated with the specified key, only if the key already exists. The new value is given by the value parameter.
- delete(*key [, time]*): Deletes the key and its associated item from the cache. If you supply a time, then adding another item with the specified key is blocked for the specified period.
- incr(*key , value*): Increments the item associated with the key by the specified value.
- decr(*key , value*): Decrements the item associated with the key by the specified value.
- flush_all: Invalidates (or expires) all the current items in the cache. Technically they still exist (they are not deleted), but they are silently destroyed the next time you try to access them.

In all implementations, most or all of these functions are duplicated through the corresponding native language interface.

When practical, use memcached to store full items, rather than caching a single column value from the database. For example, when displaying a record about an object (invoice, user history, or blog post), load all the data for the associated entry from the database, and compile it into the internal structure that would normally be required by the application. Save the complete object in the cache.

Complex data structures cannot be stored directly. Most interfaces serialize the data for you, that is, put it in a textual form that can reconstruct the original pointers and nesting. Perl uses Storable, PHP uses serialize, Python uses cPickle (or Pickle) and Java uses the Serializable interface. In most cases, the serialization interface used is customizable. To share data stored in memcached instances between different language interfaces, consider using a common serialization solution such as JSON (Javascript Object Notation).

16.2.3.2 Using memcached as a MySQL Caching Layer

When using memcached to cache MySQL data, your application must retrieve data from the database and load the appropriate key-value pairs into the cache. Then, subsequent lookups can be done directly from the cache.

Because MySQL has its own in-memory caching mechanisms for queried data, such as the InnoDB buffer pool and the MySQL query cache, look for opportunities beyond loading individual column values or rows into the cache. Prefer to cache composite values, such as those retrieved from multiple tables through a join query, or result sets assembled from multiple rows.

Caution

Limit the information in the cache to non-sensitive data, because there is no security required to access or update the information within a memcached instance. Anybody with access to the machine has the ability to read, view and potentially update the information. To keep the data secure, encrypt the information before caching it. To restrict the users capable of connecting to the server, either disable network access, or use IPTables or similar techniques to restrict access to the memcached ports to a select set of hosts.

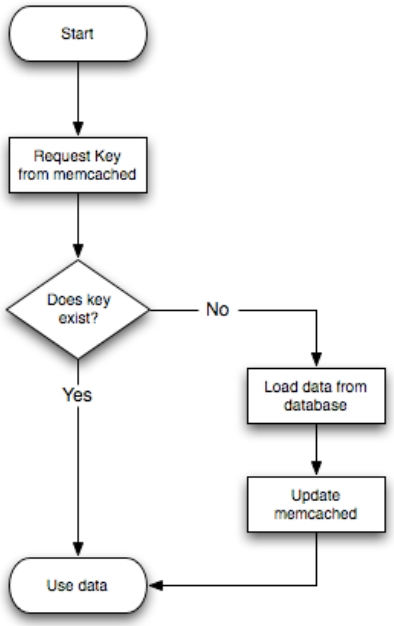
You can introduce memcached to an existing application, even if caching was not part of the original design. In many languages and environments the changes to the application is just a few lines, first to attempt to read from the cache when loading data, fall back to the old method if the information is not cached, and to update the cache with information once the data has been read.

The general sequence for using memcached in any language as a caching solution for MySQL is as follows:

1. Request the item from the cache.
2. If the item exists, use the item data.
3. If the item does not exist, load the data from MySQL, and store the value into the cache. This means the value is available to the next client that requests it from the cache.

For a flow diagram of this sequence, see Figure 16.6, “Typical memcached Application Flowchart”.

Figure 16.6 Typical memcached Application Flowchart



Adapting Database Best Practices to memcached Applications

The most direct way to cache MySQL data is to use a 2-column table, where the first column is a primary key. Because of the uniqueness requirements for memcached keys, make sure your database schema makes appropriate use of primary keys and unique constraints.

If you combine multiple column values into a single memcached item value, choose data types to make it easy to parse the value back into its components, for example by using a separator character between numeric values.

The queries that map most easily to memcached lookups are those with a single WHERE clause, using an = or IN operator. For complicated WHERE clauses, or those using operators such as <, >, BETWEEN, or LIKE, memcached does not provide a simple or efficient way to scan through or filter the keys or associated values, so typically you perform those operations as SQL queries on the underlying database.

16.2.3.3 Using libmemcached with C and C++

The libmemcached library provides both C and C++ interfaces to memcached and is also the basis for a number of different additional API implementations, including Perl, Python and Ruby. Understanding the core libmemcached functions can help when using these other interfaces.

The C library is the most comprehensive interface library for memcached and provides functions and operational systems not always exposed in interfaces not based on the libmemcached library.

The different functions can be divided up according to their basic operation. In addition to functions that interface to the core API, a number of utility functions provide extended functionality, such as appending and prepending data.

To build and install libmemcached, download the libmemcached package, run configure, and then build and install:

```
shell> tar xjf libmemcached-0.21.tar.gz
shell> cd libmemcached-0.21
shell> ./configure
shell> make
shell> make install
```

On many Linux operating systems, you can install the corresponding libmemcached package through the usual yum, apt-get, or similar commands.

To build an application that uses the library, first set the list of servers. Either directly manipulate the servers configured within the main memcached_st structure, or separately populate a list of servers, and then add this list to the memcached_st structure. The latter method is used in the following example. Once the server list has been set, you can call the functions to store or retrieve data. A simple application for setting a preset value to localhost is provided here:

```
#include <stdio.h>
#include <string.h>
#include <unistd.h>
#include <libmemcached/memcached.h>

int main(int argc, char *argv[])
{
  memcached_server_st *servers = NULL;
  memcached_st *memc;
  memcached_return rc;
  char *key= "keystring";
  char *value= "keyvalue";

  memcached_server_st *memcached_servers_parse (char *server_strings);
  memc= memcached_create(NULL);

  servers= memcached_server_list_append(servers, "localhost", 11211, &rc);
```

```
  rc= memcached_server_push(memc, servers);

  if (rc == MEMCACHED_SUCCESS)
    fprintf(stderr,"Added server successfully\n");
  else
    fprintf(stderr,"Couldn't add server: %s\n",memcached_strerror(memc, rc));

  rc= memcached_set(memc, key, strlen(key), value, strlen(value), (time_t)0, (uint32_t)0);

  if (rc == MEMCACHED_SUCCESS)
    fprintf(stderr,"Key stored successfully\n");
  else
    fprintf(stderr,"Couldn't store key: %s\n",memcached_strerror(memc, rc));

  return 0;
}
```

To test the success of an operation, use the return value, or populated result code, for a given function. The value is always set to `MEMCACHED_SUCCESS` if the operation succeeded. In the event of a failure, use the `memcached_strerror()` function to translate the result code into a printable string.

To build the application, specify the `memcached` library:

```
shell> gcc -o memc_basic memc_basic.c -lmemcached
```

Running the above sample application, after starting a `memcached` server, should return a success message:

```
shell> memc_basic
Added server successfully
Key stored successfully
```

`libmemcached` Base Functions

The base `libmemcached` functions let you create, destroy and clone the main `memcached_st` structure that is used to interface with the `memcached` servers. The main functions are defined below:

```
memcached_st *memcached_create (memcached_st *ptr);
```

Creates a new `memcached_st` structure for use with the other `libmemcached` API functions. You can supply an existing, static, `memcached_st` structure, or `NULL` to have a new structured allocated. Returns a pointer to the created structure, or `NULL` on failure.

```
void memcached_free (memcached_st *ptr);
```

Frees the structure and memory allocated to a previously created `memcached_st` structure.

```
memcached_st *memcached_clone(memcached_st *clone, memcached_st *source);
```

Clones an existing `memcached` structure from the specified `source`, copying the defaults and list of servers defined in the structure.

`libmemcached` Server Functions

The `libmemcached` API uses a list of servers, stored within the `memcached_server_st` structure, to act as the list of servers used by the rest of the functions. To use `memcached`, you first create the server list, and then apply the list of servers to a valid `libmemcached` object.

Because the list of servers, and the list of servers within an active `libmemcached` object can be manipulated separately, you can update and manage server lists while an active `libmemcached` interface is running.

The functions for manipulating the list of servers within a `memcached_st` structure are:

```
memcached_return
   memcached_server_add (memcached_st *ptr,
```

```
                          char *hostname,
                          unsigned int port);
```

Adds a server, using the given hostname and port into the memcached_st structure given in ptr.

```
memcached_return
   memcached_server_add_unix_socket (memcached_st *ptr,
                                     char *socket);
```

Adds a Unix socket to the list of servers configured in the memcached_st structure.

```
unsigned int memcached_server_count (memcached_st *ptr);
```

Returns a count of the number of configured servers within the memcached_st structure.

```
memcached_server_st *
   memcached_server_list (memcached_st *ptr);
```

Returns an array of all the defined hosts within a memcached_st structure.

```
memcached_return
   memcached_server_push (memcached_st *ptr,
                          memcached_server_st *list);
```

Pushes an existing list of servers onto list of servers configured for a current memcached_st structure. This adds servers to the end of the existing list, and duplicates are not checked.

The memcached_server_st structure can be used to create a list of memcached servers which can then be applied individually to memcached_st structures.

```
memcached_server_st *
   memcached_server_list_append (memcached_server_st *ptr,
                                 char *hostname,
                                 unsigned int port,
                                 memcached_return *error);
```

Adds a server, with hostname and port, to the server list in ptr. The result code is handled by the error argument, which should point to an existing memcached_return variable. The function returns a pointer to the returned list.

```
unsigned int memcached_server_list_count (memcached_server_st *ptr);
```

Returns the number of the servers in the server list.

```
void memcached_server_list_free (memcached_server_st *ptr);
```

Frees the memory associated with a server list.

```
memcached_server_st *memcached_servers_parse (char *server_strings);
```

Parses a string containing a list of servers, where individual servers are separated by a comma, space, or both, and where individual servers are of the form *server*[:*port*]. The return value is a server list structure.

libmemcached Set Functions

The set-related functions within libmemcached provide the same functionality as the core functions supported by the memcached protocol. The full definition for the different functions is the same for all the base functions (add, replace, prepend, append). For example, the function definition for memcached_set() is:

```
memcached_return
   memcached_set (memcached_st *ptr,
                  const char *key,
                  size_t key_length,
                  const char *value,
```

```
                size_t value_length,
                time_t expiration,
                uint32_t flags);
```

The ptr is the memcached_st structure. The key and key_length define the key name and length, and value and value_length the corresponding value and length. You can also set the expiration and optional flags. For more information, see Controlling libmemcached Behaviors.

The following table outlines the remainder of the set-related libmemcached functions and the equivalent core functions supported by the memcached protocol.

libmemcached Function	Equivalent Core Function
memcached_set(memc, key, key_length, value, value_length, expiration, flags)	Generic set() operation.
memcached_add(memc, key, key_length, value, value_length, expiration, flags)	Generic add() function.
memcached_replace(memc, key, key_length, value, value_length, expiration, flags)	Generic replace().
memcached_prepend(memc, key, key_length, value, value_length, expiration, flags)	Prepends the specified value before the current value of the specified key.
memcached_append(memc, key, key_length, value, value_length, expiration, flags)	Appends the specified value after the current value of the specified key.
memcached_cas(memc, key, key_length, value, value_length, expiration, flags, cas)	Overwrites the data for a given key as long as the corresponding cas value is still the same within the server.
memcached_set_by_key(memc, master_key, master_key_length, key, key_length, value, value_length, expiration, flags)	Similar to the generic set(), but has the option of an additional master key that can be used to identify an individual server.
memcached_add_by_key(memc, master_key, master_key_length, key, key_length, value, value_length, expiration, flags)	Similar to the generic add(), but has the option of an additional master key that can be used to identify an individual server.
memcached_replace_by_key(memc, master_key, master_key_length, key, key_length, value, value_length, expiration, flags)	Similar to the generic replace(), but has the option of an additional master key that can be used to identify an individual server.
memcached_prepend_by_key(memc, master_key, master_key_length, key, key_length, value, value_length, expiration, flags)	Similar to the memcached_prepend(), but has the option of an additional master key that can be used to identify an individual server.
memcached_append_by_key(memc, master_key, master_key_length, key, key_length, value, value_length, expiration, flags)	Similar to the memcached_append(), but has the option of an additional master key that can be used to identify an individual server.
memcached_cas_by_key(memc, master_key, master_key_length, key, key_length, value, value_length, expiration, flags)	Similar to the memcached_cas(), but has the option of an additional master key that can be used to identify an individual server.

The by_key methods add two further arguments that define the master key, to be used and applied during the hashing stage for selecting the servers. You can see this in the following definition:

```
memcached_return
   memcached_set_by_key(memcached_st *ptr,
                        const char *master_key,
                        size_t master_key_length,
                        const char *key,
                        size_t key_length,
                        const char *value,
                        size_t value_length,
                        time_t expiration,
                        uint32_t flags);
```

All the functions return a value of type memcached_return, which you can compare against the MEMCACHED_SUCCESS constant.

libmemcached Get Functions

The libmemcached functions provide both direct access to a single item, and a multiple-key request mechanism that provides much faster responses when fetching a large number of keys simultaneously.

The main get-style function, which is equivalent to the generic get() is memcached_get(). This function returns a string pointer, pointing to the value associated with the specified key.

```
char *memcached_get (memcached_st *ptr,
                     const char *key, size_t key_length,
                     size_t *value_length,
                     uint32_t *flags,
                     memcached_return *error);
```

A multi-key get, memcached_mget(), is also available. Using a multiple key get operation is much quicker to do in one block than retrieving the key values with individual calls to memcached_get(). To start the multi-key get, call memcached_mget():

```
memcached_return
    memcached_mget (memcached_st *ptr,
                    char **keys, size_t *key_length,
                    unsigned int number_of_keys);
```

The return value is the success of the operation. The keys parameter should be an array of strings containing the keys, and key_length an array containing the length of each corresponding key. number_of_keys is the number of keys supplied in the array.

To fetch the individual values, use memcached_fetch() to get each corresponding value.

```
char *memcached_fetch (memcached_st *ptr,
                       const char *key, size_t *key_length,
                       size_t *value_length,
                       uint32_t *flags,
                       memcached_return *error);
```

The function returns the key value, with the key, key_length and value_length parameters being populated with the corresponding key and length information. The function returns NULL when there are no more values to be returned. A full example, including the populating of the key data and the return of the information is provided here.

```
#include <stdio.h>
#include <sstring.h>
#include <unistd.h>
#include <libmemcached/memcached.h>

int main(int argc, char *argv[])
{
  memcached_server_st *servers = NULL;
  memcached_st *memc;
  memcached_return rc;
  char *keys[]= {"huey", "dewey", "louie"};
```

```
  size_t key_length[3];
  char *values[]= {"red", "blue", "green"};
  size_t value_length[3];
  unsigned int x;
  uint32_t flags;

  char return_key[MEMCACHED_MAX_KEY];
  size_t return_key_length;
  char *return_value;
  size_t return_value_length;

  memc= memcached_create(NULL);

  servers= memcached_server_list_append(servers, "localhost", 11211, &rc);
  rc= memcached_server_push(memc, servers);

  if (rc == MEMCACHED_SUCCESS)
    fprintf(stderr,"Added server successfully\n");
  else
    fprintf(stderr,"Couldn't add server: %s\n",memcached_strerror(memc, rc));

  for(x= 0; x < 3; x++)
    {
      key_length[x] = strlen(keys[x]);
      value_length[x] = strlen(values[x]);

      rc= memcached_set(memc, keys[x], key_length[x], values[x],
                        value_length[x], (time_t)0, (uint32_t)0);
      if (rc == MEMCACHED_SUCCESS)
        fprintf(stderr,"Key %s stored successfully\n",keys[x]);
      else
        fprintf(stderr,"Couldn't store key: %s\n",memcached_strerror(memc, rc));
    }

  rc= memcached_mget(memc, keys, key_length, 3);

  if (rc == MEMCACHED_SUCCESS)
    {
      while ((return_value= memcached_fetch(memc, return_key, &return_key_length,
                                            &return_value_length, &flags, &rc)) != NULL)
        {
          if (rc == MEMCACHED_SUCCESS)
            {
              fprintf(stderr,"Key %s returned %s\n",return_key, return_value);
            }
        }
    }

  return 0;
}
```

Running the above application produces the following output:

```
shell> memc_multi_fetch
Added server successfully
Key huey stored successfully
Key dewey stored successfully
Key louie stored successfully
Key huey returned red
Key dewey returned blue
Key louie returned green
```

Controlling libmemcached Behaviors

The behavior of libmemcached can be modified by setting one or more behavior flags. These can either be set globally, or they can be applied during the call to individual functions. Some behaviors also accept an additional setting, such as the hashing mechanism used when selecting servers.

To set global behaviors:

```
memcached_return
```

```
   memcached_behavior_set (memcached_st *ptr,
                           memcached_behavior flag,
                           uint64_t data);
```

To get the current behavior setting:

```
uint64_t
   memcached_behavior_get (memcached_st *ptr,
                           memcached_behavior flag);
```

The following table describes libmemcached behavior flags.

Behavior	Description
MEMCACHED_BEHAVIOR_NO_BLOCK	Caused libmemcached to use asynchronous I/O.
MEMCACHED_BEHAVIOR_TCP_NODELAY	Turns on no-delay for network sockets.
MEMCACHED_BEHAVIOR_HASH	Without a value, sets the default hashing algorithm for keys to use MD5. Other valid values include MEMCACHED_HASH_DEFAULT, MEMCACHED_HASH_MD5, MEMCACHED_HASH_CRC, MEMCACHED_HASH_FNV1_64, MEMCACHED_HASH_FNV1A_64, MEMCACHED_HASH_FNV1_32, and MEMCACHED_HASH_FNV1A_32.
MEMCACHED_BEHAVIOR_DISTRIBUTION	Changes the method of selecting the server used to store a given value. The default method is MEMCACHED_DISTRIBUTION_MODULA. You can enable consistent hashing by setting MEMCACHED_DISTRIBUTION_CONSISTENT. MEMCACHED_DISTRIBUTION_CONSISTENT is an alias for the value MEMCACHED_DISTRIBUTION_CONSISTENT_KETAMA.
MEMCACHED_BEHAVIOR_CACHE_LOOKUPS	Cache the lookups made to the DNS service. This can improve the performance if you are using names instead of IP addresses for individual hosts.
MEMCACHED_BEHAVIOR_SUPPORT_CAS	Support CAS operations. By default, this is disabled because it imposes a performance penalty.
MEMCACHED_BEHAVIOR_KETAMA	Sets the default distribution to MEMCACHED_DISTRIBUTION_CONSISTENT_KETAMA and the hash to MEMCACHED_HASH_MD5.
MEMCACHED_BEHAVIOR_POLL_TIMEOUT	Modify the timeout value used by poll(). Supply a signed int pointer for the timeout value.
MEMCACHED_BEHAVIOR_BUFFER_REQUESTS	Buffers IO requests instead of them being sent. A get operation, or closing the connection causes the data to be flushed.
MEMCACHED_BEHAVIOR_VERIFY_KEY	Forces libmemcached to verify that a specified key is valid.
MEMCACHED_BEHAVIOR_SORT_HOSTS	If set, hosts added to the list of configured hosts for a memcached_st structure are placed into the host list in sorted order. This breaks consistent hashing if that behavior has been enabled.
MEMCACHED_BEHAVIOR_CONNECT_TIMEOUT	In nonblocking mode this changes the value of the timeout during socket connection.

libmemcached Command-Line Utilities

In addition to the main C library interface, libmemcached also includes a number of command-line utilities that can be useful when working with and debugging memcached applications.

All of the command-line tools accept a number of arguments, the most critical of which is servers, which specifies the list of servers to connect to when returning information.

The main tools are:

- memcat: Display the value for each ID given on the command line:

```
shell> memcat --servers=localhost hwkey
Hello world
```

- memcp: Copy the contents of a file into the cache, using the file name as the key:

```
shell> echo "Hello World" > hwkey
shell> memcp --servers=localhost hwkey
shell> memcat --servers=localhost hwkey
Hello world
```

- memrm: Remove an item from the cache:

```
shell> memcat --servers=localhost hwkey
Hello world
shell> memrm --servers=localhost hwkey
shell> memcat --servers=localhost hwkey
```

- memslap: Test the load on one or more memcached servers, simulating get/set and multiple client operations. For example, you can simulate the load of 100 clients performing get operations:

```
shell> memslap --servers=localhost --concurrency=100 --flush --test=get
memslap --servers=localhost --concurrency=100 --flush --test=get Threads connecting to servers 100
 Took 13.571 seconds to read data
```

- memflush: Flush (empty) the contents of the memcached cache.

```
shell> memflush --servers=localhost
```

16.2.3.4 Using MySQL and memcached with Perl

The Cache::Memcached module provides a native interface to the Memcache protocol, and provides support for the core functions offered by memcached. Install the module using your operating system's package management system, or using CPAN:

```
root-shell> perl -MCPAN -e 'install Cache::Memcached'
```

To use memcached from Perl through the Cache::Memcached module, first create a new Cache::Memcached object that defines the list of servers and other parameters for the connection. The only argument is a hash containing the options for the cache interface. For example, to create a new instance that uses three memcached servers:

```
use Cache::Memcached;

my $cache = new Cache::Memcached {
    'servers' => [
        '198.51.100.100:11211',
        '198.51.100.101:11211',
        '198.51.100.102:11211',
 ],
};
```

Note

When using the Cache::Memcached interface with multiple servers, the API automatically performs certain operations across all the servers in the group. For example, getting statistical information through Cache::Memcached returns a hash that contains data on a host-by-host basis, as well as generalized statistics for all the servers in the group.

You can set additional properties on the cache object instance when it is created by specifying the option as part of the option hash. Alternatively, you can use a corresponding method on the instance:

- servers or method set_servers(): Specifies the list of the servers to be used. The servers list should be a reference to an array of servers, with each element as the address and port number combination (separated by a colon). You can also specify a local connection through a Unix socket (for example /tmp/sock/memcached). To specify the server with a weight (indicating how much more frequently the server should be used during hashing), specify an array reference with the memcached server instance and a weight number. Higher numbers give higher priority.
- compress_threshold or method set_compress_threshold(): Specifies the threshold when values are compressed. Values larger than the specified number are automatically compressed (using zlib) during storage and retrieval.
- no_rehash or method set_norehash(): Disables finding a new server if the original choice is unavailable.
- readonly or method set_readonly(): Disables writes to the memcached servers.

Once the Cache::Memcached object instance has been configured, you can use the set() and get() methods to store and retrieve information from the memcached servers. Objects stored in the cache are automatically serialized and deserialized using the Storable module.

The Cache::Memcached interface supports the following methods for storing/retrieving data, and relate to the generic methods as shown in the table.

Cache::Memcached Function	**Equivalent Generic Method**
get()	Generic get().
get_multi(keys)	Gets multiple keys from memcache using just one query. Returns a hash reference of key-value pairs.
set()	Generic set().
add()	Generic add().
replace()	Generic replace().
delete()	Generic delete().
incr()	Generic incr().
decr()	Generic decr().

Below is a complete example for using memcached with Perl and the Cache::Memcached module:

```
#!/usr/bin/perl

use Cache::Memcached;
use DBI;
use Data::Dumper;

# Configure the memcached server

my $cache = new Cache::Memcached {
    'servers' => [
                  'localhost:11211',
                  ],
    };

# Get the film name from the command line
# memcached keys must not contain spaces, so create
# a key name by replacing spaces with underscores

my $filmname = shift or die "Must specify the film name\n";
my $filmkey = $filmname;
```

```
$filmkey =~ s/ /_/;

# Load the data from the cache

my $filmdata = $cache->get($filmkey);

# If the data wasn't in the cache, then we load it from the database

if (!defined($filmdata))
{
    $filmdata = load_filmdata($filmname);

    if (defined($filmdata))
    {

# Set the data into the cache, using the key

 if ($cache->set($filmkey,$filmdata))
        {
            print STDERR "Film data loaded from database and cached\n";
        }
        else
        {
            print STDERR "Couldn't store to cache\n";
 }
    }
    else
    {
      die "Couldn't find $filmname\n";
    }
}
else
{
    print STDERR "Film data loaded from Memcached\n";
}

sub load_filmdata
{
    my ($filmname) = @_;

    my $dsn = "DBI:mysql:database=sakila;host=localhost;port=3306";

    $dbh = DBI->connect($dsn, 'sakila','password');

    my ($filmbase) = $dbh->selectrow_hashref(sprintf('select * from film where title = %s',
                                                     $dbh->quote($filmname)));

    if (!defined($filmname))
    {
      return (undef);
    }

    $filmbase->{stars} =
 $dbh->selectall_arrayref(sprintf('select concat(first_name," ",last_name) ' .
                                          'from film_actor left join (actor) ' .
                                          'on (film_actor.actor_id = actor.actor_id) ' .
                                          ' where film_id=%s',
                                          $dbh->quote($filmbase->{film_id})));

    return($filmbase);
}
```

The example uses the Sakila database, obtaining film data from the database and writing a composite record of the film and actors to memcached. When calling it for a film does not exist, you get this result:

```
shell> memcached-sakila.pl "ROCK INSTINCT"
Film data loaded from database and cached
```

When accessing a film that has already been added to the cache:

```
shell> memcached-sakila.pl "ROCK INSTINCT"
Film data loaded from Memcached
```

16.2.3.5 Using MySQL and memcached with Python

The Python memcache module interfaces to memcached servers, and is written in pure Python (that is, without using one of the C APIs). You can download and install a copy from Python Memcached.

To install, download the package and then run the Python installer:

```
python setup.py install
running install
running bdist_egg
running egg_info
creating python_memcached.egg-info
...
removing 'build/bdist.linux-x86_64/egg' (and everything under it)
Processing python_memcached-1.43-py2.4.egg
creating /usr/lib64/python2.4/site-packages/python_memcached-1.43-py2.4.egg
Extracting python_memcached-1.43-py2.4.egg to /usr/lib64/python2.4/site-packages
Adding python-memcached 1.43 to easy-install.pth file

Installed /usr/lib64/python2.4/site-packages/python_memcached-1.43-py2.4.egg
Processing dependencies for python-memcached==1.43
Finished processing dependencies for python-memcached==1.43
```

Once installed, the memcache module provides a class-based interface to your memcached servers. When you store Python data structures as memcached items, they are automatically serialized (turned into string values) using the Python cPickle or pickle modules.

To create a new memcache interface, import the memcache module and create a new instance of the memcache.Client class. For example, if the memcached daemon is running on localhost using the default port:

```
import memcache
memc = memcache.Client(['127.0.0.1:11211'])
```

The first argument is an array of strings containing the server and port number for each memcached instance to use. To enable debugging, set the optional debug parameter to 1.

By default, the hashing mechanism used to divide the items among multiple servers is crc32. To change the function used, set the value of memcache.serverHashFunction to the alternate function to use. For example:

```
from zlib import adler32
memcache.serverHashFunction = adler32
```

Once you have defined the servers to use within the memcache instance, the core functions provide the same functionality as in the generic interface specification. The following table provides a summary of the supported functions.

Python memcache Function	Equivalent Generic Function
get()	Generic get().
get_multi(keys)	Gets multiple values from the supplied array of keys. Returns a hash reference of key-value pairs.
set()	Generic set().
set_multi(dict [, expiry [, key_prefix]])	Sets multiple key-value pairs from the supplied dict.
add()	Generic add().
replace()	Generic replace().
prepend(key, value [, expiry])	Prepends the supplied value to the value of the existing key.

Python memcache Function	Equivalent Generic Function
append(key, value [, expiry[)	Appends the supplied value to the value of the existing key.
delete()	Generic delete().
delete_multi(keys [, expiry [, key_prefix]])	Deletes all the keys from the hash matching each string in the array keys.
incr()	Generic incr().
decr()	Generic decr().

Note

Within the Python memcache module, all the *_multi()functions support an optional key_prefix parameter. If supplied, then the string is used as a prefix to all key lookups. For example, if you call:

```
memc.get_multi(['a','b'], key_prefix='users:')
```

The function retrieves the keys users:a and users:b from the servers.

Here is an example showing the storage and retrieval of information to a memcache instance, loading the raw data from MySQL:

```
import sys
import MySQLdb
import memcache

memc = memcache.Client(['127.0.0.1:11211'], debug=1);

try:
    conn = MySQLdb.connect (host = "localhost",
                            user = "sakila",
                            passwd = "password",
                            db = "sakila")
except MySQLdb.Error, e:
     print "Error %d: %s" % (e.args[0], e.args[1])
     sys.exit (1)

popularfilms = memc.get('top5films')

if not popularfilms:
    cursor = conn.cursor()
    cursor.execute('select film_id,title from film order by rental_rate desc limit 5')
    rows = cursor.fetchall()
    memc.set('top5films',rows,60)
    print "Updated memcached with MySQL data"
else:
    print "Loaded data from memcached"
    for row in popularfilms:
        print "%s, %s" % (row[0], row[1])
```

When executed for the first time, the data is loaded from the MySQL database and stored to the memcached server.

```
shell> python memc_python.py
Updated memcached with MySQL data
```

Because the data is automatically serialized using cPickle/pickle, when you load the data back from memcached, you can use the object directly. In the example above, the information stored to memcached is in the form of rows from a Python DB cursor. When accessing the information (within the 60 second expiry time), the data is loaded from memcached and dumped:

```
shell> python memc_python.py
Loaded data from memcached
2, ACE GOLDFINGER
```

```
7, AIRPLANE SIERRA
8, AIRPORT POLLOCK
10, ALADDIN CALENDAR
13, ALI FOREVER
```

The serialization and deserialization happens automatically. Because serialization of Python data may be incompatible with other interfaces and languages, you can change the serialization module used during initialization. For example, you might use JSON format when you store complex data structures using a script written in one language, and access them in a script written in a different language.

16.2.3.6 Using MySQL and memcached with PHP

PHP provides support for the Memcache functions through a PECL extension. To enable the PHP memcache extensions, build PHP using the --enable-memcache option to configure when building from source.

If you are installing on a Red Hat-based server, you can install the php-pecl-memcache RPM:

```
root-shell> yum --install php-pecl-memcache
```

On Debian-based distributions, use the php-memcache package.

To set global runtime configuration options, specify the configuration option values within your php.ini file. The following table provides the name, default value, and a description for each global runtime configuration option.

Configuration option	Default	Description
memcache.allow_failover	1	Specifies whether another server in the list should be queried if the first server selected fails.
memcache.max_failover_attempts	20	Specifies the number of servers to try before returning a failure.
memcache.chunk_size	8192	Defines the size of network chunks used to exchange data with the memcached server.
memcache.default_port	11211	Defines the default port to use when communicating with the memcached servers.
memcache.hash_strategy	standard	Specifies which hash strategy to use. Set to consistent to enable servers to be added or removed from the pool without causing the keys to be remapped to other servers. When set to standard, an older (modula) strategy is used that potentially uses different servers for storage.
memcache.hash_function	crc32	Specifies which function to use when mapping keys to servers. crc32 uses the standard CRC32 hash. fnv uses the FNV-1a hashing algorithm.

To create a connection to a memcached server, create a new Memcache object and then specify the connection options. For example:

```
<?php

$cache = new Memcache;
$cache->connect('localhost',11211);
?>
```

This opens an immediate connection to the specified server.

To use multiple memcached servers, you need to add servers to the memcache object using addServer():

```
bool Memcache::addServer ( string $host [, int $port [, bool $persistent
                [, int $weight [, int $timeout [, int $retry_interval
                [, bool $status [, callback $failure_callback
                ]]]]]] )
```

The server management mechanism within the php-memcache module is a critical part of the interface as it controls the main interface to the memcached instances and how the different instances are selected through the hashing mechanism.

To create a simple connection to two memcached instances:

```
<?php

$cache = new Memcache;
$cache->addServer('198.51.100.100',11211);
$cache->addServer('198.51.100.101',11211);
?>
```

In this scenario, the instance connection is not explicitly opened, but only opened when you try to store or retrieve a value. To enable persistent connections to memcached instances, set the $persistent argument to true. This is the default setting, and causes the connections to remain open.

To help control the distribution of keys to different instances, use the global memcache.hash_strategy setting. This sets the hashing mechanism used to select. You can also add another weight to each server, which effectively increases the number of times the instance entry appears in the instance list, therefore increasing the likelihood of the instance being chosen over other instances. To set the weight, set the value of the $weight argument to more than one.

The functions for setting and retrieving information are identical to the generic functional interface offered by memcached, as shown in the following table.

PECL memcache Function	Generic Function
get()	Generic get().
set()	Generic set().
add()	Generic add().
replace()	Generic replace().
delete()	Generic delete().
increment()	Generic incr().
decrement()	Generic decr().

A full example of the PECL memcache interface is provided below. The code loads film data from the Sakila database when the user provides a film name. The data stored into the memcached instance is recorded as a mysqli result row, and the API automatically serializes the information for you.

```
<?php

$memc = new Memcache;
$memc->addServer('localhost','11211');

if(empty($_POST['film'])) {
?>
  <html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
    <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
      <title>Simple Memcache Lookup</title>
    </head>
    <body>
      <form method="post">
        <p><b>Film</b>: <input type="text" size="20" name="film"></p>
```

```
        <input type="submit">
      </form>
      <hr/>
<?php

} else {

    echo "Loading data...\n";

    $film   = htmlspecialchars($_POST['film'], ENT_QUOTES, 'UTF-8');
    $mfilms = $memc->get($film);

    if ($mfilms) {

        printf("<p>Film data for %s loaded from memcache</p>", $mfilms['title']);

        foreach (array_keys($mfilms) as $key) {
            printf("<p><b>%s</b>: %s</p>", $key, $mfilms[$key]);
        }

    } else {

        $mysqli = mysqli('localhost','sakila','<replaceable>password</replaceable>','sakila');

        if (mysqli_connect_error()) {
            sprintf("Database error: (%d) %s", mysqli_connect_errno(), mysqli_connect_error());
            exit;
        }

        $sql = sprintf('SELECT * FROM film WHERE title="%s"', $mysqli->real_escape_string($film));

        $result = $mysqli->query($sql);

        if (!$result) {
            sprintf("Database error: (%d) %s", $mysqli->errno, $mysqli->error);
            exit;
        }

        $row = $result->fetch_assoc();

        $memc->set($row['title'], $row);

        printf("<p>Loaded (%s) from MySQL</p>", htmlspecialchars($row['title'], ENT_QUOTES, 'UTF-8');
    }
}
?>
  </body>
</html>
```

With PHP, the connections to the memcached instances are kept open as long as the PHP and associated Apache instance remain running. When adding or removing servers from the list in a running instance (for example, when starting another script that mentions additional servers), the connections are shared, but the script only selects among the instances explicitly configured within the script.

To ensure that changes to the server list within a script do not cause problems, make sure to use the consistent hashing mechanism.

16.2.3.7 Using MySQL and memcached with Ruby

There are a number of different modules for interfacing to memcached within Ruby. The Ruby-MemCache client library provides a native interface to memcached that does not require any external libraries, such as libmemcached. You can obtain the installer package from http://www.deveiate.org/projects/RMemCache.

To install, extract the package and then run install.rb:

```
shell> install.rb
```

If you have RubyGems, you can install the Ruby-MemCache gem:

```
shell> gem install Ruby-MemCache
Bulk updating Gem source index for: http://gems.rubyforge.org
Install required dependency io-reactor? [Yn]  y
Successfully installed Ruby-MemCache-0.0.1
Successfully installed io-reactor-0.05
Installing ri documentation for io-reactor-0.05...
Installing RDoc documentation for io-reactor-0.05...
```

To use a memcached instance from within Ruby, create a new instance of the MemCache object.

```
require 'memcache'
memc = MemCache::new '198.51.100.100:11211'
```

You can add a weight to each server to increase the likelihood of the server being selected during hashing by appending the weight count to the server host name/port string:

```
require 'memcache'
memc = MemCache::new '198.51.100.100:11211:3'
```

To add servers to an existing list, you can append them directly to the MemCache object:

```
memc += ["198.51.100.101:11211"]
```

To set data into the cache, you can just assign a value to a key within the new cache object, which works just like a standard Ruby hash object:

```
memc["key"] = "value"
```

Or to retrieve the value:

```
print memc["key"]
```

For more explicit actions, you can use the method interface, which mimics the main memcached API functions, as summarized in the following table.

Ruby MemCache Method	Equivalent memcached API Functions
get()	Generic get().
get_hash(keys)	Get the values of multiple keys, returning the information as a hash of the keys and their values.
set()	Generic set().
set_many(pairs)	Set the values of the keys and values in the hash pairs.
add()	Generic add().
replace()	Generic replace().
delete()	Generic delete().
incr()	Generic incr().
decr()	Generic decr().

16.2.3.8 Using MySQL and memcached with Java

The com.danga.MemCached class within Java provides a native interface to memcached instances. You can obtain the client from https://github.com/gwhalin/Memcached-Java-Client/downloads. The Java class uses hashes that are compatible with libmemcached, so you can mix and match Java and libmemcached applications accessing the same memcached instances. The serialization between Java and other interfaces are not compatible. If this is a problem, use JSON or a similar nonbinary serialization format.

On most systems, you can download the package and use the jar directly.

To use the com.danga.MemCached interface, you create a MemCachedClient instance and then configure the list of servers by configuring the SockIOPool. Through the pool specification you set up the server list, weighting, and the connection parameters to optimized the connections between your client and the memcached instances that you configure.

Generally, you can configure the memcached interface once within a single class, then use this interface throughout the rest of your application.

For example, to create a basic interface, first configure the MemCachedClient and base SockIOPool settings:

```
public class MyClass {

    protected static MemCachedClient mcc = new MemCachedClient();

    static {

        String[] servers =
            {
                "localhost:11211",
            };

        Integer[] weights = { 1 };

        SockIOPool pool = SockIOPool.getInstance();

        pool.setServers( servers );
        pool.setWeights( weights );
```

In the above sample, the list of servers is configured by creating an array of the memcached instances to use. You can then configure individual weights for each server.

The remainder of the properties for the connection are optional, but you can set the connection numbers (initial connections, minimum connections, maximum connections, and the idle timeout) by setting the pool parameters:

```
pool.setInitConn( 5 );
pool.setMinConn( 5 );
pool.setMaxConn( 250 );
pool.setMaxIdle( 1000 * 60 * 60 * 6
```

Once the parameters have been configured, initialize the connection pool:

```
pool.initialize();
```

The pool, and the connection to your memcached instances should now be ready to use.

To set the hashing algorithm used to select the server used when storing a given key, use pool.setHashingAlg():

```
pool.setHashingAlg( SockIOPool.NEW_COMPAT_HASH );
```

Valid values are NEW_COMPAT_HASH, OLD_COMPAT_HASH and NATIVE_HASH are also basic modula hashing algorithms. For a consistent hashing algorithm, use CONSISTENT_HASH. These constants are equivalent to the corresponding hash settings within libmemcached.

The following table outlines the Java com.danga.MemCached methods and the equivalent generic methods in the memcached interface specification.

Java com.danga.MemCached Method	Equivalent Generic Method
get()	Generic get().
getMulti(keys)	Get the values of multiple keys, returning the information as Hash map using java.lang.String for the keys and

Java com.danga.MemCached Method	Equivalent Generic Method
	java.lang.Object for the corresponding values.
set()	Generic set().
add()	Generic add().
replace()	Generic replace().
delete()	Generic delete().
incr()	Generic incr().
decr()	Generic decr().

16.2.3.9 Using the memcached TCP Text Protocol

Communicating with a memcached server can be achieved through either the TCP or UDP protocols. When using the TCP protocol, you can use a simple text based interface for the exchange of information.

When communicating with memcached, you can connect to the server using the port configured for the server. You can open a connection with the server without requiring authorization or login. As soon as you have connected, you can start to send commands to the server. When you have finished, you can terminate the connection without sending any specific disconnection command. Clients are encouraged to keep their connections open to decrease latency and improve performance.

Data is sent to the memcached server in two forms:

- Text lines, which are used to send commands to the server, and receive responses from the server.
- Unstructured data, which is used to receive or send the value information for a given key. Data is returned to the client in exactly the format it was provided.

Both text lines (commands and responses) and unstructured data are always terminated with the string \r\n. Because the data being stored may contain this sequence, the length of the data (returned by the client before the unstructured data is transmitted should be used to determine the end of the data.

Commands to the server are structured according to their operation:

- **Storage commands**: set, add, replace, append, prepend, cas

  Storage commands to the server take the form:

  ```
  command key [flags] [exptime] length [noreply]
  ```

  Or when using compare and swap (cas):

  ```
  cas key [flags] [exptime] length [casunique] [noreply]
  ```

  Where:

  - command: The command name.
    - set: Store value against key
    - add: Store this value against key if the key does not already exist
    - replace: Store this value against key if the key already exists
    - append: Append the supplied value to the end of the value for the specified key. The flags and exptime arguments should not be used.
    - prepend: Append value currently in the cache to the end of the supplied value for the specified key. The flags and exptime arguments should not be used.

    - cas: Set the specified key to the supplied value, only if the supplied casunique matches. This is effectively the equivalent of change the information if nobody has updated it since I last fetched it.

  - key: The key. All data is stored using a the specific key. The key cannot contain control characters or whitespace, and can be up to 250 characters in size.
  - flags: The flags for the operation (as an integer). Flags in memcached are transparent. The memcached server ignores the contents of the flags. They can be used by the client to indicate any type of information. In memcached 1.2.0 and lower the value is a 16-bit integer value. In memcached 1.2.1 and higher the value is a 32-bit integer.
  - exptime: The expiry time, or zero for no expiry.
  - length: The length of the supplied value block in bytes, excluding the terminating \r\n characters.
  - casunique: A unique 64-bit value of an existing entry. This is used to compare against the existing value. Use the value returned by the gets command when issuing cas updates.
  - noreply: Tells the server not to reply to the command.

  For example, to store the value abcdef into the key xyzkey, you would use:

```
set xyzkey 0 0 6\r\nabcdef\r\n
```

  The return value from the server is one line, specifying the status or error information. For more information, see Table 16.3, “memcached Protocol Responses”.

- **Retrieval commands**: get, gets

  Retrieval commands take the form:

```
get key1 [key2 .... keyn]
gets key1 [key2 ... keyn]
```

  You can supply multiple keys to the commands, with each requested key separated by whitespace.

  The server responds with an information line of the form:

```
VALUE key flags bytes [casunique]
```

  Where:

  - key: The key name.
  - flags: The value of the flag integer supplied to the memcached server when the value was stored.
  - bytes: The size (excluding the terminating \r\n character sequence) of the stored value.
  - casunique: The unique 64-bit integer that identifies the item.

  The information line is immediately followed by the value data block. For example:

```
get xyzkey\r\n
VALUE xyzkey 0 6\r\n
abcdef\r\n
```

  If you have requested multiple keys, an information line and data block is returned for each key found. If a requested key does not exist in the cache, no information is returned.

- **Delete commands**: delete

Deletion commands take the form:

```
delete key [time] [noreply]
```

Where:

- key: The key name.
- time: The time in seconds (or a specific Unix time) for which the client wishes the server to refuse add or replace commands on this key. All add, replace, get, and gets commands fail during this period. set operations succeed. After this period, the key is deleted permanently and all commands are accepted.

  If not supplied, the value is assumed to be zero (delete immediately).
- noreply: Tells the server not to reply to the command.

Responses to the command are either DELETED to indicate that the key was successfully removed, or NOT_FOUND to indicate that the specified key could not be found.

- **Increment/Decrement**: incr, decr

  The increment and decrement commands change the value of a key within the server without performing a separate get/set sequence. The operations assume that the currently stored value is a 64-bit integer. If the stored value is not a 64-bit integer, then the value is assumed to be zero before the increment or decrement operation is applied.

  Increment and decrement commands take the form:

  ```
  incr key value [noreply]
  decr key value [noreply]
  ```

  Where:

  - key: The key name.
  - value: An integer to be used as the increment or decrement value.
  - noreply: Tells the server not to reply to the command.

  The response is:

  - NOT_FOUND: The specified key could not be located.
  - value: The new value associated with the specified key.

  Values are assumed to be unsigned. For decr operations, the value is never decremented below 0. For incr operations, the value wraps around the 64-bit maximum.

- **Statistics commands**: `stats`

  The `stats` command provides detailed statistical information about the current status of the `memcached` instance and the data it is storing.

  Statistics commands take the form:

  ```
  STAT [name] [value]
  ```

  Where:

  - `name`: The optional name of the statistics to return. If not specified, the general statistics are returned.
  - `value`: A specific value to be used when performing certain statistics operations.

  The return value is a list of statistics data, formatted as follows:

  ```
  STAT name value
  ```

  The statistics are terminated with a single line, `END`.

  For more information, see Section 16.2.4, "Getting memcached Statistics".

For reference, a list of the different commands supported and their formats is provided below.

Table 16.2 memcached Command Reference

Command	Command Formats
`set`	`set key flags exptime length`, `set key flags exptime length noreply`
`add`	`add key flags exptime length`, `add key flags exptime length noreply`
`replace`	`replace key flags exptime length`, `replace key flags exptime length noreply`
`append`	`append key length`, `append key length noreply`
`prepend`	`prepend key length`, `prepend key length noreply`
`cas`	`cas key flags exptime length casunique`, `cas key flags exptime length casunique noreply`
`get`	`get key1 [key2 ... keyn]`
`gets`	
`delete`	`delete key`, `delete key noreply`, `delete key expiry`, `delete key expiry noreply`
`incr`	`incr key`, `incr key noreply`, `incr key value`, `incr key value noreply`
`decr`	`decr key`, `decr key noreply`, `decr key value`, `decr key value noreply`
`stat`	`stat`, `stat name`, `stat name value`

When sending a command to the server, the response from the server is one of the settings in the following table. All response values from the server are terminated by `\r\n`:

Table 16.3 memcached Protocol Responses

String	Description
`STORED`	Value has successfully been stored.

String	Description
NOT_STORED	The value was not stored, but not because of an error. For commands where you are adding a or updating a value if it exists (such as add and replace), or where the item has already been set to be deleted.
EXISTS	When using a cas command, the item you are trying to store already exists and has been modified since you last checked it.
NOT_FOUND	The item you are trying to store, update or delete does not exist or has already been deleted.
ERROR	You submitted a nonexistent command name.
CLIENT_ERROR errorstring	There was an error in the input line, the detail is contained in errorstring.
SERVER_ERROR errorstring	There was an error in the server that prevents it from returning the information. In extreme conditions, the server may disconnect the client after this error occurs.
VALUE keys flags length	The requested key has been found, and the stored key, flags and data block are returned, of the specified length.
DELETED	The requested key was deleted from the server.
STAT name value	A line of statistics data.
END	The end of the statistics data.

16.2.4 Getting memcached Statistics

The memcached system has a built-in statistics system that collects information about the data being stored into the cache, cache hit ratios, and detailed information on the memory usage and distribution of information through the slab allocation used to store individual items. Statistics are provided at both a basic level that provide the core statistics, and more specific statistics for specific areas of the memcached server.

This information can be useful to ensure that you are getting the correct level of cache and memory usage, and that your slab allocation and configuration properties are set at an optimal level.

The stats interface is available through the standard memcached protocol, so the reports can be accessed by using telnet to connect to the memcached. The supplied memcached-tool includes support for obtaining the Section 16.2.4.2, “memcached Slabs Statistics” and Section 16.2.4.1, “memcached General Statistics” information. For more information, see Section 16.2.4.6, “Using memcached-tool”.

Alternatively, most of the language API interfaces provide a function for obtaining the statistics from the server.

For example, to get the basic stats using telnet:

```
shell> telnet localhost 11211
Trying ::1...
Connected to localhost.
Escape character is '^]'.
stats
STAT pid 23599
STAT uptime 675
STAT time 1211439587
STAT version 1.2.5
STAT pointer_size 32
STAT rusage_user 1.404992
STAT rusage_system 4.694685
STAT curr_items 32
STAT total_items 56361
STAT bytes 2642
STAT curr_connections 53
STAT total_connections 438
```

```
STAT connection_structures 55
STAT cmd_get 113482
STAT cmd_set 80519
STAT get_hits 78926
STAT get_misses 34556
STAT evictions 0
STAT bytes_read 6379783
STAT bytes_written 4860179
STAT limit_maxbytes 67108864
STAT threads 1
END
```

When using Perl and the Cache::Memcached module, the stats() function returns information about all the servers currently configured in the connection object, and total statistics for all the memcached servers as a whole.

For example, the following Perl script obtains the stats and dumps the hash reference that is returned:

```
use Cache::Memcached;
use Data::Dumper;

my $memc = new Cache::Memcached;
$memc->set_servers(\@ARGV);

print Dumper($memc->stats());
```

When executed on the same memcached as used in the Telnet example above we get a hash reference with the host by host and total statistics:

```
$VAR1 = {
    'hosts' => {
           'localhost:11211' => {
                      'misc' => {
                            'bytes' => '2421',
                            'curr_connections' => '3',
                            'connection_structures' => '56',
                            'pointer_size' => '32',
                            'time' => '1211440166',
                            'total_items' => '410956',
                            'cmd_set' => '588167',
                            'bytes_written' => '35715151',
                            'evictions' => '0',
                            'curr_items' => '31',
                            'pid' => '23599',
                            'limit_maxbytes' => '67108864',
                            'uptime' => '1254',
                            'rusage_user' => '9.857805',
                            'cmd_get' => '838451',
                            'rusage_system' => '34.096988',
                            'version' => '1.2.5',
                            'get_hits' => '581511',
                            'bytes_read' => '46665716',
                            'threads' => '1',
                            'total_connections' => '3104',
                            'get_misses' => '256940'
                          },
                      'sizes' => {
                             '128' => '16',
                             '64' => '15'
                           }
                    }
         },
    'self' => {},
    'total' => {
           'cmd_get' => 838451,
           'bytes' => 2421,
           'get_hits' => 581511,
           'connection_structures' => 56,
           'bytes_read' => 46665716,
           'total_items' => 410956,
           'total_connections' => 3104,
```

```
          'cmd_set' => 588167,
          'bytes_written' => 35715151,
          'curr_items' => 31,
          'get_misses' => 256940
        }
      };
```

The statistics are divided up into a number of distinct sections, and then can be requested by adding the type to the `stats` command. Each statistics output is covered in more detail in the following sections.

- General statistics, see Section 16.2.4.1, "memcached General Statistics".
- Slab statistics (`slabs`), see Section 16.2.4.2, "memcached Slabs Statistics".
- Item statistics (`items`), see Section 16.2.4.3, "memcached Item Statistics".
- Size statistics (`sizes`), see Section 16.2.4.4, "memcached Size Statistics".
- Detailed status (`detail`), see Section 16.2.4.5, "memcached Detail Statistics".

16.2.4.1 memcached General Statistics

The output of the general statistics provides an overview of the performance and use of the `memcached` instance. The statistics returned by the command and their meaning is shown in the following table.

The following terms are used to define the value type for each statistics value:

- `32u`: 32-bit unsigned integer
- `64u`: 64-bit unsigned integer
- `32u:32u`: Two 32-bit unsigned integers separated by a colon
- `String`: Character string

Statistic	Data type	Description	Version
pid	32u	Process ID of the memcached instance.	
uptime	32u	Uptime (in seconds) for this memcached instance.	
time	32u	Current time (as epoch).	
version	string	Version string of this instance.	
pointer_size	string	Size of pointers for this host specified in bits (32 or 64).	
rusage_user	32u:32u	Total user time for this instance (seconds:microseconds).	
rusage_system	32u:32u	Total system time for this instance (seconds:microseconds).	
curr_items	32u	Current number of items stored by this instance.	
total_items	32u	Total number of items stored during the life of this instance.	
bytes	64u	Current number of bytes used by this server to store items.	
curr_connections	32u	Current number of open connections.	
total_connections	32u	Total number of connections opened since the server started running.	
connection_structures	32u	Number of connection structures allocated by the server.	

Statistic	Data type	Description	Version
cmd_get	64u	Total number of retrieval requests (get operations).	
cmd_set	64u	Total number of storage requests (set operations).	
get_hits	64u	Number of keys that have been requested and found present.	
get_misses	64u	Number of items that have been requested and not found.	
delete_hits	64u	Number of keys that have been deleted and found present.	1.3.x
delete_misses	64u	Number of items that have been delete and not found.	1.3.x
incr_hits	64u	Number of keys that have been incremented and found present.	1.3.x
incr_misses	64u	Number of items that have been incremented and not found.	1.3.x
decr_hits	64u	Number of keys that have been decremented and found present.	1.3.x
decr_misses	64u	Number of items that have been decremented and not found.	1.3.x
cas_hits	64u	Number of keys that have been compared and swapped and found present.	1.3.x
cas_misses	64u	Number of items that have been compared and swapped and not found.	1.3.x
cas_badvalue	64u	Number of keys that have been compared and swapped, but the comparison (original) value did not match the supplied value.	1.3.x
evictions	64u	Number of valid items removed from cache to free memory for new items.	
bytes_read	64u	Total number of bytes read by this server from network.	
bytes_written	64u	Total number of bytes sent by this server to network.	
limit_maxbytes	32u	Number of bytes this server is permitted to use for storage.	
threads	32u	Number of worker threads requested.	
conn_yields	64u	Number of yields for connections (related to the -R option).	1.4.0

The most useful statistics from those given here are the number of cache hits, misses, and evictions.

A large number of get_misses may just be an indication that the cache is still being populated with information. The number should, over time, decrease in comparison to the number of cache get_hits. If, however, you have a large number of cache misses compared to cache hits after an extended period of execution, it may be an indication that the size of the cache is too small and you either need to increase the total memory size, or increase the number of the memcached instances to improve the hit ratio.

A large number of evictions from the cache, particularly in comparison to the number of items stored is a sign that your cache is too small to hold the amount of information that you regularly want to keep cached. Instead of items being retained in the cache, items are being evicted to make way for new items keeping the turnover of items in the cache high, reducing the efficiency of the cache.

16.2.4.2 memcached Slabs Statistics

To get the slabs statistics, use the stats slabs command, or the API equivalent.

The slab statistics provide you with information about the slabs that have created and allocated for storing information within the cache. You get information both on each individual slab-class and total statistics for the whole slab.

```
STAT 1:chunk_size 104
STAT 1:chunks_per_page 10082
STAT 1:total_pages 1
STAT 1:total_chunks 10082
STAT 1:used_chunks 10081
STAT 1:free_chunks 1
STAT 1:free_chunks_end 10079
STAT 9:chunk_size 696
STAT 9:chunks_per_page 1506
STAT 9:total_pages 63
STAT 9:total_chunks 94878
STAT 9:used_chunks 94878
STAT 9:free_chunks 0
STAT 9:free_chunks_end 0
STAT active_slabs 2
STAT total_malloced 67083616
END
```

Individual stats for each slab class are prefixed with the slab ID. A unique ID is given to each allocated slab from the smallest size up to the largest. The prefix number indicates the slab class number in relation to the calculated chunk from the specified growth factor. Hence in the example, 1 is the first chunk size and 9 is the 9th chunk allocated size.

The parameters returned for each chunk size and a description of each parameter are provided in the following table.

Statistic	Description	Version
chunk_size	Space allocated to each chunk within this slab class.	
chunks_per_page	Number of chunks within a single page for this slab class.	
total_pages	Number of pages allocated to this slab class.	
total_chunks	Number of chunks allocated to the slab class.	
used_chunks	Number of chunks allocated to an item..	
free_chunks	Number of chunks not yet allocated to items.	
free_chunks_end	Number of free chunks at the end of the last allocated page.	
get_hits	Number of get hits to this chunk	1.3.x
cmd_set	Number of set commands on this chunk	1.3.x
delete_hits	Number of delete hits to this chunk	1.3.x
incr_hits	Number of increment hits to this chunk	1.3.x
decr_hits	Number of decrement hits to this chunk	1.3.x
cas_hits	Number of CAS hits to this chunk	1.3.x
cas_badval	Number of CAS hits on this chunk where the existing value did not match	1.3.x
mem_requested	The true amount of memory of memory requested within this chunk	1.4.1

The following additional statistics cover the information for the entire server, rather than on a chunk by chunk basis:

Statistic	Description	Version
active_slabs	Total number of slab classes allocated.	

Statistic	Description	Version
total_malloced	Total amount of memory allocated to slab pages.	

The key values in the slab statistics are the chunk_size, and the corresponding total_chunks and used_chunks parameters. These given an indication of the size usage of the chunks within the system. Remember that one key-value pair is placed into a chunk of a suitable size.

From these stats, you can get an idea of your size and chunk allocation and distribution. If you store many items with a number of largely different sizes, consider adjusting the chunk size growth factor to increase in larger steps to prevent chunk and memory wastage. A good indication of a bad growth factor is a high number of different slab classes, but with relatively few chunks actually in use within each slab. Increasing the growth factor creates fewer slab classes and therefore makes better use of the allocated pages.

16.2.4.3 memcached Item Statistics

To get the items statistics, use the stats items command, or the API equivalent.

The items statistics give information about the individual items allocated within a given slab class.

```
STAT items:2:number 1
STAT items:2:age 452
STAT items:2:evicted 0
STAT items:2:evicted_nonzero 0
STAT items:2:evicted_time 2
STAT items:2:outofmemory 0
STAT items:2:tailrepairs 0
...
STAT items:27:number 1
STAT items:27:age 452
STAT items:27:evicted 0
STAT items:27:evicted_nonzero 0
STAT items:27:evicted_time 2
STAT items:27:outofmemory 0
STAT items:27:tailrepairs 0
```

The prefix number against each statistics relates to the corresponding chunk size, as returned by the stats slabs statistics. The result is a display of the number of items stored within each chunk within each slab size, and specific statistics about their age, eviction counts, and out of memory counts. A summary of the statistics is given in the following table.

Statistic	Description	Version
number	The number of items currently stored in this slab class.	
age	The age of the oldest item within the slab class, in seconds.	
evicted	The number of items evicted to make way for new entries.	
evicted_time	The time of the last evicted entry	
evicted_nonzero	The time of the last evicted non-zero entry	1.4.0
outofmemory	The number of items for this slab class that have triggered an out of memory error (only value when the -M command line option is in effect).	
tailrepairs	Number of times the entries for a particular ID need repairing	

Item level statistics can be used to determine how many items are stored within a given slab and their freshness and recycle rate. You can use this to help identify whether there are certain slab classes that are triggering a much larger number of evictions that others.

16.2.4.4 memcached Size Statistics

To get size statistics, use the stats sizes command, or the API equivalent.

The size statistics provide information about the sizes and number of items of each size within the cache. The information is returned as two columns, the first column is the size of the item (rounded up to the nearest 32 byte boundary), and the second column is the count of the number of items of that size within the cache:

```
96 35
128 38
160 807
192 804
224 410
256 222
288 83
320 39
352 53
384 33
416 64
448 51
480 30
512 54
544 39
576 10065
```

Caution

Running this statistic locks up your cache as each item is read from the cache and its size calculated. On a large cache, this may take some time and prevent any set or get operations until the process completes.

The item size statistics are useful only to determine the sizes of the objects you are storing. Since the actual memory allocation is relevant only in terms of the chunk size and page size, the information is only useful during a careful debugging or diagnostic session.

16.2.4.5 memcached Detail Statistics

For memcached 1.3.x and higher, you can enable and obtain detailed statistics about the get, set, and del operations on theindividual keys stored in the cache, and determine whether the attempts hit (found) a particular key. These operations are only recorded while the detailed stats analysis is turned on.

To enable detailed statistics, you must send the `stats detail on` command to the memcached server:

```
$ telnet localhost 11211
Trying 127.0.0.1...
Connected to tiger.
Escape character is '^]'.
stats detail on
OK
```

Individual statistics are recorded for every `get`, `set` and `del` operation on a key, including keys that are not currently stored in the server. For example, if an attempt is made to obtain the value of key `abckey` and it does not exist, the `get` operating on the specified key are recorded while detailed statistics are in effect, even if the key is not currently stored. The `hits`, that is, the number of `get` or `del` operations for a key that exists in the server are also counted.

To turn detailed statistics off, send the `stats detail off` command to the memcached server:

```
$ telnet localhost 11211
Trying 127.0.0.1...
Connected to tiger.
Escape character is '^]'.
stats detail off
OK
```

To obtain the detailed statistics recorded during the process, send the `stats detail dump` command to the memcached server:

```
stats detail dump
PREFIX hykkey get 0 hit 0 set 1 del 0
PREFIX xyzkey get 0 hit 0 set 1 del 0
PREFIX yukkey get 1 hit 0 set 0 del 0
PREFIX abckey get 3 hit 3 set 1 del 0
END
```

You can use the detailed statistics information to determine whether your memcached clients are using a large number of keys that do not exist in the server by comparing the hit and get or del counts. Because the information is recorded by key, you can also determine whether the failures or operations are clustered around specific keys.

16.2.4.6 Using memcached-tool

The memcached-tool, located within the scripts directory within the memcached source directory. The tool provides convenient access to some reports and statistics from any memcached instance.

The basic format of the command is:

```
shell> ./memcached-tool hostname:port [command]
```

The default output produces a list of the slab allocations and usage. For example:

```
shell> memcached-tool localhost:11211 display
  #  Item_Size  Max_age   Pages   Count   Full?  Evicted Evict_Time OOM
  1      80B        93s       1      20      no        0        0    0
  2     104B        93s       1      16      no        0        0    0
  3     136B      1335s       1      28      no        0        0    0
  4     176B      1335s       1      24      no        0        0    0
  5     224B      1335s       1      32      no        0        0    0
  6     280B      1335s       1      34      no        0        0    0
  7     352B      1335s       1      36      no        0        0    0
  8     440B      1335s       1      46      no        0        0    0
  9     552B      1335s       1      58      no        0        0    0
 10     696B      1335s       1      66      no        0        0    0
 11     872B      1335s       1      89      no        0        0    0
 12     1.1K      1335s       1     112      no        0        0    0
 13     1.3K      1335s       1     145      no        0        0    0
 14     1.7K      1335s       1     123      no        0        0    0
 15     2.1K      1335s       1     198      no        0        0    0
 16     2.6K      1335s       1     199      no        0        0    0
 17     3.3K      1335s       1     229      no        0        0    0
 18     4.1K      1335s       1     248     yes       36        2    0
 19     5.2K      1335s       2     328      no        0        0    0
 20     6.4K      1335s       2     316     yes      387        1    0
 21     8.1K      1335s       3     381     yes      492        1    0
 22    10.1K      1335s       3     303     yes      598        2    0
 23    12.6K      1335s       5     405     yes      605        1    0
 24    15.8K      1335s       6     384     yes      766        2    0
 25    19.7K      1335s       7     357     yes      908      170    0
 26    24.6K      1336s       7     287     yes     1012        1    0
 27    30.8K      1336s       7     231     yes     1193      169    0
 28    38.5K      1336s       4     104     yes     1323      169    0
 29    48.1K      1336s       1      21     yes     1287        1    0
 30    60.2K      1336s       1      17     yes     1093      169    0
 31    75.2K      1337s       1      13     yes      713      168    0
 32    94.0K      1337s       1      10     yes      278      168    0
 33   117.5K      1336s       1       3      no        0        0    0
```

This output is the same if you specify the command as display:

```
shell> memcached-tool localhost:11211 display
  #  Item_Size  Max_age   Pages   Count   Full?  Evicted Evict_Time OOM
  1      80B        93s       1      20      no        0        0    0
  2     104B        93s       1      16      no        0        0    0
...
```

The output shows a summarized version of the output from the slabs statistics. The columns provided in the output are shown below:

- `#`: The slab number
- `Item_Size`: The size of the slab
- `Max_age`: The age of the oldest item in the slab
- `Pages`: The number of pages allocated to the slab
- `Count`: The number of items in this slab
- `Full?`: Whether the slab is fully populated
- `Evicted`: The number of objects evicted from this slab
- `Evict_Time`: The time (in seconds) since the last eviction
- `OOM`: The number of items that have triggered an out of memory error

You can also obtain a dump of the general statistics for the server using the `stats` command:

```
shell> memcached-tool localhost:11211 stats
#localhost:11211   Field       Value
         accepting_conns           1
                   bytes         162
              bytes_read         485
           bytes_written        6820
              cas_badval           0
                cas_hits           0
              cas_misses           0
               cmd_flush           0
                 cmd_get           4
                 cmd_set           2
             conn_yields           0
   connection_structures          11
        curr_connections          10
              curr_items           2
               decr_hits           0
             decr_misses           1
             delete_hits           0
           delete_misses           0
               evictions           0
                get_hits           4
              get_misses           0
               incr_hits           0
             incr_misses           2
          limit_maxbytes    67108864
     listen_disabled_num           0
                     pid       12981
           pointer_size          32
          rusage_system    0.013911
            rusage_user    0.011876
                threads           4
                   time  1255518565
       total_connections          20
             total_items           2
                  uptime         880
                 version       1.4.2
```

16.2.5 `memcached` FAQ

16.2.5.1 Can memcached be run on a Windows environment? 2217
16.2.5.2 What is the maximum size of an object you can store in memcached? Is that configurable? 2217
16.2.5.3 Is it true that `memcached` is much more effective with db-read-intensive applications than with db-write-intensive applications? 2217
16.2.5.4 Is there any overhead in not using persistent connections? If persistent is always recommended, what are the downsides (for example, locking up)? 2217

16.2.5.5 How is an event such as a crash of one of the memcached servers handled by the memcached client? 2217
16.2.5.6 What is a recommended hardware configuration for a memcached server? 2218
16.2.5.7 Is memcached more effective for video and audio as opposed to textual read/writes? 2218
16.2.5.8 Can memcached work with ASPX? 2218
16.2.5.9 How expensive is it to establish a memcache connection? Should those connections be pooled? 2218
16.2.5.10 How is the data handled when the memcached server is down? 2218
16.2.5.11 How are auto-increment columns in the MySQL database coordinated across multiple instances of memcached? 2218
16.2.5.12 Is compression available? 2219
16.2.5.13 Can we implement different types of memcached as different nodes in the same server, so can there be deterministic and non-deterministic in the same server? 2219
16.2.5.14 What are best practices for testing an implementation, to ensure that it improves performance, and to measure the impact of memcached configuration changes? And would you recommend keeping the configuration very simple to start? 2219

16.2.5.1 Can memcached be run on a Windows environment?

No. Currently memcached is available only on the Unix/Linux platform. There is an unofficial port available, see http://www.codeplex.com/memcachedproviders.

16.2.5.2 What is the maximum size of an object you can store in memcached? Is that configurable?

The default maximum object size is 1MB. In memcached 1.4.2 and later, you can change the maximum size of an object using the `-I` command line option.

For versions before this, to increase this size, you have to re-compile memcached. You can modify the value of the `POWER_BLOCK` within the `slabs.c` file within the source.

In memcached 1.4.2 and higher, you can configure the maximum supported object size by using the `-I` command-line option. For example, to increase the maximum object size to 5MB:

```
$ memcached -I 5m
```

If an object is larger than the maximum object size, you must manually split it. memcached is very simple: you give it a key and some data, it tries to cache it in RAM. If you try to store more than the default maximum size, the value is just truncated for speed reasons.

16.2.5.3 Is it true that memcached is much more effective with db-read-intensive applications than with db-write-intensive applications?

Yes. memcached plays no role in database writes; it is a method of caching data already read from the database in RAM.

16.2.5.4 Is there any overhead in not using persistent connections? If persistent is always recommended, what are the downsides (for example, locking up)?

If you do not use persistent connections when communicating with memcached, there is a small increase in the latency of opening the connection each time. The effect is comparable to use nonpersistent connections with MySQL.

In general, the chance of locking or other issues with persistent connections is minimal, because there is very little locking within memcached. If there is a problem, eventually your request times out and returns no result, meaning that your application must load from MySQL again.

16.2.5.5 How is an event such as a crash of one of the memcached servers handled by the memcached client?

There is no automatic handling of this. If your client fails to get a response from a server, code a fallback mechanism to load the data from the MySQL database.

The client APIs all provide the ability to add and remove memcached instances on the fly. If within your application you notice that memcached server is no longer responding, you can remove the server from the list of servers, and keys are automatically redistributed to another memcached server in the list. If retaining the cache content on all your servers is important, make sure you use an API that supports a consistent hashing algorithm. For more information, see Section 16.2.2.5, “memcached Hashing/Distribution Types”.

16.2.5.6. What is a recommended hardware configuration for a memcached server?

memcached has a very low processing overhead. All that is required is spare physical RAM capacity. A memcached server does not require a dedicated machine. If you have web, application, or database servers that have spare RAM capacity, then use them with memcached.

To build and deploy a dedicated memcached server, use a relatively low-power CPU, lots of RAM, and one or more Gigabit Ethernet interfaces.

16.2.5.7. Is memcached more effective for video and audio as opposed to textual read/writes?

memcached works equally well for all kinds of data. To memcached, any value you store is just a stream of data. Remember, though, that the maximum size of an object you can store in memcached is 1MB, but can be configured to be larger by using the `-I` option in memcached 1.4.2 and later, or by modifying the source in versions before 1.4.2. If you plan on using memcached with audio and video content, you should to increase the maximum object size. Also remember that memcached is a solution for caching information for reading. It shouldn't be used for writes, except when updating the information in the cache.

16.2.5.8. Can memcached work with ASPX?

There are ports and interfaces for many languages and environments. ASPX relies on an underlying language such as C# or VisualBasic, and if you are using ASP.NET then there is a C# memcached library. For more information, see https://sourceforge.net/projects/memcacheddotnet/.

16.2.5.9. How expensive is it to establish a memcache connection? Should those connections be pooled?

Opening the connection is relatively inexpensive, because there is no security, authentication or other handshake taking place before you can start sending requests and getting results. Most APIs support a persistent connection to a memcached instance to reduce the latency. Connection pooling would depend on the API you are using, but if you are communicating directly over TCP/IP, then connection pooling would provide some small performance benefit.

16.2.5.10. How is the data handled when the memcached server is down?

The behavior is entirely application dependent. Most applications fall back to loading the data from the database (just as if they were updating the memcached information). If you are using multiple memcached servers, you might also remove a downed server from the list to prevent it from affecting performance. Otherwise, the client continues to attempt to communicate with the memcached server that corresponds to the key you are trying to load.

16.2.5.11. How are auto-increment columns in the MySQL database coordinated across multiple instances of memcached?

They aren't. There is no relationship between MySQL and memcached unless your application (or, if you are using the MySQL UDFs for memcached, your database definition) creates one.

If you are storing information based on an auto-increment key into multiple instances of memcached, the information is only stored on one of the memcached instances anyway. The client uses the key value to determine which memcached instance to store the information. It doesn't store the same information across all the instances, as that would be a waste of cache memory.

16.2.5.12. Is compression available?

Yes. Most of the client APIs support some sort of compression, and some even allow you to specify the threshold at which a value is deemed appropriate for compression during storage.

16.2.5.13. Can we implement different types of `memcached` as different nodes in the same server, so can there be deterministic and non-deterministic in the same server?

Yes. You can run multiple instances of `memcached` on a single server, and in your client configuration you choose the list of servers you want to use.

16.2.5.14. What are best practices for testing an implementation, to ensure that it improves performance, and to measure the impact of `memcached` configuration changes? And would you recommend keeping the configuration very simple to start?

The best way to test the performance is to start up a `memcached` instance. First, modify your application so that it stores the data just before the data is about to be used or displayed into `memcached`. Since the APIs handle the serialization of the data, it should just be a one-line modification to your code. Then, modify the start of the process that would normally load that information from MySQL with the code that requests the data from `memcached`. If the data cannot be loaded from `memcached`, default to the MySQL process.

All of the changes required should not require many lines of code. To get the most benefit, make sure you cache entire objects (for example, all the components of a web page, blog post, discussion thread, and so on), rather than using `memcached` as a simple cache of individual rows of MySQL tables.

Keeping the configuration simple at the start, or even over the long term, is easy with `memcached`. Once you have the basic structure up and running, often the only ongoing change is to add more servers into the list of servers used by your applications. You don't need to manage the `memcached` servers, and there is no complex configuration; just add more servers to the list and let the client API and the `memcached` servers make the decisions.

Chapter 17 Replication

Table of Contents

17.1 Replication Configuration 2222
17.1.1 How to Set Up Replication 2223
17.1.2 Replication Formats 2233
17.1.3 Replication with Global Transaction Identifiers 2240
17.1.4 Replication and Binary Logging Options and Variables 2249
17.1.5 Common Replication Administration Tasks 2310
17.2 Replication Implementation 2313
17.2.1 Replication Threads 2313
17.2.2 Relay Log and Replication Metadata Repositories 2316
17.2.3 How Servers Evaluate Replication Filtering Rules 2323
17.3 Replication Solutions 2328
17.3.1 Using Replication for Backups 2329
17.3.2 Handling an Unexpected Halt of a Replica Server 2332
17.3.3 Using Replication with Different Source and Replica Storage Engines 2334
17.3.4 Using Replication for Scale-Out 2335
17.3.5 Replicating Different Databases to Different Replicas 2337
17.3.6 Improving Replication Performance 2338
17.3.7 Switching Sources During Failover 2339
17.3.8 Setting Up Replication to Use Encrypted Connections 2341
17.3.9 Semisynchronous Replication 2343
17.3.10 Delayed Replication 2347
17.4 Replication Notes and Tips 2348
17.4.1 Replication Features and Issues 2348
17.4.2 Replication Compatibility Between MySQL Versions 2370
17.4.3 Upgrading a Replication Setup 2371
17.4.4 Troubleshooting Replication 2373
17.4.5 How to Report Replication Bugs or Problems 2374

Replication enables data from one MySQL database server (the source) to be replicated to one or more MySQL database servers (the replicas). Replication is asynchronous by default, therefore replicas do not need to be connected permanently to receive updates from the source. This means that updates can occur over long-distance connections and even over temporary or intermittent connections such as a dial-up service. Depending on the configuration, you can replicate all databases, selected databases, or even selected tables within a database.

For answers to some questions often asked by those who are new to MySQL Replication, see Section A.14, “MySQL 5.6 FAQ: Replication”.

Advantages of replication in MySQL include:

- Scale-out solutions - spreading the load among multiple replicas to improve performance. In this environment, all writes and updates must take place on the replication source server. Reads, however, may take place on one or more replicas. This model can improve the performance of writes (since the source is dedicated to updates), while dramatically increasing read speed across an increasing number of replicas.
- Data security - because data is replicated to the replica, and the replica can pause the replication process, it is possible to run backup services on the replica without corrupting the corresponding data on the source.
- Analytics - live data can be created on the source, while the analysis of the information can take place on the replica without affecting the performance of the source.

- Long-distance data distribution - if a branch office would like to work with a copy of your main data, you can use replication to create a local copy of the data for their use without requiring permanent access to the source.

Replication in MySQL features support for one-way, asynchronous replication, in which one server acts as the source, while one or more other servers act as replicas. This is in contrast to the *synchronous* replication which is a characteristic of NDB Cluster (see Chapter 18, *MySQL NDB Cluster 7.3 and NDB Cluster 7.4*). In MySQL 5.6, an interface to semisynchronous replication is supported in addition to the built-in asynchronous replication. With semisynchronous replication, a commit performed on the source blocks before returning to the session that performed the transaction until at least one replica acknowledges that it has received and logged the events for the transaction. See Section 17.3.9, “Semisynchronous Replication” MySQL 5.6 also supports delayed replication such that a replica server deliberately lags behind the source by at least a specified amount of time. See Section 17.3.10, “Delayed Replication”. For scenarios where *synchronous* replication is required, use NDB Cluster (see Chapter 18, *MySQL NDB Cluster 7.3 and NDB Cluster 7.4*).

There are a number of solutions available for setting up replication between two servers, but the best method to use depends on the presence of data and the engine types you are using. For more information on the available options, see Section 17.1.1, “How to Set Up Replication”.

There are two core types of replication format, Statement Based Replication (SBR), which replicates entire SQL statements, and Row Based Replication (RBR), which replicates only the changed rows. You may also use a third variety, Mixed Based Replication (MBR). For more information on the different replication formats, see Section 17.1.2, “Replication Formats”. In MySQL 5.6, statement-based format is the default.

MySQL 5.6.5 and later supports transactional replication based on *global transaction identifiers* (GTIDs). When using this type of replication, it is not necessary to work directly with log files or positions within these files, which greatly simplifies many common replication tasks. Because replication using GTIDs is entirely transactional, consistency between source and replica is guaranteed as long as all transactions committed on the source have also been applied on the replica. For more information about GTIDs and GTID-based replication, see Section 17.1.3, “Replication with Global Transaction Identifiers”.

Replication is controlled through a number of different options and variables. These control the core operation of the replication, timeouts, and the databases and filters that can be applied on databases and tables. For more information on the available options, see Section 17.1.4, “Replication and Binary Logging Options and Variables”.

You can use replication to solve a number of different problems, including problems with performance, supporting the backup of different databases, and as part of a larger solution to alleviate system failures. For information on how to address these issues, see Section 17.3, “Replication Solutions”.

For notes and tips on how different data types and statements are treated during replication, including details of replication features, version compatibility, upgrades, and problems and their resolution, including an FAQ, see Section 17.4, “Replication Notes and Tips”.

For detailed information on the implementation of replication, how replication works, the process and contents of the binary log, background threads and the rules used to decide how statements are recorded and replication, see Section 17.2, “Replication Implementation”.

17.1 Replication Configuration

Replication between servers in MySQL is based on the binary logging mechanism. The MySQL instance operating as the replication source server (the source of the database changes) writes updates and changes as “events” to the binary log. The information in the binary log is stored in different logging formats according to the database changes being recorded. Replicas are configured to read the binary log from the source and to execute the events in the binary log on the replica's local database.

Important

You cannot configure the source to log only certain events.

The source is “dumb” in this scenario. Once binary logging has been enabled, all statements are recorded in the binary log. Each replica receives a copy of the entire contents of the binary log. It is the responsibility of the replica to decide which statements in the binary log should be executed; you cannot configure the source to log only certain events. If you do not specify otherwise, all events in the source's binary log are executed on the replica. If required, you can configure the replica to process only events that apply to particular databases or tables.

Each replica keeps a record of the binary log coordinates: The file name and position within the file that it has read and processed from the source. This means that multiple replicas can be connected to the source and executing different parts of the same binary log. Because the replicas control this process, individual replicas can be connected and disconnected from the server without affecting the source's operation. Also, because each replica records the current position within the binary log, it is possible for replicas to be disconnected, reconnect and then resume processing.

The source and each replica must be configured with a unique ID (using the `server_id` system variable). In addition, each replica must be configured with information about the source's host name, log file name, and position within that file. These details can be controlled from within a MySQL session using the `CHANGE MASTER TO` statement on the replica. The details are stored within the replica's connection metadata repository, which can be either a file or a table (see Section 17.2.2, “Relay Log and Replication Metadata Repositories”).

This section describes the setup and configuration required for a replication environment, including step-by-step instructions for creating a new replication environment. The major components of this section are:

- For a guide to setting up two or more servers for replication, Section 17.1.1, “How to Set Up Replication”, deals with the configuration of the systems and provides methods for copying data between the source and replicas.
- Events in the binary log are recorded using a number of formats. These are referred to as statement-based replication (SBR) or row-based replication (RBR). A third type, mixed-format replication (MIXED), uses SBR or RBR replication automatically to take advantage of the benefits of both SBR and RBR formats when appropriate. The different formats are discussed in Section 17.1.2, “Replication Formats”.
- Detailed information on the different configuration options and variables that apply to replication is provided in Section 17.1.4, “Replication and Binary Logging Options and Variables”.
- Once started, the replication process should require little administration or monitoring. However, for advice on common tasks that you may want to execute, see Section 17.1.5, “Common Replication Administration Tasks”.

17.1.1 How to Set Up Replication

This section describes how to set up complete replication of a MySQL server. There are a number of different methods for setting up replication, and the exact method to use depends on how you are setting up replication, and whether you already have data within your source database.

There are some generic tasks that are common to all replication setups:

- On the source, you must enable binary logging and configure a unique server ID. This might require a server restart. See Section 17.1.1.1, “Setting the Replication Source Configuration”.
- On each replica that you want to connect to the source, you must configure a unique server ID. This might require a server restart. See Section 17.1.1.2, “Setting the Replica Configuration”.

- Optionally, create a separate user for your replicas to use during authentication with the source when reading the binary log for replication. See Section 17.1.1.3, “Creating a User for Replication”.
- Before creating a data snapshot or starting the replication process, you should record the position of the binary log on the source. You need this information when configuring the replica so that the replica knows where within the binary log to start executing events. See Section 17.1.1.4, “Obtaining the Replication Source Binary Log Coordinates”.
- If you already have data on your source and want to use it to synchronize your replica, you need to create a data snapshot. There are different methods to create the database snapshot, depending on the size of the database and the location of the files. Create a snapshot using `mysqldump` (see Section 17.1.1.5, “Creating a Data Snapshot Using mysqldump”) or by copying the data files directly (see Section 17.1.1.6, “Creating a Data Snapshot Using Raw Data Files”).
- Configure the replica with settings for connecting to the source, such as the host name, login credentials, and binary log file name and position. See Section 17.1.1.10, “Setting the Source Configuration on the Replica”.

After configuring the basic options, select your scenario:

- To set up replication for a fresh installation of a source and replicas that contain no data, see Section 17.1.1.7, “Setting Up Replication with New Source and Replicas”.
- To set up replication of a new source using the data from an existing MySQL server, see Section 17.1.1.8, “Setting Up Replication with Existing Data”.
- To add replicas to an existing replication environment, see Section 17.1.1.9, “Introducing Additional Replicas to an Existing Replication Environment”.

Before administering MySQL replication servers, read this entire chapter and try all statements mentioned in Section 13.4.1, “SQL Statements for Controlling Replication Source Servers”, and Section 13.4.2, “SQL Statements for Controlling Replica Servers”. Also familiarize yourself with the replication startup options described in Section 17.1.4, “Replication and Binary Logging Options and Variables”.

Note

Certain steps within the setup process require the `SUPER` privilege. If you do not have this privilege, it might not be possible to enable replication.

17.1.1.1 Setting the Replication Source Configuration

On a replication source server, you must enable binary logging and establish a unique server ID. If this has not already been done, a server restart is required.

Binary logging *must* be enabled on the source because the binary log is the basis for replicating changes from the source to its replicas. If binary logging is not enabled using the `log-bin` option, replication is not possible.

Each server within a replication group must be configured with a unique server ID. This ID is used to identify individual servers within the group, and must be a positive integer between 1 and (2^{32})−1. How you organize and select the numbers is your choice.

To configure the binary log and server ID options, shut down the MySQL server and edit the `my.cnf` or `my.ini` file. Within the `[mysqld]` section of the configuration file, add the `log-bin` and `server-id` options. If these options already exist, but are commented out, uncomment the options and alter them according to your needs. For example, to enable binary logging using a log file name prefix of `mysql-bin`, and configure a server ID of 1, use these lines:

```
[mysqld]
log-bin=mysql-bin
```

```
server-id=1
```

After making the changes, restart the server.

Note

If you do not set `server_id` (or set it explicitly to its default value of 0), the source refuses any connections from replicas.

Note

For the greatest possible durability and consistency in a replication setup using `InnoDB` with transactions, you should use `innodb_flush_log_at_trx_commit=1` and `sync_binlog=1` in the source `my.cnf` file.

Note

Ensure that the `skip_networking` system variable is not enabled on your replication source server. If networking has been disabled, the replica can not communicate with the source and replication fails.

17.1.1.2 Setting the Replica Configuration

On a replica, you must establish a unique server ID. If this has not already been done, this part of replica setup requires a server restart.

If the replica server ID is not already set, or the current value conflicts with the value that you have chosen for the replication source server, shut down the replica server and edit the `[mysqld]` section of the configuration file to specify a unique server ID. For example:

```
[mysqld]
server-id=2
```

After making the changes, restart the server.

If you are setting up multiple replicas, each one must have a unique `server_id` value that differs from that of the source and from any of the other replicas.

Note

If you do not set `server_id` (or set it explicitly to its default value of 0), the replica refuses to connect to a source.

You do not have to enable binary logging on the replica for replication to be set up. However, if you enable binary logging on the replica, you can use the replica's binary log for data backups and crash recovery, and also use the replica as part of a more complex replication topology. For example, where this replica then acts as a source to other replicas.

17.1.1.3 Creating a User for Replication

Each replica connects to the source using a MySQL user name and password, so there must be a user account on the source server that the replica can use to connect. Any account can be used for this operation, providing it has been granted the `REPLICATION SLAVE` privilege. You can choose to create a different account for each replica, or connect to the source using the same account for each replica.

Although you do not have to create an account specifically for replication, you should be aware that the replication user name and password are stored in plain text in the replica's connection metadata repository file or table (see Section 17.2.2.2, “Replication Metadata Repositories”). Therefore, you may want to create a separate account that has privileges only for the replication process, to minimize the possibility of compromise to other accounts.

To create a new account, use `CREATE USER`. To grant this account the privileges required for replication, use the `GRANT` statement. If you create an account solely for the purposes of replication, that account needs only the `REPLICATION SLAVE` privilege. For example, to set up a new user, `repl`, that can connect for replication from any host within the `example.com` domain, issue these statements on the source:

```
mysql> CREATE USER 'repl'@'%.example.com' IDENTIFIED BY 'password';
mysql> GRANT REPLICATION SLAVE ON *.* TO 'repl'@'%.example.com';
```

See Section 13.7.1, “Account Management Statements”, for more information on statements for manipulation of user accounts.

17.1.1.4 Obtaining the Replication Source Binary Log Coordinates

To configure the replica to start the replication process at the correct point, you need to note the source's current coordinates within its binary log.

Warning

This procedure uses `FLUSH TABLES WITH READ LOCK`, which blocks `COMMIT` operations for `InnoDB` tables.

If you have existing data on your source server that you want to synchronize on your replicas before starting the replication process, you must stop processing statements on the source, and then obtain its current binary log coordinates and dump its data, before permitting the source to continue executing statements. If you do not stop the execution of statements, the data dump and the source status information that you use does not match and you may end up with inconsistent or corrupted databases on the replicas.

If you are planning to shut down the source to create a data snapshot, you can optionally skip this procedure and instead store a copy of the binary log index file along with the data snapshot. In that situation, the source creates a new binary log file on restart. The source's binary log coordinates where the replica must start the replication process are therefore the start of that new file, which is the next binary log file on the source following after the files that are listed in the copied binary log index file.

To obtain the source's binary log coordinates, follow these steps:

1. Start a session on the source by connecting to it with the command-line client, and flush all tables and block write statements by executing the `FLUSH TABLES WITH READ LOCK` statement:

   ```
   mysql> FLUSH TABLES WITH READ LOCK;
   ```

   Warning

   Leave the client from which you issued the `FLUSH TABLES` statement running so that the read lock remains in effect. If you exit the client, the lock is released.

2. In a different session on the source, use the `SHOW MASTER STATUS` statement to determine the current binary log file name and position:

   ```
   mysql > SHOW MASTER STATUS;
   +------------------+----------+--------------+------------------+
   | File             | Position | Binlog_Do_DB | Binlog_Ignore_DB |
   +------------------+----------+--------------+------------------+
   | mysql-bin.000003 | 73       | test         | manual,mysql     |
   +------------------+----------+--------------+------------------+
   ```

   The `File` column shows the name of the log file and `Position` shows the position within the file. In this example, the binary log file is `mysql-bin.000003` and the position is 73. Record these values. You need them later when you are setting up the replica. They represent the replication coordinates at which the replica should begin processing new updates from the source.

If the source has been running previously without binary logging enabled, the log file name and position values displayed by `SHOW MASTER STATUS` or `mysqldump --master-data` is empty. In that case, the values that you need to use later when specifying the binary log file and position are the empty string (`''`) and `4`.

You now have the information you need to enable the replica to start reading from the binary log in the correct place to start replication.

If you have existing data that needs be to synchronized with the replica before you start replication, leave the client running so that the lock remains in place and then proceed to Section 17.1.1.5, “Creating a Data Snapshot Using mysqldump”, or Section 17.1.1.6, “Creating a Data Snapshot Using Raw Data Files”. The idea here is to prevent any further changes so that the data copied to the replicas is in synchrony with the source.

If you are setting up a brand new source and replica replication group, you can exit the first session to release the read lock.

17.1.1.5 Creating a Data Snapshot Using mysqldump

One way to create a snapshot of the data in an existing source database is to use the `mysqldump` tool to create a dump of all the databases you want to replicate. Once the data dump has been completed, you then import this data into the replica before starting the replication process.

The example shown here dumps all databases to a file named `dbdump.db`, and includes the `--master-data` option which automatically appends the `CHANGE MASTER TO` statement required on the replica to start the replication process:

```
shell> mysqldump --all-databases --master-data > dbdump.db
```

If you do not use `--master-data`, then it is necessary to lock all tables in a separate session manually (using `FLUSH TABLES WITH READ LOCK`) prior to running `mysqldump`, then exiting or running UNLOCK TABLES from the second session to release the locks. You must also obtain binary log position information matching the snapshot, using `SHOW MASTER STATUS`, and use this to issue the appropriate `CHANGE MASTER TO` statement when starting the replica.

When choosing databases to include in the dump, remember that you need to filter out databases on each replica that you do not want to include in the replication process.

To import the data, either copy the dump file to the replica, or access the file from the source when connecting remotely to the replica.

17.1.1.6 Creating a Data Snapshot Using Raw Data Files

If your database is large, copying the raw data files can be more efficient than using `mysqldump` and importing the file on each replica. This technique skips the overhead of updating indexes as the `INSERT` statements are replayed.

Using this method with tables in storage engines with complex caching or logging algorithms requires extra steps to produce a perfect “point in time” snapshot: the initial copy command might leave out cache information and logging updates, even if you have acquired a global read lock. How the storage engine responds to this depends on its crash recovery abilities.

This method also does not work reliably if the source and replica have different values for `ft_stopword_file`, `ft_min_word_len`, or `ft_max_word_len` and you are copying tables having full-text indexes.

If you use `InnoDB` tables, you can use the `mysqlbackup` command from the MySQL Enterprise Backup component to produce a consistent snapshot. This command records the log name and offset corresponding to the snapshot to be later used on the replica. MySQL Enterprise Backup is a

commercial product that is included as part of a MySQL Enterprise subscription. See Section 24.2, “MySQL Enterprise Backup Overview” for detailed information.

Otherwise, use the cold backup technique to obtain a reliable binary snapshot of `InnoDB` tables: copy all data files after doing a slow shutdown of the MySQL Server.

To create a raw data snapshot of `MyISAM` tables, you can use standard copy tools such as `cp` or `copy`, a remote copy tool such as `scp` or `rsync`, an archiving tool such as `zip` or `tar`, or a file system snapshot tool such as `dump`, providing that your MySQL data files exist on a single file system. If you are replicating only certain databases, copy only those files that relate to those tables. (For `InnoDB`, all tables in all databases are stored in the system tablespace files, unless you have the `innodb_file_per_table` option enabled.)

You might want to specifically exclude the following files from your archive:

- Files relating to the `mysql` database.
- The replica's connection metadata repository file, if used (see Section 17.2.2, “Relay Log and Replication Metadata Repositories”).
- The source's binary log files, with the exception of the binary log index file if you are going to use this to locate the source's binary log coordinates for the replica.
- Any relay log files.

To get the most consistent results with a raw data snapshot, shut down the source server during the process, as follows:

1. Acquire a read lock and get the source's status. See Section 17.1.1.4, “Obtaining the Replication Source Binary Log Coordinates”.
2. In a separate session, shut down the source server:

   ```
   shell> mysqladmin shutdown
   ```

3. Make a copy of the MySQL data files. The following examples show common ways to do this. You need to choose only one of them:

   ```
   shell> tar cf /tmp/db.tar ./data
   shell> zip -r /tmp/db.zip ./data
   shell> rsync --recursive ./data /tmp/dbdata
   ```

4. Restart the source server.

If you are not using `InnoDB` tables, you can get a snapshot of the system from a source without shutting down the server as described in the following steps:

1. Acquire a read lock and get the source's status. See Section 17.1.1.4, “Obtaining the Replication Source Binary Log Coordinates”.
2. Make a copy of the MySQL data files. The following examples show common ways to do this. You need to choose only one of them:

   ```
   shell> tar cf /tmp/db.tar ./data
   shell> zip -r /tmp/db.zip ./data
   shell> rsync --recursive ./data /tmp/dbdata
   ```

3. In the client where you acquired the read lock, release the lock:

   ```
   mysql> UNLOCK TABLES;
   ```

Once you have created the archive or copy of the database, copy the files to each replica before starting the replication process.

17.1.1.7 Setting Up Replication with New Source and Replicas

The easiest and most straightforward method for setting up replication is to use new source and replica servers.

You can also use this method if you are setting up new servers but have an existing dump of the databases from a different server that you want to load into your replication configuration. By loading the data into a new source, the data is automatically replicated to the replicas.

To set up replication between a new source and replica:

1. Configure the MySQL source with the necessary configuration properties. See Section 17.1.1.1, “Setting the Replication Source Configuration”.
2. Start up the MySQL source.
3. Set up a user. See Section 17.1.1.3, “Creating a User for Replication”.
4. Obtain the source's status information, or a copy of the source's binary log index file made during a shutdown for the data snapshot. See Section 17.1.1.4, “Obtaining the Replication Source Binary Log Coordinates”.
5. On the source, release the read lock:

   ```
   mysql> UNLOCK TABLES;
   ```

6. On the replica, edit the MySQL configuration. See Section 17.1.1.2, “Setting the Replica Configuration”.
7. Start up the MySQL replica server.
8. Execute a `CHANGE MASTER TO` statement to set the replication source server configuration. See Section 17.1.1.10, “Setting the Source Configuration on the Replica”.

Perform the replica setup steps on each replica.

Because there is no data to load or exchange on a new server configuration you do not need to copy or import any information.

If you are setting up a new replication environment using the data from a different existing database server, you must now run the dump file generated from that server on the new source. The database updates are automatically propagated to the replicas:

```
shell> mysql -h source < fulldb.dump
```

17.1.1.8 Setting Up Replication with Existing Data

When setting up replication with existing data, you must decide how best to get the data from the source to the replica before starting the replication service.

The basic process for setting up replication with existing data is as follows:

1. With the MySQL source running, create a user to be used by the replica when connecting to the source during replication. See Section 17.1.1.3, “Creating a User for Replication”.
2. If you have not already configured the `server_id` system variable and enabled binary logging on the source server, you must shut it down to configure these options. See Section 17.1.1.1, “Setting the Replication Source Configuration”.

   If you have to shut down your source server, this is a good opportunity to take a snapshot of its databases. You should obtain the source status (see Section 17.1.1.4, “Obtaining the Replication

Source Binary Log Coordinates") before taking down the source, updating the configuration and taking a snapshot. For information on how to create a snapshot using raw data files, see Section 17.1.1.6, "Creating a Data Snapshot Using Raw Data Files".

3. If your source server is already correctly configured, obtain its status (see Section 17.1.1.4, "Obtaining the Replication Source Binary Log Coordinates") and then use `mysqldump` to take a snapshot (see Section 17.1.1.5, "Creating a Data Snapshot Using mysqldump") or take a raw snapshot of the live server using the guide in Section 17.1.1.6, "Creating a Data Snapshot Using Raw Data Files".

4. Update the configuration of the replica. See Section 17.1.1.2, "Setting the Replica Configuration".

5. The next step depends on how you created the snapshot of data on the source.

   If you used `mysqldump`:

   a. Start the replica, using the `--skip-slave-start` option so that replication does not start.

   b. Import the dump file:

   ```
   shell> mysql < fulldb.dump
   ```

   If you created a snapshot using the raw data files:

   a. Extract the data files into the replica's data directory. For example:

   ```
   shell> tar xvf dbdump.tar
   ```

   You may need to set permissions and ownership on the files so that the replica server can access and modify them.

   b. Start the replica, using the `--skip-slave-start` option so that replication does not start.

6. Configure the replica with the replication coordinates from the source. This tells the replica the binary log file and position within the file where replication needs to start. Also, configure the replica with the login credentials and host name of the source. For more information on the `CHANGE MASTER TO` statement required, see Section 17.1.1.10, "Setting the Source Configuration on the Replica".

7. Start the replication threads:

   ```
   mysql> START SLAVE;
   ```

After you have performed this procedure, the replica should connect to the source and catch up on any updates that have occurred since the snapshot was taken.

If you have forgotten to set the `server_id` system variable for the source, replicas cannot connect to it.

If you have forgotten to set the `server_id` system variable for the replica, you get the following error in the replica's error log:

```
Warning: You should set server-id to a non-0 value if master_host
is set; we will force server id to 2, but this MySQL server will
not act as a slave.
```

You also find error messages in the replica's error log if it is not able to replicate for any other reason.

The replica uses information stored in its connection metadata repository to keep track of how much of the source's binary log it has processed. The repository can be in the form of files or a table, as determined by the value set for the `master_info_repository` system variable. When a replica runs with `master_info_repository=FILE`, you can find in its data directory two files, named `master.info` and `relay-log.info`. If `master_info_repository=TABLE` instead, this

information is saved in the table `master_slave_info` in the `mysql` system database. In either case, do *not* remove or edit the files or table unless you know exactly what you are doing and fully understand the implications. Even in that case, it is preferred that you use the `CHANGE MASTER TO` statement to change replication parameters. The replica can use the values specified in the statement to update the status files automatically. See Section 17.2.2, "Relay Log and Replication Metadata Repositories", for more information.

Note

The contents of the connection metadata repository override some of the server options specified on the command line or in `my.cnf`. See Section 17.1.4, "Replication and Binary Logging Options and Variables", for more details.

A single snapshot of the source suffices for multiple replicas. To set up additional replicas, use the same source snapshot and follow the replica portion of the procedure just described.

17.1.1.9 Introducing Additional Replicas to an Existing Replication Environment

You can add another replica to an existing replication configuration without stopping the source. To do this, you can set up the new replica by copying the data directory of an existing replica, and giving the new replica a different server ID (which is user-specified) and server UUID (which is generated at startup).

To duplicate an existing replica:

1. Stop the existing replica and record the replica status information, particularly the source's binary log file and relay log file positions. You can view the replica status by issuing `SHOW SLAVE STATUS` as follows:

   ```
   mysql> STOP SLAVE;
   mysql> SHOW SLAVE STATUS\G
   ```

2. Shut down the existing replica:

   ```
   shell> mysqladmin shutdown
   ```

3. Copy the data directory from the existing replica to the new replica, including the log files and relay log files. You can do this by creating an archive using `tar` or `WinZip`, or by performing a direct copy using a tool such as `cp` or `rsync`.

Important

- Before copying, verify that all the files relating to the existing replica actually are stored in the data directory. For example, the `InnoDB` system tablespace, undo tablespace, and redo log might be stored in an alternative location. `InnoDB` tablespace files and file-per-table tablespaces might have been created in other directories. The binary logs and relay logs for the replica might be in their own directories outside the data directory. Check through the system variables that are set for the existing replica and look for any alternative paths that have been specified. If you find any, copy these directories over as well.
- During copying, if files have been used for the replication metadata repositories (see Section 17.2.2, "Relay Log and Replication Metadata Repositories"), which is the default in MySQL 5.6, ensure that you also copy these files from the existing replica to the new replica. If tables have been used for the repositories, the tables are in the data directory.
- After copying, delete the `auto.cnf` file from the copy of the data directory on the new replica, so that the new replica is started with a different generated server UUID. The server UUID must be unique.

A common problem that is encountered when adding new replicas is that the new slave fails with a series of warning and error messages like these:

```
071118 16:44:10 [Warning] Neither --relay-log nor --relay-log-index were used; so
replication may break when this MySQL server acts as a slave and has his hostname
changed!! Please use '--relay-log=new_replica_hostname-relay-bin' to avoid this problem.
071118 16:44:10 [ERROR] Failed to open the relay log './old_replica_hostname-relay-bin.003525'
(relay_log_pos 22940879)
071118 16:44:10 [ERROR] Could not find target log during relay log initialization
071118 16:44:10 [ERROR] Failed to initialize the master info structure
```

This situation can occur if the `relay_log` system variable is not specified, as the relay log files contain the host name as part of their file names. This is also true of the relay log index file if the `relay_log_index` system variable is not used. For more information about these variables, see Section 17.1.4, “Replication and Binary Logging Options and Variables”.

To avoid this problem, use the same value for `relay_log` on the new replica that was used on the existing replica. If this option was not set explicitly on the existing replica, use *`existing_replica_hostname`*`-relay-bin`. If this is not possible, copy the existing replica's relay log index file to the new replica and set the `relay_log_index` system variable on the new replica to match what was used on the existing replica. If this option was not set explicitly on the existing replica, use *`existing_replica_hostname`*`-relay-bin.index`. Alternatively, if you have already tried to start the new replica after following the remaining steps in this section and have encountered errors like those described previously, then perform the following steps:

a. If you have not already done so, issue `STOP SLAVE` on the new replica.

   If you have already started the existing replica again, issue `STOP SLAVE` on the existing replica as well.

b. Copy the contents of the existing replica's relay log index file into the new replica's relay log index file, making sure to overwrite any content already in the file.

c. Proceed with the remaining steps in this section.

4. When copying is complete, restart the existing replica.

5. On the new replica, edit the configuration and give the new replica a unique server ID (using the `server_id` system variable) that is not used by the source or any of the existing replicas.

6. Start the new replica server, specifying the `--skip-slave-start` option so that replication does not start yet. Issue `SHOW SLAVE STATUS` to confirm that the new replica has the correct settings when compared with the existing replica. Also display the server ID and server UUID and verify that these are correct and unique for the new replica.

7. Start the replication threads by issuing a `START SLAVE` statement:

   ```
   mysql> START SLAVE;
   ```

   The new replica now uses the information in its connection metadata repository to start the replication process.

17.1.1.10 Setting the Source Configuration on the Replica

To set up the replica to communicate with the source for replication, you must tell the replica the necessary connection information. To do this, execute the following statement on the replica, replacing the option values with the actual values relevant to your system:

```
mysql> CHANGE MASTER TO
    ->     MASTER_HOST='source_host_name',
    ->     MASTER_USER='replication_user_name',
    ->     MASTER_PASSWORD='replication_password',
```

```
    ->     MASTER_LOG_FILE='recorded_log_file_name',
    ->     MASTER_LOG_POS=recorded_log_position;
```

Note

Replication cannot use Unix socket files. You must be able to connect to the replication source server using TCP/IP.

The `CHANGE MASTER TO` statement has other options as well. For example, it is possible to set up secure replication using SSL. For a full list of options, and information about the maximum permissible length for the string-valued options, see Section 13.4.2.1, “CHANGE MASTER TO Statement”.

17.1.2 Replication Formats

Replication works because events written to the binary log are read from the source and then processed on the replica. The events are recorded within the binary log in different formats according to the type of event. The different replication formats used correspond to the binary logging format used when the events were recorded in the source's binary log. The correlation between binary logging formats and the terms used during replication are:

- When using statement-based binary logging, the source writes SQL statements to the binary log. Replication of the source to the replica works by executing the SQL statements on the replica. This is called *statement-based replication* (often abbreviated as *SBR*), which corresponds to the standard MySQL statement-based binary logging format. Replication capabilities in MySQL version 5.1.4 and earlier used this format exclusively.

- When using row-based logging, the source writes *events* to the binary log that indicate how individual table rows are changed. Replication of the source to the replica works by copying the events representing the changes to the table rows to the replica. This is called *row-based replication* (often abbreviated as *RBR*). In row-based replication, the source writes *events* to the binary log that indicate how individual table rows are changed.

- You can also configure MySQL to use a mix of both statement-based and row-based logging, depending on which is most appropriate for the change to be logged. This is called *mixed-format logging*. When using mixed-format logging, a statement-based log is used by default. Depending on certain statements, and also the storage engine being used, the log is automatically switched to row-based in particular cases. Replication using the mixed format is often referred to as *mixed-based replication* or *mixed-format replication*. For more information, see Section 5.4.4.3, “Mixed Binary Logging Format”.

In MySQL 5.6, statement-based format is the default.

NDB Cluster. The default binary logging format in all MySQL NDB Cluster 7.3 and MySQL NDB Cluster 7.4 releases is `MIXED`. You should note that NDB Cluster Replication always uses row-based replication, and that the `NDB` storage engine is incompatible with statement-based replication. See Section 18.6.2, “General Requirements for NDB Cluster Replication”, for more information.

When using `MIXED` format, the binary logging format is determined in part by the storage engine being used and the statement being executed. For more information on mixed-format logging and the rules governing the support of different logging formats, see Section 5.4.4.3, “Mixed Binary Logging Format”.

The logging format in a running MySQL server is controlled by setting the `binlog_format` server system variable. This variable can be set with session or global scope. The rules governing when and how the new setting takes effect are the same as for other MySQL server system variables. Setting the variable for the current session lasts only until the end of that session, and the change is not visible to other sessions. Setting the variable globally takes effect for clients that connect after the change, but not for any current client sessions, including the session where the variable setting was changed. To make the global system variable setting permanent so that it applies across server restarts, you must set it in an option file. For more information, see Section 13.7.4.1, “SET Syntax for Variable Assignment”.

There are conditions under which you cannot change the binary logging format at runtime or doing so causes replication to fail. See Section 5.4.4.2, “Setting The Binary Log Format”.

Changing the global `binlog_format` value requires privileges sufficient to set global system variables. Changing the session `binlog_format` value requires privileges sufficient to set restricted session system variables. See Section 5.1.8.1, “System Variable Privileges”.

The statement-based and row-based replication formats have different issues and limitations. For a comparison of their relative advantages and disadvantages, see Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”.

With statement-based replication, you may encounter issues with replicating stored routines or triggers. You can avoid these issues by using row-based replication instead. For more information, see Section 20.7, “Stored Program Binary Logging”.

17.1.2.1 Advantages and Disadvantages of Statement-Based and Row-Based Replication

Each binary logging format has advantages and disadvantages. For most users, the mixed replication format should provide the best combination of data integrity and performance. If, however, you want to take advantage of the features specific to the statement-based or row-based replication format when performing certain tasks, you can use the information in this section, which provides a summary of their relative advantages and disadvantages, to determine which is best for your needs.

- Advantages of statement-based replication
- Disadvantages of statement-based replication
- Advantages of row-based replication
- Disadvantages of row-based replication

Advantages of statement-based replication

- Proven technology.
- Less data written to log files. When updates or deletes affect many rows, this results in *much* less storage space required for log files. This also means that taking and restoring from backups can be accomplished more quickly.
- Log files contain all statements that made any changes, so they can be used to audit the database.

Disadvantages of statement-based replication

- **Statements that are unsafe for SBR.**
  Not all statements which modify data (such as `INSERT DELETE`, `UPDATE`, and `REPLACE` statements) can be replicated using statement-based replication. Any nondeterministic behavior is difficult to replicate when using statement-based replication. Examples of such DML (Data Modification Language) statements include the following:

  - A statement that depends on a UDF or stored program that is nondeterministic, since the value returned by such a UDF or stored program or depends on factors other than the parameters supplied to it. (Row-based replication, however, simply replicates the value returned by the UDF or stored program, so its effect on table rows and data is the same on both the source and replica.) See Section 17.4.1.16, “Replication of Invoked Features”, for more information.
  - `DELETE` and `UPDATE` statements that use a `LIMIT` clause without an `ORDER BY` are nondeterministic. See Section 17.4.1.17, “Replication and LIMIT”.
  - Statements using any of the following functions cannot be replicated properly using statement-based replication:

    - `LOAD_FILE()`

- `UUID()`, `UUID_SHORT()`
- `USER()`
- `FOUND_ROWS()`
- `SYSDATE()` (unless both the source and the replica are started with the `--sysdate-is-now` option)
- `GET_LOCK()`
- `IS_FREE_LOCK()`
- `IS_USED_LOCK()`
- `MASTER_POS_WAIT()`
- `RAND()`
- `RELEASE_LOCK()`
- `SLEEP()`
- `VERSION()`

  However, all other functions are replicated correctly using statement-based replication, including `NOW()` and so forth.

  For more information, see Section 17.4.1.15, “Replication and System Functions”.

Statements that cannot be replicated correctly using statement-based replication are logged with a warning like the one shown here:

```
[Warning] Statement is not safe to log in statement format.
```

A similar warning is also issued to the client in such cases. The client can display it using `SHOW WARNINGS`.

- `INSERT ... SELECT` requires a greater number of row-level locks than with row-based replication.
- `UPDATE` statements that require a table scan (because no index is used in the `WHERE` clause) must lock a greater number of rows than with row-based replication.
- For `InnoDB`: An `INSERT` statement that uses `AUTO_INCREMENT` blocks other nonconflicting `INSERT` statements.
- For complex statements, the statement must be evaluated and executed on the replica before the rows are updated or inserted. With row-based replication, the replica only has to modify the affected rows, not execute the full statement.
- If there is an error in evaluation on the replica, particularly when executing complex statements, statement-based replication may slowly increase the margin of error across the affected rows over time. See Section 17.4.1.27, “Replica Errors During Replication”.
- Stored functions execute with the same `NOW()` value as the calling statement. However, this is not true of stored procedures.
- Deterministic UDFs must be applied on the replicas.
- Table definitions must be (nearly) identical on source and replica. See Section 17.4.1.9, “Replication with Differing Table Definitions on Source and Replica”, for more information.

Advantages of row-based replication

- All changes can be replicated. This is the safest form of replication.

Note

Statements that update the information in the `mysql` database, such as `GRANT`, `REVOKE` and the manipulation of triggers, stored routines (including stored procedures), and views, are all replicated to replicas using statement-based replication.

For statements such as `CREATE TABLE ... SELECT`, a `CREATE` statement is generated from the table definition and replicated using statement-based format, while the row insertions are replicated using row-based format.

- The technology is the same as in most other database management systems; knowledge about other systems transfers to MySQL.
- Fewer row locks are required on the source, which thus achieves higher concurrency, for the following types of statements:
  - `INSERT ... SELECT`
  - `INSERT` statements with `AUTO_INCREMENT`
  - `UPDATE` or `DELETE` statements with `WHERE` clauses that do not use keys or do not change most of the examined rows.
- Fewer row locks are required on the replica for any `INSERT`, `UPDATE`, or `DELETE` statement.

Disadvantages of row-based replication

- RBR can generate more data that must be logged. To replicate a DML statement (such as an `UPDATE` or `DELETE` statement), statement-based replication writes only the statement to the binary log. By contrast, row-based replication writes each changed row to the binary log. If the statement changes many rows, row-based replication may write significantly more data to the binary log; this is true even for statements that are rolled back. This also means that taking and restoring from backup can require more time. In addition, the binary log is locked for a longer time to write the data, which may cause concurrency problems. Use `binlog_row_image=minimal` to reduce the disadvantage considerably.
- Deterministic UDFs that generate large `BLOB` values take longer to replicate with row-based replication than with statement-based replication. This is because the `BLOB` column value is logged, rather than the statement generating the data.
- You cannot see on the replica what statements were received from the source and executed. However, you can see what data was changed using `mysqlbinlog` with the options `--base64-output=DECODE-ROWS` and `--verbose`.

  Alternatively, use the `binlog_rows_query_log_events` variable added in MySQL 5.6.2, which if enabled adds a `Rows_query` event with the statement to `mysqlbinlog` output when the `-vv` option is used.
- For tables using the `MyISAM` storage engine, a stronger lock is required on the replica for `INSERT` statements when applying them as row-based events to the binary log than when applying them as statements. This means that concurrent inserts on `MyISAM` tables are not supported when using row-based replication.

17.1.2.2 Usage of Row-Based Logging and Replication

MySQL uses statement-based logging (SBL), row-based logging (RBL) or mixed-format logging. The type of binary log used impacts the size and efficiency of logging. Therefore the choice between row-based replication (RBR) or statement-based replication (SBR) depends on your application and

environment. This section describes known issues when using a row-based format log, and discusses some best practices using it in replication.

For additional information, see Section 17.1.2, “Replication Formats”, and Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”.

For information about issues specific to NDB Cluster Replication (which depends on row-based replication), see Section 18.6.3, “Known Issues in NDB Cluster Replication”.

- **Row-based logging of temporary tables.** As noted in Section 17.4.1.29, “Replication and Temporary Tables”, temporary tables are not replicated when using row-based format. When using mixed format logging, “safe” statements involving temporary tables are logged using statement-based format. For more information, see Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”.

  Temporary tables are not replicated when using row-based format because there is no need. In addition, because temporary tables can be read only from the thread which created them, there is seldom if ever any benefit obtained from replicating them, even when using statement-based format.

  In MySQL 5.6, you can switch from statement-based to row-based binary logging mode even when temporary tables have been created. However, while using the row-based format, the MySQL server cannot determine the logging mode that was in effect when a given temporary table was created. For this reason, the server in such cases logs a `DROP TEMPORARY TABLE IF EXISTS` statement for each temporary table that still exists for a given client session when that session ends. While this means that it is possible that an unnecessary `DROP TEMPORARY TABLE` statement might be logged in some cases, the statement is harmless, and does not cause an error even if the table does not exist, due to the presence of the `IF EXISTS` option.

  Note

  From MySQL 8.0, this behavior is changed because the MySQL server tracks the logging mode that was in effect when each temporary table was created. The `DROP TEMPORARY TABLE IF EXISTS` statement is therefore not necessarily logged for each temporary table. From that release, when a given client session ends, the server logs a `DROP TEMPORARY TABLE IF EXISTS` statement for each temporary table that still exists and was created when statement-based binary logging was in use. If row-based or mixed format binary logging was in use when the table was created, the `DROP TEMPORARY TABLE IF EXISTS` statement is not logged.

  Nontransactional DML statements involving temporary tables are allowed when using `binlog_format=ROW`, as long as any nontransactional tables affected by the statements are temporary tables (Bug #14272672).

- **RBL and synchronization of nontransactional tables.** When many rows are affected, the set of changes is split into several events; when the statement commits, all of these events are written to the binary log. When executing on the replica, a table lock is taken on all tables involved, and then the rows are applied in batch mode. (This may or may not be effective, depending on the engine used for the replica's copy of the table.)

- **Latency and binary log size.** RBL writes changes for each row to the binary log and so its size can increase quite rapidly. This can significantly increase the time required to make changes on the replica that match those on the source. You should be aware of the potential for this delay in your applications.

- **Reading the binary log.** `mysqlbinlog` displays row-based events in the binary log using the `BINLOG` statement (see Section 13.7.6.1, “BINLOG Statement”). This statement displays an event as a base 64-encoded string, the meaning of which is not evident. When invoked with the `--base64-output=DECODE-ROWS` and `--verbose` options, `mysqlbinlog` formats the contents of the binary log to be human readable. When binary log events were written in row-based format and you want to

read or recover from a replication or database failure you can use this command to read contents of the binary log. For more information, see Section 4.6.8.2, “mysqlbinlog Row Event Display”.

- **Binary log execution errors and slave_exec_mode.** If `slave_exec_mode` is `IDEMPOTENT`, a failure to apply changes from RBL because the original row cannot be found does not trigger an error or cause replication to fail. This means that it is possible that updates are not applied on the replica, so that the source and replica are no longer synchronized. Latency issues and use of nontransactional tables with RBR when `slave_exec_mode` is `IDEMPOTENT` can cause the source and replica to diverge even further. For more information about `slave_exec_mode`, see Section 5.1.7, “Server System Variables”.

  Note

  `slave_exec_mode=IDEMPOTENT` is generally useful only for circular replication or multi-source replication with NDB Cluster, for which `IDEMPOTENT` is the default value.

  For other scenarios, setting `slave_exec_mode` to `STRICT` is normally sufficient; this is the default value.

  Note

  Formerly, the default value when using NDB Cluster was `slave_exec_mode=IDEMPOTENT`, but this is no longer the case in MySQL NDB Cluster 7.3 and later.

- **Filtering based on server ID not supported.** In MySQL 5.6, you can filter based on server ID by using the `IGNORE_SERVER_IDS` option for the `CHANGE MASTER TO` statement. This option works with statement-based and row-based logging formats. Another method to filter out changes on some replicas is to use a `WHERE` clause that includes the relation `@@server_id <> id_value` clause with `UPDATE` and `DELETE` statements. For example, `WHERE @@server_id <> 1`. However, this does not work correctly with row-based logging. To use the `server_id` system variable for statement filtering, use statement-based logging.

- **Database-level replication options.** The effects of the `--replicate-do-db`, `--replicate-ignore-db`, and `--replicate-rewrite-db` options differ considerably depending on whether row-based or statement-based logging is used. Therefore, it is recommended to avoid database-level options and instead use table-level options such as `--replicate-do-table` and `--replicate-ignore-table`. For more information about these options and the impact replication format has on how they operate, see Section 17.1.4, “Replication and Binary Logging Options and Variables”.

- **RBL, nontransactional tables, and stopped replicas.** When using row-based logging, if the replica server is stopped while a replication thread is updating a nontransactional table, the replica database can reach an inconsistent state. For this reason, it is recommended that you use a transactional storage engine such as `InnoDB` for all tables replicated using the row-based format. Use of `STOP SLAVE` or `STOP SLAVE SQL_THREAD` prior to shutting down the replica MySQL server helps prevent issues from occurring, and is always recommended regardless of the logging format or storage engine you use.

17.1.2.3 Determination of Safe and Unsafe Statements in Binary Logging

The “safeness” of a statement in MySQL Replication, refers to whether the statement and its effects can be replicated correctly using statement-based format. If this is true of the statement, we refer to the statement as *safe*; otherwise, we refer to it as *unsafe*.

In general, a statement is safe if it deterministic, and unsafe if it is not. However, certain nondeterministic functions are *not* considered unsafe (see Nondeterministic functions not considered unsafe, later in this section). In addition, statements using results from floating-point math functions—which are hardware-dependent—are always considered unsafe (see Section 17.4.1.12, “Replication and Floating-Point Values”).

Handling of safe and unsafe statements. A statement is treated differently depending on whether the statement is considered safe, and with respect to the binary logging format (that is, the current value of `binlog_format`).

- When using row-based logging, no distinction is made in the treatment of safe and unsafe statements.
- When using mixed-format logging, statements flagged as unsafe are logged using the row-based format; statements regarded as safe are logged using the statement-based format.
- When using statement-based logging, statements flagged as being unsafe generate a warning to this effect. Safe statements are logged normally.

Each statement flagged as unsafe generates a warning. Formerly, if a large number of such statements were executed on the source, this could lead to excessively large error log files. To prevent this, MySQL provides a warning suppression mechanism (introduced in MySQL 5.6.7), which behaves as follows: Whenever the 50 most recent `ER_BINLOG_UNSAFE_STATEMENT` warnings have been generated more than 50 times in any 50-second period, warning suppression is enabled. When activated, this causes such warnings not to be written to the error log; instead, for each 50 warnings of this type, a note `The last warning was repeated N times in last S seconds` is written to the error log. This continues as long as the 50 most recent such warnings were issued in 50 seconds or less; once the rate has decreased below this threshold, the warnings are once again logged normally. Warning suppression has no effect on how the safety of statements for statement-based logging is determined, nor on how warnings are sent to the client. MySQL clients still receive one warning for each such statement.

For more information, see Section 17.1.2, “Replication Formats”.

Statements considered unsafe.
Statements with the following characteristics are considered unsafe:

- **Statements containing system functions that may return a different value on replica.** These functions include `FOUND_ROWS()`, `GET_LOCK()`, `IS_FREE_LOCK()`, `IS_USED_LOCK()`, `LOAD_FILE()`, `MASTER_POS_WAIT()`, `PASSWORD()`, `RAND()`, `RELEASE_LOCK()`, `ROW_COUNT()`, `SESSION_USER()`, `SLEEP()`, `SYSDATE()`, `SYSTEM_USER()`, `USER()`, `UUID()`, and `UUID_SHORT()`.

  Nondeterministic functions not considered unsafe. Although these functions are not deterministic, they are treated as safe for purposes of logging and replication: `CONNECTION_ID()`, `CURDATE()`, `CURRENT_DATE()`, `CURRENT_TIME()`, `CURRENT_TIMESTAMP()`, `CURTIME()`,, `LAST_INSERT_ID()`, `LOCALTIME()`, `LOCALTIMESTAMP()`, `NOW()`, `UNIX_TIMESTAMP()`, `UTC_DATE()`, `UTC_TIME()`, and `UTC_TIMESTAMP()`.

  For more information, see Section 17.4.1.15, “Replication and System Functions”.

- **References to system variables.** Most system variables are not replicated correctly using the statement-based format. See Section 17.4.1.35, “Replication and Variables”. For exceptions, see Section 5.4.4.3, “Mixed Binary Logging Format”.
- **UDFs.** Since we have no control over what a UDF does, we must assume that it is executing unsafe statements.
- **Trigger or stored program updates a table having an AUTO_INCREMENT column.** This is unsafe because the order in which the rows are updated may differ on the source and the replica.

  In addition, an `INSERT` into a table that has a composite primary key containing an `AUTO_INCREMENT` column that is not the first column of this composite key is unsafe.

  For more information, see Section 17.4.1.1, “Replication and AUTO_INCREMENT”.

- **INSERT DELAYED statement.** This statement is considered unsafe because the insertion of the rows may interleave with concurrently executing statements.

- **INSERT ... ON DUPLICATE KEY UPDATE statements on tables with multiple primary or unique keys.** When executed against a table that contains more than one primary or unique key, this statement is considered unsafe, being sensitive to the order in which the storage engine checks the keys, which is not deterministic, and on which the choice of rows updated by the MySQL Server depends.

  An `INSERT ... ON DUPLICATE KEY UPDATE` statement against a table having more than one unique or primary key is marked as unsafe for statement-based replication beginning with MySQL 5.6.6. (Bug #11765650, Bug #58637)

- **Updates using LIMIT.** The order in which rows are retrieved is not specified, and is therefore considered unsafe. See Section 17.4.1.17, “Replication and LIMIT”.

- **Accesses or references log tables.** The contents of the system log table may differ between source and replica.

- **Nontransactional operations after transactional operations.** Within a transaction, allowing any nontransactional reads or writes to execute after any transactional reads or writes is considered unsafe.

  For more information, see Section 17.4.1.32, “Replication and Transactions”.

- **Accesses or references self-logging tables.** All reads and writes to self-logging tables are considered unsafe. Within a transaction, any statement following a read or write to self-logging tables is also considered unsafe.

- **LOAD DATA statements.** `LOAD DATA` is treated as unsafe and when `binlog_format=mixed` the statement is logged in row-based format. When `binlog_format=statement` `LOAD DATA` does not generate a warning, unlike other unsafe statements.

For additional information, see Section 17.4.1, “Replication Features and Issues”.

17.1.3 Replication with Global Transaction Identifiers

This section explains transaction-based replication using *global transaction identifiers* (GTIDs), introduced in MySQL 5.6.5. When using GTIDs, each transaction can be identified and tracked as it is committed on the originating server and applied by any replicas; this means that it is not necessary when using GTIDs to refer to log files or positions within those files when starting a new replica or failing over to a new source, which greatly simplifies these tasks. Because GTID-based replication is completely transaction-based, it is simple to determine whether sources and replicas are consistent; as long as all transactions committed on a source are also committed on a replica, consistency between the two is guaranteed. You can use either statement-based or row-based replication with GTIDs (see Section 17.1.2, “Replication Formats”); however, for best results, we recommend that you use the row-based format.

This section discusses the following topics:

- How GTIDs are defined and created, and how they are represented in the MySQL Server (see Section 17.1.3.1, “GTID Concepts”).

- A general procedure for setting up and starting GTID-based replication (see Section 17.1.3.2, “Setting Up Replication Using GTIDs”).

- Suggested methods for provisioning new replication servers when using GTIDs (see Section 17.1.3.3, “Using GTIDs for Failover and Scaleout”).

- Restrictions and limitations that you should be aware of when using GTID-based replication (see Section 17.1.3.4, “Restrictions on Replication with GTIDs”).

- A procedure for disabling GTIDs, which you must do if you have enabled GTIDs and are downgrading to a MySQL release that does not support GTIDs (see Section 17.1.3.5, “Disabling GTID Transactions”).

For information about MySQL Server options and variables relating to GTID-based replication, see Section 17.1.4.5, “Global Transaction ID Options and Variables”. See also Section 12.18, “Functions Used with Global Transaction IDs”, which describes SQL functions supported by MySQL 5.6 for use with GTIDs.

Note

GTIDs are not compatible or supported with the `NDB` storage engine used by NDB Cluster. Enabling GTIDs in NDB Cluster is very likely to cause problems with `NDB`, and to cause NDB Cluster Replication to fail as well.

17.1.3.1 GTID Concepts

A global transaction identifier (GTID) is a unique identifier created and associated with each transaction committed on the server of origin (source). This identifier is unique not only to the server on which it originated, but is unique across all servers in a given replication setup. There is a 1-to-1 mapping between all transactions and all GTIDs.

A GTID is represented as a pair of coordinates, separated by a colon character (`:`), as shown here:

```
GTID = source_id:transaction_id
```

The *`source_id`* identifies the originating server. Normally, the server's `server_uuid` is used for this purpose. The *`transaction_id`* is a sequence number determined by the order in which the transaction was committed on this server; for example, the first transaction to be committed has `1` as its *`transaction_id`*, and the tenth transaction to be committed on the same originating server is assigned a *`transaction_id`* of `10`. It is not possible for a transaction to have `0` as a sequence number in a GTID. For example, the twenty-third transaction to be committed originally on the server with the UUID `3E11FA47-71CA-11E1-9E33-C80AA9429562` has this GTID:

```
3E11FA47-71CA-11E1-9E33-C80AA9429562:23
```

This format is used to represent GTIDs in the output of statements such as `SHOW SLAVE STATUS` as well as in the binary log. They can also be seen when viewing the log file with `mysqlbinlog --base64-output=DECODE-ROWS` or in the output from `SHOW BINLOG EVENTS`.

As written in the output of statements such as `SHOW MASTER STATUS` or `SHOW SLAVE STATUS`, a sequence of GTIDs originating from the same server may be collapsed into a single expression, as shown here.

```
3E11FA47-71CA-11E1-9E33-C80AA9429562:1-5
```

The example just shown represents the first through fifth transactions originating on the MySQL Server whose `server_uuid` is `3E11FA47-71CA-11E1-9E33-C80AA9429562`.

In MySQL 5.6.6 and later, this format is also used to supply the argument required by the `START SLAVE` options `SQL_BEFORE_GTIDS` and `SQL_AFTER_GTIDS`.

GTID Sets

A GTID set is a set of global transaction identifiers which is represented as shown here:

```
gtid_set:
    uuid_set [, uuid_set] ...
    | ''

uuid_set:
    uuid:interval[:interval]...

uuid:
    hhhhhhhh-hhhh-hhhh-hhhh-hhhhhhhhhhhh

h:
    [0-9|A-F]

interval:
```

```
n[-n]

(n >= 1)
```

GTID sets are used in the MySQL Server in several ways. For example, the values stored by the `gtid_executed` and `gtid_purged` system variables are represented as GTID sets. In addition, the functions `GTID_SUBSET()` and `GTID_SUBTRACT()` require GTID sets as input.

GTIDs are always preserved between source and replica. This means that you can always determine the source for any transaction applied on any replica by examining its binary log. In addition, once a transaction with a given GTID is committed on a given server, any subsequent transaction having the same GTID is ignored by that server. Thus, a transaction committed on the source can be applied no more than once on the replica, which helps to guarantee consistency.

When GTIDs are in use, the replica has no need for any nonlocal data, such as the name of a file on the source and a position within that file. All necessary information for synchronizing with the source is obtained directly from the replication data stream. From the perspective of the database administrator or developer, GTIDs entirely take the place of the file-offset pairs previously required to determine points for starting, stopping, or resuming the flow of data between source and replica. This means that, when you are using GTIDs for replication, you do not need (or want) to include `MASTER_LOG_FILE` or `MASTER_LOG_POS` options in the `CHANGE MASTER TO` statement used to direct a replica to replicate from a given source; in place of these options, it is necessary only to enable the `MASTER_AUTO_POSITION` option introduced in MySQL 5.6.5. For the exact steps needed to configure and start sources and replicas using GTID-based replication, see Section 17.1.3.2, “Setting Up Replication Using GTIDs”.

The generation and lifecycle of a GTID consists of the following steps:

1. A transaction is executed and committed on the source.

   This transaction is assigned a GTID using the source's UUID and the smallest nonzero transaction sequence number not yet used on this server; the GTID is written to the source's binary log (immediately preceding the transaction itself in the log).

2. After the binary log data is transmitted to the replica and stored in the replica's relay log (using established mechanisms for this process—see Section 17.2, “Replication Implementation”, for details), the replica reads the GTID and sets the value of its `gtid_next` system variable as this GTID. This tells the replica that the next transaction must be logged using this GTID.

   The replica sets `gtid_next` in a session context.

3. The replica checks to make sure that this GTID has not already been used to log a transaction in its own binary log. If and only if this GTID has not been used, the replica then writes the GTID and applies the transaction (and writes the transaction to its binary log). By reading and checking the transaction's GTID first, before processing the transaction itself, the replica guarantees not only that no previous transaction having this GTID has been applied on the replica, but also that no other session has already read this GTID but has not yet committed the associated transaction. In other words, multiple clients are not permitted to apply the same transaction concurrently.

4. Because `gtid_next` is not empty, the replica does not attempt to generate a GTID for this transaction but instead writes the GTID stored in this variable, that is, the GTID obtained from the source, immediately preceding the transaction in its binary log.

17.1.3.2 Setting Up Replication Using GTIDs

This section describes a process for configuring and starting GTID-based replication in MySQL 5.6. This is a “cold start” procedure that assumes either that you are starting the replication source server for the first time, or that it is possible to stop it; for information about provisioning replicas using GTIDs from a running source, see Section 17.1.3.3, “Using GTIDs for Failover and Scaleout”.

The key steps in this startup process for the simplest possible GTID replication topology, consisting of one source and one replica, are as follows:

1. If replication is already running, synchronize both servers by making them read-only.
2. Stop both servers.
3. Restart both servers with GTIDs, binary logging, and replica update logging enabled, and with statements that are unsafe for GTID-based replication disabled. In addition, the servers should be started in read-only mode, and the replication SQL and I/O threads should be prevented from starting on the replica.

   The `mysqld` options necessary to start the servers as described are discussed in the example that follows later in this section.
4. Instruct the replica to use the source as the replication data source and to use auto-positioning. The SQL statements needed to accomplish this step are described in the example that follows later in this section.
5. Take a new backup. Binary logs containing transactions without GTIDs cannot be used on servers where GTIDs are enabled, so backups taken before this point cannot be used with your new configuration.
6. Start the replica, then disable read-only mode again on both servers, so that they can accept updates.

In the following example, two servers are already running as source and replica, using MySQL's “classic” file-based replication protocol.

Most of the steps that follow require the use of the MySQL `root` account or another MySQL user account that has the `SUPER` privilege. `mysqladmin shutdown` requires either the `SUPER` privilege or the `SHUTDOWN` privilege.

Step 1: Synchronize the servers. Make the servers read-only. To do this, enable the `read_only` system variable by executing the following statement on both servers:

```
mysql> SET @@GLOBAL.read_only = ON;
```

Wait for all ongoing transactions to commit or roll back. Then, allow the replica to catch up with the source. *It is extremely important that you make sure the replica has processed all updates before continuing*.

If you use binary logs for anything other than replication, for example to do point in time backup and restore, wait until you do not need the old binary logs containing transactions without GTIDs. Ideally, wait for the server to purge all binary logs, and wait for any existing backup to expire.

Important

It is important to understand that logs containing transactions without GTIDs cannot be used on servers where GTIDs are enabled. Before proceeding, you must be sure that transactions without GTIDs do not exist anywhere in the topology.

Step 2: Stop both servers. Stop each server using `mysqladmin` as shown here, where *`username`* is the user name for a MySQL user having sufficient privileges to shut down the server:

```
shell> mysqladmin -uusername -p shutdown
```

Then supply this user's password at the prompt.

Step 3: Restart both servers with GTIDs enabled. To enable binary logging with global transaction identifiers, each server must be started with GTID mode, binary logging, replica update logging enabled, and with statements that are unsafe for GTID-based replication disabled. In addition, you should prevent unwanted or accidental updates from being performed on either server by starting both in read-only mode. This means that both servers must be started with (at least) the options shown in the following invocation of `mysqld_safe`:

```
shell> mysqld_safe --gtid_mode=ON --log-bin --log-slave-updates --enforce-gtid-consistency &
```

In addition, you should start the replica with the `--skip-slave-start` option along with the other server options specified in the example just shown.

Note

`gtid_mode` is not a boolean, but an enumeration. Use one of the values `ON` or `OFF` only, when setting this option. Using a numeric value such as 0 or 1 can lead to unexpected results.

For more information about the `gtid_mode` and `enforce_gtid_consistency` system variables, see Section 17.1.4.5, “Global Transaction ID Options and Variables”.

Depending on your configuration, supply additional options to `mysqld_safe` or other `mysqld` startup script.

Step 4: Direct the replica to use the source. Tell the replica to use the replication source server as the data source, and to use GTID-based auto-positioning rather than file-based positioning. Execute a `CHANGE MASTER TO` statement on the replica, using the `MASTER_AUTO_POSITION` option to tell the replica that transactions are identified by GTIDs.

You may also need to supply appropriate values for the source's host name and port number as well as the user name and password for a replication user account which can be used by the replica to connect to the source; if these have already been set prior to Step 1 and no further changes need to be made, the corresponding options can safely be omitted from the statement shown here.

```
mysql> CHANGE MASTER TO
     >     MASTER_HOST = host,
     >     MASTER_PORT = port,
     >     MASTER_USER = user,
     >     MASTER_PASSWORD = password,
     >     MASTER_AUTO_POSITION = 1;
```

Neither the `MASTER_LOG_FILE` option nor the `MASTER_LOG_POS` option may be used with `MASTER_AUTO_POSITION` set equal to 1. Attempting to do so causes the `CHANGE MASTER TO` statement to fail with an error.

Step 5: Take a new backup. Existing backups that were made before you enabled GTIDs can no longer be used on these servers now that you have enabled GTIDs. Take a new backup at this point, so that you are not left without a usable backup.

For instance, you can execute `FLUSH LOGS` on the server where you are taking backups. Then either explicitly take a backup or wait for the next iteration of any periodic backup routine you may have set up.

Step 6: Start the replica and disable read-only mode. Start the replica like this:

```
mysql> START SLAVE;
```

Allow the source to begin accepting updates once again by running the following statement:

```
mysql> SET @@GLOBAL.read_only = OFF;
```

GTID-based replication should now be running, and you can begin (or resume) activity on the source as before. Section 17.1.3.3, “Using GTIDs for Failover and Scaleout”, discusses creation of new replicas when using GTIDs.

17.1.3.3 Using GTIDs for Failover and Scaleout

There are a number of techniques when using MySQL Replication with Global Transaction Identifiers (GTIDs) in MySQL 5.6.9 and later for provisioning a new replica which can then be used for scaleout, being promoted to source as necessary for failover. In this section, we discuss the four techniques listed here:

- Simple replication
- Copying data and transactions to the replica
- Injecting empty transactions
- Excluding transactions with gtid_purged

Global transaction identifiers were added to MySQL Replication for the purpose of simplifying in general management of the replication data flow and of failover activities in particular. Each identifier uniquely identifies a set of binary log events that together make up a transaction. GTIDs play a key role in applying changes to the database: the server automatically skips any transaction having an identifier which the server recognizes as one that it has processed before. This behavior is critical for automatic replication positioning and correct failover.

The mapping between identifiers and sets of events comprising a given transaction is captured in the binary log. This poses some challenges when provisioning a new server with data from another existing server. To reproduce the identifier set on the new server, it is necessary to copy the identifiers from the old server to the new one, and to preserve the relationship between the identifiers and the actual events This is neccessary for restoring a replica that is immediately available as a candidate to become a new source on failover or switchover.

Simple replication. This is the easiest way to reproduce all identifiers and transactions on a new server; you simply make the new server into the replica of a source that has the entire execution history, and enable global transaction identifiers on both servers. See Section 17.1.3.2, “Setting Up Replication Using GTIDs”, for more information.

Once replication is started, the new server copies the entire binary log from the source and thus obtains all information about all GTIDs.

This method is simple and effective, but requires the replica to read the binary log from the source; it can sometimes take a comparatively long time for the new replica to catch up with the source, so this method is not suitable for fast failover or restoring from backup. This section explains how to avoid fetching all of the execution history from the source by copying binary log files to the new server.

Copying data and transactions to the replica. Playing back the entire transaction history can be time-consuming, and represents a major bottleneck when setting up a new replica. To eliminate this requirement, a snapshot of the data set, the binary logs and the global transaction information the source contains is imported to the replica. The the binary log is played back, after which replication can be started, allowing the replica to become current with any remaining transactions.

There are several variants of this method, the difference being in the manner in which data dumps and transactions from binary logs are transferred to the replica, as outlined here:

Data Set

1. Use the `mysql` client to import a dump file created with `mysqldump`. Use the `--master-data` option to include binary logging information and `--set-gtid-purged` (available in MySQL 5.6.9 and later) to `AUTO` (the default) or `ON`, to include information about executed transactions. You should have `gtid_mode=ON` while importing the dump on the replica. (Bug #14832472)

2. Stop the replica, copy the contents of the source's data directory to the replica's data directory, then restart the replica.

Transaction History

If `gtid_mode` is not `ON`, restart the server with GTID mode enabled.

1. Import the binary log using `mysqlbinlog`, with the `--read-from-remote-server` and `--read-from-remote-master` options.

2. Copy the source's binary log files to the replica. You can make copies from the replica using `mysqlbinlog --read-from-remote-server --raw`. These can be read in to the replica in either of the following ways:

   - Update the replica's `binlog.index` file to point to the copied log files. Then execute a `CHANGE MASTER TO` statement in the `mysql` client to point to the first log file, and `START SLAVE` to read them.

   - Use `mysqlbinlog > file` (without the `--raw` option) to export the binary log files to SQL files that can be processed by the `mysql` client.

See also Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”.

This method has the advantage that a new server is available almost immediately; only those transactions that were committed while the snapshot or dump file was being replayed still need to be obtained from the existing source. This means that the replica's availability is not instantaneous, but only a relatively short amount of time should be required for the replica to catch up with these few remaining transactions.

Copying over binary logs to the target server in advance is usually faster than reading the entire transaction execution history from the source in real time. However, it may not always be feasible to move these files to the target when required, due to size or other considerations. The two remaining methods for provisioning a new replica discussed in this section use other means to transfer information about transactions to the new replica.

Injecting empty transactions. The source's global `gtid_executed` variable contains the set of all transactions executed on the source. Rather than copy the binary logs when taking a snapshot to provision a new server, you can instead note the content of `gtid_executed` on the server from which the snapshot was taken. Before adding the new server to the replication chain, simply commit an empty transaction on the new server for each transaction identifier contained in the source's `gtid_executed`, like this:

```
SET GTID_NEXT='aaa-bbb-ccc-ddd:N';

BEGIN;
COMMIT;

SET GTID_NEXT='AUTOMATIC';
```

Once all transaction identifiers have been reinstated in this way using empty transactions, you must flush and purge the replica's binary logs, as shown here, where *`N`* is the nonzero suffix of the current binary log file name:

```
FLUSH LOGS;
PURGE BINARY LOGS TO 'source-bin.00000N';
```

You should do this to prevent this server from flooding the replication stream with false transactions in the event that it is later promoted to source. (The `FLUSH LOGS` statement forces the creation of a new binary log file; `PURGE BINARY LOGS` purges the empty transactions, but retains their identifiers.)

This method creates a server that is essentially a snapshot, but in time is able to become a source as its binary log history converges with that of the replication stream (that is, as it catches up with the source or sources). This outcome is similar in effect to that obtained using the remaining provisioning method, which we discuss in the next few paragraphs.

Excluding transactions with gtid_purged. The source's global `gtid_purged` variable contains the set of all transactions that have been purged from the source's binary log. As with the method discussed previously (see Injecting empty transactions), you can record the value of `gtid_executed`

on the server from which the snapshot was taken (in place of copying the binary logs to the new server). Unlike the previous method, there is no need to commit empty transactions (or to issue `PURGE BINARY LOGS`); instead, you can set `gtid_purged` on the replica directly, based on the value of `gtid_executed` on the server from which the backup or snapshot was taken.

Note

Prior to MySQL 5.6.9, `gtid_purged` was not settable. (Bug #14797808)

As with the method using empty transactions, this method creates a server that is functionally a snapshot, but in time is able to become a source as its binary log history converges with that of the replication source server or group.

17.1.3.4 Restrictions on Replication with GTIDs

Because GTID-based replication is dependent on transactions, some features otherwise available in MySQL are not supported when using it. This section provides information about restrictions on and limitations of replication with GTIDs.

Updates involving nontransactional storage engines. When using GTIDs, updates to tables using nontransactional storage engines such as `MyISAM` cannot be made in the same statement or transaction as updates to tables using transactional storage engines such as `InnoDB`.

This restriction is due to the fact that updates to tables that use a nontransactional storage engine mixed with updates to tables that use a transactional storage engine within the same transaction can result in multiple GTIDs being assigned to the same transaction.

Such problems can also occur when the source and the replica use different storage engines for their respective versions of the same table, where one storage engine is transactional and the other is not.

In any of the cases just mentioned, the one-to-one correspondence between transactions and GTIDs is broken, with the result that GTID-based replication cannot function correctly.

CREATE TABLE ... SELECT statements. `CREATE TABLE ... SELECT` is not safe for statement-based replication. When using row-based replication, this statement is actually logged as two separate events—one for the creation of the table, and another for the insertion of rows from the source table into the new table just created. When this statement is executed within a transaction, it is possible in some cases for these two events to receive the same transaction identifier, which means that the transaction containing the inserts is skipped by the replica. Therefore, `CREATE TABLE ... SELECT` is not supported when using GTID-based replication.

Temporary tables. `CREATE TEMPORARY TABLE` and `DROP TEMPORARY TABLE` statements are not supported inside transactions when using GTIDs (that is, when the server was started with the `--enforce-gtid-consistency` option). It is possible to use these statements with GTIDs enabled, but only outside of any transaction, and only with `autocommit=1`.

Preventing execution of unsupported statements. In order to prevent execution of statements that would cause GTID-based replication to fail, all servers must be started with the `--enforce-gtid-consistency` option when enabling GTIDs. This causes statements of any of the types discussed previously in this section to fail with an error.

For information about other required startup options when enabling GTIDs, see Section 17.1.3.2, “Setting Up Replication Using GTIDs”.

`sql_slave_skip_counter` is not supported when using GTIDs. If you need to skip transactions, use the value of the source's `gtid_executed` variable instead; see Injecting empty transactions, for more information.

GTID mode and mysqldump. In MySQL 5.6.9 and later, it is possible to import a dump made using `mysqldump` into a MySQL Server running with GTID mode enabled, provided that there are no GTIDs in the target server's binary log.

Prior to MySQL 5.6.9, `mysqldump` did not record global transaction IDs, and it was necessary to use the binary log and `mysqlbinlog` to restore GTIDs. (Bug #14797808, Bug #14832472)

GTID mode and mysql_upgrade. Prior to MySQL 5.6.7, `mysql_upgrade` could not connect to a MySQL Server that was running with global transaction identifiers (GTIDs) enabled (`gtid_mode=ON`) unless `mysql_upgrade` was run with `--write-binlog=OFF`. Otherwise, `mysqld` had to be restarted with `gtid_mode=OFF` before running `mysql_upgrade`, then restarted with `gtid_mode=ON` afterwards. In MySQL 5.6.7 and later, where `mysql_upgrade` runs with `--write-binlog=OFF` by default. (Bug #13833710). Do not enable this option when the server is running with (`gtid_mode=ON`).

17.1.3.5 Disabling GTID Transactions

If you have enabled GTIDs in MySQL 5.6 and want to downgrade to a MySQL release that does not support GTIDs, you must carry out this procedure to disable GTIDs before downgrading. In MySQL 5.6, you must take the servers offline in order to disable GTIDs.

1. On each replica, disable auto-positioning by running the following statements:

```
STOP SLAVE;
CHANGE MASTER TO MASTER_AUTO_POSITION = 0, MASTER_LOG_FILE = file, \
MASTER_LOG_POS = position;
START SLAVE;
```

2. On each server, stop updates by running the following statement:

```
SET @@GLOBAL.READ_ONLY = ON;
```

3. Wait for all ongoing transactions to commit or roll back. Then, wait for a safe period of time, depending on your deployment, for all transactions that currently exist in any binary log to replicate to all replicas. *It is extremely important that you make sure all replicas have processed all updates before continuing*.

   If you use binary logs for anything other than replication, for example to do point in time backup and restore, wait until you do not need the old binary logs containing GTID transactions. Ideally, wait for the server to purge all binary logs, and wait for any existing backup to expire.

Important

It is important to understand that logs containing GTID transactions cannot be used on servers where GTIDs are disabled. Before proceeding, you must be sure that GTID transactions do not exist anywhere in the topology.

4. Stop each server using `mysqladmin` as shown here, where *`username`* is the user name for a MySQL user having sufficient privileges to shut down the server:

```
shell> mysqladmin -uusername -p shutdown
```

   Then supply this user's password at the prompt.

5. On each server, set `gtid_mode=OFF` and `enforce_gtid_consistency=OFF` in `my.cnf`.

6. Restart each server in read-only mode, using `mysqld_safe` or another `mysqld` startup script, and specifying the option `--read_only=ON` on the command line. Starting the servers in read-only mode prevents unwanted or accidental updates from being performed on any server.

7. Take a new backup at this point, so that you are not left without a usable backup. Existing backups that were made before you disabled GTIDs can no longer be used on these servers now that you have disabled GTIDs. For instance, you can execute `FLUSH LOGS` on the server where you are taking backups. Then either explicitly take a backup or wait for the next iteration of any periodic backup routine you may have set up.

8. On each server, re-enable updates by running the following statement:

```
SET @@GLOBAL.READ_ONLY = OFF;
```

If you want to downgrade to an earlier version of MySQL, you can do so now, using the normal downgrade procedure.

17.1.4 Replication and Binary Logging Options and Variables

The following sections contain information about `mysqld` options and server variables that are used in replication and for controlling the binary log. Options and variables for use on replication source servers and replicas are covered separately, as are options and variables relating to binary logging. A set of quick-reference tables providing basic information about these options and variables is also included.

Of particular importance is the `server_id` system variable.

Command-Line Format	`--server-id=#`
System Variable	`server_id`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`4294967295`

This variable specifies the server ID.

On a replication source server and each replica, you *must* specify `server_id` to establish a unique replication ID in the range from 1 to $2^{32} - 1$. “Unique”, means that each ID must be different from every other ID in use by any other replication source server or replica. For additional information, see Section 17.1.4.2, “Replication Source Options and Variables”, and Section 17.1.4.3, “Replica Server Options and Variables”.

If you do not specify `server_id`, the default server ID is 0. If the server ID is set to 0, binary logging takes place, but a source with a server ID of 0 refuses any connections from replicas, and a replica with a server ID of 0 refuses to connect to a source. Note that although you can change the server ID dynamically to a nonzero value, doing so does not enable replication to start immediately. You must change the server ID and then restart the server to initialize the replica.

In MySQL 5.6, whether the server ID is set to 0 explicitly or the default is allowed to be used, the server sets the `server_id` system variable to 1; this is a known issue that is fixed in MySQL 5.7.

For more information, see Section 17.1.1.2, “Setting the Replica Configuration”.

`server_uuid`

Beginning with MySQL 5.6, the server generates a true UUID in addition to the `server_id` value supplied by the user. This is available as the global, read-only `server_uuid` system variable.

System Variable	`server_uuid`
Scope	Global
Dynamic	No
Type	String

When starting, the MySQL server automatically obtains a UUID as follows:

1. Attempt to read and use the UUID written in the file *`data_dir`*`/auto.cnf` (where *`data_dir`* is the server's data directory).

2. If *`data_dir`*`/auto.cnf` is not found, generate a new UUID and save it to this file, creating the file if necessary.

The `auto.cnf` file has a format similar to that used for `my.cnf` or `my.ini` files. In MySQL 5.6, `auto.cnf` has only a single `[auto]` section containing a single `server_uuid` setting and value; the file's contents appear similar to what is shown here:

```
[auto]
server_uuid=8a94f357-aab4-11df-86ab-c80aa9429562
```

Important

The `auto.cnf` file is automatically generated; do not attempt to write or modify this file.

Also beginning with MySQL 5.6, when using MySQL replication, sources and replicas know one another's UUIDs. The value of a replica's UUID can be seen in the output of `SHOW SLAVE HOSTS`. Once `START SLAVE` has been executed (but not before), the value of the source's UUID is available on the replica in the output of `SHOW SLAVE STATUS`.

Note

Issuing a `STOP SLAVE` or `RESET SLAVE` statement does *not* reset the source's UUID as used on the replica.

A server's `server_uuid` is also used in GTIDs for transactions originating on that server. For more information, see Section 17.1.3, “Replication with Global Transaction Identifiers”.

When starting, the replication I/O thread generates an error and aborts if its source's UUID is equal to its own unless the `--replicate-same-server-id` option has been set. In addition, the replication I/O thread generates a warning if either of the following is true:

- No source having the expected `server_uuid` exists.
- The source's `server_uuid` has changed, although no `CHANGE MASTER TO` statement has ever been executed.

Note

The addition of the `server_uuid` system variable in MySQL 5.6 does not change the requirement for setting a unique `server_id` value for each MySQL server as part of preparing and running MySQL replication, as described earlier in this section.

17.1.4.1 Replication and Binary Logging Option and Variable Reference

The following two sections provide basic information about the MySQL command-line options and system variables applicable to replication and the binary log.

Replication Options and Variables

The command-line options and system variables in the following list relate to replication source servers and replicas. Section 17.1.4.2, “Replication Source Options and Variables” provides more detailed information about options and variables relating to replication source servers. For more information about options and variables relating to replicas, see Section 17.1.4.3, “Replica Server Options and Variables”.

- `abort-slave-event-count`: Option used by mysql-test for debugging and testing of replication.
- `auto_increment_increment`: AUTO_INCREMENT columns are incremented by this value.
- `auto_increment_offset`: Offset added to AUTO_INCREMENT columns.

- `binlog_gtid_simple_recovery`: Controls how binary logs are iterated during GTID recovery.
- `Com_change_master`: Count of CHANGE MASTER TO statements.
- `Com_show_master_status`: Count of SHOW MASTER STATUS statements.
- `Com_show_slave_hosts`: Count of SHOW REPLICAS and SHOW SLAVE HOSTS statements.
- `Com_show_slave_status`: Count of SHOW REPLICA STATUS and SHOW SLAVE STATUS statements.
- `Com_slave_start`: Count of START REPLICA and START SLAVE statements.
- `Com_slave_stop`: Count of STOP REPLICA and STOP SLAVE statements.
- `disconnect-slave-event-count`: Option used by mysql-test for debugging and testing of replication.
- `enforce_gtid_consistency`: Prevents execution of statements that cannot be logged in transactionally safe manner.
- `expire_logs_days`: Purge binary logs after this many days.
- `gtid_executed`: Global: All GTIDs in binary log (global) or current transaction (session). Read-only.
- `gtid_mode`: Controls whether GTID based logging is enabled and what type of transactions logs can contain.
- `gtid_next`: Specifies GTID for next statement to execute; see documentation for details.
- `gtid_owned`: Set of GTIDs owned by this client (session), or by all clients, together with thread ID of owner (global). Read-only.
- `gtid_purged`: Set of all GTIDs that have been purged from binary log.
- `init_slave`: Statements that are executed when replica connects to source.
- `log_bin_trust_function_creators`: If equal to 0 (default), then when --log-bin is used, stored function creation is allowed only to users having SUPER privilege and only if function created does not break binary logging.
- `master-info-file`: Location and name of file that remembers source and where I/O replication thread is in source's binary log.
- `master-retry-count`: Number of tries replica makes to connect to source before giving up.
- `master_info_repository`: Whether to write connection metadata repository, containing source information and replication I/O thread location in source's binary log, to file or table.
- `max_relay_log_size`: If nonzero, relay log is rotated automatically when its size exceeds this value. If zero, size at which rotation occurs is determined by value of max_binlog_size.
- `relay_log`: Location and base name to use for relay logs.
- `relay_log_basename`: Complete path to relay log, including file name.
- `relay_log_index`: Location and name to use for file that keeps list of last relay logs.
- `relay_log_info_file`: File name for applier metadata repository in which replica records information about relay logs.
- `relay_log_info_repository`: Whether to write location of replication SQL thread in relay logs to file or table.

- `relay_log_purge`: Determines whether relay logs are purged.
- `relay_log_recovery`: Whether automatic recovery of relay log files from source at startup is enabled; must be enabled for crash-safe replica.
- `relay_log_space_limit`: Maximum space to use for all relay logs.
- `replicate-do-db`: Tells replication SQL thread to restrict replication to specified database.
- `replicate-do-table`: Tells replication SQL thread to restrict replication to specified table.
- `replicate-ignore-db`: Tells replication SQL thread not to replicate to specified database.
- `replicate-ignore-table`: Tells replication SQL thread not to replicate to specified table.
- `replicate-rewrite-db`: Updates to database with different name from original.
- `replicate-same-server-id`: In replication, if enabled, do not skip events having our server id.
- `replicate-wild-do-table`: Tells replication SQL thread to restrict replication to tables that match specified wildcard pattern.
- `replicate-wild-ignore-table`: Tells replication SQL thread not to replicate to tables that match given wildcard pattern.
- `report_host`: Host name or IP of replica to be reported to source during replica registration.
- `report_password`: Arbitrary password which replica server should report to source; not same as password for replication user account.
- `report_port`: Port for connecting to replica reported to source during replica registration.
- `report_user`: Arbitrary user name which replica server should report to source; not same as name used for replication user account.
- `Rpl_semi_sync_master_clients`: Number of semisynchronous replicas.
- `rpl_semi_sync_master_enabled`: Whether semisynchronous replication is enabled on source.
- `Rpl_semi_sync_master_net_avg_wait_time`: Average time source has waited for replies from replica.
- `Rpl_semi_sync_master_net_wait_time`: Total time source has waited for replies from replica.
- `Rpl_semi_sync_master_net_waits`: Total number of times source waited for replies from replica.
- `Rpl_semi_sync_master_no_times`: Number of times source turned off semisynchronous replication.
- `Rpl_semi_sync_master_no_tx`: Number of commits not acknowledged successfully.
- `Rpl_semi_sync_master_status`: Whether semisynchronous replication is operational on source.
- `Rpl_semi_sync_master_timefunc_failures`: Number of times source failed when calling time functions.
- `rpl_semi_sync_master_timeout`: Number of milliseconds to wait for replica acknowledgment.
- `rpl_semi_sync_master_trace_level`: Semisynchronous replication debug trace level on source.
- `Rpl_semi_sync_master_tx_avg_wait_time`: Average time source waited for each transaction.
- `Rpl_semi_sync_master_tx_wait_time`: Total time source waited for transactions.

- `Rpl_semi_sync_master_tx_waits`: Total number of times source waited for transactions.
- `rpl_semi_sync_master_wait_no_slave`: Whether source waits for timeout even with no replicas.
- `Rpl_semi_sync_master_wait_pos_backtraverse`: Total number of times source has waited for event with binary coordinates lower than events waited for previously.
- `Rpl_semi_sync_master_wait_sessions`: Number of sessions currently waiting for replica replies.
- `Rpl_semi_sync_master_yes_tx`: Number of commits acknowledged successfully.
- `rpl_semi_sync_slave_enabled`: Whether semisynchronous replication is enabled on replica.
- `Rpl_semi_sync_slave_status`: Whether semisynchronous replication is operational on replica.
- `rpl_semi_sync_slave_trace_level`: Semisynchronous replication debug trace level on replica.
- `rpl_stop_slave_timeout`: Number of seconds that STOP REPLICA or STOP SLAVE waits before timing out.
- `server_uuid`: Server's globally unique ID, automatically (re)generated at server start.
- `show-slave-auth-info`: Show user name and password in SHOW REPLICAS and SHOW SLAVE HOSTS on this source.
- `simplified_binlog_gtid_recovery`: Renamed to binlog_gtid_simple_recovery.
- `skip-slave-start`: If set, replication is not autostarted when replica server starts.
- `slave_load_tmpdir`: Location where replica should put its temporary files when replicating LOAD DATA statements.
- `slave_net_timeout`: Number of seconds to wait for more data from source/replica connection before aborting read.
- `slave-skip-errors`: Tells replication thread to continue replication when query returns error from provided list.
- `slave_checkpoint_group`: Maximum number of transactions processed by multithreaded replica before checkpoint operation is called to update progress status. Not supported by NDB Cluster.
- `slave_checkpoint_period`: Update progress status of multithreaded replica and flush relay log info to disk after this number of milliseconds. Not supported by NDB Cluster.
- `slave_compressed_protocol`: Use compression of source/replica protocol.
- `slave_exec_mode`: Allows for switching replication thread between IDEMPOTENT mode (key and some other errors suppressed) and STRICT mode; STRICT mode is default, except for NDB Cluster, where IDEMPOTENT is always used.
- `Slave_heartbeat_period`: Replica's replication heartbeat interval, in seconds.
- `Slave_last_heartbeat`: Shows when latest heartbeat signal was received, in TIMESTAMP format.
- `slave_max_allowed_packet`: Maximum size, in bytes, of packet that can be sent from replication source server to replica; overrides max_allowed_packet.
- `Slave_open_temp_tables`: Number of temporary tables that replication SQL thread currently has open.
- `slave_parallel_workers`: Number of applier threads for executing replication transactions in parallel. 0 disables replica multithreading. Not supported by MySQL Cluster.

- `slave_pending_jobs_size_max`: Maximum size of replica worker queues holding events not yet applied.
- `Slave_received_heartbeats`: Number of heartbeats received by replica since previous reset.
- `Slave_retried_transactions`: Total number of times since startup that replication SQL thread has retried transactions.
- `slave_rows_search_algorithms`: Determines search algorithms used for replica update batching. Any 2 or 3 from this list: INDEX_SEARCH, TABLE_SCAN, HASH_SCAN.
- `Slave_rows_last_search_algorithm_used`: Search algorithm most recently used by this replica to locate rows for row-based replication (index, table, or hash scan).
- `Slave_running`: State of this server as replica (replication I/O thread status).
- `slave_transaction_retries`: Number of times replication SQL thread retries transaction in case it failed with deadlock or elapsed lock wait timeout, before giving up and stopping.
- `slave_type_conversions`: Controls type conversion mode on replica. Value is list of zero or more elements from this list: ALL_LOSSY, ALL_NON_LOSSY. Set to empty string to disallow type conversions between source and replica.
- `sql_log_bin`: Controls binary logging for current session.
- `sql_slave_skip_counter`: Number of events from source that replica should skip. Not compatible with GTID replication.
- `sync_master_info`: Synchronize master.info to disk after every #th event.
- `sync_relay_log`: Synchronize relay log to disk after every #th event.
- `sync_relay_log_info`: Synchronize relay.info file to disk after every #th event.

For a listing of all command-line options, system variables, and status variables used with `mysqld`, see Section 5.1.3, “Server Option, System Variable, and Status Variable Reference”.

Binary Logging Options and Variables

The command-line options and system variables in the following list relate to the binary log. Section 17.1.4.4, “Binary Log Options and Variables”, provides more detailed information about options and variables relating to binary logging. For additional general information about the binary log, see Section 5.4.4, “The Binary Log”.

- `binlog-checksum`: Enable/disable binary log checksums.
- `binlog-do-db`: Limits binary logging to specific databases.
- `binlog_format`: Specifies format of binary log.
- `binlog-ignore-db`: Tells source that updates to given database should not be written to binary log.
- `binlog-row-event-max-size`: Binary log max event size.
- `Binlog_cache_disk_use`: Number of transactions which used temporary file instead of binary log cache.
- `binlog_cache_size`: Size of cache to hold SQL statements for binary log during transaction.
- `Binlog_cache_use`: Number of transactions that used temporary binary log cache.
- `binlog_checksum`: Enable/disable binary log checksums.
- `binlog_direct_non_transactional_updates`: Causes updates using statement format to nontransactional engines to be written directly to binary log. See documentation before using.

- `binlog_error_action`: Controls what happens when server cannot write to binary log.
- `binlog_max_flush_queue_time`: How long to read transactions before flushing to binary log.
- `binlog_order_commits`: Whether to commit in same order as writes to binary log.
- `binlog_row_image`: Use full or minimal images when logging row changes.
- `binlog_rows_query_log_events`: When enabled, enables logging of rows query log events when using row-based logging. Disabled by default. Do not enable when producing logs for pre-5.6 replicas/readers.
- `Binlog_stmt_cache_disk_use`: Number of nontransactional statements that used temporary file instead of binary log statement cache.
- `binlog_stmt_cache_size`: Size of cache to hold nontransactional statements for binary log during transaction.
- `Binlog_stmt_cache_use`: Number of statements that used temporary binary log statement cache.
- `binlogging_impossible_mode`: Deprecated and later removed. Use binlog_error_action instead.
- `Com_show_binlog_events`: Count of SHOW BINLOG EVENTS statements.
- `Com_show_binlogs`: Count of SHOW BINLOGS statements.
- `log-bin`: Base name for binary log files.
- `log-bin-index`: Name of binary log index file.
- `log_bin`: Whether binary log is enabled.
- `log_bin_basename`: Path and base name for binary log files.
- `log_bin_use_v1_row_events`: Whether server is using version 1 binary log row events.
- `log_slave_updates`: Whether replica should log updates performed by its replication SQL thread to its own binary log.
- `master_verify_checksum`: Cause source to examine checksums when reading from binary log.
- `max-binlog-dump-events`: Option used by mysql-test for debugging and testing of replication.
- `max_binlog_cache_size`: Can be used to restrict total size used to cache multi-statement transaction.
- `max_binlog_size`: Binary log is rotated automatically when size exceeds this value.
- `max_binlog_stmt_cache_size`: Can be used to restrict total size used to cache all nontransactional statements during transaction.
- `slave-sql-verify-checksum`: Cause replica to examine checksums when reading from relay log.
- `slave_sql_verify_checksum`: Cause replica to examine checksums when reading from relay log.
- `sporadic-binlog-dump-fail`: Option used by mysql-test for debugging and testing of replication.
- `sync_binlog`: Synchronously flush binary log to disk after every #th event.

For a listing of all command-line options, system and status variables used with `mysqld`, see Section 5.1.3, “Server Option, System Variable, and Status Variable Reference”.

17.1.4.2 Replication Source Options and Variables

This section describes the server options and system variables that you can use on replication source servers. You can specify the options either on the command line or in an option file. You can specify system variable values using `SET`.

On the source and each replica, you must set the `server_id` system variable to establish a unique replication ID. For each server, you should pick a unique positive integer in the range from 1 to 2^{32} − 1, and each ID must be different from every other ID in use by any other replication source server or replica. Example: `server-id=3`.

For options used on the source for controlling binary logging, see Section 17.1.4.4, “Binary Log Options and Variables”.

Startup Options for Replication Source Servers

The following list describes startup options for controlling replication source servers. Replication-related system variables are discussed later in this section.

- `--show-slave-auth-info`

Command-Line Format	`--show-slave-auth-info[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

Display replica user names and passwords in the output of `SHOW SLAVE HOSTS` on the replication source server for replicas started with the `--report-user` and `--report-password` options.

System Variables Used on Replication Sources

The following system variables are used to control replication source servers:

- `auto_increment_increment`

Command-Line Format	`--auto-increment-increment=#`
System Variable	`auto_increment_increment`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`1`
Minimum Value	`1`
Maximum Value	`65535`

`auto_increment_increment` and `auto_increment_offset` are intended for use with source-to-source replication, and can be used to control the operation of `AUTO_INCREMENT` columns. Both variables have global and session values, and each can assume an integer value between 1 and 65,535 inclusive. Setting the value of either of these two variables to 0 causes its value to be set to 1 instead. Attempting to set the value of either of these two variables to an integer greater than 65,535 or less than 0 causes its value to be set to 65,535 instead. Attempting to set the value of `auto_increment_increment` or `auto_increment_offset` to a noninteger value produces an error, and the actual value of the variable remains unchanged.

Note

`auto_increment_increment` is also supported for use with `NDB` tables.

These two variables affect `AUTO_INCREMENT` column behavior as follows:

- `auto_increment_increment` controls the interval between successive column values. For example:

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 1     |
| auto_increment_offset    | 1     |
+--------------------------+-------+
2 rows in set (0.00 sec)

mysql> CREATE TABLE autoinc1
    -> (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
  Query OK, 0 rows affected (0.04 sec)

mysql> SET @@auto_increment_increment=10;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 10    |
| auto_increment_offset    | 1     |
+--------------------------+-------+
2 rows in set (0.01 sec)

mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
|   1 |
|  11 |
|  21 |
|  31 |
+-----+
4 rows in set (0.00 sec)
```

- `auto_increment_offset` determines the starting point for the `AUTO_INCREMENT` column value. Consider the following, assuming that these statements are executed during the same session as the example given in the description for `auto_increment_increment`:

```
mysql> SET @@auto_increment_offset=5;
Query OK, 0 rows affected (0.00 sec)

mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 10    |
| auto_increment_offset    | 5     |
+--------------------------+-------+
2 rows in set (0.00 sec)

mysql> CREATE TABLE autoinc2
    -> (col INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO autoinc2 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT col FROM autoinc2;
+-----+
| col |
```

```
+-----+
|   5 |
|  15 |
|  25 |
|  35 |
+-----+
4 rows in set (0.02 sec)
```

When the value of `auto_increment_offset` is greater than that of `auto_increment_increment`, the value of `auto_increment_offset` is ignored.

If either of these variables is changed, and then new rows inserted into a table containing an `AUTO_INCREMENT` column, the results may seem counterintuitive because the series of `AUTO_INCREMENT` values is calculated without regard to any values already present in the column, and the next value inserted is the least value in the series that is greater than the maximum existing value in the `AUTO_INCREMENT` column. The series is calculated like this:

`auto_increment_offset` + *N* × `auto_increment_increment`

where *N* is a positive integer value in the series [1, 2, 3, ...]. For example:

```
mysql> SHOW VARIABLES LIKE 'auto_inc%';
+--------------------------+-------+
| Variable_name            | Value |
+--------------------------+-------+
| auto_increment_increment | 10    |
| auto_increment_offset    | 5     |
+--------------------------+-------+
2 rows in set (0.00 sec)

mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
|   1 |
|  11 |
|  21 |
|  31 |
+-----+
4 rows in set (0.00 sec)

mysql> INSERT INTO autoinc1 VALUES (NULL), (NULL), (NULL), (NULL);
Query OK, 4 rows affected (0.00 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> SELECT col FROM autoinc1;
+-----+
| col |
+-----+
|   1 |
|  11 |
|  21 |
|  31 |
|  35 |
|  45 |
|  55 |
|  65 |
+-----+
8 rows in set (0.00 sec)
```

The values shown for `auto_increment_increment` and `auto_increment_offset` generate the series 5 + *N* × 10, that is, [5, 15, 25, 35, 45, ...]. The highest value present in the `col` column prior to the `INSERT` is 31, and the next available value in the `AUTO_INCREMENT` series is 35, so the inserted values for `col` begin at that point and the results are as shown for the `SELECT` query.

It is not possible to restrict the effects of these two variables to a single table; these variables control the behavior of all `AUTO_INCREMENT` columns in *all* tables on the MySQL server. If the global value of either variable is set, its effects persist until the global value is changed or overridden by

setting the session value, or until `mysqld` is restarted. If the local value is set, the new value affects `AUTO_INCREMENT` columns for all tables into which new rows are inserted by the current user for the duration of the session, unless the values are changed during that session.

The default value of `auto_increment_increment` is 1. See Section 17.4.1.1, “Replication and AUTO_INCREMENT”.

- `auto_increment_offset`

Command-Line Format	`--auto-increment-offset=#`
System Variable	`auto_increment_offset`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`1`
Minimum Value	`1`
Maximum Value	`65535`

This variable has a default value of 1. For more information, see the description for `auto_increment_increment`.

Note

`auto_increment_offset` is also supported for use with `NDB` tables.

- `rpl_semi_sync_master_enabled`

Command-Line Format	`--rpl-semi-sync-master-enabled[={OFF|ON}]`
System Variable	`rpl_semi_sync_master_enabled`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

Controls whether semisynchronous replication is enabled on the source. To enable or disable the plugin, set this variable to `ON` or `OFF` (or 1 or 0), respectively. The default is `OFF`.

This variable is available only if the source-side semisynchronous replication plugin is installed.

- `rpl_semi_sync_master_timeout`

Command-Line Format	`--rpl-semi-sync-master-timeout=#`
System Variable	`rpl_semi_sync_master_timeout`
Scope	Global
Dynamic	Yes
Type	Integer

Default Value	10000

A value in milliseconds that controls how long the source waits on a commit for acknowledgment from a replica before timing out and reverting to asynchronous replication. The default value is 10000 (10 seconds).

This variable is available only if the source-side semisynchronous replication plugin is installed.

- rpl_semi_sync_master_trace_level

Command-Line Format	--rpl-semi-sync-master-trace-level=#
System Variable	rpl_semi_sync_master_trace_level
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	32

The semisynchronous replication debug trace level on the source. Four levels are defined:

- 1 = general level (for example, time function failures)
- 16 = detail level (more verbose information)
- 32 = net wait level (more information about network waits)
- 64 = function level (information about function entry and exit)

This variable is available only if the source-side semisynchronous replication plugin is installed.

- rpl_semi_sync_master_wait_no_slave

Command-Line Format	--rpl-semi-sync-master-wait-no-slave[={OFF|ON}]
System Variable	rpl_semi_sync_master_wait_no_slave
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	ON

With semisynchronous replication, for each transaction, the source waits until timeout for acknowledgment of receipt from some semisynchronous replica. If no response occurs during this period, the source reverts to normal replication. This variable controls whether the source waits for the timeout to expire before reverting to normal replication even if the replica count drops to zero during the timeout period.

If the value is ON (the default), it is permissible for the replica count to drop to zero during the timeout period (for example, if replicas disconnect). The source still waits for the timeout, so as long as some replica reconnects and acknowledges the transaction within the timeout interval, semisynchronous replication continues.

If the value is OFF, the source reverts to normal replication if the replica count drops to zero during the timeout period.

This variable is available only if the source-side semisynchronous replication plugin is installed.

17.1.4.3 Replica Server Options and Variables

- Startup Options for Replica Servers
- Options for Logging Replica Status to Tables
- Obsolete Replica Options
- System Variables Used on Replica Servers

Specify the options either on the command line or in an option file. Many of the options can be set while the server is running by using the `CHANGE MASTER TO` statement. Specify system variable values using `SET`.

Server ID. On the source and each slave, you must set the `server_id` system variable to establish a unique replication ID in the range from 1 to $2^{32} - 1$. “Unique” means that each ID must be different from every other ID in use by any other replication source server or replica. Example `my.cnf` file:

```
[mysqld]
server-id=3
```

Startup Options for Replica Servers

This section explains startup options for controlling replicas. Many of these options can be set while the server is running by using the `CHANGE MASTER TO` statement. Others, such as the `--replicate-*` options, can be set only when the replica server starts. Replication-related system variables are discussed later in this section.

- `--abort-slave-event-count`

Command-Line Format	`--abort-slave-event-count=#`
Type	Integer
Default Value	`0`
Minimum Value	`0`

When this option is set to some positive integer *`value`* other than 0 (the default) it affects replication behavior as follows: After the replication SQL thread has started, *`value`* log events are permitted to be executed; after that, the replication SQL thread does not receive any more events, just as if the network connection from the source were cut. The thread continues to run, and the output from `SHOW SLAVE STATUS` displays `Yes` in both the `Slave_IO_Running` and the `Slave_SQL_Running` columns, but no further events are read from the relay log.

This option is used internally by the MySQL test suite for replication testing and debugging. It is not intended for use in a production setting.

- `--disconnect-slave-event-count`

Command-Line Format	`--disconnect-slave-event-count=#`
Type	Integer
Default Value	`0`

This option is used internally by the MySQL test suite for replication testing and debugging.

- `--log-slow-slave-statements`

Command-Line Format	`--log-slow-slave-statements[={OFF|ON}]` (5.6.10)
Removed	5.6.11
Type	Boolean
Default Value	`OFF`

This command-line option was removed in MySQL 5.6.11 and replaced by the `log_slow_slave_statements` system variable. The system variable can be set on the command line or in option files the same way as the option, so there is no need for any changes at server startup, but the system variable also makes it possible to examine or set the value at runtime.

- `--log-warnings[=level]`

Command-Line Format	`--log-warnings[=#]`
System Variable	`log_warnings`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`1`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

Causes the server to record more messages to the error log about what it is doing. With respect to replication, the server generates warnings that it succeeded in reconnecting after a network or connection failure, and provides information about how each replication thread started. This option is enabled (1) by default; to disable it, use `--log-warnings=0`. If the value is greater than 1, aborted connections are written to the error log, and access-denied errors for new connection attempts are written. See Section B.3.2.10, “Communication Errors and Aborted Connections”.

Note

The effects of this option are not limited to replication. It affects diagnostic messages across a spectrum of server activities.

- `--master-info-file=file_name`

Command-Line Format	`--master-info-file=file_name`
Type	File name
Default Value	`master.info`

The name to use for the file in which the replica records information about the replication source server. The default name is `master.info` in the data directory. For information about the format of this file, see Section 17.2.2.2, “Replication Metadata Repositories”.

- `--master-retry-count=count`

Command-Line Format	`--master-retry-count=#`
Deprecated	Yes
Type	Integer
Default Value	`86400`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

The number of times that the replica tries to connect to the replication source server before giving up. Reconnects are attempted at intervals set by the `MASTER_CONNECT_RETRY` option of the `CHANGE MASTER TO` statement (default 60). Reconnection attempts are triggered when the replica reaches its connection timeout (specified by the `slave_net_timeout` system variable) without receiving data from the source. The default value is 86400. A value of 0 means “infinite”; the replica attempts to connect forever.

This option is deprecated; expect it be removed in a future MySQL release. Applications should be updated to use the `MASTER_RETRY_COUNT` option of the `CHANGE MASTER TO` statement instead.

- `--max-relay-log-size=size`

Command-Line Format	`--max-relay-log-size=#`
System Variable	`max_relay_log_size`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1073741824`

The size at which the server rotates relay log files automatically. If this value is nonzero, the relay log is rotated automatically when its size exceeds this value. If this value is zero (the default), the size at which relay log rotation occurs is determined by the value of `max_binlog_size`. For more information, see Section 17.2.2.1, “The Relay Log”.

- `--relay-log-purge={0|1}`

Command-Line Format	`--relay-log-purge[={OFF|ON}]`
System Variable	`relay_log_purge`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`ON`

Disable or enable automatic purging of relay logs as soon as they are no longer needed. The default value is 1 (enabled). This is a global variable that can be changed dynamically with `SET GLOBAL relay_log_purge = N`. Disabling purging of relay logs when enabling the `--relay-log-recovery` option puts data consistency at risk.

- `--relay-log-space-limit=size`

Command-Line Format	`--relay-log-space-limit=#`
System Variable	`relay_log_space_limit`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`

Maximum Value (32-bit platforms)	`4294967295`

This option places an upper limit on the total size in bytes of all relay logs on the replica. A value of 0 means “no limit”. This is useful for a replica server host that has limited disk space. When the limit is reached, the replication I/O thread stops reading binary log events from the replication source server until the SQL thread has caught up and deleted some unused relay logs. Note that this limit is not absolute: There are cases where the SQL thread needs more events before it can delete relay logs. In that case, the I/O thread exceeds the limit until it becomes possible for the SQL thread to delete some relay logs because not doing so would cause a deadlock. You should not set `--relay-log-space-limit` to less than twice the value of `--max-relay-log-size` (or `--max-binlog-size` if `--max-relay-log-size` is 0). In that case, there is a chance that the I/O thread waits for free space because `--relay-log-space-limit` is exceeded, but the SQL thread has no relay log to purge and is unable to satisfy the I/O thread. This forces the I/O thread to ignore `--relay-log-space-limit` temporarily.

- `--replicate-do-db=`*`db_name`*

Command-Line Format	`--replicate-do-db=name`
Type	String

The effects of this option depend on whether statement-based or row-based replication is in use.

Statement-based replication. Tell the replication SQL thread to restrict replication to statements where the default database (that is, the one selected by `USE`) is *`db_name`*. To specify more than one database, use this option multiple times, once for each database; however, doing so does *not* replicate cross-database statements such as `UPDATE` *`some_db.some_table`* `SET foo='bar'` while a different database (or no database) is selected.

Warning

To specify multiple databases you *must* use multiple instances of this option. Because database names can contain commas, if you supply a comma separated list, then the list is treated as the name of a single database.

An example of what does not work as you might expect when using statement-based replication: If the replica is started with `--replicate-do-db=sales` and you issue the following statements on the source, the `UPDATE` statement is *not* replicated:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

The main reason for this “check just the default database” behavior is that it is difficult from the statement alone to know whether it should be replicated (for example, if you are using multiple-table `DELETE` statements or multiple-table `UPDATE` statements that act across multiple databases). It is also faster to check only the default database rather than all databases if there is no need.

Row-based replication. Tells the replication SQL thread to restrict replication to database *`db_name`*. Only tables belonging to *`db_name`* are changed; the current database has no effect on this. Suppose that the replica is started with `--replicate-do-db=sales` and row-based replication is in effect, and then the following statements are run on the source:

```
USE prices;
UPDATE sales.february SET amount=amount+100;
```

The `february` table in the `sales` database on the replica is changed in accordance with the `UPDATE` statement; this occurs whether or not the `USE` statement was issued. However, issuing the

following statements on the source has no effect on the replica when using row-based replication and `--replicate-do-db=sales`:

```
USE prices;
UPDATE prices.march SET amount=amount-25;
```

Even if the statement `USE prices` were changed to `USE sales`, the `UPDATE` statement's effects would still not be replicated.

Another important difference in how `--replicate-do-db` is handled in statement-based replication as opposed to row-based replication occurs with regard to statements that refer to multiple databases. Suppose that the replica is started with `--replicate-do-db=db1`, and the following statements are executed on the source:

```
USE db1;
UPDATE db1.table1, db2.table2 SET db1.table1.col1 = 10, db2.table2.col2 = 20;
```

If you are using statement-based replication, then both tables are updated on the replica. However, when using row-based replication, only `table1` is affected on the replica; since `table2` is in a different database, `table2` on the replica is not changed by the `UPDATE`. Now suppose that, instead of the `USE db1` statement, a `USE db4` statement had been used:

```
USE db4;
UPDATE db1.table1, db2.table2 SET db1.table1.col1 = 10, db2.table2.col2 = 20;
```

In this case, the `UPDATE` statement would have no effect on the replica when using statement-based replication. However, if you are using row-based replication, the `UPDATE` would change `table1` on the replica, but not `table2`—in other words, only tables in the database named by `--replicate-do-db` are changed, and the choice of default database has no effect on this behavior.

If you need cross-database updates to work, use `--replicate-wild-do-table=db_name.%` instead. See Section 17.2.3, “How Servers Evaluate Replication Filtering Rules”.

Note

This option affects replication in the same manner that `--binlog-do-db` affects binary logging, and the effects of the replication format on how `--replicate-do-db` affects replication behavior are the same as those of the logging format on the behavior of `--binlog-do-db`.

This option has no effect on `BEGIN`, `COMMIT`, or `ROLLBACK` statements.

- `--replicate-ignore-db=db_name`

Command-Line Format	`--replicate-ignore-db=name`
Type	String

As with `--replicate-do-db`, the effects of this option depend on whether statement-based or row-based replication is in use.

Statement-based replication. Tells the replication SQL thread not to replicate any statement where the default database (that is, the one selected by `USE`) is *db_name*.

Row-based replication. Tells the replication SQL thread not to update any tables in the database *db_name*. The default database has no effect.

When using statement-based replication, the following example does not work as you might expect. Suppose that the replica is started with `--replicate-ignore-db=sales` and you issue the following statements on the source:

```
USE prices;
```

```
UPDATE sales.january SET amount=amount+1000;
```

The `UPDATE` statement *is* replicated in such a case because `--replicate-ignore-db` applies only to the default database (determined by the `USE` statement). Because the `sales` database was specified explicitly in the statement, the statement has not been filtered. However, when using row-based replication, the `UPDATE` statement's effects are *not* propagated to the replica, and the replica's copy of the `sales.january` table is unchanged; in this instance, `--replicate-ignore-db=sales` causes *all* changes made to tables in the source's copy of the `sales` database to be ignored by the replica.

To specify more than one database to ignore, use this option multiple times, once for each database. Because database names can contain commas, if you supply a comma separated list, then the list is treated as the name of a single database.

You should not use this option if you are using cross-database updates and you do not want these updates to be replicated. See Section 17.2.3, “How Servers Evaluate Replication Filtering Rules”.

If you need cross-database updates to work, use `--replicate-wild-ignore-table=`*`db_name`*`.%` instead. See Section 17.2.3, “How Servers Evaluate Replication Filtering Rules”.

Note

This option affects replication in the same manner that `--binlog-ignore-db` affects binary logging, and the effects of the replication format on how `--replicate-ignore-db` affects replication behavior are the same as those of the logging format on the behavior of `--binlog-ignore-db`.

This option has no effect on `BEGIN`, `COMMIT`, or `ROLLBACK` statements.

- `--replicate-do-table=`*`db_name.tbl_name`*

Command-Line Format	`--replicate-do-table=name`
Type	String

Creates a replication filter by telling the replication SQL thread to restrict replication to a given table. To specify more than one table, use this option multiple times, once for each table. This works for both cross-database updates and default database updates, in contrast to `--replicate-do-db`. See Section 17.2.3, “How Servers Evaluate Replication Filtering Rules”.

This option affects only statements that apply to tables. It does not affect statements that apply only to other database objects, such as stored routines. To filter statements operating on stored routines, use one or more of the `--replicate-*-db` options.

- `--replicate-ignore-table=`*`db_name.tbl_name`*

Command-Line Format	`--replicate-ignore-table=name`
Type	String

Creates a replication filter by telling the replication SQL thread not to replicate any statement that updates the specified table, even if any other tables might be updated by the same statement. To specify more than one table to ignore, use this option multiple times, once for each table. This works for cross-database updates, in contrast to `--replicate-ignore-db`. See Section 17.2.3, “How Servers Evaluate Replication Filtering Rules”.

This option affects only statements that apply to tables. It does not affect statements that apply only to other database objects, such as stored routines. To filter statements operating on stored routines, use one or more of the `--replicate-*-db` options.

- `--replicate-rewrite-db=from_name->to_name`

Command-Line Format	`--replicate-rewrite-db=old_name->new_name`
Type	String

Tells the replica to create a replication filter that translates the specified database to *`to_name`* if it was *`from_name`* on the source. Only statements involving tables are affected, not statements such as `CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE`.

To specify multiple rewrites, use this option multiple times. The server uses the first one with a *`from_name`* value that matches. The database name translation is done *before* the `--replicate-*` rules are tested. You can also create such a filter by issuing a `CHANGE REPLICATION FILTER REPLICATE_REWRITE_DB` statement.

If you use the `--replicate-rewrite-db` option on the command line and the `>` character is special to your command interpreter, quote the option value. For example:

```
shell> mysqld --replicate-rewrite-db="olddb->newdb"
```

The effect of the `--replicate-rewrite-db` option differs depending on whether statement-based or row-based binary logging format is used for the query. With statement-based format, DML statements are translated based on the current database, as specified by the `USE` statement. With row-based format, DML statements are translated based on the database where the modified table exists. DDL statements are always filtered based on the current database, as specified by the `USE` statement, regardless of the binary logging format.

To ensure that rewriting produces the expected results, particularly in combination with other replication filtering options, follow these recommendations when you use the `--replicate-rewrite-db` option:

- Create the *`from_name`* and *`to_name`* databases manually on the source and the replica with different names.
- If you use statement-based or mixed binary logging format, do not use cross-database queries, and do not specify database names in queries. For both DDL and DML statements, rely on the `USE` statement to specify the current database, and use only the table name in queries.
- If you use row-based binary logging format exclusively, for DDL statements, rely on the `USE` statement to specify the current database, and use only the table name in queries. For DML statements, you can use a fully qualified table name (*`db`*.*`table`*) if you want.

If these recommendations are followed, it is safe to use the `--replicate-rewrite-db` option in combination with table-level replication filtering options such as `--replicate-do-table`.

- `--replicate-same-server-id`

Command-Line Format	`--replicate-same-server-id[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

To be used on replica servers. Usually you should use the default setting of 0, to prevent infinite loops caused by circular replication. If set to 1, the replica does not skip events having its own server ID. Normally, this is useful only in rare configurations. Cannot be set to 1 if `log_slave_updates` is enabled. By default, the replication I/O thread does not write binary log events to the relay log if they have the replica's server ID (this optimization helps save disk usage). If you want to use `--replicate-same-server-id`, be sure to start the replica with this option before you make the replica read its own events that you want the replication SQL thread to execute.

- `--replicate-wild-do-table=db_name.tbl_name`

Command-Line Format	`--replicate-wild-do-table=name`
Type	String

Creates a replication filter by telling the replication SQL thread to restrict replication to statements where any of the updated tables match the specified database and table name patterns. Patterns can contain the `%` and `_` wildcard characters, which have the same meaning as for the `LIKE` pattern-matching operator. To specify more than one table, use this option multiple times, once for each table. This works for cross-database updates. See Section 17.2.3, "How Servers Evaluate Replication Filtering Rules".

This option applies to tables, views, and triggers. It does not apply to stored procedures and functions, or events. To filter statements operating on the latter objects, use one or more of the `--replicate-*-db` options.

Example: `--replicate-wild-do-table=foo%.bar%` replicates only updates that use a table where the database name starts with `foo` and the table name starts with `bar`.

If the table name pattern is `%`, it matches any table name and the option also applies to database-level statements (`CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE`). For example, if you use `--replicate-wild-do-table=foo%.%`, database-level statements are replicated if the database name matches the pattern `foo%`.

To include literal wildcard characters in the database or table name patterns, escape them with a backslash. For example, to replicate all tables of a database that is named `my_own%db`, but not replicate tables from the `my1ownAABCdb` database, you should escape the `_` and `%` characters like this: `--replicate-wild-do-table=my\_own\%db`. If you use the option on the command line, you might need to double the backslashes or quote the option value, depending on your command interpreter. For example, with the `bash` shell, you would need to type `--replicate-wild-do-table=my\\_own\\%db`.

- `--replicate-wild-ignore-table=`*`db_name.tbl_name`*

Command-Line Format	`--replicate-wild-ignore-table=name`
Type	String

Creates a replication filter which keeps the replication SQL thread from replicating a statement in which any table matches the given wildcard pattern. To specify more than one table to ignore, use this option multiple times, once for each table. This works for cross-database updates. See Section 17.2.3, "How Servers Evaluate Replication Filtering Rules".

Example: `--replicate-wild-ignore-table=foo%.bar%` does not replicate updates that use a table where the database name starts with `foo` and the table name starts with `bar`.

For information about how matching works, see the description of the `--replicate-wild-do-table` option. The rules for including literal wildcard characters in the option value are the same as for `--replicate-wild-ignore-table` as well.

- `--skip-slave-start`

Command-Line Format	`--skip-slave-start[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

Tells the replica server not to start the replication threads when the server starts. To start the threads later, use a `START SLAVE` statement.

- `--slave-skip-errors=[err_code1,err_code2,...|all|ddl_exist_errors]`

Command-Line Format	`--slave-skip-errors=name`
System Variable	`slave_skip_errors`
Scope	Global
Dynamic	No
Type	String
Default Value	`OFF`

Valid Values	`OFF`
	`[list of error codes]`
	`all`
	`ddl_exist_errors`

Normally, replication stops when an error occurs on the replica. This gives you the opportunity to resolve the inconsistency in the data manually. This option tells the replication SQL thread to continue replication when a statement returns any of the errors listed in the option value.

Do not use this option unless you fully understand why you are getting errors. If there are no bugs in your replication setup and client programs, and no bugs in MySQL itself, an error that stops replication should never occur. Indiscriminate use of this option results in replicas becoming hopelessly out of synchrony with the source, with you having no idea why this has occurred.

For error codes, you should use the numbers provided by the error message in the replica's error log and in the output of `SHOW SLAVE STATUS`. Appendix B, *Error Messages and Common Problems*, lists server error codes.

You can also (but should not) use the very nonrecommended value of `all` to cause the replica to ignore all error messages and keeps going regardless of what happens. Needless to say, if you use `all`, there are no guarantees regarding the integrity of your data. Please do not complain (or file bug reports) in this case if the replica's data is not anywhere close to what it is on the source. *You have been warned*.

MySQL 5.6 as well as MySQL NDB Cluster 7.3 and later support an additional shorthand value `ddl_exist_errors`, which is equivalent to the error code list `1007,1008,1050,1051,1054,1060,1061,1068,1094,1146`.

Examples:

```
--slave-skip-errors=1062,1053
--slave-skip-errors=all
--slave-skip-errors=ddl_exist_errors
```

- `--slave-sql-verify-checksum={0|1}`

Command-Line Format	`--slave-sql-verify-checksum[={OFF|ON}]`
Type	Boolean
Default Value	`ON`

When this option is enabled, the replica examines checksums read from the relay log. In the event of a mismatch, replication stops with an error.

Options for Logging Replica Status to Tables

MySQL 5.6 and higher supports logging of replica status information to tables rather than files. Writing of the replica's connection metadata repository and applier metadata repository can be configured separately using these two system variables:

- `master_info_repository`
- `relay_log_info_repository`

For information about these variables, see Section 17.1.4.3, “Replica Server Options and Variables”.

The replication metadata repositories and their contents are considered local to a given MySQL Server. They are not replicated, and changes to them are not written to the binary log.

For more information, see Section 17.2.2, “Relay Log and Replication Metadata Repositories”.

Obsolete Replica Options

The following options have been removed and replaced by the functions of the `CHANGE MASTER TO ...` statement (see Section 13.4.2.1, “CHANGE MASTER TO Statement”). If you attempt to start `mysqld` with any of these options in MySQL 5.6, the server aborts with an `unknown variable` error.

- `--master-host`
- `--master-user`
- `--master-password`
- `--master-port`
- `--master-connect-retry`
- `--master-ssl`
- `--master-ssl-ca`
- `--master-ssl-capath`
- `--master-ssl-cert`
- `--master-ssl-cipher`
- `--master-ssl-key`

System Variables Used on Replica Servers

The following list describes system variables for controlling replica servers. They can be set at server startup and some of them can be changed at runtime using `SET`. Server options used with replicas are listed earlier in this section.

- `init_slave`

Command-Line Format	`--init-slave=name`
System Variable	`init_slave`
Scope	Global
Dynamic	Yes
Type	String

This variable is similar to `init_connect`, but is a string to be executed by a replica server each time the replication SQL thread starts. The format of the string is the same as for the `init_connect` variable.

Note

The replication SQL thread sends an acknowledgment to the client before it executes `init_slave`. Therefore, it is not guaranteed that `init_slave` has been executed when `START SLAVE` returns. See Section 13.4.2.5, "START SLAVE Statement", for more information.

- `log_slow_slave_statements`

Command-Line Format	`--log-slow-slave-statements[={OFF|ON}]`
Introduced	5.6.11
System Variable	`log_slow_slave_statements`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

When the slow query log is enabled, this variable enables logging for queries that have taken more than `long_query_time` seconds to execute on the replica. Note that if row-based replication is in use (`binlog_format=ROW`), `log_slow_slave_statements` has no effect. Queries are only added to the replica's slow query log when they are logged in statement format in the binary log, that is, when `binlog_format=STATEMENT` is set, or when `binlog_format=MIXED` is set and the statement is logged in statement format. Slow queries that are logged in row format when `binlog_format=MIXED` is set, or that are logged when `binlog_format=ROW` is set, are not added to the replica's slow query log, even if `log_slow_slave_statements` is enabled.

Setting `log_slow_slave_statements` has no immediate effect. The state of the variable applies on all subsequent `START SLAVE` statements. Also note that the global setting for `long_query_time` applies for the lifetime of the SQL thread. If you change that setting, you must stop and restart the replication SQL thread to implement the change there (for example, by issuing `STOP SLAVE` and `START SLAVE` statements with the `SQL_THREAD` option).

- `master_info_repository`

Command-Line Format	`--master-info-repository={FILE|TABLE}`
System Variable	`master_info_repository`
Scope	Global
Dynamic	Yes
Type	String
Default Value	`FILE`

Valid Values	`FILE`
	
	`TABLE`

The setting of this variable determines whether the replica server logs source status and connection information to a `FILE` (`master.info`), or to a `TABLE` (`mysql.slave_master_info`).

The setting of this variable also has a direct influence on the effect had by the setting of the `sync_master_info` system variable; see that variable description for further information.

For `FILE` logging, you can change the name of the file using the `--master-info-file` server option.

- `max_relay_log_size`

Command-Line Format	`--max-relay-log-size=#`
System Variable	`max_relay_log_size`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1073741824`

If a write by a replica to its relay log causes the current log file size to exceed the value of this variable, the replica rotates the relay logs (closes the current file and opens the next one). If `max_relay_log_size` is 0, the server uses `max_binlog_size` for both the binary log and the relay log. If `max_relay_log_size` is greater than 0, it constrains the size of the relay log, which enables you to have different sizes for the two logs. You must set `max_relay_log_size` to between 4096 bytes and 1GB (inclusive), or to 0. The default value is 0. See Section 17.2.1, “Replication Threads”.

- `relay_log`

Command-Line Format	`--relay-log=file_name`
System Variable	`relay_log`
Scope	Global
Dynamic	No
Type	File name

The base name for the relay log. The default base name is *`host_name`*`-relay-bin`.

The server writes the file in the data directory unless the base name is given with a leading absolute path name to specify a different directory. The server creates relay log files in sequence by adding a numeric suffix to the base name.

Due to the manner in which MySQL parses server options, if you specify this variable at server startup, you must supply a value; *the default base name is used only if the option is not actually specified*. If you specify the `relay_log` system variable at server startup without specifying a value, unexpected behavior is likely to result; this behavior depends on the other options used, the order in which they are specified, and whether they are specified on the command line or in an option file. For more information about how MySQL handles server options, see Section 4.2.2, “Specifying Program Options”.

If you specify this variable, the value specified is also used as the base name for the relay log index file. You can override this behavior by specifying a different relay log index file base name using the `relay_log_index` system variable.

When the server reads an entry from the index file, it checks whether the entry contains a relative path. If it does, the relative part of the path is replaced with the absolute path set using the

`relay_log` system variable. An absolute path remains unchanged; in such a case, the index must be edited manually to enable the new path or paths to be used.

You may find the `relay_log` system variable useful in performing the following tasks:

- Creating relay logs whose names are independent of host names.
- If you need to put the relay logs in some area other than the data directory because your relay logs tend to be very large and you do not want to decrease `max_relay_log_size`.
- To increase speed by using load-balancing between disks.

You can obtain the relay log file name (and path) from the `relay_log_basename` system variable.

- `relay_log_basename`

System Variable	`relay_log_basename`
Scope	Global
Dynamic	No
Type	File name
Default Value	`datadir + '/' + hostname + '-relay-bin'`

Holds the base name and complete path to the relay log file. The maximum variable length is 256. This variable is set by the server and is read only.

- `relay_log_index`

Command-Line Format	`--relay-log-index=file_name`
System Variable	`relay_log_index`
Scope	Global
Dynamic	No
Type	File name
Default Value	`*host_name*-relay-bin.index`

The name for the relay log index file. The maximum variable length is 256. The default name is *`host_name`*`-relay-bin.index` in the data directory, where *`host_name`* is the name of the replica server.

Due to the manner in which MySQL parses server options, if you specify this variable at server startup, you must supply a value; *the default base name is used only if the option is not actually specified*. If you specify the `relay_log_index` system variable at server startup without specifying a value, unexpected behavior is likely to result; this behavior depends on the other options used, the order in which they are specified, and whether they are specified on the command line or in an option file. For more information about how MySQL handles server options, see Section 4.2.2, “Specifying Program Options”.

- `relay_log_info_file`

Command-Line Format	`--relay-log-info-file=file_name`
System Variable	`relay_log_info_file`
Scope	Global
Dynamic	No
Type	File name

Default Value	`relay-log.info`

The name of the file in which the replica records information about the relay logs. The default name is `relay-log.info` in the data directory. For information about the format of this file, see Section 17.2.2.2, “Replication Metadata Repositories”.

- `relay_log_info_repository`

Command-Line Format	`--relay-log-info-repository=value`
System Variable	`relay_log_info_repository`
Scope	Global
Dynamic	Yes
Type	String
Default Value	`FILE`
Valid Values	`FILE` `TABLE`

This variable determines whether the replica's position in the relay logs is written to a `FILE` (`relay-log.info`) or to a `TABLE` (`mysql.slave_relay_log_info`).

The setting of this variable also has a direct influence on the effect had by the setting of the `sync_relay_log_info` system variable; see that variable descrption for further information.

This option can be used to make a replica resilient to unexpected halts. See Section 17.3.2, “Handling an Unexpected Halt of a Replica Server”, for more information.

- `relay_log_purge`

Command-Line Format	`--relay-log-purge[={OFF|ON}]`
System Variable	`relay_log_purge`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`ON`

Disables or enables automatic purging of relay log files as soon as they are not needed any more. The default value is 1 (`ON`).

- `relay_log_recovery`

Command-Line Format	`--relay-log-recovery[={OFF|ON}]`
System Variable	`relay_log_recovery`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`OFF`

If enabled, this variable enables automatic relay log recovery immediately following server startup. The recovery process creates a new relay log file, initializes the SQL thread position to this new relay log, and initializes the I/O thread to the SQL thread position. Reading of the relay log from the source then continues. This global variable is read-only at runtime. Its value can set with the `--relay-log-recovery` option at replica startup, which should be used following an unexpected halt of a

replica to ensure that no possibly corrupted relay logs are processed. This option can be enabled to make a replica resilient to unexpected halts. See Section 17.3.2, “Handling an Unexpected Halt of a Replica Server” for more information.

This variable also interacts with the `relay_log_purge` variable, which controls purging of logs when they are no longer needed. Enabling `relay_log_recovery` when `relay_log_purge` is disabled risks reading the relay log from files that were not purged, leading to data inconsistency.

When `relay_log_recovery` is enabled and the replica has stopped due to errors encountered while running in multithreaded mode, you cannot execute `CHANGE MASTER TO` if there are any gaps in the log. The solution in this situation is to issue `START SLAVE UNTIL SQL_AFTER_MTS_GAPS` to ensure that all gaps are processed before switching back to single-threaded mode or executing a `CHANGE MASTER TO` statement.

- `relay_log_space_limit`

Command-Line Format	`--relay-log-space-limit=#`
System Variable	`relay_log_space_limit`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

The maximum amount of space to use for all relay logs.

- `report_host`

Command-Line Format	`--report-host=host_name`
System Variable	`report_host`
Scope	Global
Dynamic	No
Type	String

The host name or IP address of the replica to be reported to the replication source server during replica registration. This value appears in the output of `SHOW SLAVE HOSTS` on the source server. Leave the value unset if you do not want the replica to register itself with the source.

Note

It is not sufficient for the source to simply read the IP address of the replica from the TCP/IP socket after the replica connects. Due to NAT and other routing issues, that IP may not be valid for connecting to the replica from the source or other hosts.

- `report_password`

Command-Line Format	`--report-password=name`
System Variable	`report_password`
Scope	Global

Dynamic	No
Type	String

The account password to be reported to the source during replica registration. This value appears in the output of `SHOW SLAVE HOSTS` on the source if the source was started with `--show-slave-auth-info`.

Although the name of this variable might imply otherwise, `report_password` is not connected to the MySQL user privilege system and so is not necessarily (or even likely to be) the same as the password for the MySQL replication user account.

- `report_port`

Command-Line Format	`--report-port=port_num`
System Variable	`report_port`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`[slave_port]`
Minimum Value	`0`
Maximum Value	`65535`

The TCP/IP port number for connecting to the replica, to be reported to the source during replica registration. Set this only if the replica is listening on a nondefault port or if you have a special tunnel from the source or other clients to the replica. If you are not sure, do not use this option.

The default value for this option is the port number actually used by the replica. This is also the default value displayed by `SHOW SLAVE HOSTS`.

- `report_user`

Command-Line Format	`--report-user=name`
System Variable	`report_user`
Scope	Global
Dynamic	No
Type	String

The account user name of the replica to be reported to the replication source server during replica registration. This value appears in the output of `SHOW SLAVE HOSTS` on the source if the source was started with `--show-slave-auth-info`.

Although the name of this variable might imply otherwise, `report_user` is not connected to the MySQL user privilege system and so is not necessarily (or even likely to be) the same as the name of the MySQL replication user account.

- `rpl_semi_sync_slave_enabled`

Command-Line Format	`--rpl-semi-sync-slave-enabled[={OFF|ON}]`
System Variable	`rpl_semi_sync_slave_enabled`
Scope	Global
Dynamic	Yes
Type	Boolean

Default Value	OFF

Controls whether semisynchronous replication is enabled on the replica. To enable or disable the plugin, set this variable to ON or OFF (or 1 or 0), respectively. The default is OFF.

This variable is available only if the replica-side semisynchronous replication plugin is installed.

- rpl_semi_sync_slave_trace_level

Command-Line Format	--rpl-semi-sync-slave-trace-level=#
System Variable	rpl_semi_sync_slave_trace_level
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	32

The semisynchronous replication debug trace level on the replica. See rpl_semi_sync_master_trace_level for the permissible values.

This variable is available only if the replica-side semisynchronous replication plugin is installed.

- rpl_stop_slave_timeout

Command-Line Format	--rpl-stop-slave-timeout=seconds
Introduced	5.6.13
System Variable	rpl_stop_slave_timeout
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	31536000
Minimum Value	2
Maximum Value	31536000

In MySQL 5.6.13 and higher, you can control the length of time (in seconds) that STOP SLAVE waits before timing out by setting this variable. This can be used to avoid deadlocks between STOP SLAVE and other SQL statements using different client connections to the replica.

The maximum and default value of rpl_stop_slave_timeout is 31536000 seconds (1 year). The minimum is 2 seconds. Changes to this variable take effect for subsequent STOP SLAVE statements.

This variable affects only the client that issues a STOP SLAVE statement. When the timeout is reached, the issuing client returns an error message stating that the command execution is incomplete. The client then stops waiting for the replication threads to stop, but the replication threads continue to try to stop, and the STOP SLAVE instruction remains in effect. Once the replication threads are no longer busy, the STOP SLAVE statement is executed and the replica stops.

- slave_checkpoint_group

Command-Line Format	--slave-checkpoint-group=#
System Variable	slave_checkpoint_group
Scope	Global
Dynamic	Yes

Type	Integer
Default Value	`512`
Minimum Value	`32`
Maximum Value	`524280`
Block Size	`8`

Sets the maximum number of transactions that can be processed by a multithreaded replica before a checkpoint operation is called to update its status as shown by `SHOW SLAVE STATUS`. Setting this variable has no effect on replicas for which multithreading is not enabled.

Note

Multithreaded replicas are not currently supported by NDB Cluster, which silently ignores the setting for this variable. See Section 18.6.3, “Known Issues in NDB Cluster Replication”, for more information.

This variable works in combination with the `slave_checkpoint_period` system variable in such a way that, when either limit is exceeded, the checkpoint is executed and the counters tracking both the number of transactions and the time elapsed since the last checkpoint are reset.

The minimum allowed value for this variable is 32, unless the server was built using `-DWITH_DEBUG`, in which case the minimum value is 1. The effective value is always a multiple of 8; you can set it to a value that is not such a multiple, but the server rounds it down to the next lower multiple of 8 before storing the value. (*Exception*: No such rounding is performed by the debug server.) Regardless of how the server was built, the default value is 512, and the maximum allowed value is 524280.

- `slave_checkpoint_period`

Command-Line Format	`--slave-checkpoint-period=#`
System Variable	`slave_checkpoint_period`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`300`
Minimum Value	`1`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

Unit	milliseconds

Sets the maximum time (in milliseconds) that is allowed to pass before a checkpoint operation is called to update the status of a multithreaded replica as shown by SHOW SLAVE STATUS. Setting this variable has no effect on replicas for which multithreading is not enabled.

Note

Multithreaded replicas are not currently supported by NDB Cluster, which silently ignores the setting for this variable. See Section 18.6.3, "Known Issues in NDB Cluster Replication", for more information.

This variable works in combination with the slave_checkpoint_group system variable in such a way that, when either limit is exceeded, the checkpoint is executed and the counters tracking both the number of transactions and the time elapsed since the last checkpoint are reset.

The minimum allowed value for this variable is 1, unless the server was built using -DWITH_DEBUG, in which case the minimum value is 0. Regardless of how the server was built, the default value is 300, and the maximum possible value is 4294967296 (4GB).

- slave_compressed_protocol

Command-Line Format	--slave-compressed-protocol[={OFF|ON}]
System Variable	slave_compressed_protocol
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	OFF

Whether to use compression of the source/replica protocol if both source and replica support it. If this variable is disabled (the default), connections are uncompressed. See also Section 4.2.6, "Connection Compression Control".

- slave_exec_mode

Command-Line Format	--slave-exec-mode=mode
System Variable	slave_exec_mode
Scope	Global
Dynamic	Yes
Type	Enumeration
Default Value	IDEMPOTENT (NDB) STRICT (Other)
Valid Values	IDEMPOTENT STRICT

Controls how a replication thread resolves conflicts and errors during replication. IDEMPOTENT mode causes suppression of duplicate-key and no-key-found errors; STRICT means no such suppression takes place.

IDEMPOTENT mode is intended for use in multi-source replication, circular replication, and some other special replication scenarios for NDB Cluster Replication. (See Section 18.6.10, "NDB Cluster Replication: Bidirectional and Circular Replication", and Section 18.6.11, "NDB Cluster

Replication Conflict Resolution", for more information.) NDB Cluster ignores any value explicitly set for `slave_exec_mode`, and always treats it as `IDEMPOTENT`.

In MySQL Server 5.6, `STRICT` mode is the default value.

For storage engines other than `NDB`, *`IDEMPOTENT` mode should be used only when you are absolutely sure that duplicate-key errors and key-not-found errors can safely be ignored*. It is meant to be used in fail-over scenarios for NDB Cluster where multi-source replication or circular replication is employed, and is not recommended for use in other cases.

- `slave_load_tmpdir`

Command-Line Format	`--slave-load-tmpdir=dir_name`
System Variable	`slave_load_tmpdir`
Scope	Global
Dynamic	No
Type	Directory name
Default Value	`Value of --tmpdir`

The name of the directory where the replica creates temporary files. Setting this variable takes effect for all replication channels immediately, including running channels. The variable value is by default equal to the value of the `tmpdir` system variable, or the default that applies when that system variable is not specified.

When the replication SQL thread replicates a `LOAD DATA` statement, it extracts the file to be loaded from the relay log into temporary files, and then loads these into the table. If the file loaded on the source is huge, the temporary files on the replica are huge, too. Therefore, it might be advisable to use this option to tell the replica to put temporary files in a directory located in some file system that has a lot of available space. In that case, the relay logs are huge as well, so you might also want to set the `relay_log` system variable to place the relay logs in that file system.

The directory specified by this option should be located in a disk-based file system (not a memory-based file system) so that the temporary files used to replicate `LOAD DATA` statements can survive machine restarts. The directory also should not be one that is cleared by the operating system during the system startup process. However, replication can now continue after a restart if the temporary files have been removed.

- `slave_max_allowed_packet`

Command-Line Format	`--slave-max-allowed-packet=#`
System Variable	`slave_max_allowed_packet`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`1073741824`
Minimum Value	`1024`
Maximum Value	`1073741824`

This variable sets the maximum packet size for the replication SQL and I/O threads, so that large updates using row-based replication do not cause replication to fail because an update exceeded `max_allowed_packet`.

This global variable always has a value that is a positive integer multiple of 1024; if you set it to some value that is not, the value is rounded down to the next highest multiple of 1024 for it is stored or used; setting `slave_max_allowed_packet` to 0 causes 1024 to be used. (A truncation warning

is issued in all such cases.) The default and maximum value is 1073741824 (1 GB); the minimum is 1024.

- `slave_net_timeout`

Command-Line Format	`--slave-net-timeout=#`
System Variable	`slave_net_timeout`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`3600`
Minimum Value	`1`
Maximum Value	`4294967295`

The number of seconds to wait for more data from the source before the replica considers the connection broken, aborts the read, and tries to reconnect. The first retry occurs immediately after the timeout. The interval between retries is controlled by the `MASTER_CONNECT_RETRY` option for the `CHANGE MASTER TO` statement, and the number of reconnection attempts is limited by the `--master-retry-count` option. The default is 3600 seconds (one hour).

- `slave_parallel_workers`

Command-Line Format	`--slave-parallel-workers=#`
System Variable	`slave_parallel_workers`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1024`

Sets the number of worker threads on the replica for executing replication events (transactions) in parallel. Setting this variable to 0 (the default) disables parallel execution. The maximum is 1024.

Note

Multithreaded replicas are not currently supported by NDB Cluster, which silently ignores the setting for this variable. See Section 18.6.3, “Known Issues in NDB Cluster Replication”, for more information.

When parallel execution is enabled, the replication SQL thread acts as the coordinator for the worker threads, among which transactions are distributed on a per-database basis. This means that a worker thread on the replica can process successive transactions on a given database without waiting for updates to other databases to complete. The current implementation of multithreading on the replica assumes that the data is partitioned per database, and that updates within a given database occur in the same relative order as they do on the source, in order to work correctly. However, transactions do not need to be coordinated between any two databases.

Due to the fact that transactions on different databases can occur in a different order on the replica than on the source, checking for the most recently executed transaction does not guarantee that all previous transactions from the source have been executed on the replica. This has implications for logging and recovery when using a multithreaded replica. For information about how to interpret binary logging information when using multithreading on the replica, see Section 13.7.5.35, “SHOW

SLAVE STATUS Statement". In addition, this means that `START SLAVE UNTIL` is not supported with a multithreaded replica.

When multithreading is enabled, `slave_transaction_retries` is treated as equal to 0, and cannot be changed. (Currently, retrying of transactions is not supported with multithreaded replicas.)

You should also note that enforcing foreign key relationships between tables in different databases causes multithreaded replicas to use sequential rather than parallel mode, which can have a negative impact on performance. (Bug #14092635)

- `slave_pending_jobs_size_max`

Command-Line Format	`--slave-pending-jobs-size-max=#`
System Variable	`slave_pending_jobs_size_max`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`16M`
Minimum Value	`1024`
Maximum Value	`16EiB`
Unit	`bytes`
Block Size	`1024`

For multithreaded replicas, this variable sets the maximum amount of memory (in bytes) available to worker queues on the replica holding events not yet applied. Setting this variable has no effect on replicas for which multithreading is not enabled.

The minimum possible value for this variable is 1024; the default is 16MB. The maximum possible value is 18446744073709551615 (16 exabytes). Values that are not exact multiples of 1024 are rounded down to the next-highest multiple of 1024 prior to being stored.

The value of this variable is a soft limit and can be set to match the normal workload. If an unusually large event exceeds this size, the transaction is held until all the worker threads on the replica have empty queues, and then processed. All subsequent transactions are held until the large transaction has been completed.

- `slave_rows_search_algorithms`

Command-Line Format	`--slave-rows-search-algorithms=value`
System Variable	`slave_rows_search_algorithms`
Scope	Global
Dynamic	Yes
Type	Set
Default Value	`TABLE_SCAN,INDEX_SCAN`

Valid Values	`TABLE_SCAN,INDEX_SCAN`
	
	`INDEX_SCAN,HASH_SCAN`
	
	`TABLE_SCAN,HASH_SCAN`

	`TABLE_SCAN,INDEX_SCAN,HASH_SCAN` (equivalent to INDEX_SCAN,HASH_SCAN)

When preparing batches of rows for row-based logging and replication, this variable controls how the rows are searched for matches—that is, whether or not hashing is used for searches using a primary or unique key, some other key, or using no key at all. Setting this variable takes effect for all replication channels immediately, including running channels. option.

Specify a comma-separated list of any 2 (or all 3) values from the list `INDEX_SCAN`, `TABLE_SCAN`, `HASH_SCAN`. The value is expected as a string, so if set at runtime rather than at server startup, the value must be quoted. In addition, the value must not contain any spaces. Possible combinations (lists) and their effects are shown in the following table:

Index used / option value	**`INDEX_SCAN,HASH_SCAN` or `INDEX_SCAN,TABLE_SCAN,HASH_SCAN`**	**`INDEX_SCAN,TABLE_SCAN`**	**`TABLE_SCAN,HASH_SCAN`**
Primary key or unique key	Index scan	Index scan	Hash scan over index
(Other) Key	Hash scan over index	Index scan	Hash scan over index
No index	Hash scan	Table scan	Hash scan

The order in which the algorithms are specified in the list makes no difference to the order in which they are displayed by a `SELECT` or `SHOW VARIABLES` statement (which is the same as that used in the table just shown previously).

- The default value is `TABLE_SCAN,INDEX_SCAN`, which means that all searches that can use indexes do use them, and searches without any indexes use table scans.
- To use hashing for any searches that do not use a primary or unique key, set this option to `INDEX_SCAN,HASH_SCAN`. Specifying `INDEX_SCAN,TABLE_SCAN,HASH_SCAN` has the same effect as specifying `INDEX_SCAN,HASH_SCAN`.
- To force hashing for *all* searches, set this option to `TABLE_SCAN,HASH_SCAN`.

It is possible to specify a single value, but this is not optimal, because setting a single value limits searches to using only that algorithm. In particular, setting `INDEX_SCAN` alone is not recommended, as in that case searches are unable to find rows at all if no index is present.

Note

There is a performance advantage for `INDEX_SCAN` and `HASH_SCAN` only if the row events are big enough. The size of row events is configured using `--binlog-row-event-max-size`. For example, suppose a `DELETE` statement which deletes 25,000 rows generates large `Delete_row_event` events. In this case if `slave_rows_search_algorithms` is set to `INDEX_SCAN` or `HASH_SCAN` there is a performance improvement. However, if there are 25,000 `DELETE` statements and each is represented by a separate event then setting `slave_rows_search_algorithms` to `INDEX_SCAN` or `HASH_SCAN` provides no performance improvement while executing these separate events.

- `slave_skip_errors`

Command-Line Format	`--slave-skip-errors=name`
System Variable	`slave_skip_errors`
Scope	Global

Dynamic	No
Type	String
Default Value	`OFF`

Valid Values	`OFF`
	`[list of error codes]`
	`all`
	`ddl_exist_errors`

Normally, replication stops when an error occurs on the replica. This gives you the opportunity to resolve the inconsistency in the data manually. This variable tells the replication SQL thread to continue replication when a statement returns any of the errors listed in the variable value.

- `slave_sql_verify_checksum`

Command-Line Format	`--slave-sql-verify-checksum[={OFF|ON}]`
System Variable	`slave_sql_verify_checksum`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`ON`

Cause the replication SQL thread to verify data using the checksums read from the relay log. In the event of a mismatch, the replica stops with an error.

Note

The replication I/O thread always reads checksums if possible when accepting events from over the network.

- `slave_transaction_retries`

Command-Line Format	`--slave-transaction-retries=#`
System Variable	`slave_transaction_retries`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`10`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

If a replication SQL thread fails to execute a transaction because of an `InnoDB` deadlock or because the transaction's execution time exceeded `InnoDB`'s `innodb_lock_wait_timeout` or `NDB`'s `TransactionDeadlockDetectionTimeout` or `TransactionInactiveTimeout`,

it automatically retries `slave_transaction_retries` times before stopping with an error. The default value is 10.

Transactions cannot be retried when using a multithreaded replica. In other words, whenever `slave_parallel_workers` is greater than 0, `slave_transaction_retries` is treated as equal to 0, and cannot be changed.

- `slave_type_conversions`

Command-Line Format	`--slave-type-conversions=set`
System Variable	`slave_type_conversions`
Scope	Global
Dynamic	No
Type	Set
Default Value	

Valid Values (≥ 5.6.13)	`ALL_LOSSY`
	`ALL_NON_LOSSY`
	`ALL_SIGNED`
	`ALL_UNSIGNED`

Valid Values (≤ 5.6.12)	`ALL_LOSSY` `ALL_NON_LOSSY`

Controls the type conversion mode in effect on the replica when using row-based replication. In MySQL 5.6.13 and higher, its value is a comma-delimited set of zero or more elements from the list: `ALL_LOSSY`, `ALL_NON_LOSSY`, `ALL_SIGNED`, `ALL_UNSIGNED`. Set this variable to an empty string to disallow type conversions between the source and the replica. Changes require a restart of the replica to take effect.

`ALL_SIGNED` and `ALL_UNSIGNED` were added in MySQL 5.6.13 (Bug#15831300). For additional information on type conversion modes applicable to attribute promotion and demotion in row-based replication, see Row-based replication: attribute promotion and demotion.

- `sql_slave_skip_counter`

System Variable	`sql_slave_skip_counter`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

The number of events from the source that a replica server should skip.

This option is incompatible with GTID-based replication, and must not be set to a nonzero value when `gtid_mode=ON`. In MySQL 5.6.10 and higher, trying to do so is specifically disallowed. (Bug

#15833516) If you need to skip transactions when employing GTIDs, use `gtid_executed` from the source instead. See Injecting empty transactions, for information about how to do this.

Important

If skipping the number of events specified by setting this variable would cause the replica to begin in the middle of an event group, the replica continues to skip until it finds the beginning of the next event group and begins from that point. For more information, see Section 13.4.2.4, "SET GLOBAL sql_slave_skip_counter Statement".

- `sync_master_info`

Command-Line Format	`--sync-master-info=#`
System Variable	`sync_master_info`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`10000`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

The effects of this variable on a replica depend on whether the replica's `master_info_repository` is set to `FILE` or `TABLE`, as explained in the following paragraphs.

master_info_repository = FILE. If the value of `sync_master_info` is greater than 0, the replica synchronizes its `master.info` file to disk (using `fdatasync()`) after every `sync_master_info` events. If it is 0, the MySQL server performs no synchronization of the `master.info` file to disk; instead, the server relies on the operating system to flush its contents periodically as with any other file.

master_info_repository = TABLE. If the value of `sync_master_info` is greater than 0, the replica updates its connection metadata repository table after every `sync_master_info` events. If it is 0, the table is never updated.

- `sync_relay_log`

Command-Line Format	`--sync-relay-log=#`
System Variable	`sync_relay_log`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`10000`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`18446744073709551615`

Maximum Value (32-bit platforms)	4294967295

If the value of this variable is greater than 0, the MySQL server synchronizes its relay log to disk (using fdatasync()) after every sync_relay_log events are written to the relay log.

Setting sync_relay_log to 0 causes no synchronization to be done to disk; in this case, the server relies on the operating system to flush the relay log's contents from time to time as for any other file.

A value of 1 is the safest choice because in the event of an unexpected exit you lose at most one event from the relay log. However, it is also the slowest choice (unless the disk has a battery-backed cache, which makes synchronization very fast).

- sync_relay_log_info

Command-Line Format	--sync-relay-log-info=#
System Variable	sync_relay_log_info
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	10000
Minimum Value	0
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295

The default value for sync_relay_log_info is 10000.

The effects of this variable on the replica depend on the server's relay_log_info_repository setting (FILE or TABLE). If the setting is TABLE, the effects of the variable also depend on whether the storage engine used by the relay log info table is transactional (such as InnoDB) or not transactional (MyISAM). The effects of these factors on the behavior of the server for sync_relay_log_info values of zero and greater than zero are as follows:

sync_relay_log_info = 0

- If relay_log_info_repository is set to FILE, the MySQL server performs no synchronization of the relay-log.info file to disk; instead, the server relies on the operating system to flush its contents periodically as with any other file.
- If relay_log_info_repository is set to TABLE, and the storage engine for that table is transactional, the table is updated after each transaction. (The sync_relay_log_info setting is effectively ignored in this case.)
- If relay_log_info_repository is set to TABLE, and the storage engine for that table is not transactional, the table is never updated.

sync_relay_log_info = N > 0

- If relay_log_info_repository is set to FILE, the replica synchronizes its relay-log.info file to disk (using fdatasync()) after every N transactions.
- If relay_log_info_repository is set to TABLE, and the storage engine for that table is transactional, the table is

updated after each transaction. (The `sync_relay_log_info` setting is effectively ignored in this case.)

- If `relay_log_info_repository` is set to `TABLE`, and the storage engine for that table is not transactional, the table is updated after every *`N`* events.

17.1.4.4 Binary Log Options and Variables

- Startup Options Used with Binary Logging
- System Variables Used with Binary Logging

You can use the `mysqld` options and system variables that are described in this section to affect the operation of the binary log as well as to control which statements are written to the binary log. For additional information about the binary log, see Section 5.4.4, “The Binary Log”. For additional information about using MySQL server options and system variables, see Section 5.1.6, “Server Command Options”, and Section 5.1.7, “Server System Variables”.

Startup Options Used with Binary Logging

The following list describes startup options for enabling and configuring the binary log. System variables used with binary logging are discussed later in this section.

- `--binlog-row-event-max-size=`*`N`*

Command-Line Format	`--binlog-row-event-max-size=#`
Type	Integer
Default Value	`8192`
Minimum Value	`256`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

Specify the maximum size of a row-based binary log event, in bytes. Rows are grouped into events smaller than this size if possible. The value should be a multiple of 256. See Section 17.1.2, “Replication Formats”.

- `--log-bin[=`*`base_name`*`]`

Command-Line Format	`--log-bin=file_name`
Type	File name

Enables binary logging. With binary logging enabled, the server logs all statements that change data to the binary log, which is used for backup and replication. The binary log is a sequence of files with a base name and numeric extension. For information on the format and management of the binary log, see Section 5.4.4, “The Binary Log”.

The option value, if given, is the base name for the log sequence. The server creates binary log files in sequence by adding a numeric suffix to the base name. It is recommended that you specify a base name (see Section B.3.7, “Known Issues in MySQL”, for the reason). Otherwise, MySQL uses *`host_name`*`-bin` as the base name.

If you supply a value for the `--log-bin` option, the value is used as the base name for the log sequence. The server creates binary log files in sequence by adding a numeric suffix to the base name. In MySQL 5.6, the default base name is the name of the process ID file, with the suffix `-bin`. That name can be set with the `--pid-file` option, and it defaults to the name of the host machine.

It is recommended that you specify a base name using the `--log-bin` option, so that you can continue to use the same binary log file names regardless of changes to the default name.

The default location for binary log files is the data directory. You can use the `--log-bin` option to specify an alternative location, by adding a leading absolute path name to the base name to specify a different directory. When the server reads an entry from the binary log index file, which tracks the binary log files that have been used, it checks whether the entry contains a relative path. If it does, the relative part of the path is replaced with the absolute path set using the `--log-bin` option. An absolute path recorded in the binary log index file remains unchanged; in such a case, the index file must be edited manually to enable a new path or paths to be used. (In older versions of MySQL, manual intervention was required whenever relocating the binary log or relay log files.) (Bug #11745230, Bug #12133)

Setting this option causes the `log_bin` system variable to be set to `ON` (or `1`), and not to the base name. The binary log file base name and any specified path are available as the `log_bin_basename` system variable.

If you want to disable binary logging for a server start but keep the `--log-bin` setting intact, you can specify the `--skip-log-bin` or `--disable-log-bin` option at startup. Specify the option after the `--log-bin` option, so that it takes precedence. When binary logging is disabled, the `log_bin` system variable is set to OFF.

- `--log-bin-index[=file_name]`

Command-Line Format	`--log-bin-index=file_name`
System Variable	`log_bin_index`
Scope	Global
Dynamic	No
Type	File name

The name for the binary log index file, which contains the names of the binary log files. By default, it has the same location and base name as the value specified for the binary log files using the `--log-bin` option, plus the extension `.index`. If you do not specify `--log-bin`, the default binary log index file name is `binlog.index`. If you omit the file name and do not specify one with `--log-bin`, the default binary log index file name is *`host_name`*`-bin.index`, using the name of the host machine.

For information on the format and management of the binary log, see Section 5.4.4, “The Binary Log”.

Statement selection options. The options in the following list affect which statements are written to the binary log, and thus sent by a replication source server to its replicas. There are also options for replica servers that control which statements received from the source should be executed or ignored. For details, see Section 17.1.4.3, “Replica Server Options and Variables”.

- `--binlog-do-db=`*`db_name`*

Command-Line Format	`--binlog-do-db=name`
Type	String

This option affects binary logging in a manner similar to the way that `--replicate-do-db` affects replication.

The effects of this option depend on whether the statement-based or row-based logging format is in use, in the same way that the effects of `--replicate-do-db` depend on whether statement-based or row-based replication is in use. You should keep in mind that the format used to log a given statement may not necessarily be the same as that indicated by the value of `binlog_format`. For example, DDL statements such as `CREATE TABLE` and `ALTER TABLE` are always logged as

statements, without regard to the logging format in effect, so the following statement-based rules for `--binlog-do-db` always apply in determining whether or not the statement is logged.

Statement-based logging. Only those statements are written to the binary log where the default database (that is, the one selected by `USE`) is *`db_name`*. To specify more than one database, use this option multiple times, once for each database; however, doing so does *not* cause cross-database statements such as `UPDATE some_db.some_table SET foo='bar'` to be logged while a different database (or no database) is selected.

Warning

To specify multiple databases you *must* use multiple instances of this option. Because database names can contain commas, the list is treated as the name of a single database if you supply a comma-separated list.

An example of what does not work as you might expect when using statement-based logging: If the server is started with `--binlog-do-db=sales` and you issue the following statements, the `UPDATE` statement is *not* logged:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

The main reason for this “just check the default database” behavior is that it is difficult from the statement alone to know whether it should be replicated (for example, if you are using multiple-table `DELETE` statements or multiple-table `UPDATE` statements that act across multiple databases). It is also faster to check only the default database rather than all databases if there is no need.

Another case which may not be self-evident occurs when a given database is replicated even though it was not specified when setting the option. If the server is started with `--binlog-do-db=sales`, the following `UPDATE` statement is logged even though `prices` was not included when setting `--binlog-do-db`:

```
USE sales;
UPDATE prices.discounts SET percentage = percentage + 10;
```

Because `sales` is the default database when the `UPDATE` statement is issued, the `UPDATE` is logged.

Row-based logging. Logging is restricted to database *`db_name`*. Only changes to tables belonging to *`db_name`* are logged; the default database has no effect on this. Suppose that the server is started with `--binlog-do-db=sales` and row-based logging is in effect, and then the following statements are executed:

```
USE prices;
UPDATE sales.february SET amount=amount+100;
```

The changes to the `february` table in the `sales` database are logged in accordance with the `UPDATE` statement; this occurs whether or not the `USE` statement was issued. However, when using the row-based logging format and `--binlog-do-db=sales`, changes made by the following `UPDATE` are not logged:

```
USE prices;
UPDATE prices.march SET amount=amount-25;
```

Even if the `USE prices` statement were changed to `USE sales`, the `UPDATE` statement's effects would still not be written to the binary log.

Another important difference in `--binlog-do-db` handling for statement-based logging as opposed to the row-based logging occurs with regard to statements that refer to multiple databases. Suppose that the server is started with `--binlog-do-db=db1`, and the following statements are executed:

```
USE db1;
```

```
UPDATE db1.table1, db2.table2 SET db1.table1.col1 = 10, db2.table2.col2 = 20;
```

If you are using statement-based logging, the updates to both tables are written to the binary log. However, when using the row-based format, only the changes to `table1` are logged; `table2` is in a different database, so it is not changed by the `UPDATE`. Now suppose that, instead of the `USE db1` statement, a `USE db4` statement had been used:

```
USE db4;
UPDATE db1.table1, db2.table2 SET db1.table1.col1 = 10, db2.table2.col2 = 20;
```

In this case, the `UPDATE` statement is not written to the binary log when using statement-based logging. However, when using row-based logging, the change to `table1` is logged, but not that to `table2`—in other words, only changes to tables in the database named by `--binlog-do-db` are logged, and the choice of default database has no effect on this behavior.

- `--binlog-ignore-db=`*`db_name`*

Command-Line Format	`--binlog-ignore-db=name`
Type	String

This option affects binary logging in a manner similar to the way that `--replicate-ignore-db` affects replication.

The effects of this option depend on whether the statement-based or row-based logging format is in use, in the same way that the effects of `--replicate-ignore-db` depend on whether statement-based or row-based replication is in use. You should keep in mind that the format used to log a given statement may not necessarily be the same as that indicated by the value of `binlog_format`. For example, DDL statements such as `CREATE TABLE` and `ALTER TABLE` are always logged as statements, without regard to the logging format in effect, so the following statement-based rules for `--binlog-ignore-db` always apply in determining whether or not the statement is logged.

Statement-based logging. Tells the server to not log any statement where the default database (that is, the one selected by `USE`) is *`db_name`*.

Prior to MySQL 5.6.12, this option caused any statements containing fully qualified table names not to be logged if there was no default database specified (that is, when `SELECT DATABASE()` returned `NULL`). In MySQL 5.6.12 and higher, when there is no default database, no `--binlog-ignore-db` options are applied, and such statements are always logged. (Bug #11829838, Bug #60188)

Row-based format. Tells the server not to log updates to any tables in the database *`db_name`*. The current database has no effect.

When using statement-based logging, the following example does not work as you might expect. Suppose that the server is started with `--binlog-ignore-db=sales` and you issue the following statements:

```
USE prices;
UPDATE sales.january SET amount=amount+1000;
```

The `UPDATE` statement *is* logged in such a case because `--binlog-ignore-db` applies only to the default database (determined by the `USE` statement). Because the `sales` database was specified explicitly in the statement, the statement has not been filtered. However, when using row-based logging, the `UPDATE` statement's effects are *not* written to the binary log, which means that no changes to the `sales.january` table are logged; in this instance, `--binlog-ignore-db=sales`

causes *all* changes made to tables in the source's copy of the `sales` database to be ignored for purposes of binary logging.

To specify more than one database to ignore, use this option multiple times, once for each database. Because database names can contain commas, the list is treated as the name of a single database if you supply a comma-separated list.

You should not use this option if you are using cross-database updates and you do not want these updates to be logged.

Checksum options. MySQL supports reading and writing of binary log checksums. These are enabled using the two options listed here:

- `--binlog-checksum={NONE|CRC32}`

Command-Line Format	`--binlog-checksum=type`
Type	String
Default Value	`CRC32`

Valid Values	`NONE`
	
	`CRC32`

Enabling this option causes the source to write checksums for events written to the binary log. Set to `NONE` to disable, or the name of the algorithm to be used for generating checksums; currently, only CRC32 checksums are supported.

To control reading of checksums by the replica (from the relay log), use the `--slave-sql-verify-checksum` option.

Testing and debugging options. The following binary log options are used in replication testing and debugging. They are not intended for use in normal operations.

- `--max-binlog-dump-events=N`

Command-Line Format	`--max-binlog-dump-events=#`
Type	Integer
Default Value	`0`

This option is used internally by the MySQL test suite for replication testing and debugging.

- `--sporadic-binlog-dump-fail`

Command-Line Format	`--sporadic-binlog-dump-fail[={OFF|ON}]`
Type	Boolean
Default Value	`OFF`

This option is used internally by the MySQL test suite for replication testing and debugging.

System Variables Used with Binary Logging

The following list describes system variables for controlling binary logging. They can be set at server startup and some of them can be changed at runtime using `SET`. Server options used to control binary logging are listed earlier in this section.

- `binlog_cache_size`

Command-Line Format	`--binlog-cache-size=#`
System Variable	`binlog_cache_size`

Scope	Global
Dynamic	Yes
Type	Integer
Default Value	32768
Minimum Value	4096
Maximum Value (64-bit platforms)	18446744073709551615
Maximum Value (32-bit platforms)	4294967295

The size of the cache to hold changes to the binary log during a transaction. A binary log cache is allocated for each client if the server supports any transactional storage engines and if the server has the binary log enabled (`--log-bin` option). If you often use large transactions, you can increase this cache size to get better performance. The `Binlog_cache_use` and `Binlog_cache_disk_use` status variables can be useful for tuning the size of this variable. See Section 5.4.4, "The Binary Log".

`binlog_cache_size` sets the size for the transaction cache only; the size of the statement cache is governed by the `binlog_stmt_cache_size` system variable.

- `binlog_checksum`

Command-Line Format	`--binlog-checksum=name`
System Variable	`binlog_checksum`
Scope	Global
Dynamic	Yes
Type	String
Default Value	`CRC32`
Valid Values	`NONE` `CRC32`

When enabled, this variable causes the replication source server to write a checksum for each event in the binary log. `binlog_checksum` supports the values `NONE` (disabled) and `CRC32`. The default is `CRC32`.

When `binlog_checksum` is disabled (value `NONE`), the server verifies that it is writing only complete events to the binary log by writing and checking the event length (rather than a checksum) for each event.

Changing the value of this variable causes the binary log to be rotated; checksums are always written to an entire binary log file, and never to only part of one.

Setting this variable on the source to a value unrecognized by the replica causes the replica to set its own `binlog_checksum` value to `NONE`, and to stop replication with an error. (Bug #13553750, Bug #61096) If backward compatibility with older replicas is a concern, you may want to set the value explicitly to `NONE`.

- `binlog_direct_non_transactional_updates`

Command-Line Format	`--binlog-direct-non-transactional-updates[={OFF|ON}]`
System Variable	`binlog_direct_non_transactional_updates`

Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

Due to concurrency issues, a replica can become inconsistent when a transaction contains updates to both transactional and nontransactional tables. MySQL tries to preserve causality among these statements by writing nontransactional statements to the transaction cache, which is flushed upon commit. However, problems arise when modifications done to nontransactional tables on behalf of a transaction become immediately visible to other connections because these changes may not be written immediately into the binary log.

The `binlog_direct_non_transactional_updates` variable offers one possible workaround to this issue. By default, this variable is disabled. Enabling `binlog_direct_non_transactional_updates` causes updates to nontransactional tables to be written directly to the binary log, rather than to the transaction cache.

`binlog_direct_non_transactional_updates` works only for statements that are replicated using the statement-based binary logging format; that is, it works only when the value of `binlog_format` is `STATEMENT`, or when `binlog_format` is `MIXED` and a given statement is being replicated using the statement-based format. This variable has no effect when the binary log format is `ROW`, or when `binlog_format` is set to `MIXED` and a given statement is replicated using the row-based format.

Important

Before enabling this variable, you must make certain that there are no dependencies between transactional and nontransactional tables; an example of such a dependency would be the statement `INSERT INTO myisam_table SELECT * FROM innodb_table`. Otherwise, such statements are likely to cause the replica to diverge from the source.

In MySQL 5.6, this variable has no effect when the binary log format is `ROW` or `MIXED`. (Bug #51291)

- `binlog_error_action`

Command-Line Format	`--binlog-error-action[=value]`
Introduced	5.6.22
System Variable	`binlog_error_action`
Scope	Global
Dynamic	Yes
Type	Enumeration
Default Value	`IGNORE_ERROR`

Valid Values	`IGNORE_ERROR`
	
	`ABORT_SERVER`

Controls what happens when the server encounters an error such as not being able to write to, flush or synchronize the binary log, which can cause the source's binary log to become inconsistent and replicas to lose synchronization.

In MySQL 5.6, this variable defaults to `IGNORE_ERROR`. If the server encounters such an error, it continues the ongoing transaction, logs the error then halts logging, and continues performing

updates. To resume binary logging `log_bin` must be enabled again, which requires a server restart. This setting provides backward compatibility with older versions of MySQL.

Setting this variable to `ABORT_SERVER` makes the server halt logging and shut down whenever it encounters such an error with the binary log. On restart, recovery proceeds as in the case of an unexpected server halt (see Section 17.3.2, “Handling an Unexpected Halt of a Replica Server”). This is the recommended setting, particularly in complex replication environments.

In previous releases this variable was named `binlogging_impossible_mode`.

- `binlog_format`

Command-Line Format	`--binlog-format=format`
System Variable	`binlog_format`
Scope	Global, Session
Dynamic	Yes
Type	Enumeration
Default Value (≥ 5.6.10-ndb-7.3.1)	`MIXED`
Default Value	`STATEMENT`
Valid Values	`ROW` `STATEMENT`

	`MIXED`

This variable sets the binary logging format, and can be any one of `STATEMENT`, `ROW`, or `MIXED`. See Section 17.1.2, “Replication Formats”.

`binlog_format` can be set at startup or at runtime, except that under some conditions, changing this variable at runtime is not possible or causes replication to fail, as described later.

In MySQL 5.6, the default format is `STATEMENT`. *Exception*: In MySQL NDB Cluster 7.3 and later, the default is `MIXED`; statement-based replication is not supported for NDB Cluster.

Setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.8.1, “System Variable Privileges”.

The rules governing when changes to this variable take effect and how long the effect lasts are the same as for other MySQL server system variables. For more information, see Section 13.7.4.1, “SET Syntax for Variable Assignment”.

When `MIXED` is specified, statement-based replication is used, except for cases where only row-based replication is guaranteed to lead to proper results. For example, this happens when statements contain user-defined functions (UDF) or the `UUID()` function.

For details of how stored programs (stored procedures and functions, triggers, and events) are handled when each binary logging format is set, see Section 20.7, “Stored Program Binary Logging”.

There are exceptions when you cannot switch the replication format at runtime:

- From within a stored function or a trigger.
- If the session is currently in row-based replication mode and has open temporary tables.
- From within a transaction.

Trying to switch the format in those cases results in an error.

Changing the logging format on a replication source server does not cause a replica to change its logging format to match. Switching the replication format while replication is ongoing can cause issues if a replica has binary logging enabled, and the change results in the replica using `STATEMENT` format logging while the source is using `ROW` or `MIXED` format logging. A replica is not able to convert binary log entries received in `ROW` logging format to `STATEMENT` format for use in its own binary log, so this situation can cause replication to fail. For more information, see Section 5.4.4.2, “Setting The Binary Log Format”.

The binary log format affects the behavior of the following server options:

- `--replicate-do-db`
- `--replicate-ignore-db`
- `--binlog-do-db`
- `--binlog-ignore-db`

These effects are discussed in detail in the descriptions of the individual options.

- `binlogging_impossible_mode`

Command-Line Format	`--binlogging-impossible-mode[=value]`
Introduced	5.6.20

Deprecated	5.6.22
System Variable	`binlogging_impossible_mode`
Scope	Global, Session
Dynamic	Yes
Type	Enumeration
Default Value	`IGNORE_ERROR`
Valid Values	`IGNORE_ERROR` `ABORT_SERVER`

This option is deprecated; expect it to be removed in a future MySQL release. Use the `binlog_error_action` variable to control what happens when the server cannot write to the binary log.

- `binlog_max_flush_queue_time`

Command-Line Format	`--binlog-max-flush-queue-time=#`
System Variable	`binlog_max_flush_queue_time`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`100000`

How long in microseconds to keep reading transactions from the flush queue before proceeding with the group commit (and syncing the log to disk, if `sync_binlog` is greater than 0). If the value is 0 (the default), there is no timeout and the server keeps reading new transactions until the queue is empty.

Normally, `binlog_max_flush_queue_time` can remain set to 0. If the server processes a large number of connections (for example, 100 or more) and many short transactions with low-latency requirements, it may be useful to set the value larger than 0 to force more frequent flushes to disk.

- `binlog_order_commits`

Command-Line Format	`--binlog-order-commits[={OFF|ON}]`
System Variable	`binlog_order_commits`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`ON`

When this variable is enabled on a replication source server (which is the default), transaction commit instructions issued to storage engines are serialized on a single thread, so that transactions are always committed in the same order as they are written to the binary log. Disabling this variable permits transaction commit instructions to be issued using multiple threads. Used in combination with binary log group commit, this prevents the commit rate of a single transaction being a bottleneck to throughput, and might therefore produce a performance improvement.

Transactions are written to the binary log at the point when all the storage engines involved have confirmed that the transaction is prepared to commit. The binary log group commit

logic then commits a group of transactions after their binary log write has taken place. When `binlog_order_commits` is disabled, because multiple threads are used for this process, transactions in a commit group might be committed in a different order from their order in the binary log. (Transactions from a single client always commit in chronological order.) In many cases this does not matter, as operations carried out in separate transactions should produce consistent results, and if that is not the case, a single transaction ought to be used instead.

- `binlog_row_image`

Command-Line Format	`--binlog-row-image=image_type`
System Variable	`binlog_row_image`
Scope	Global, Session
Dynamic	Yes
Type	Enumeration
Default Value	`full`
Valid Values	`full` (Log all columns) `minimal` (Log only changed columns, and columns needed to identify rows) `noblob` (Log all columns, except for unneeded BLOB and TEXT columns)

For MySQL row-based replication, this variable determines how row images are written to the binary log.

In MySQL row-based replication, each row change event contains two images, a “before” image whose columns are matched against when searching for the row to be updated, and an “after” image containing the changes. Normally, MySQL logs full rows (that is, all columns) for both the before and after images. However, it is not strictly necessary to include every column in both images, and we can often save disk, memory, and network usage by logging only those columns which are actually required.

Note

When deleting a row, only the before image is logged, since there are no changed values to propagate following the deletion. When inserting a row, only the after image is logged, since there is no existing row to be matched. Only when updating a row are both the before and after images required, and both written to the binary log.

For the before image, it is necessary only that the minimum set of columns required to uniquely identify rows is logged. If the table containing the row has a primary key, then only the primary key column or columns are written to the binary log. Otherwise, if the table has a unique key all of whose columns are `NOT NULL`, then only the columns in the unique key need be logged. (If the table has neither a primary key nor a unique key without any `NULL` columns, then all columns must be used in the before image, and logged.) In the after image, it is necessary to log only the columns which have actually changed.

You can cause the server to log full or minimal rows using the `binlog_row_image` system variable. This variable actually takes one of three possible values, as shown in the following list:

- `full`: Log all columns in both the before image and the after image.
- `minimal`: Log only those columns in the before image that are required to identify the row to be changed; log only those columns in the after image where a value was specified by the SQL statement, or generated by auto-increment.

- `noblob`: Log all columns (same as `full`), except for `BLOB` and `TEXT` columns that are not required to identify rows, or that have not changed.

Note

This variable is not supported by NDB Cluster; setting it has no effect on the logging of `NDB` tables. (Bug #16316828)

The default value is `full`.

When using `minimal` or `noblob`, deletes and updates are guaranteed to work correctly for a given table if and only if the following conditions are true for both the source and destination tables:

- All columns must be present and in the same order; each column must use the same data type as its counterpart in the other table.
- The tables must have identical primary key definitions.

(In other words, the tables must be identical with the possible exception of indexes that are not part of the tables' primary keys.)

If these conditions are not met, it is possible that the primary key column values in the destination table may prove insufficient to provide a unique match for a delete or update. In this event, no warning or error is issued; the source and replica silently diverge, thus breaking consistency.

Setting this variable has no effect when the binary logging format is `STATEMENT`. When `binlog_format` is `MIXED`, the setting for `binlog_row_image` is applied to changes that are logged using row-based format, but this setting has no effect on changes logged as statements.

Setting `binlog_row_image` on either the global or session level does not cause an implicit commit; this means that this variable can be changed while a transaction is in progress without affecting the transaction.

- `binlog_rows_query_log_events`

Command-Line Format	`--binlog-rows-query-log-events[={OFF|ON}]`
System Variable	`binlog_rows_query_log_events`
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

This system variable affects row-based logging only. When enabled, it causes the server to write informational log events such as row query log events into its binary log. This information can be used for debugging and related purposes, such as obtaining the original query issued on the source when it cannot be reconstructed from the row updates.

These informational events are normally ignored by MySQL programs reading the binary log and so cause no issues when replicating or restoring from backup. To view them, increase the verbosity level by using mysqlbinlog's `--verbose` option twice, either as `-vv` or `--verbose --verbose`.

- `binlog_stmt_cache_size`

Command-Line Format	`--binlog-stmt-cache-size=#`
System Variable	`binlog_stmt_cache_size`
Scope	Global

Dynamic	Yes
Type	Integer
Default Value	`32768`
Minimum Value	`4096`
Maximum Value (64-bit platforms)	`18446744073709551615`
Maximum Value (32-bit platforms)	`4294967295`

This variable determines the size of the cache for the binary log to hold nontransactional statements issued during a transaction. Separate binary log transaction and statement caches are allocated for each client if the server supports any transactional storage engines and if the server has the binary log enabled (`--log-bin` option). If you often use large nontransactional statements during transactions, you can increase this cache size to get better performance. The `Binlog_stmt_cache_use` and `Binlog_stmt_cache_disk_use` status variables can be useful for tuning the size of this variable. See Section 5.4.4, “The Binary Log”.

The `binlog_cache_size` system variable sets the size for the transaction cache.

- `expire_logs_days`

Command-Line Format	`--expire-logs-days=#`
System Variable	`expire_logs_days`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`99`

The number of days for automatic binary log file removal. The default is 0, which means “no automatic removal.” Possible removals happen at startup and when the binary log is flushed. Log flushing occurs as indicated in Section 5.4, “MySQL Server Logs”.

To remove binary log files manually, use the `PURGE BINARY LOGS` statement. See Section 13.4.1.1, “PURGE BINARY LOGS Statement”.

- `log_bin`

System Variable	`log_bin`
Scope	Global
Dynamic	No
Type	Boolean

Whether the binary log is enabled. If the `--log-bin` option is used, then the value of this variable is `ON`; otherwise it is `OFF`. This variable reports only on the status of binary logging (enabled or disabled); it does not actually report the value to which `--log-bin` is set.

See Section 5.4.4, “The Binary Log”.

- `log_bin_basename`

System Variable	`log_bin_basename`

Scope	Global
Dynamic	No
Type	File name

Holds the base name and path for the binary log files, which can be set with the `--log-bin` server option. The maximum variable length is 256. In MySQL 5.6, the default base name is the name of the process ID file, with the suffix `-bin`. That name can be set with the `--pid-file` option, and it defaults to the name of the host machine. The default location for the binary log files is the data directory.

- `log_bin_index`

Command-Line Format	`--log-bin-index=file_name`
System Variable	`log_bin_index`
Scope	Global
Dynamic	No
Type	File name

Holds the base name and path for the binary log index file, which can be set with the `--log-bin-index` server option. The maximum variable length is 256.

- `log_bin_trust_function_creators`

Command-Line Format	`--log-bin-trust-function-creators[={OFF|ON}]`
System Variable	`log_bin_trust_function_creators`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

This variable applies when binary logging is enabled. It controls whether stored function creators can be trusted not to create stored functions that cause unsafe events to be written to the binary log. If set to 0 (the default), users are not permitted to create or alter stored functions unless they have the `SUPER` privilege in addition to the `CREATE ROUTINE` or `ALTER ROUTINE` privilege. A setting of 0 also enforces the restriction that a function must be declared with the `DETERMINISTIC` characteristic, or with the `READS SQL DATA` or `NO SQL` characteristic. If the variable is set to 1, MySQL does not enforce these restrictions on stored function creation. This variable also applies to trigger creation. See Section 20.7, “Stored Program Binary Logging”.

- `log_bin_use_v1_row_events`

Command-Line Format	`--log-bin-use-v1-row-events[={OFF|ON}]`
System Variable	`log_bin_use_v1_row_events`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`OFF`

Whether Version 2 binary logging is in use. If this variable is 0 (disabled, the default), Version 2 binary log events are in use. If this variable is 1 (enabled), the server writes the binary log using

Version 1 logging events (the only version of binary log events used in previous releases), and thus produces a binary log that can be read by older replicas.

MySQL 5.6 uses Version 2 binary log row events by default. However, Version 2 events cannot be read by MySQL Server releases prior to MySQL 5.6.6. Enabling `log_bin_use_v1_row_events` causes `mysqld` to write the binary log using Version 1 logging events.

This variable is read-only at runtime. To switch between Version 1 and Version 2 binary event binary logging, it is necessary to set `log_bin_use_v1_row_events` at server startup.

Other than when performing upgrades of NDB Cluster Replication, `log_bin_use_v1_row_events` is chiefly of interest when setting up replication conflict detection and resolution using `NDB $EPOCH_TRANS()` as the conflict detection function, which requires Version 2 binary log row events. Thus, this variable and `--ndb-log-transaction-id` are not compatible.

Note

MySQL NDB Cluster 7.3 and higher use Version 2 binary log row events by default. You should keep this mind when planning upgrades or downgrades, and for setups using NDB Cluster Replication.

For more information, see Section 18.6.11, “NDB Cluster Replication Conflict Resolution”.

- `log_slave_updates`

Command-Line Format	`--log-slave-updates[={OFF|ON}]`
System Variable	`log_slave_updates`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`OFF`

Whether updates received by a replica from a replication source server should be logged to the replica's own binary log.

Normally, a replica does not log to its own binary log any updates that are received from a source. Enabling this variable causes the replica to write the updates performed by its replication SQL thread to its own binary log. For this option to have any effect, the replica must also be started with the `--log-bin` option to enable binary logging. See Section 17.1.4, “Replication and Binary Logging Options and Variables”. A warning is issued if you enable `log_slave_updates` without also starting the server with the `--log-bin` option.

`log_slave_updates` is enabled when you want to chain replication servers. For example, you might want to set up replication servers using this arrangement:

```
A -> B -> C
```

Here, `A` serves as the source for the replica `B`, and `B` serves as the source for the replica `C`. For this to work, `B` must be both a source *and* a replica. You must start both `A` and `B` with `--log-bin` to enable binary logging, and `B` with `log_slave_updates` enabled so that updates received from `A` are logged by `B` to its binary log.

- `master_verify_checksum`

Command-Line Format	`--master-verify-checksum[={OFF|ON}]`
System Variable	`master_verify_checksum`
Scope	Global

Dynamic	Yes
Type	Boolean
Default Value	`OFF`

Enabling this variable causes the source to verify events read from the binary log by examining checksums, and to stop with an error in the event of a mismatch. `master_verify_checksum` is disabled by default; in this case, the source uses the event length from the binary log to verify events, so that only complete events are read from the binary log.

- `max_binlog_cache_size`

Command-Line Format	`--max-binlog-cache-size=#`
System Variable	`max_binlog_cache_size`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`18446744073709551615`
Minimum Value	`4096`
Maximum Value	`18446744073709551615`

If a transaction requires more than this many bytes of memory, the server generates a `Multi-statement transaction required more than 'max_binlog_cache_size' bytes of storage` error. The minimum value is 4096. The maximum possible value is 16EB (exabytes). The maximum recommended value is 4GB; this is due to the fact that MySQL currently cannot work with binary log positions greater than 4GB.

`max_binlog_cache_size` sets the size for the transaction cache only; the upper limit for the statement cache is governed by the `max_binlog_stmt_cache_size` system variable.

In MySQL 5.6, the visibility to sessions of `max_binlog_cache_size` matches that of the `binlog_cache_size` system variable; in other words, changing its value effects only new sessions that are started after the value is changed.

- `max_binlog_size`

Command-Line Format	`--max-binlog-size=#`
System Variable	`max_binlog_size`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`1073741824`
Minimum Value	`4096`
Maximum Value	`1073741824`

If a write to the binary log causes the current log file size to exceed the value of this variable, the server rotates the binary logs (closes the current file and opens the next one). The minimum value is 4096 bytes. The maximum and default value is 1GB.

A transaction is written in one chunk to the binary log, so it is never split between several binary logs. Therefore, if you have big transactions, you might see binary log files larger than `max_binlog_size`.

If `max_relay_log_size` is 0, the value of `max_binlog_size` applies to relay logs as well.

- `max_binlog_stmt_cache_size`

Command-Line Format	`--max-binlog-stmt-cache-size=#`
System Variable	`max_binlog_stmt_cache_size`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`18446744073709547520`
Minimum Value	`4096`
Maximum Value	`18446744073709547520`

If nontransactional statements within a transaction require more than this many bytes of memory, the server generates an error. The minimum value is 4096. The maximum and default values are 4GB on 32-bit platforms and 16EB (exabytes) on 64-bit platforms.

`max_binlog_stmt_cache_size` sets the size for the statement cache only; the upper limit for the transaction cache is governed exclusively by the `max_binlog_cache_size` system variable.

- `sql_log_bin`

System Variable	`sql_log_bin`
Scope	Session
Dynamic	Yes
Type	Boolean
Default Value	`ON`

This variable controls whether logging to the binary log is enabled for the current session (assuming that the binary log itself is enabled). The default value is `ON`. To disable or enable binary logging for the current session, set the session `sql_log_bin` variable to `OFF` or `ON`.

Set this variable to `OFF` for a session to temporarily disable binary logging while making changes to the source you do not want replicated to the replica.

Setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.8.1, “System Variable Privileges”.

It is not possible to set the session value of `sql_log_bin` within a transaction or subquery.

Setting this variable to `OFF` prevents GTIDs from being assigned to transactions in the binary log. If you are using GTIDs for replication, this means that, even when binary logging is later enabled once again, the GTIDs written into the log from this point do not account for any transactions that occurred in the meantime—in effect, those transactions are lost.

As of MySQL 5.6.22, the global `sql_log_bin` variable is read only and cannot be modified. The global scope is deprecated; expect it to be removed in a future MySQL release. Prior to 5.6.22, `sql_log_bin` can be set as a global or session variable. Setting `sql_log_bin` globally is only detected when a new session is started. Any sessions previously running are not impacted when setting `sql_log_bin` globally.

Warning

Incorrect use of `sql_log_bin` with a global scope means any changes made in an already running session are *still* being recorded to the binary log and therefore replicated. Exercise extreme caution using `sql_log_bin`

> with a global scope as the above situation could cause unexpected results including replication failure.

- `sync_binlog`

Command-Line Format	`--sync-binlog=#`
System Variable	`sync_binlog`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`4294967295`

Controls how often the MySQL server synchronizes the binary log to disk.

- `sync_binlog=0`: Disables synchronization of the binary log to disk by the MySQL server. Instead, the MySQL server relies on the operating system to flush the binary log to disk from time to time as it does for any other file. This setting provides the best performance, but in the event of a power failure or operating system crash, it is possible that the server has committed transactions that have not been synchronized to the binary log.
- `sync_binlog=1`: Enables synchronization of the binary log to disk before transactions are committed. This is the safest setting but can have a negative impact on performance due to the increased number of disk writes. In the event of a power failure or operating system crash, transactions that are missing from the binary log are only in a prepared state. This permits the automatic recovery routine to roll back the transactions, which guarantees that no transaction is lost from the binary log.
- `sync_binlog=N`, where `N` is a value other than 0 or 1: The binary log is synchronized to disk after `N` binary log commit groups have been collected. In the event of a power failure or operating system crash, it is possible that the server has committed transactions that have not been flushed to the binary log. This setting can have a negative impact on performance due to the increased number of disk writes. A higher value improves performance, but with an increased risk of data loss.

For the greatest possible durability and consistency in a replication setup that uses `InnoDB` with transactions, use these settings:

- `sync_binlog=1`.
- `innodb_flush_log_at_trx_commit=1`.

> **Caution**
>
> Many operating systems and some disk hardware fool the flush-to-disk operation. They may tell `mysqld` that the flush has taken place, even though it has not. In this case, the durability of transactions is not guaranteed even with the recommended settings, and in the worst case, a power outage can corrupt `InnoDB` data. Using a battery-backed disk cache in the SCSI disk controller or in the disk itself speeds up file flushes, and makes the operation safer. You can also try to disable the caching of disk writes in hardware caches.

17.1.4.5 Global Transaction ID Options and Variables

- System Variables Used with GTID Replication

The MySQL Server system variables described in this section are used to monitor and control Global Transaction Identifiers (GTIDs). For additional information, see Section 17.1.3, “Replication with Global Transaction Identifiers”.

System Variables Used with GTID Replication

The following system variables are used with GTID-based replication:

- `binlog_gtid_simple_recovery`

Command-Line Format	`--binlog-gtid-simple-recovery[={OFF|ON}]`
Introduced	5.6.23
System Variable	`binlog_gtid_simple_recovery`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`OFF`

This variable controls how binary log files are iterated during the search for GTIDs when MySQL starts or restarts. In MySQL version 5.6.21, this variable was added as `simplified_binlog_gtid_recovery` and in MySQL version 5.6.23 it was renamed to `binlog_gtid_simple_recovery`.

When `binlog_gtid_simple_recovery=FALSE`, the iteration starts from the newest file to initialize `gtid_executed`, and starts from the oldest file to initialize `gtid_purged`. This process could take a long time if you had a large number of binary log files without GTID events, for example created when `gtid_mode=OFF`.

When `binlog_gtid_simple_recovery=TRUE`, the server does not open more than two binary logs when iterating to populate `gtid_purged` and `gtid_executed`, either during server restart or when binary logs are being purged.

Note

If this option is enabled, `gtid_executed` and `gtid_purged` may be initialized incorrectly in the following situations:

- Some binary logs were generated when `gtid_mode` was `ON`, but `gtid_mode` was `OFF` for the newest binary log.
- A `SET GTID_PURGED` statement was issued after the oldest existing binary log was generated.

If an incorrect GTID set is computed in either situation, it remains incorrect even if the server is later restarted, regardless of the value of this option.

- `enforce_gtid_consistency`

Command-Line Format	`--enforce-gtid-consistency[=value]`
System Variable	`enforce_gtid_consistency`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`OFF`

When this variable is true, the server enforces GTID consistency by allowing execution of only those statements that can be logged in a transactionally safe manner. You *must* enable GTID consistency (by enabling `enforce_gtid_consistency`) before you can start the server with `gtid_mode=ON`; otherwise, enabling GTID mode fails with an error. You *can* (and *should*) enable GTID consistency *prior* to using `gtid_mode`, in order to test whether the system is ready to use GTIDs.

Since only transactionally safe statements can be logged when `enforce_gtid_consistency` is true, it follows that the operations listed here cannot be used when this is the case:

- `CREATE TABLE ... SELECT` statements
- `CREATE TEMPORARY TABLE` statements inside transactions
- Transactions or statements that update both transactional and nontransactional tables.

This variable is read-only at runtime and must be set at server startup.

Nontransactional DML statements involving temporary tables are allowed when using `binlog_format=ROW`, as long as any nontransactional tables affected by the statements are temporary tables (Bug #14272672).

In MySQL 5.6, it is possible but not recommended to run `mysql_upgrade` on a server where `gtid_mode=ON`, since the MySQL system tables use the `MyISAM` storage engine, which is nontransactional.

This option allows single statements updating nontransactional tables, which is intended chiefly for use with programs such as `mysql_install_db` and `mysql_upgrade`. (Bug #14722659)

- `gtid_executed`

System Variable	`gtid_executed`
Scope	Global, Session
Dynamic	No
Type	String
Unit	`set of GTIDs`

When used with global scope, this variable contains a representation of the set of all transactions that are logged in the binary log. This is the same as the value of the `Executed_Gtid_Set` column in the output of `SHOW MASTER STATUS` and `SHOW SLAVE STATUS`.

When used with session scope, this variable contains a representation of the set of transactions that are written to the cache in the current session.

The set of transactions that can be found in the binary logs at any given time is equal to `GTID_SUBTRACT(@@GLOBAL.gtid_executed, @@GLOBAL.gtid_purged)`; that is, to all transactions in the binary log that have not yet been purged.

When the server starts, `@@GLOBAL.gtid_executed` is initialized to the union of the following two sets:

- The GTIDs listed in the `Previous_gtids_log_event` of the newest binary log
- The GTIDs found in every `Gtid_log_event` in the newest binary log.

Thereafter, GTIDs are added to the set as transactions are executed.

Issuing `RESET MASTER` causes the global value (but not the session value) of this variable to be reset to an empty string. GTIDs are not otherwise removed from this set other than when the set is

cleared due to `RESET MASTER`. The set is also cleared if the server is shut down and all binary logs are removed.

- `gtid_mode`

Command-Line Format	`--gtid-mode=MODE`
System Variable	`gtid_mode`
Scope	Global
Dynamic	No
Type	Enumeration
Default Value	`OFF`

Valid Values	`OFF`
	`UPGRADE_STEP_1`
	`UPGRADE_STEP_2`
	`ON`

This variable specifies whether global transaction identifiers (GTIDs) are used to identify transactions. This variable is read-only at runtime and must be set at server startup.

Starting the server with `gtid_mode=ON` requires that the server also be started with the `--log-bin`, `--log-slave-updates`, and `--enforce-gtid-consistency` options.

Setting this option to `OFF` when there are GTIDs in the binary log or in the relay log, or to `ON` when there remain anonymous transactions to be executed, causes an error.

Important

This option does not employ boolean values; its values are in fact enumerated. You should not attempt to use numeric values when setting this option, as these may lead to unexpected results. The values `UPGRADE_STEP_1` and `UPGRADE_STEP_2` are reserved for future use, but currently are not supported in production; if you set `gtid_mode` to either of these two values, the server refuses to start.

The values of `gtid_purged` and `gtid_executed` are not persistent while `gtid_mode=off`. Therefore, after changing `gtid_mode` to `OFF`, once all binary logs containing GTIDs are purged, the values of these variables are lost.

- `gtid_next`

System Variable	`gtid_next`
Scope	Session
Dynamic	Yes
Type	Enumeration
Default Value	`AUTOMATIC`
Valid Values	`AUTOMATIC` `ANONYMOUS`

	`UUID:NUMBER`

This variable is used to specify whether and how the next GTID is obtained.

Setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.8.1, “System Variable Privileges”.

`gtid_next` can take any of the following values:

- `AUTOMATIC`: Use the next automatically-generated global transaction ID.
- `ANONYMOUS`: Transactions do not have global identifiers, and are identified by file and position only.
- A global transaction ID in *`UUID`*:*`NUMBER`* format.

Setting this variable has no effect if `gtid_mode` is `OFF`.

Prior to MySQL 5.6.20, when GTIDs were enabled but `gtid_next` was not `AUTOMATIC`, `DROP TABLE` did not work correctly when used on a combination of nontemporary tables with temporary tables, or of temporary tables using transactional storage engines with temporary tables using nontransactional storage engines. In MySQL 5.6.20 and higher, `DROP TABLE` or `DROP TEMPORARY TABLE` fails with an explicit error when used with either of these combinations of tables. (Bug #17620053)

In MySQL 5.6.11 only, you cannot execute any of the statements `CHANGE MASTER TO`, `START SLAVE`, `STOP SLAVE`, `REPAIR TABLE`, `OPTIMIZE TABLE`, `ANALYZE TABLE`, `CHECK TABLE`, `CREATE SERVER`, `ALTER SERVER`, `DROP SERVER`, `CACHE INDEX`, `LOAD INDEX INTO CACHE`, `FLUSH`, or `RESET` when `gtid_next` is set to any value other than `AUTOMATIC`; in such cases, the statement fails with an error. Such statements are *not* disallowed in MySQL 5.6.12 and later. (Bug #16062608, Bug #16715809, Bug #69045)

- `gtid_owned`

System Variable	`gtid_owned`
Scope	Global, Session
Dynamic	No
Type	String
Unit	`set of GTIDs`

This read-only variable holds a list whose contents depend on its scope. When used with session scope, the list holds all GTIDs that are owned by this client; when used with global scope, it holds a list of all GTIDs along with their owners.

- `gtid_purged`

System Variable	`gtid_purged`
Scope	Global
Dynamic	Yes
Type	String

Unit	set of GTIDs

The set of all transactions that have been purged from the binary log. This is a subset of the set of transactions in gtid_executed.

When the server starts, the global value of gtid_purged is initialized to the set of GTIDs contained by the Previous_gtid_log_event of the oldest binary log. When a binary log is purged, gtid_purged is re-read from the binary log that has now become the oldest one.

To update the value of this variable, gtid_mode must be ON, gtid_executed must be the empty string, and therefore gtid_purged is also the empty string. This can occur either when replication has not been started previously, or when replication was not previously using GTIDs.

After executing SET gtid_purged, you should note down the current binary log file name, which can be checked using SHOW MASTER STATUS. If the server is restarted before this file has been purged, then you should use binlog_gtid_simple_recovery=0 (the default in 5.6) to avoid gtid_purged or gtid_executed being computed incorrectly.

Issuing RESET MASTER causes the value of this variable to be reset to an empty string.

- simplified_binlog_gtid_recovery

Command-Line Format	--simplified-binlog-gtid-recovery[={OFF|ON}]
Introduced	5.6.21
Deprecated	5.6.23
System Variable	simplified_binlog_gtid_recovery
Scope	Global
Dynamic	No
Type	Boolean
Default Value	OFF

This option is deprecated; expect it to be removed in a future MySQL release. Use the binlog_gtid_simple_recovery variable to control how MySQL iterates through binary log files after a crash.

17.1.5 Common Replication Administration Tasks

Once replication has been started it should execute without requiring much regular administration. Depending on your replication environment, you should check the replication status of each slave periodically, daily, or even more frequently.

17.1.5.1 Checking Replication Status

The most common task when managing a replication process is to ensure that replication is taking place and that there have been no errors between the slave and the source. The primary statement for this is SHOW SLAVE STATUS, which you must execute on each slave:

```
mysql> SHOW SLAVE STATUS\G
*************************** 1. row ***************************
               Slave_IO_State: Waiting for master to send event
                  Master_Host: source1
                  Master_User: root
                  Master_Port: 3306
                Connect_Retry: 60
              Master_Log_File: mysql-bin.000004
          Read_Master_Log_Pos: 931
               Relay_Log_File: slave1-relay-bin.000056
                Relay_Log_Pos: 950
```

```
       Relay_Master_Log_File: mysql-bin.000004
            Slave_IO_Running: Yes
           Slave_SQL_Running: Yes
             Replicate_Do_DB:
         Replicate_Ignore_DB:
          Replicate_Do_Table:
      Replicate_Ignore_Table:
     Replicate_Wild_Do_Table:
  Replicate_Wild_Ignore_Table:
                   Last_Errno: 0
                   Last_Error:
                 Skip_Counter: 0
          Exec_Master_Log_Pos: 931
              Relay_Log_Space: 1365
              Until_Condition: None
               Until_Log_File:
                Until_Log_Pos: 0
           Master_SSL_Allowed: No
           Master_SSL_CA_File:
           Master_SSL_CA_Path:
              Master_SSL_Cert:
            Master_SSL_Cipher:
               Master_SSL_Key:
       Seconds_Behind_Master: 0
Master_SSL_Verify_Server_Cert: No
               Last_IO_Errno: 0
               Last_IO_Error:
              Last_SQL_Errno: 0
              Last_SQL_Error:
  Replicate_Ignore_Server_Ids: 0
```

The key fields from the status report to examine are:

- `Slave_IO_State`: The current status of the slave. See Section 8.14.7, “Replication Replica I/O Thread States”, and Section 8.14.8, “Replication Replica SQL Thread States”, for more information.
- `Slave_IO_Running`: Whether the I/O thread for reading the source's binary log is running. Normally, you want this to be `Yes` unless you have not yet started replication or have explicitly stopped it with `STOP SLAVE`.
- `Slave_SQL_Running`: Whether the SQL thread for executing events in the relay log is running. As with the I/O thread, this should normally be `Yes`.
- `Last_IO_Error`, `Last_SQL_Error`: The last errors registered by the I/O and SQL threads when processing the relay log. Ideally these should be blank, indicating no errors.
- `Seconds_Behind_Master`: The number of seconds that the slave SQL thread is behind processing the source's binary log. A high number (or an increasing one) can indicate that the slave is unable to handle events from the source in a timely fashion.

  A value of 0 for `Seconds_Behind_Master` can usually be interpreted as meaning that the slave has caught up with the source, but there are some cases where this is not strictly true. For example, this can occur if the network connection between source and slave is broken but the slave I/O thread has not yet noticed this—that is, `slave_net_timeout` has not yet elapsed.

  It is also possible that transient values for `Seconds_Behind_Master` may not reflect the situation accurately. When the slave SQL thread has caught up on I/O, `Seconds_Behind_Master` displays 0; but when the slave I/O thread is still queuing up a new event, `Seconds_Behind_Master` may show a large value until the SQL thread finishes executing the new event. This is especially likely when the events have old timestamps; in such cases, if you execute `SHOW SLAVE STATUS` several times in a relatively short period, you may see this value change back and forth repeatedly between 0 and a relatively large value.

Several pairs of fields provide information about the progress of the slave in reading events from the source's binary log and processing them in the relay log:

- (`Master_Log_file`, `Read_Master_Log_Pos`): Coordinates in the source's binary log indicating how far the slave I/O thread has read events from that log.
- (`Relay_Master_Log_File`, `Exec_Master_Log_Pos`): Coordinates in the source's binary log indicating how far the slave SQL thread has executed events received from that log.
- (`Relay_Log_File`, `Relay_Log_Pos`): Coordinates in the slave relay log indicating how far the slave SQL thread has executed the relay log. These correspond to the preceding coordinates, but are expressed in the replica's relay log coordinates rather than the source's binary log coordinates.

The `SHOW STATUS` statement also provides some information relating specifically to replication slaves. The replication heartbeat information displayed by `SHOW STATUS` lets you check that the replication connection is active even if the source has not sent events to the slave recently. The source sends a heartbeat signal to a slave if there are no updates to, and no unsent events in, the binary log for a longer period than the heartbeat interval. The `MASTER_HEARTBEAT_PERIOD` setting on the source (set by the `CHANGE MASTER TO` statement) specifies the frequency of the heartbeat, which defaults to half of the connection timeout interval for the slave (`slave_net_timeout`). The `Slave_last_heartbeat` variable for `SHOW STATUS` shows when the replication slave last received a heartbeat signal.

On the source, you can check the status of connected slaves using `SHOW PROCESSLIST` to examine the list of running processes. Slave connections have `Binlog Dump` in the `Command` field:

```
mysql> SHOW PROCESSLIST \G;
*************************** 4. row ***************************
     Id: 10
   User: root
   Host: slave1:58371
     db: NULL
Command: Binlog Dump
   Time: 777
  State: Has sent all binlog to slave; waiting for binlog to be updated
   Info: NULL
```

Because it is the slave that drives the replication process, very little information is available in this report.

For slaves that were started with the `--report-host` option and are connected to the source, the `SHOW SLAVE HOSTS` statement on the source shows basic information about the slaves. The output includes the ID of the slave server, the value of the `--report-host` option, the connecting port, and source ID:

```
mysql> SHOW SLAVE HOSTS;
+-----------+--------+------+------------------+-----------+
| Server_id | Host   | Port | Rpl_recovery_rank | Master_id |
+-----------+--------+------+------------------+-----------+
|        10 | slave1 | 3306 |                 0 |         1 |
+-----------+--------+------+------------------+-----------+
1 row in set (0.00 sec)
```

17.1.5.2 Pausing Replication on the Slave

You can stop and start the replication of statements on the slave using the `STOP SLAVE` and `START SLAVE` statements.

To stop processing of the binary log from the source, use `STOP SLAVE`:

```
mysql> STOP SLAVE;
```

When replication is stopped, the slave I/O thread stops reading events from the source's binary log and writing them to the relay log, and the SQL thread stops reading events from the relay log and executing them. You can pause the I/O or SQL thread individually by specifying the thread type:

```
mysql> STOP SLAVE IO_THREAD;
mysql> STOP SLAVE SQL_THREAD;
```

To start execution again, use the `START SLAVE` statement:

```
mysql> START SLAVE;
```

To start a particular thread, specify the thread type:

```
mysql> START SLAVE IO_THREAD;
mysql> START SLAVE SQL_THREAD;
```

For a slave that performs updates only by processing events from the source, stopping only the SQL thread can be useful if you want to perform a backup or other task. The I/O thread continues to read events from the source but they are not executed. This makes it easier for the slave to catch up when you restart the SQL thread.

Stopping only the I/O thread enables the events in the relay log to be executed by the SQL thread up to the point where the relay log ends. This can be useful when you want to pause execution to catch up with events already received from the source, when you want to perform administration on the slave but also ensure that it has processed all updates to a specific point. This method can also be used to pause event receipt on the slave while you conduct administration on the source. Stopping the I/O thread but permitting the SQL thread to run helps ensure that there is not a massive backlog of events to be executed when replication is started again.

17.2 Replication Implementation

Replication is based on the replication source server keeping track of all changes to its databases (updates, deletes, and so on) in its binary log. The binary log serves as a written record of all events that modify database structure or content (data) from the moment the server was started. Typically, `SELECT` statements are not recorded because they modify neither database structure nor content.

Each replica that connects to the replication source server requests a copy of the binary log. That is, it pulls the data from the source, rather than the source pushing the data to the replica. The replica also executes the events from the binary log that it receives. This has the effect of repeating the original changes just as they were made on the source. Tables are created or their structure modified, and data is inserted, deleted, and updated according to the changes that were originally made on the source.

Because each replica is independent, the replaying of the changes from the source's binary log occurs independently on each replica that is connected to the source. In addition, because each replica receives a copy of the binary log only by requesting it from the source, the replica is able to read and update the copy of the database at its own pace and can start and stop the replication process at will without affecting the ability to update to the latest database status on either the source or replica side.

For more information on the specifics of the replication implementation, see Section 17.2.1, “Replication Threads”.

Sources and replicas report their status in respect of the replication process regularly so that you can monitor them. See Section 8.14, “Examining Server Thread (Process) Information”, for descriptions of all replicated-related states.

The source's binary log is written to a local relay log on the replica before it is processed. The replica also records information about the current position with the source's binary log and the local relay log. See Section 17.2.2, “Relay Log and Replication Metadata Repositories”.

Database changes are filtered on the replica according to a set of rules that are applied according to the various configuration options and variables that control event evaluation. For details on how these rules are applied, see Section 17.2.3, “How Servers Evaluate Replication Filtering Rules”.

17.2.1 Replication Threads

MySQL replication capabilities are implemented using three main threads, one on the source server and two on the replica:

- **Binary log dump thread.** The source creates a thread to send the binary log contents to a replica when the replica connects. This thread can be identified in the output of `SHOW PROCESSLIST` on the source as the `Binlog Dump` thread.

  The binary log dump thread acquires a lock on the source's binary log for reading each event that is to be sent to the replica. As soon as the event has been read, the lock is released, even before the event is sent to the replica.

- **Replication I/O thread.** When a `START SLAVE` statement is issued on a replica server, the replica creates an I/O thread, which connects to the source and asks it to send the updates recorded in its binary logs.

  The replication I/O thread reads the updates that the source's `Binlog Dump` thread sends (see previous item) and copies them to local files that comprise the replica's relay log.

  The state of this thread is shown as `Slave_IO_running` in the output of `SHOW SLAVE STATUS`.

- **Replication SQL thread.** The replica creates an SQL thread to read the relay log that is written by the replication I/O thread and execute the transactions contained in it.

There are three main threads for each source/replica connection. A source that has multiple replicas creates one binary log dump thread for each currently connected replica, and each replica has its own replication I/O and SQL threads.

A replica uses two threads to separate reading updates from the source and executing them into independent tasks. Thus, the task of reading transactions is not slowed down if the process of applying them is slow. For example, if the replica server has not been running for a while, its I/O thread can quickly fetch all the binary log contents from the source when the replica starts, even if the SQL thread lags far behind. If the replica stops before the SQL thread has executed all the fetched statements, the I/O thread has at least fetched everything so that a safe copy of the transactions is stored locally in the replica's relay logs, ready for execution the next time that the replica starts.

You can enable further parallelization for tasks on a replica by setting the `slave_parallel_workers` system variable to a value greater than 0 (the default). When this system variable is set, the replica creates the specified number of worker threads to apply transactions, plus a coordinator thread to manage them. A replica with `slave_parallel_workers` set to a value greater than 0 is called a multithreaded replica. With this setup, transactions that fail can be retried.

Note

Multithreaded replicas are not currently supported by NDB Cluster, which silently ignores the setting for this variable. See Section 18.6.3, “Known Issues in NDB Cluster Replication” for more information.

17.2.1.1 Monitoring Replication Main Threads

The `SHOW PROCESSLIST` statement provides information that tells you what is happening on the source and on the replica regarding replication. For information on source states, see Section 8.14.6, “Replication Source Thread States”. For replica states, see Section 8.14.7, “Replication Replica I/O Thread States”, and Section 8.14.8, “Replication Replica SQL Thread States”.

The following example illustrates how the three main replication threads, the binary log dump thread, replicatin I/O thread, and replication SQL thread, show up in the output from `SHOW PROCESSLIST`.

On the source server, the output from `SHOW PROCESSLIST` looks like this:

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 2
   User: root
```

```
   Host: localhost:32931
     db: NULL
Command: Binlog Dump
   Time: 94
  State: Has sent all binlog to slave; waiting for binlog to
         be updated
   Info: NULL
```

Here, thread 2 is a `Binlog Dump` thread that services a connected replica. The `State` information indicates that all outstanding updates have been sent to the replica and that the source is waiting for more updates to occur. If you see no `Binlog Dump` threads on a source server, this means that replication is not running; that is, no replicas are currently connected.

On a replica server, the output from `SHOW PROCESSLIST` looks like this:

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 10
   User: system user
   Host:
     db: NULL
Command: Connect
   Time: 11
  State: Waiting for master to send event
   Info: NULL
*************************** 2. row ***************************
     Id: 11
   User: system user
   Host:
     db: NULL
Command: Connect
   Time: 11
  State: Has read all relay log; waiting for the slave I/O
         thread to update it
   Info: NULL
```

The `State` information indicates that thread 10 is the replication I/O thread that is communicating with the source server, and thread 11 is the replication SQL thread that is processing the updates stored in the relay logs. At the time that `SHOW PROCESSLIST` was run, both threads were idle, waiting for further updates.

The value in the `Time` column can show how late the replica is compared to the source. See Section A.14, “MySQL 5.6 FAQ: Replication”. If sufficient time elapses on the source side without activity on the `Binlog Dump` thread, the source determines that the replica is no longer connected. As for any other client connection, the timeouts for this depend on the values of `net_write_timeout` and `net_retry_count`; for more information about these, see Section 5.1.7, “Server System Variables”.

The `SHOW SLAVE STATUS` statement provides additional information about replication processing on a replica server. See Section 17.1.5.1, “Checking Replication Status”.

17.2.1.2 Monitoring Replication Applier Worker Threads

A multithreaded replica's coordinator thread prints statistics to the replica's error log on a regular basis if the verbosity setting is set to display informational messages. The statistics are printed depending on the volume of events that the coordinator thread has assigned to applier worker threads, with a maximum frequency of once every 120 seconds. The message lists the following statistics:

Seconds elapsed	The difference in seconds between the current time and the last time this information was printed to the error log.
Events assigned	The total number of events that the coordinator thread has queued to all applier worker threads since the coordinator thread was started.

Worker queues filled over overrun level	The current number of events that are queued to any of the applier worker threads in excess of the overrun level, which is set at 90% of the maximum queue length of 16384 events. If this value is zero, no applier worker threads are operating at the upper limit of their capacity.
Waited due to worker queue full	The number of times that the coordinator thread had to wait to schedule an event because an applier worker thread's queue was full. If this value is zero, no applier worker threads exhausted their capacity.
Waited due to the total size	The number of times that the coordinator thread had to wait to schedule an event because the `slave_pending_jobs_size_max` limit had been reached. This system variable sets the maximum amount of memory (in bytes) available to applier worker thread queues holding events not yet applied. If an unusually large event exceeds this size, the transaction is held until all the applier worker threads have empty queues, and then processed. All subsequent transactions are held until the large transaction has been completed.
Waited at clock conflicts	The number of nanoseconds that the coordinator thread had to wait to schedule an event because a transaction that the event depended on had not yet been committed. If `slave_parallel_type` is set to `DATABASE` (rather than `LOGICAL_CLOCK`), this value is always zero.
Waited (count) when workers occupied	The number of times that the coordinator thread slept for a short period, which it might do in two situations. The first situation is where the coordinator thread assigns an event and finds the applier worker thread's queue is filled beyond the underrun level of 10% of the maximum queue length, in which case it sleeps for a maximum of 1 millisecond. The second situation is where `slave_parallel_type` is set to `LOGICAL_CLOCK` and the coordinator thread needs to assign the first event of a transaction to an applier worker thread's queue, it only does this to a worker with an empty queue, so if no queues are empty, the coordinator thread sleeps until one becomes empty.
Waited when workers occupied	The number of nanoseconds that the coordinator thread slept while waiting for an empty applier worker thread queue (that is, in the second situation described above, where `slave_parallel_type` is set to `LOGICAL_CLOCK` and the first event of a transaction needs to be assigned).

17.2.2 Relay Log and Replication Metadata Repositories

A replica server creates several repositories of information to use for the replication process:

- The replica's *relay log*, which is written by the replication I/O thread, contains the transactions read from the replication source server's binary log. The transactions in the relay log are applied on the replica by the replication SQL thread. For information about the relay log, see Section 17.2.2.1, “The Relay Log”.
- The replica's *connection metadata repository* contains information that the replication I/O thread needs to connect to the replication source server and retrieve transactions from the source's binary log. The connection metadata repository is written to the `master.info` file or to the `mysql.slave_master_info` table.
- The replica's *applier metadata repository* contains information that the replication SQL thread needs to read and apply transactions from the replica's relay log. The applier metadata repository is written to the `relay-log.info` file or to the `mysql.slave_relay_log_info` table.

The connection metadata repository and applier metadata repository are collectively known as the replication metadata repositories. For information about these, see Section 17.2.2.2, “Replication Metadata Repositories”.

The `mysql.slave_master_info` and `mysql.slave_relay_log_info` tables are created using the transactional storage engine `InnoDB`. Updates to the replica's applier metadata repository table are committed together with the transactions, meaning that the replica's progress information recorded in that repository is always consistent with what has been applied to the database, even in the event of an unexpected server halt. The `--relay-log-recovery` option must be enabled on the replica to guarantee resilience. For more details, see Section 17.3.2, “Handling an Unexpected Halt of a Replica Server”.

17.2.2.1 The Relay Log

The relay log, like the binary log, consists of a set of numbered files containing events that describe database changes, and an index file that contains the names of all used relay log files.

The term “relay log file” generally denotes an individual numbered file containing database events. The term “relay log” collectively denotes the set of numbered relay log files plus the index file.

Relay log files have the same format as binary log files and can be read using `mysqlbinlog` (see Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”).

By default, relay log file names have the form *`host_name`*`-relay-bin.`*`nnnnnn`* in the data directory, where *`host_name`* is the name of the replica server host and *`nnnnnn`* is a sequence number. Successive relay log files are created using successive sequence numbers, beginning with `000001`. The replica uses an index file to track the relay log files currently in use. The default relay log index file name is *`host_name`*`-relay-bin.index` in the data directory.

The default relay log file and relay log index file names can be overridden with, respectively, the `relay_log` and `relay_log_index` system variables (see Section 17.1.4, “Replication and Binary Logging Options and Variables”).

If a replica uses the default host-based relay log file names, changing the replica server's host name after replication has been set up can cause replication to fail with the errors `Failed to open the relay log` and `Could not find target log during relay log initialization`. This is a known issue (see Bug #2122). If you anticipate that a replica's host name might change in the future (for example, if networking is set up on the replica such that its host name can be modified using DHCP), you can avoid this issue entirely by using the `relay_log` and `relay_log_index` system variables to specify relay log file names explicitly when you initially set up the replica. This makes the names independent of server host name changes.

If you encounter the issue after replication has already begun, one way to work around it is to stop the replica server, prepend the contents of the old relay log index file to the new one, and then restart the replica. On a Unix system, this can be done as shown here:

```
shell> cat new_relay_log_name.index >> old_relay_log_name.index
shell> mv old_relay_log_name.index new_relay_log_name.index
```

A replica server creates a new relay log file under the following conditions:

- Each time the I/O thread starts.
- When the logs are flushed (for example, with `FLUSH LOGS` or `mysqladmin flush-logs`).
- When the size of the current relay log file becomes “too large,” determined as follows:
  - If the value of `max_relay_log_size` is greater than 0, that is the maximum relay log file size.
  - If the value of `max_relay_log_size` is 0, `max_binlog_size` determines the maximum relay log file size.

The SQL thread automatically deletes each relay log file after it has executed all events in the file and no longer needs it. There is no explicit mechanism for deleting relay logs because the SQL thread takes care of doing so. However, `FLUSH LOGS` rotates relay logs, which influences when the SQL thread deletes them.

17.2.2.2 Replication Metadata Repositories

A replica server creates two replication metadata repositories, the connection metadata repository and the applier metadata repository. The replication metadata repositories survive a replica server's shutdown. If binary log file position based replication is in use, when the replica restarts, it reads the two repositories to determine how far it previously proceeded in reading the binary log from the source and in processing its own relay log.

- The replica's *connection metadata repository* contains information that the replication I/O thread needs to connect to the replication source server and retrieve transactions from the source's binary log. The metadata in this repository includes the connection configuration, the replication user account details, the SSL settings for the connection, and the file name and position where the replication I/O thread is currently reading from the source's binary log.
- The replica's *applier metadata repository* contains information that the replication SQL thread needs to read and apply transactions from the replica's relay log. The metadata in this repository includes the file name and position up to which the replication SQL thread has executed the transactions in the relay log, and the equivalent position in the source's binary log. It also includes metadata for the process of applying transactions, such as the number of worker threads.

By default, the replication metadata repositories are created as files in the data directory named `master.info` and `relay-log.info`, or with alternative names and locations specified by the `--master-info-file` option and `relay_log_info_file` system variable. To create the replication metadata repositories as tables, specify `master_info_repository=TABLE` and `relay_log_info_repository=TABLE` at server startup. In that case, the replica's connection metadata repository is written to the `slave_master_info` table in the `mysql` system schema, and the replica's applier metadata repository is written to the `slave_relay_log_info` table in the `mysql` system schema. A warning message is issued if `mysqld` is unable to initialize the tables for the replication metadata repositories, but the replica is allowed to continue starting. This situation is most likely to occur when upgrading from a version of MySQL that does not support the use of tables for the repositories to one in which they are supported.

Important

1. Do not attempt to update or insert rows in the `mysql.slave_master_info` or `mysql.slave_relay_log_info` tables manually. Doing so can cause undefined behavior, and is not supported. Execution of any statement requiring a write lock on either or both of the `slave_master_info` and `slave_relay_log_info` tables is disallowed while replication is ongoing (although statements that perform only reads are permitted at any time).
2. Access to the replica's connection metadata repository file or table should be restricted to the database administrator, because it contains the replication user account name and password for connecting to the source. Use a restricted access mode to protect database backups that include this repository.

`RESET SLAVE` clears the data in the replication metadata repositories, with the exception of the replication connection parameters (depending on the MySQL Server release and repository type). For details, see the description for `RESET SLAVE`.

In MySQL 5.6.5 and earlier, the `slave_master_info` and `slave_relay_log_info` tables used `MyISAM` by default, which meant that it was necessary before starting replication to change the storage engine used by these tables by issuing `ALTER TABLE ... ENGINE=InnoDB`, as shown here:

```
ALTER TABLE mysql.slave_master_info ENGINE=InnoDB;
ALTER TABLE mysql.slave_relay_log_info ENGINE=InnoDB;
```

The `ALTER TABLE` statements must be executed by the MySQL `root` or other user account with the appropriate privileges on the `mysql` system database. You should not attempt to do this while replication is running; beginning with MySQL 5.6.3, trying to execute an `ALTER TABLE` on either these tables while replication is ongoing is disallowed.

From MySQL 5.6.6, if you set `master_info_repository` and `relay_log_info_repository` to `TABLE`, the `mysql.slave_master_info` and `mysql.slave_relay_log_info` tables are created using the `InnoDB` transactional storage engine. As a table, updates to the replica's applier metadata repository are committed together with the transactions, meaning that the replica's progress information recorded in that log is always consistent with what has been applied to the database, even in the event of an unexpected server halt. The `--relay-log-recovery` option must be enabled on the replica to guarantee resilience. For more details, see Section 17.3.2, “Handling an Unexpected Halt of a Replica Server”.

When you back up the replica's data or transfer a snapshot of its data to create a new replica, ensure that you include the `mysql.slave_master_info` and `mysql.slave_relay_log_info` tables containing the replication metadata repositories, or the equivalent files (`master.info` and `relay-log.info` in the data directory, unless you specified alternative names and locations). When binary log file position based replication is in use, the replication metadata repositories are needed to resume replication after restarting the restored or copied replica. If you do not have the relay log files, but still have the replica's applier metadata repository, you can check it to determine how far the replication SQL thread has executed in the source's binary log. Then you can use a `CHANGE MASTER TO` statement with the `MASTER_LOG_FILE` and `MASTER_LOG_POS` options to tell the replica to re-read the binary logs from the source from that point (provided that the required binary logs still exist on the source).

One additional repository, the applier worker metadata repository, is created primarily for internal use, and holds status information about worker threads on a multithreaded replica. The applier worker metadata repository includes the names and positions for the relay log file and the source's binary log file for each worker thread. If the replica's applier metadata repository is created as a table, which is the default, the applier worker metadata repository is written to the `mysql.slave_worker_info` table. If the applier metadata repository is written to a file, the applier worker metadata repository is written to the `worker-relay-log.info` file. For external use, status information for worker threads is presented in the Performance Schema `replication_applier_status_by_worker` table.

The replication metadata repositories originally contained information similar to that shown in the output of the `SHOW SLAVE STATUS` statement, which is discussed in Section 13.4.2, “SQL Statements for Controlling Replica Servers”. Further information has since been added to the replication metadata repositories which is not displayed by the `SHOW SLAVE STATUS` statement.

For the connection metadata repository, the following table shows the correspondence between the columns in the `mysql.slave_master_info` table, the columns displayed by `SHOW SLAVE STATUS`, and the lines in the `master.info` file.

`master.info` File Line	`slave_master_info` Table Column	`SHOW SLAVE STATUS` Column	Description
1	`Number_of_lines`	[None]	Number of lines in the file, or columns in the table
2	`Master_log_name`	`Master_Log_File`	The name of the binary log currently being read from the source

`master.info` File Line	`slave_master_info` Table Column	`SHOW SLAVE STATUS` Column	Description
3	`Master_log_pos`	`Read_Master_Log_Pos`	The current position within the binary log that has been read from the source
4	`Host`	`Master_Host`	The host name of the source server
5	`User_name`	`Master_User`	The replication user name used to connect to the source
6	`User_password`	Password (not shown by `SHOW SLAVE STATUS`)	The password used to connect to the source
7	`Port`	`Master_Port`	The network port used to connect to the source
8	`Connect_retry`	`Connect_Retry`	The period (in seconds) that the replica waits before trying to reconnect to the source
9	`Enabled_ssl`	`Master_SSL_Allowed`	Indicates whether the server supports SSL connections
10	`Ssl_ca`	`Master_SSL_CA_File`	The file used for the Certificate Authority (CA) certificate
11	`Ssl_capath`	`Master_SSL_CA_Path`	The path to the Certificate Authority (CA) certificates
12	`Ssl_cert`	`Master_SSL_Cert`	The name of the SSL certificate file
13	`Ssl_cipher`	`Master_SSL_Cipher`	The list of possible ciphers used in the handshake for the SSL connection
14	`Ssl_key`	`Master_SSL_Key`	The name of the SSL key file
15	`Ssl_verify_server_cert`	`Master_SSL_Verify_Server_Cert`	Whether to verify the server certificate
16	`Heartbeat`	[None]	Interval between replication

`master.info` File Line	`slave_master_info` Table Column	`SHOW SLAVE STATUS` Column	Description
			heartbeats, in seconds
17	`Bind`	`Master_Bind`	Which of the replica's network interfaces should be used for connecting to the source
18	`Ignored_server_ids`	`Replicate_Ignore_Server_Ids`	The list of server IDs to be ignored. Note that for `Ignored_serv` the list of server IDs is preceded by the total number of server IDs to ignore.
19	`Uuid`	`Master_UUID`	The source's unique ID
20	`Retry_count`	`Master_Retry_Count`	Maximum numbe of reconnection attempts permitted
21	`Ssl_crl`	[None]	Path to an SSL certificate revocation-list file
22	`Ssl_crl_path`	[None]	Path to a directo containing SSL certificate revocation-list files
23	`Enabled_auto_position`	`Auto_position`	If autopositioning is in use or not (added in MySQ version 5.6.5)

For the applier metadata repository, the following table shows the correspondence between the columns in the `mysql.slave_relay_log_info` table, the columns displayed by `SHOW SLAVE STATUS`, and the lines in the `relay-log.info` file.

Line in `relay-log.info`	`slave_relay_log_info` Table Column	`SHOW SLAVE STATUS` Column	Description
1	`Number_of_lines`	[None]	Number of lines in the file or columns in the table
2	`Relay_log_name`	`Relay_Log_File`	The name of the current relay log file
3	`Relay_log_pos`	`Relay_Log_Pos`	The current positio within the relay log file; events up to this position have

Line in `relay-log.info`	`slave_relay_log_info` Table Column	`SHOW SLAVE STATUS` Column	Description
			been executed on the replica database
4	`Master_log_name`	`Relay_Master_Log_File`	The name of the source's binary log file from which the events in the relay log file were read
5	`Master_log_pos`	`Exec_Master_Log_Pos`	The equivalent position within the source's binary log file of events that have already been executed
5	`Sql_delay`	`SQL_Delay`	The number of seconds that the replica must lag the source
6	`Number_of_workers`	[None]	The number of worker threads on the replica for executing replication events (transactions) in parallel
7	`Id`	[None]	ID used for internal purposes; currently this is always 1

Prior to MySQL 5.6, the `relay-log.info` file does not include a line count or a delay value (and the `slave_relay_log_info` table is not available).

Line	Status Column	Description
1	`Relay_Log_File`	The name of the current relay log file
2	`Relay_Log_Pos`	The current position within the relay log file; events up to this position have been executed on the replica database
3	`Relay_Master_Log_File`	The name of the source's binary log file from which the events in the relay log file were read
4	`Exec_Master_Log_Pos`	The equivalent position within the source's binary log file of events that have already been executed

Note

If you downgrade a replica server to a version older than MySQL 5.6, the older server does not read the `relay-log.info` file correctly. To address this, modify the file in a text editor by deleting the initial line containing the number of lines.

The contents of the `relay-log.info` file and the states shown by the `SHOW SLAVE STATUS` statement might not match if the `relay-log.info` file has not been flushed to disk. Ideally, you should only view `relay-log.info` on a replica that is offline (that is, `mysqld` is not running). For

a running system, you can use `SHOW SLAVE STATUS`, or query the `slave_master_info` and `slave_relay_log_info` tables if you are writing the replication metadata repositories to tables.

17.2.3 How Servers Evaluate Replication Filtering Rules

If a replication source server does not write a statement to its binary log, the statement is not replicated. If the server does log the statement, the statement is sent to all replicas and each replica determines whether to execute it or ignore it.

On the source, you can control which databases to log changes for by using the `--binlog-do-db` and `--binlog-ignore-db` options to control binary logging. For a description of the rules that servers use in evaluating these options, see Section 17.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”. You should not use these options to control which databases and tables are replicated. Instead, use filtering on the replica to control the events that are executed on the replica.

On the replica side, decisions about whether to execute or ignore statements received from the source are made according to the `--replicate-*` options that the replica was started with. (See Section 17.1.4, “Replication and Binary Logging Options and Variables”.)

In the simplest case, when there are no `--replicate-*` options, the replica executes all statements that it receives from the source. Otherwise, the result depends on the particular options given.

Database-level options (`--replicate-do-db`, `--replicate-ignore-db`) are checked first; see Section 17.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”, for a description of this process. If no database-level options are used, option checking proceeds to any table-level options that may be in use (see Section 17.2.3.2, “Evaluation of Table-Level Replication Options”, for a discussion of these). If one or more database-level options are used but none are matched, the statement is not replicated.

For statements affecting databases only (that is, `CREATE DATABASE`, `DROP DATABASE`, and `ALTER DATABASE`), database-level options always take precedence over any `--replicate-wild-do-table` options. In other words, for such statements, `--replicate-wild-do-table` options are checked if and only if there are no database-level options that apply. This is a change in behavior from previous versions of MySQL, where the statement `CREATE DATABASE dbx` was not replicated if the replica had been started with `--replicate-do-db=dbx --replicate-wild-do-table=db%.t1`. (Bug #46110)

To make it easier to determine what effect an option set has, it is recommended that you avoid mixing `*-do-*` and `*-ignore-*` options, or wildcard options with options that do not employ wildcards.

If any `--replicate-rewrite-db` options were specified, they are applied before the `--replicate-*` filtering rules are tested.

Note

In MySQL 5.6, all replication filtering options follow the same rules for case sensitivity that apply to names of databases and tables elsewhere in the MySQL server, including the effects of the `lower_case_table_names` system variable.

This is a change from previous versions of MySQL. (Bug #51639)

17.2.3.1 Evaluation of Database-Level Replication and Binary Logging Options

When evaluating replication options, the replica begins by checking to see whether there are any `--replicate-do-db` or `--replicate-ignore-db` options that apply. When using `--binlog-do-db` or `--binlog-ignore-db`, the process is similar, but the options are checked on the source.

The database that is checked for a match depends on the binary log format of the statement that is being handled. If the statement has been logged using the row format, the database where data is to

be changed is the database that is checked. If the statement has been logged using the statement format, the default database (specified with a `USE` statement) is the database that is checked.

Note

Only DML statements can be logged using the row format. DDL statements are always logged as statements, even when `binlog_format=ROW`. All DDL statements are therefore always filtered according to the rules for statement-based replication. This means that you must select the default database explicitly with a `USE` statement in order for a DDL statement to be applied.

For replication, the steps involved are listed here:

1. Which logging format is used?
   - **STATEMENT.** Test the default database.
   - **ROW.** Test the database affected by the changes.
2. Are there any `--replicate-do-db` options?
   - **Yes.** Does the database match any of them?
     - **Yes.** Continue to Step 4.
     - **No.** Ignore the update and exit.
   - **No.** Continue to step 3.
3. Are there any `--replicate-ignore-db` options?
   - **Yes.** Does the database match any of them?
     - **Yes.** Ignore the update and exit.
     - **No.** Continue to step 4.
   - **No.** Continue to step 4.
4. Proceed to checking the table-level replication options, if there are any. For a description of how these options are checked, see Section 17.2.3.2, “Evaluation of Table-Level Replication Options”.

Important

A statement that is still permitted at this stage is not yet actually executed. The statement is not executed until all table-level options (if any) have also been checked, and the outcome of that process permits execution of the statement.

For binary logging, the steps involved are listed here:

1. Are there any `--binlog-do-db` or `--binlog-ignore-db` options?
   - **Yes.** Continue to step 2.
   - **No.** Log the statement and exit.
2. Is there a default database (has any database been selected by `USE`)?
   - **Yes.** Continue to step 3.
   - **No.** Ignore the statement and exit.
3. There is a default database. Are there any `--binlog-do-db` options?

- **Yes.** Do any of them match the database?
  - **Yes.** Log the statement and exit.
  - **No.** Ignore the statement and exit.
- **No.** Continue to step 4.

4. Do any of the `--binlog-ignore-db` options match the database?
   - **Yes.** Ignore the statement and exit.
   - **No.** Log the statement and exit.

Important

For statement-based logging, an exception is made in the rules just given for the `CREATE DATABASE`, `ALTER DATABASE`, and `DROP DATABASE` statements. In those cases, the database being *created, altered, or dropped* replaces the default database when determining whether to log or ignore updates.

`--binlog-do-db` can sometimes mean “ignore other databases”. For example, when using statement-based logging, a server running with only `--binlog-do-db=sales` does not write to the binary log statements for which the default database differs from `sales`. When using row-based logging with the same option, the server logs only those updates that change data in `sales`.

17.2.3.2 Evaluation of Table-Level Replication Options

The replica checks for and evaluates table options only if either of the following two conditions is true:

- No matching database options were found.
- One or more database options were found, and were evaluated to arrive at an “execute” condition according to the rules described in the previous section (see Section 17.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”).

First, as a preliminary condition, the replica checks whether statement-based replication is enabled. If so, and the statement occurs within a stored function, the replica executes the statement and exits. If row-based replication is enabled, the replica does not know whether a statement occurred within a stored function on the source, so this condition does not apply.

Note

For statement-based replication, replication events represent statements (all changes making up a given event are associated with a single SQL statement); for row-based replication, each event represents a change in a single table row (thus a single statement such as `UPDATE mytable SET mycol = 1` may yield many row-based events). When viewed in terms of events, the process of checking table options is the same for both row-based and statement-based replication.

Having reached this point, if there are no table options, the replica simply executes all events. If there are any `--replicate-do-table` or `--replicate-wild-do-table` options, the event must match one of these if it is to be executed; otherwise, it is ignored. If there are any `--replicate-ignore-table` or `--replicate-wild-ignore-table` options, all events are executed except those that match any of these options.

The following steps describe this evaluation in more detail. The starting point is the end of the evaluation of the database-level options, as described in Section 17.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”.

1. Are there any table replication options?

- **Yes.** Continue to step 2.
- **No.** Execute the update and exit.

2. Which logging format is used?
   - **STATEMENT.** Carry out the remaining steps for each statement that performs an update.
   - **ROW.** Carry out the remaining steps for each update of a table row.
3. Are there any `--replicate-do-table` options?
   - **Yes.** Does the table match any of them?
     - **Yes.** Execute the update and exit.
     - **No.** Continue to step 4.
   - **No.** Continue to step 4.
4. Are there any `--replicate-ignore-table` options?
   - **Yes.** Does the table match any of them?
     - **Yes.** Ignore the update and exit.
     - **No.** Continue to step 5.
   - **No.** Continue to step 5.
5. Are there any `--replicate-wild-do-table` options?
   - **Yes.** Does the table match any of them?
     - **Yes.** Execute the update and exit.
     - **No.** Continue to step 6.
   - **No.** Continue to step 6.
6. Are there any `--replicate-wild-ignore-table` options?
   - **Yes.** Does the table match any of them?
     - **Yes.** Ignore the update and exit.
     - **No.** Continue to step 7.
   - **No.** Continue to step 7.
7. Is there another table to be tested?
   - **Yes.** Go back to step 3.
   - **No.** Continue to step 8.
8. Are there any `--replicate-do-table` or `--replicate-wild-do-table` options?
   - **Yes.** Ignore the update and exit.
   - **No.** Execute the update and exit.

Note

Statement-based replication stops if a single SQL statement operates on both a table that is included by a `--replicate-do-table` or `--replicate-wild-do-table` option, and another table that is ignored by a `--replicate-ignore-table` or `--replicate-wild-ignore-table` option. The replica must either execute or ignore the complete statement (which forms a replication event), and it cannot logically do this. This also applies to row-based replication for DDL statements, because DDL statements are always logged as statements, without regard to the logging format in effect. The only type of statement that can update both an included and an ignored table and still be replicated successfully is a DML statement that has been logged with `binlog_format=ROW`.

17.2.3.3 Interactions Between Replication Filtering Options

If you use a combination of database-level and table-level replication filtering options, the replica first accepts or ignores events using the database options, then it evaluates all events permitted by those options according to the table options. This can sometimes lead to results that seem counterintuitive. It is also important to note that the results vary depending on whether the operation is logged using statement-based or row-based binary logging format. If you want to be sure that your replication filters always operate in the same way independently of the binary logging format, which is particularly important if you are using mixed binary logging format, follow the guidance in this topic.

The effect of the replication filtering options differs between binary logging formats because of the way the database name is identified. With statement-based format, DML statements are handled based on the current database, as specified by the `USE` statement. With row-based format, DML statements are handled based on the database where the modified table exists. DDL statements are always filtered based on the current database, as specified by the `USE` statement, regardless of the binary logging format.

An operation that involves multiple tables can also be affected differently by replication filtering options depending on the binary logging format. Operations to watch out for include transactions involving multi-table `UPDATE` statements, triggers, cascading foreign keys, stored functions that update multiple tables, and DML statements that invoke stored functions that update one or more tables. If these operations update both filtered-in and filtered-out tables, the results can vary with the binary logging format.

If you need to guarantee that your replication filters operate consistently regardless of the binary logging format, particularly if you are using mixed binary logging format (`binlog_format=MIXED`), use only table-level replication filtering options, and do not use database-level replication filtering options. Also, do not use multi-table DML statements that update both filtered-in and filtered-out tables.

If you need to use a combination of database-level and table-level replication filters, and want these to operate as consistently as possible, choose one of the following strategies:

1. If you use row-based binary logging format (`binlog_format=ROW`), for DDL statements, rely on the `USE` statement to set the database and do not specify the database name. You can consider changing to row-based binary logging format for improved consistency with replication filtering. See Section 5.4.4.2, “Setting The Binary Log Format” for the conditions that apply to changing the binary logging format.

2. If you use statement-based or mixed binary logging format (`binlog_format=STATEMENT` or `MIXED`), for both DML and DDL statements, rely on the `USE` statement and do not use the database name. Also, do not use multi-table DML statements that update both filtered-in and filtered-out tables.

Example 17.1 A `--replicate-ignore-db` option and a `--replicate-do-table` option

On the replication source server, the following statements are issued:

```
USE db1;
CREATE TABLE t2 LIKE t1;
INSERT INTO db2.t3 VALUES (1);
```

The replica has the following replication filtering options set:

```
replicate-ignore-db = db1
replicate-do-table = db2.t3
```

The DDL statement `CREATE TABLE` creates the table in `db1`, as specified by the preceding `USE` statement. The replica filters out this statement according to its `--replicate-ignore-db = db1` option, because `db1` is the current database. This result is the same whatever the binary logging format is on the replication source server. However, the result of the DML `INSERT` statement is different depending on the binary logging format:

- If row-based binary logging format is in use on the source (`binlog_format=ROW`), the replica evaluates the `INSERT` operation using the database where the table exists, which is named as `db2`. The database-level option `--replicate-ignore-db = db1`, which is evaluated first, therefore does not apply. The table-level option `--replicate-do-table = db2.t3` does apply, so the replica applies the change to table `t3`.
- If statement-based binary logging format is in use on the source (`binlog_format=STATEMENT`), the replica evaluates the `INSERT` operation using the default database, which was set by the `USE` statement to `db1` and has not been changed. According to its database-level `--replicate-ignore-db = db1` option, it therefore ignores the operation and does not apply the change to table `t3`. The table-level option `--replicate-do-table = db2.t3` is not checked, because the statement already matched a database-level option and was ignored.

If the `--replicate-ignore-db = db1` option on the replica is necessary, and the use of statement-based (or mixed) binary logging format on the source is also necessary, the results can be made consistent by omitting the database name from the `INSERT` statement and relying on a `USE` statement instead, as follows:

```
USE db1;
CREATE TABLE t2 LIKE t1;
USE db2;
INSERT INTO t3 VALUES (1);
```

In this case, the replica always evaluates the `INSERT` statement based on the database `db2`. Whether the operation is logged in statement-based or row-based binary format, the results remain the same.

17.3 Replication Solutions

Replication can be used in many different environments for a range of purposes. This section provides general notes and advice on using replication for specific solution types.

For information on using replication in a backup environment, including notes on the setup, backup procedure, and files to back up, see Section 17.3.1, “Using Replication for Backups”.

For advice and tips on using different storage engines on the source and replicas, see Section 17.3.3, “Using Replication with Different Source and Replica Storage Engines”.

Using replication as a scale-out solution requires some changes in the logic and operation of applications that use the solution. See Section 17.3.4, “Using Replication for Scale-Out”.

For performance or data distribution reasons, you may want to replicate different databases to different replicas. See Section 17.3.5, “Replicating Different Databases to Different Replicas”

As the number of replicas increases, the load on the source can increase and lead to reduced performance (because of the need to replicate the binary log to each replica). For tips on improving your replication performance, including using a single secondary server as a replication source server, see Section 17.3.6, “Improving Replication Performance”.

For guidance on switching sources, or converting replicas into sources as part of an emergency failover solution, see Section 17.3.7, “Switching Sources During Failover”.

To secure your replication communication, you can encrypt the communication channel. For step-by-step instructions, see Section 17.3.8, “Setting Up Replication to Use Encrypted Connections”.

17.3.1 Using Replication for Backups

To use replication as a backup solution, replicate data from the source to a replica, and then back up the replica. The replica can be paused and shut down without affecting the running operation of the source, so you can produce an effective snapshot of “live” data that would otherwise require the source to be shut down.

How you back up a database depends on its size and whether you are backing up only the data, or the data and the replica state so that you can rebuild the replica in the event of failure. There are therefore two choices:

- If you are using replication as a solution to enable you to back up the data on the source, and the size of your database is not too large, the `mysqldump` tool may be suitable. See Section 17.3.1.1, “Backing Up a Replica Using mysqldump”.
- For larger databases, where `mysqldump` would be impractical or inefficient, you can back up the raw data files instead. Using the raw data files option also means that you can back up the binary and relay logs that enable you to recreate the replica in the event of a replica failure. For more information, see Section 17.3.1.2, “Backing Up Raw Data from a Replica”.

Another backup strategy, which can be used for either source or replica servers, is to put the server in a read-only state. The backup is performed against the read-only server, which then is changed back to its usual read/write operational status. See Section 17.3.1.3, “Backing Up a Source or Replica by Making It Read Only”.

17.3.1.1 Backing Up a Replica Using mysqldump

Using `mysqldump` to create a copy of a database enables you to capture all of the data in the database in a format that enables the information to be imported into another instance of MySQL Server (see Section 4.5.4, “`mysqldump` — A Database Backup Program”). Because the format of the information is SQL statements, the file can easily be distributed and applied to running servers in the event that you need access to the data in an emergency. However, if the size of your data set is very large, `mysqldump` may be impractical.

When using `mysqldump`, you should stop replication on the replica before starting the dump process to ensure that the dump contains a consistent set of data:

1. Stop the replica from processing requests. You can stop replication completely on the replica using `mysqladmin`:

   ```
   shell> mysqladmin stop-slave
   ```

   Alternatively, you can stop only the replication SQL thread to pause event execution:

   ```
   shell> mysql -e 'STOP SLAVE SQL_THREAD;'
   ```

   This enables the replica to continue to receive data change events from the source's binary log and store them in the relay logs using the I/O thread, but prevents the replica from executing these events and changing its data. Within busy replication environments, permitting the I/O thread to run during backup may speed up the catch-up process when you restart the replication SQL thread.

2. Run `mysqldump` to dump your databases. You may either dump all databases or select databases to be dumped. For example, to dump all databases:

   ```
   shell> mysqldump --all-databases > fulldb.dump
   ```

3. Once the dump has completed, start replication operations again:

```
shell> mysqladmin start-slave
```

In the preceding example, you may want to add login credentials (user name, password) to the commands, and bundle the process up into a script that you can run automatically each day.

If you use this approach, make sure you monitor the replication process to ensure that the time taken to run the backup does not affect the replica's ability to keep up with events from the source. See Section 17.1.5.1, “Checking Replication Status”. If the replica is unable to keep up, you may want to add another replica and distribute the backup process. For an example of how to configure this scenario, see Section 17.3.5, “Replicating Different Databases to Different Replicas”.

17.3.1.2 Backing Up Raw Data from a Replica

To guarantee the integrity of the files that are copied, backing up the raw data files on your replica should take place while the replica server is shut down. If the MySQL server is still running, background tasks may still be updating the database files, particularly those involving storage engines with background processes such as `InnoDB`. With `InnoDB`, these problems should be resolved during crash recovery, but since the replica server can be shut down during the backup process without affecting the execution of the source it makes sense to take advantage of this capability.

To shut down the server and back up the files:

1. Shut down the replica MySQL server:

```
shell> mysqladmin shutdown
```

2. Copy the data files. You can use any suitable copying or archive utility, including `cp`, `tar` or `WinZip`. For example, assuming that the data directory is located under the current directory, you can archive the entire directory as follows:

```
shell> tar cf /tmp/dbbackup.tar ./data
```

3. Start the MySQL server again. Under Unix:

```
shell> mysqld_safe &
```

Under Windows:

```
C:\> "C:\Program Files\MySQL\MySQL Server 5.6\bin\mysqld"
```

Normally you should back up the entire data directory for the replica MySQL server. If you want to be able to restore the data and operate as a replica (for example, in the event of failure of the replica), then in addition to the replica's data, you should also back up the replication metadata repositories, and the relay log files. These files are needed to resume replication after you restore the replica's data.

If you lose the relay logs but still have the `relay-log.info` file, you can check it to determine how far the SQL thread has executed in the source's binary logs. Then you can use `CHANGE MASTER TO` with the `MASTER_LOG_FILE` and `MASTER_LOG_POS` options to tell the replica to re-read the binary logs from that point. This requires that the binary logs still exist on the replication source server.

If your replica is replicating `LOAD DATA` statements, you should also back up any `SQL_LOAD-*` files that exist in the directory that the replica uses for this purpose. The replica needs these files to resume replication of any interrupted `LOAD DATA` operations. The location of this directory is the value of the `slave_load_tmpdir` system variable. If the server was not started with that variable set, the directory location is the value of the `tmpdir` system variable.

17.3.1.3 Backing Up a Source or Replica by Making It Read Only

It is possible to back up either source or replica servers in a replication setup by acquiring a global read lock and manipulating the `read_only` system variable to change the read-only state of the server to be backed up:

1. Make the server read-only, so that it processes only retrievals and blocks updates.
2. Perform the backup.
3. Change the server back to its normal read/write state.

Note

The instructions in this section place the server to be backed up in a state that is safe for backup methods that get the data from the server, such as `mysqldump` (see Section 4.5.4, “`mysqldump` — A Database Backup Program”). You should not attempt to use these instructions to make a binary backup by copying files directly because the server may still have modified data cached in memory and not flushed to disk.

The following instructions describe how to do this for a replication source server and for a replica. For both scenarios discussed here, suppose that you have the following replication setup:

- A replication source server S1
- A replica server R1 that has S1 as its source
- A client C1 connected to S1
- A client C2 connected to R1

In either scenario, the statements to acquire the global read lock and manipulate the `read_only` variable are performed on the server to be backed up and do not propagate to any replicas of that server.

Scenario 1: Backup with a Read-Only Source

Put the source S1 in a read-only state by executing these statements on it:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SET GLOBAL read_only = ON;
```

While S1 is in a read-only state, the following properties are true:

- Requests for updates sent by C1 to S1 block because the server is in read-only mode.
- Requests for query results sent by C1 to S1 succeed.
- Making a backup on S1 is safe.
- Making a backup on R1 is not safe. This server is still running, and might be processing the binary log or update requests coming from client C2.

While S1 is read only, perform the backup. For example, you can use `mysqldump`.

After the backup operation on S1 completes, restore S1 to its normal operational state by executing these statements:

```
mysql> SET GLOBAL read_only = OFF;
mysql> UNLOCK TABLES;
```

Although performing the backup on S1 is safe (as far as the backup is concerned), it is not optimal for performance because clients of S1 are blocked from executing updates.

This strategy applies to backing up a source server in a replication setup, but can also be used for a single server in a nonreplication setting.

Scenario 2: Backup with a Read-Only Replica

Put the replica R1 in a read-only state by executing these statements on it:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SET GLOBAL read_only = ON;
```

While R1 is in a read-only state, the following properties are true:

- The source S1 continues to operate, so making a backup on S1 is not safe.
- The replica R1 is stopped, so making a backup on R1 is safe.

These properties provide the basis for a popular backup scenario: Having one replica busy performing a backup for a while is not a problem because it does not affect the entire network, and the system is still running during the backup. In particular, clients can still perform updates on the replication source server, which remains unaffected by backup activity on the replica.

While R1 is read only, perform the backup. For example, you can use `mysqldump`.

After the backup operation on R1 completes, restore R1 to its normal operational state by executing these statements:

```
mysql> SET GLOBAL read_only = OFF;
mysql> UNLOCK TABLES;
```

After the replica is restored to normal operation, it again synchronizes to the source by catching up with any outstanding updates from the binary log of the source.

17.3.2 Handling an Unexpected Halt of a Replica Server

In order for replication to be resilient to unexpected halts of the server (sometimes described as crash-safe) it must be possible for the replica to recover its state before halting. This section describes the impact of an unexpected halt of a replica during replication and how to configure a replica for the best chance of recovery to continue replication.

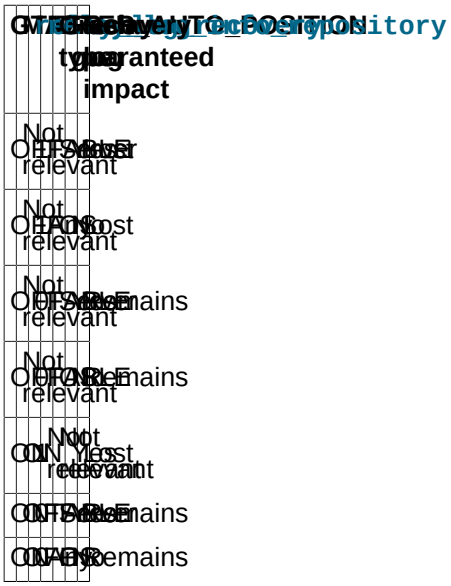
After an unexpected halt of a replica, upon restart the replica's SQL thread must recover which transactions have been executed already. The information required for recovery is stored in the replica's applier metadata repository. In older MySQL Server versions, this log could only be created as a file in the data directory that was updated after the transaction had been applied. This held the risk of losing synchrony with the source depending at which stage of processing a transaction the replica halted, or even corruption of the file itself. In MySQL 5.6 you can instead use an `InnoDB` table to store the applier metadata repository. By using this transactional storage engine the information is always recoverable upon restart. As a table, updates to the applier metadata repository are committed together with the transactions, meaning that the replica's progress information recorded in that repository is always consistent with what has been applied to the database, even in the event of an unexpected server halt.

To configure MySQL 5.6 to store the applier metadata repository as an `InnoDB` table, set the system variable `relay_log_info_repository` to `TABLE`. The server then stores information required for the recovery of the replication SQL thread in the `mysql.slave_relay_log_info` table. For further information on the replication metadata repositories, see Section 17.2.2, “Relay Log and Replication Metadata Repositories”.

Exactly how a replica recovers from an unexpected halt is influenced by the chosen method of replication, whether the replica is single-threaded or multithreaded, the setting of variables such as `relay_log_recovery`, and whether features such as `MASTER_AUTO_POSITION` are being used.

The following table shows the impact of these different factors on how a single-threaded replica recovers from an unexpected halt.

Table 17.1 Factors Influencing Single-threaded Replica Recovery



As the table shows, when using a single-threaded replica the following configurations are most resilient to unexpected halts:

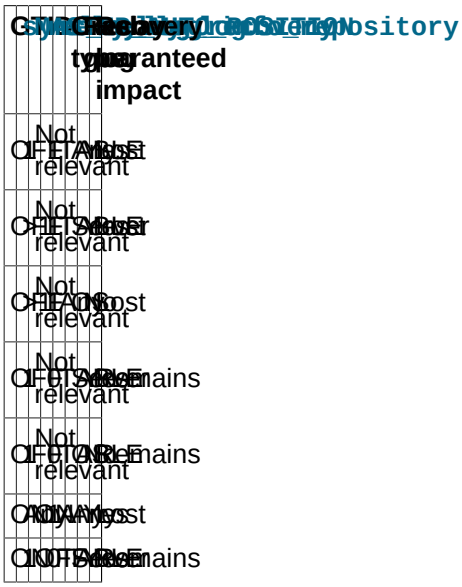
- When using GTIDs and `MASTER_AUTO_POSITION`, set `relay_log_recovery=1`. With this configuration the setting of `relay_log_info_repository` and other variables does not impact on recovery. Note that to guarantee recovery, `sync_binlog=1` must also be set on the replica, so that the replica's binary log is synchronized to disk at each write. Otherwise, committed transactions might not be present in the replica's binary log.
- When using file position based replication, set `relay_log_recovery=1` and `relay_log_info_repository=TABLE`.

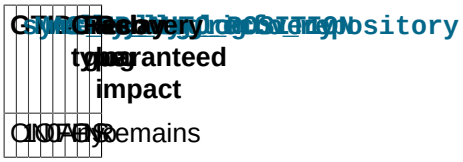
Note

During recovery the relay log is lost.

The following table shows the impact of these different factors on how a multithreaded replica recovers from an unexpected halt.

Table 17.2 Factors Influencing Multithreaded Replica Recovery





As the table shows, when using a multithreaded replica the following configurations are most resilient to unexpected halts:

- When using GTIDs and `MASTER_AUTO_POSITION`, set `relay_log_recovery=1`. With this configuration the setting of `relay_log_info_repository` and other variables does not impact on recovery.
- When using file position based replication, set `relay_log_recovery=1`, `sync_relay_log=1`, and `relay_log_info_repository=TABLE`.

Note

During recovery the relay log is lost.

It is important to note the impact of `sync_relay_log=1`, which requires a write of to the relay log per transaction. Although this setting is the most resilient to an unexpected halt, with at most one unwritten transaction being lost, it also has the potential to greatly increase the load on storage. Without `sync_relay_log=1`, the effect of an unexpected halt depends on how the relay log is handled by the operating system. Also note that when `relay_log_recovery=0`, the next time the replica is started after an unexpected halt the relay log is processed as part of recovery. After this process completes, the relay log is deleted.

An unexpected halt of a multithreaded replica using the recommended file position based replication configuration above may result in a relay log with transaction inconsistencies (gaps in the sequence of transactions) caused by the unexpected halt. See Replication and Transaction Inconsistencies. In MySQL 5.7.13 and later, if the relay log recovery process encounters such transaction inconsistencies they are filled and the recovery process continues automatically. In MySQL versions prior to MySQL 5.7.13, this process is not automatic and requires starting the server with `relay_log_recovery=0`, starting the replica with `START SLAVE UNTIL SQL_AFTER_MTS_GAPS` to fix any transaction inconsistencies and then restarting the replica with `relay_log_recovery=1`.

When you are using multi-source replication and `relay_log_recovery=1`, after restarting due to an unexpected halt all replication channels go through the relay log recovery process. Any inconsistencies found in the relay log due to an unexpected halt of a multithreaded replica are filled.

17.3.3 Using Replication with Different Source and Replica Storage Engines

It does not matter for the replication process whether the source table on the replication source server and the replicated table on the replica use different engine types. In fact, the `default_storage_engine` and `storage_engine` system variables are not replicated.

This provides a number of benefits in the replication process in that you can take advantage of different engine types for different replication scenarios. For example, in a typical scale-out scenario (see Section 17.3.4, “Using Replication for Scale-Out”), you want to use `InnoDB` tables on the source to take advantage of the transactional functionality, but use `MyISAM` on the replicas where transaction support is not required because the data is only read. When using replication in a data-logging environment you may want to use the `Archive` storage engine on the replica.

Configuring different engines on the source and replica depends on how you set up the initial replication process:

- If you used `mysqldump` to create the database snapshot on your source, you could edit the dump file text to change the engine type used on each table.

Another alternative for `mysqldump` is to disable engine types that you do not want to use on the replica before using the dump to build the data on the replica. For example, you can add the `--skip-federated` option on your replica to disable the `FEDERATED` engine. If a specific engine does not exist for a table to be created, MySQL will use the default engine type, usually `MyISAM`. (This requires that the `NO_ENGINE_SUBSTITUTION` SQL mode is not enabled.) If you want to disable additional engines in this way, you may want to consider building a special binary to be used on the replica that only supports the engines you want.

- If you are using raw data files (a binary backup) to set up the replica, you cannot change the initial table format. Instead, use `ALTER TABLE` to change the table types after the replica has been started.
- For new source/replica replication setups where there are currently no tables on the source, avoid specifying the engine type when creating new tables.

If you are already running a replication solution and want to convert your existing tables to another engine type, follow these steps:

1. Stop the replica from running replication updates:

   ```
   mysql> STOP SLAVE;
   ```

   This enables you to change engine types without interruptions.

2. Execute an `ALTER TABLE ... ENGINE='engine_type'` for each table to be changed.

3. Start the replication process again:

   ```
   mysql> START SLAVE;
   ```

Although the `default_storage_engine` variable is not replicated, be aware that `CREATE TABLE` and `ALTER TABLE` statements that include the engine specification are correctly replicated to the replica. For example, if you have a CSV table and you execute:

```
mysql> ALTER TABLE csvtable Engine='MyISAM';
```

The previous statement is replicated to the replica and the engine type on the replica is converted to `MyISAM`, even if you have previously changed the table type on the replica to an engine other than CSV. If you want to retain engine differences on the source and replica, you should be careful to use the `default_storage_engine` variable on the source when creating a new table. For example, instead of:

```
mysql> CREATE TABLE tablea (columna int) Engine=MyISAM;
```

Use this format:

```
mysql> SET default_storage_engine=MyISAM;
mysql> CREATE TABLE tablea (columna int);
```

When replicated, the `default_storage_engine` variable is ignored, and the `CREATE TABLE` statement executes on the replica using the replica's default engine.

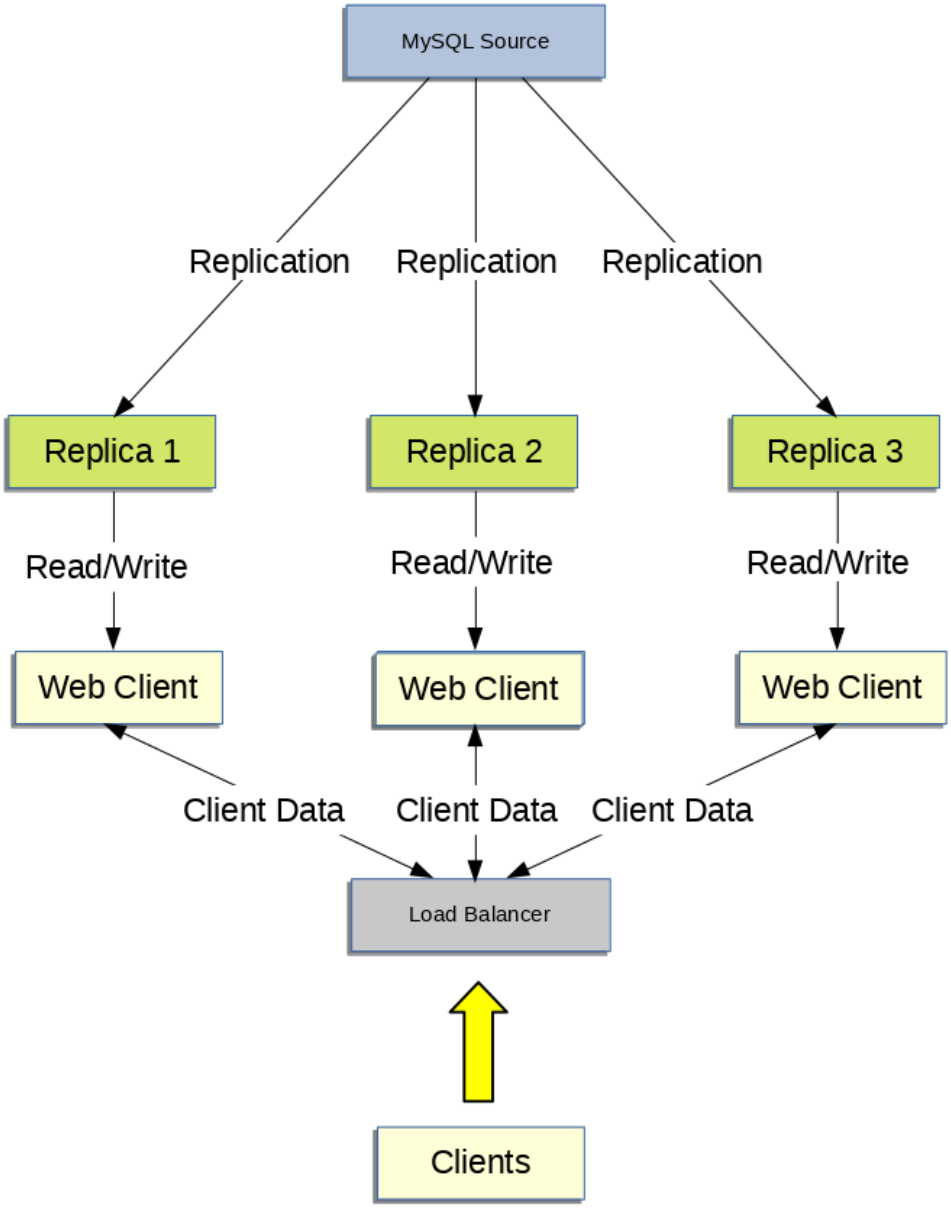
17.3.4 Using Replication for Scale-Out

You can use replication as a scale-out solution; that is, where you want to split up the load of database queries across multiple database servers, within some reasonable limitations.

Because replication works from the distribution of one source to one or more replicas, using replication for scale-out works best in an environment where you have a high number of reads and low number of writes/updates. Most websites fit into this category, where users are browsing the website, reading articles, posts, or viewing products. Updates only occur during session management, or when making a purchase or adding a comment/message to a forum.

Replication in this situation enables you to distribute the reads over the replicas, while still enabling your web servers to communicate with the replication source server when a write is required. You can see a sample replication layout for this scenario in Figure 17.1, “Using Replication to Improve Performance During Scale-Out”.

Figure 17.1 Using Replication to Improve Performance During Scale-Out



If the part of your code that is responsible for database access has been properly abstracted/modularized, converting it to run with a replicated setup should be very smooth and easy. Change the implementation of your database access to send all writes to the source, and to send reads to either the source or a replica. If your code does not have this level of abstraction, setting up a replicated system gives you the opportunity and motivation to clean it up. Start by creating a wrapper library or module that implements the following functions:

- `safe_writer_connect()`
- `safe_reader_connect()`

- `safe_reader_statement()`
- `safe_writer_statement()`

`safe_` in each function name means that the function takes care of handling all error conditions. You can use different names for the functions. The important thing is to have a unified interface for connecting for reads, connecting for writes, doing a read, and doing a write.

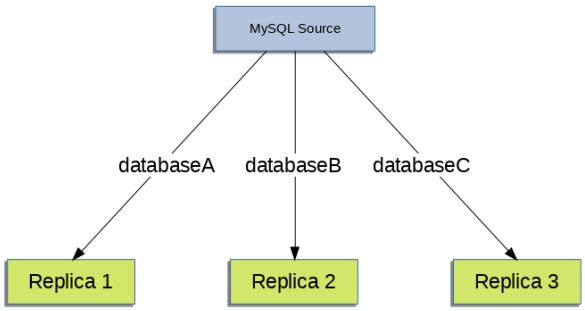
Then convert your client code to use the wrapper library. This may be a painful and scary process at first, but it pays off in the long run. All applications that use the approach just described are able to take advantage of a source/replica configuration, even one involving multiple replicas. The code is much easier to maintain, and adding troubleshooting options is trivial. You need modify only one or two functions (for example, to log how long each statement took, or which statement among those issued gave you an error).

If you have written a lot of code, you may want to automate the conversion task by using the `replace` utility that comes with standard MySQL distributions, or write your own conversion script. Ideally, your code uses consistent programming style conventions. If not, then you are probably better off rewriting it anyway, or at least going through and manually regularizing it to use a consistent style.

17.3.5 Replicating Different Databases to Different Replicas

There may be situations where you have a single source and want to replicate different databases to different replicas. For example, you may want to distribute different sales data to different departments to help spread the load during data analysis. A sample of this layout is shown in Figure 17.2, “Replicating Databases to Separate Replicas”.

Figure 17.2 Replicating Databases to Separate Replicas



You can achieve this separation by configuring the source and replicas as normal, and then limiting the binary log statements that each replica processes by using the `--replicate-wild-do-table` configuration option on each replica.

Important

You should *not* use `--replicate-do-db` for this purpose when using statement-based replication, since statement-based replication causes this option's effects to vary according to the database that is currently selected. This applies to mixed-format replication as well, since this enables some updates to be replicated using the statement-based format.

However, it should be safe to use `--replicate-do-db` for this purpose if you are using row-based replication only, since in this case the currently selected database has no effect on the option's operation.

For example, to support the separation as shown in Figure 17.2, “Replicating Databases to Separate Replicas”, you should configure each replica as follows, before executing `START SLAVE`:

- Replica 1 should use `--replicate-wild-do-table=databaseA.%`.
- Replica 2 should use `--replicate-wild-do-table=databaseB.%`.
- Replica 3 should use `--replicate-wild-do-table=databaseC.%`.

Each replica in this configuration receives the entire binary log from the source, but executes only those events from the binary log that apply to the databases and tables included by the `--replicate-wild-do-table` option in effect on that replica.

If you have data that must be synchronized to the replicas before replication starts, you have a number of choices:

- Synchronize all the data to each replica, and delete the databases, tables, or both that you do not want to keep.
- Use `mysqldump` to create a separate dump file for each database and load the appropriate dump file on each replica.
- Use a raw data file dump and include only the specific files and databases that you need for each replica.

Note

This does not work with `InnoDB` databases unless you use `innodb_file_per_table`.

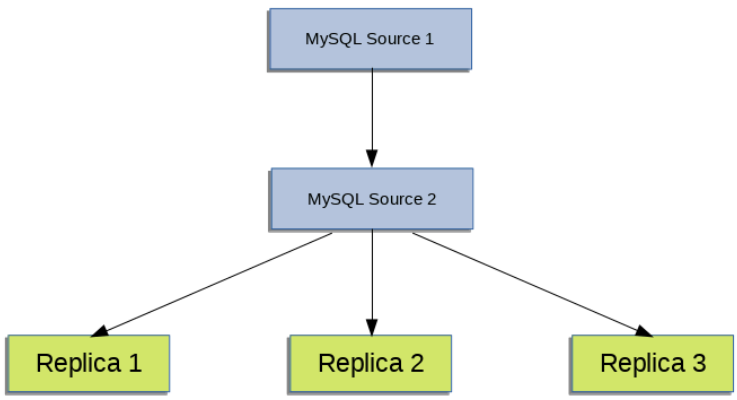
17.3.6 Improving Replication Performance

As the number of replicas connecting to a replication source server increases, the load, although minimal, also increases, as each replica uses a client connection to the source. Also, as each replica must receive a full copy of the source's binary log, the network load on the source may also increase and create a bottleneck.

If you are using a large number of replicas connected to one source, and that source is also busy processing requests (for example, as part of a scale-out solution), then you may want to improve the performance of the replication process.

One way to improve the performance of the replication process is to create a deeper replication structure that enables the source to replicate to only one replica, and for the remaining replicas to connect to this primary replica for their individual replication requirements. A sample of this structure is shown in Figure 17.3, “Using an Additional Replication Source to Improve Performance”.

Figure 17.3 Using an Additional Replication Source to Improve Performance



For this to work, you must configure the MySQL instances as follows:

- Source 1 is the primary source where all changes and updates are written to the database. Binary logging should be enabled on this machine.
- Source 2 is the replica to Source 1 that provides the replication functionality to the remainder of the replicas in the replication structure. Source 2 is the only machine permitted to connect to Source 1. Source 2 also has binary logging enabled, and the `log_slave_updates` system variable enabled so that replication instructions from Source 1 are also written to Source 2's binary log so that they can then be replicated to the true replicas.
- Replica 1, Replica 2, and Replica 3 act as replicas to Source 2, and replicate the information from Source 2, which actually consists of the updates logged on Source 1.

The above solution reduces the client load and the network interface load on the primary source, which should improve the overall performance of the primary source when used as a direct database solution.

If your replicas are having trouble keeping up with the replication process on the source, there are a number of options available:

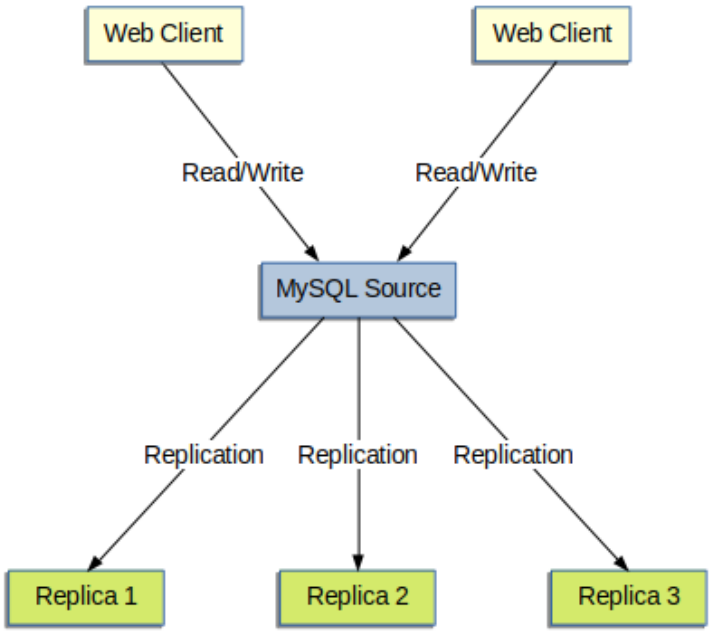
- If possible, put the relay logs and the data files on different physical drives. To do this, set the `relay_log` system variable to specify the location of the relay log.
- If the replicas are significantly slower than the source, you may want to divide up the responsibility for replicating different databases to different replicas. See Section 17.3.5, “Replicating Different Databases to Different Replicas”.
- If your source makes use of transactions and you are not concerned about transaction support on your replicas, use `MyISAM` or another nontransactional engine on the replicas. See Section 17.3.3, “Using Replication with Different Source and Replica Storage Engines”.
- If your replicas are not acting as sources, and you have a potential solution in place to ensure that you can bring up a source in the event of failure, then you can disable the `log_slave_updates` system variable. This prevents “dumb” replicas from also logging events they have executed into their own binary log.

17.3.7 Switching Sources During Failover

You can tell a replica to change to a new source using the `CHANGE MASTER TO` statement. The replica does not check whether the databases on the source are compatible with those on the replica; it simply begins reading and executing events from the specified coordinates in the new source's binary log. In a failover situation, all the servers in the group are typically executing the same events from the same binary log file, so changing the source of the events should not affect the structure or integrity of the database, provided that you exercise care in making the change.

Replicas should be run with the `--log-bin` option, and if not using GTIDs then they should also be run without enabling the `log_slave_updates` system variable. In this way, the replica is ready to become a source without restarting the replica `mysqld`. Assume that you have the structure shown in Figure 17.4, “Redundancy Using Replication, Initial Structure”.

Figure 17.4 Redundancy Using Replication, Initial Structure



In this diagram, the `MySQL Source` holds the source database, the `Replica` hosts are replicas, and the `Web Client` machines are issuing database reads and writes. Web clients that issue only reads (and would normally be connected to the replicas) are not shown, as they do not need to switch to a new server in the event of failure. For a more detailed example of a read/write scale-out replication structure, see Section 17.3.4, “Using Replication for Scale-Out”.

Each replica (`Replica 1`, `Replica 2`, and `Replica 3`) is a replica running with `--log-bin` and without enabling the `log_slave_updates` system variable. Because updates received by a replica from the source are not logged in the binary log unless `log_slave_updates` is enabled, the binary log on each replica is empty initially. If for some reason `MySQL Source` becomes unavailable, you can pick one of the replicas to become the new source. For example, if you pick `Replica 1`, all `Web Clients` should be redirected to `Replica 1`, which writes the updates to its binary log. `Replica 2` and `Replica 3` should then replicate from `Replica 1`.

The reason for running the replica without `log_slave_updates` enabled is to prevent replicas from receiving updates twice in case you cause one of the replicas to become the new source. If `Replica 1` has `log_slave_updates` enabled, it writes any updates that it receives from `Source` in its own binary log. This means that, when `Replica 2` changes from `Source` to `Replica 1` as its source, it may receive updates from `Replica 1` that it has already received from `Source`.

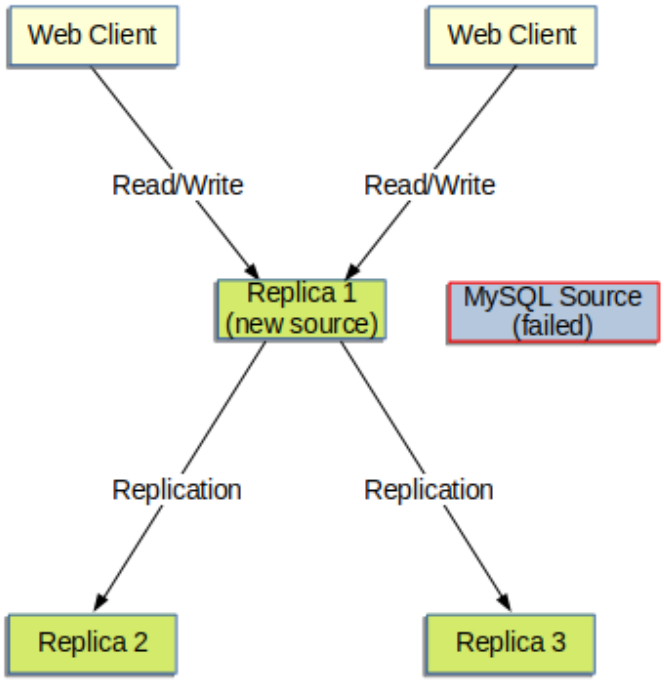
Make sure that all replicas have processed any statements in their relay log. On each replica, issue `STOP SLAVE IO_THREAD`, then check the output of `SHOW PROCESSLIST` until you see `Has read all relay log`. When this is true for all replicas, they can be reconfigured to the new setup. On the replica `Replica 1` being promoted to become the source, issue `STOP SLAVE` and `RESET MASTER`.

On the other replicas `Replica 2` and `Replica 3`, use `STOP SLAVE` and `CHANGE MASTER TO MASTER_HOST='Replica1'` (where `'Replica1'` represents the real host name of `Replica 1`). To use `CHANGE MASTER TO`, add all information about how to connect to `Replica 1` from `Replica 2` or `Replica 3` (*`user`*, *`password`*, *`port`*). When issuing the `CHANGE MASTER TO` statement in this, there is no need to specify the name of the `Replica 1` binary log file or log position to read from, since the first binary log file and position 4, are the defaults. Finally, execute `START SLAVE` on `Replica 2` and `Replica 3`.

Once the new replication setup is in place, you need to tell each `Web Client` to direct its statements to `Replica 1`. From that point on, all updates statements sent by `Web Client` to `Replica 1` are written to the binary log of `Replica 1`, which then contains every update statement sent to `Replica 1` since `Source` stopped working.

The resulting server structure is shown in Figure 17.5, “Redundancy Using Replication, After Source Failure”.

Figure 17.5 Redundancy Using Replication, After Source Failure



When `Source` becomes available again, you should make it a replica of `Replica 1`. To do this, issue on `Source` the same `CHANGE MASTER TO` statement as that issued on `Replica 2` and `Replica 3` previously. `Source` then becomes a replica of `Replica 1` and picks up the `Web Client` writes that it missed while it was offline.

To make `Source` a source again, use the preceding procedure as if `Replica 1` was unavailable and `Source` was to be the new source. During this procedure, do not forget to run `RESET MASTER` on `Source` before making `Replica 1`, `Replica 2`, and `Replica 3` replicas of `Source`. If you fail to do this, the replicas may pick up stale writes from the `Web Client` applications dating from before the point at which `Source` became unavailable.

You should be aware that there is no synchronization between replicas, even when they share the same source, and thus some replicas might be considerably ahead of others. This means that in some cases the procedure outlined in the previous example might not work as expected. In practice, however, relay logs on all replicas should be relatively close together.

One way to keep applications informed about the location of the source is to have a dynamic DNS entry for the source. With `bind` you can use `nsupdate` to update the DNS dynamically.

17.3.8 Setting Up Replication to Use Encrypted Connections

To use an encrypted connection for the transfer of the binary log required during replication, both the source and the replica servers must support encrypted network connections. If either server does not

support encrypted connections (because it has not been compiled or configured for them), replication through an encrypted connection is not possible.

Setting up encrypted connections for replication is similar to doing so for client/server connections. You must obtain (or create) a suitable security certificate that you can use on the source, and a similar certificate (from the same certificate authority) on each replica. You must also obtain suitable key files.

For more information on setting up a server and client for encrypted connections, see Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”.

To enable encrypted connections on the source, you must create or obtain suitable certificate and key files, and then add the following configuration parameters to the source's configuration within the `[mysqld]` section of the source's `my.cnf` file, changing the file names as necessary:

```
[mysqld]
ssl_ca=cacert.pem
ssl_cert=server-cert.pem
ssl_key=server-key.pem
```

The paths to the files may be relative or absolute; we recommend that you always use complete paths for this purpose.

The configuration parameters are as follows:

- `ssl_ca`: The path name of the Certificate Authority (CA) certificate file. (`--ssl-capath` is similar but specifies the path name of a directory of CA certificate files.)
- `ssl_cert`: The path name of the server public key certificate file. This certificate can be sent to the client and authenticated against the CA certificate that it has.
- `ssl_key`: The path name of the server private key file.

On the replica, there are two ways to specify the information required for connecting using encryption to the source. You can either name the replica certificate and key files in the `[client]` section of the replica's `my.cnf` file, or you can explicitly specify that information using the `CHANGE MASTER TO` statement:

- To name the replica certificate and key files using an option file, add the following lines to the `[client]` section of the replica's `my.cnf` file, changing the file names as necessary:

  ```
  [client]
  ssl-ca=cacert.pem
  ssl-cert=client-cert.pem
  ssl-key=client-key.pem
  ```

  Restart the replica server, using the `--skip-slave-start` option to prevent the replica from connecting to the source. Use `CHANGE MASTER TO` to specify the source configuration, using the `MASTER_SSL` option to connect using encryption:

  ```
  mysql> CHANGE MASTER TO
      -> MASTER_HOST='source_hostname',
      -> MASTER_USER='replicate',
      -> MASTER_PASSWORD='password',
      -> MASTER_SSL=1;
  ```

- To specify the certificate and key names using the `CHANGE MASTER TO` statement, append the appropriate `MASTER_SSL_xxx` options:

  ```
  mysql> CHANGE MASTER TO
      -> MASTER_HOST='source_hostname',
      -> MASTER_USER='replicate',
      -> MASTER_PASSWORD='password',
      -> MASTER_SSL=1,
      -> MASTER_SSL_CA = 'ca_file_name',
      -> MASTER_SSL_CAPATH = 'ca_directory_name',
  ```

```
    -> MASTER_SSL_CERT = 'cert_file_name',
    -> MASTER_SSL_KEY = 'key_file_name';
```

After the source information has been updated, start the replication process:

```
mysql> START SLAVE;
```

You can use the `SHOW SLAVE STATUS` statement to confirm that an encrypted connection was established successfully.

For more information on the `CHANGE MASTER TO` statement, see Section 13.4.2.1, “CHANGE MASTER TO Statement”.

If you want to enforce the use of encrypted connections during replication, create a user with the `REPLICATION SLAVE` privilege and use the `REQUIRE SSL` option for that user. For example:

```
mysql> CREATE USER 'repl'@'%.example.com' IDENTIFIED BY 'password';
mysql> GRANT REPLICATION SLAVE ON *.*
    -> TO 'repl'@'%.example.com' REQUIRE SSL;
```

If the account already exists, you can add `REQUIRE SSL` to it with this statement:

```
mysql> GRANT USAGE ON *.*
    -> TO 'repl'@'%.example.com' REQUIRE SSL;
```

17.3.9 Semisynchronous Replication

In addition to the built-in asynchronous replication, MySQL 5.6 supports an interface to semisynchronous replication that is implemented by plugins. This section discusses what semisynchronous replication is and how it works. The following sections cover the administrative interface to semisynchronous replication and how to install, configure, and monitor it.

MySQL replication by default is asynchronous. The source writes events to its binary log and replicas request them when they are ready. The source does not know whether or when a replica has retrieved and processed the transactions, and there is no guarantee that any event ever reaches any replica. With asynchronous replication, if the source crashes, transactions that it has committed might not have been transmitted to any replica. Failover from source to replica in this case might result in failover to a server that is missing transactions relative to the source.

With fully synchronous replication, when a source commits a transaction, all replicas must also have committed the transaction before the source returns to the session that performed the transaction. Fully synchronous replication means failover from the source to any replica is possible at any time. The drawback of fully synchronous replication is that there might be a lot of delay to complete a transaction.

Semisynchronous replication falls between asynchronous and fully synchronous replication. The source waits until at least one replica has received and logged the events, and then commits the transaction. The source does not wait for all replicas to acknowledge receipt, and it requires only an acknowledgement from the replicas, not that the events have been fully executed and committed on the replica side. Semisynchronous replication therefore guarantees that if the source crashes, all the transactions that it has committed have been transmitted to at least one replica.

Important

With semisynchronous replication, if the source crashes and a failover to a replica is carried out, the failed source should not be reused as the replication source server, and should be discarded. It could have transactions that were not acknowledged by any replica, which were therefore not committed before the failover.

The performance impact of semisynchronous replication compared to asynchronous replication is the tradeoff for increased data integrity. The amount of slowdown is at least the TCP/IP roundtrip time to send the commit to the replica and wait for the acknowledgment of receipt by the replica. This means

that semisynchronous replication works best for close servers communicating over fast networks, and worst for distant servers communicating over slow networks. Semisynchronous replication also places a rate limit on busy sessions by constraining the speed at which binary log events can be sent from source to replica. When one user is too busy, this slows it down, which can be useful in some deployment situations.

Semisynchronous replication between a source and its replicas operates as follows:

- A replica indicates whether it is semisynchronous-capable when it connects to the source.
- If semisynchronous replication is enabled on the source side and there is at least one semisynchronous replica, a thread that performs a transaction commit on the source blocks after the commit is done and waits until at least one semisynchronous replica acknowledges that it has received all events for the transaction, or until a timeout occurs.
- The replica acknowledges receipt of a transaction's events only after the events have been written to its relay log and flushed to disk.
- If a timeout occurs without any replica having acknowledged the transaction, the source reverts to asynchronous replication. When at least one semisynchronous replica catches up, the source returns to semisynchronous replication.
- Semisynchronous replication must be enabled on both the source and replica sides. If semisynchronous replication is disabled on the source, or enabled on the source but on no replicas, the source uses asynchronous replication.

While the source is blocking (waiting for acknowledgment from a replica after having performed a commit), it does not return to the session that performed the transaction. When the block ends, the source returns to the session, which then can proceed to execute other statements. At this point, the transaction has committed on the source side, and receipt of its events has been acknowledged by at least one replica.

Blocking also occurs after rollbacks that are written to the binary log, which occurs when a transaction that modifies nontransactional tables is rolled back. The rolled-back transaction is logged even though it has no effect for transactional tables because the modifications to the nontransactional tables cannot be rolled back and must be sent to replicas.

For statements that do not occur in transactional context (that is, when no transaction has been started with `START TRANSACTION` or `SET autocommit = 0`), autocommit is enabled and each statement commits implicitly. With semisynchronous replication, the source blocks after committing each such statement, just as it does for explicit transaction commits.

17.3.9.1 Semisynchronous Replication Administrative Interface

The administrative interface to semisynchronous replication has several components:

- Two plugins implement semisynchronous capability. There is one plugin for the source side and one for the replica side.
- System variables control plugin behavior. Some examples:
  - `rpl_semi_sync_master_enabled`

    Controls whether semisynchronous replication is enabled on the source. To enable or disable the plugin, set this variable to 1 or 0, respectively. The default is 0 (off).
  - `rpl_semi_sync_master_timeout`

    A value in milliseconds that controls how long the source waits on a commit for acknowledgment from a replica before timing out and reverting to asynchronous replication. The default value is 10000 (10 seconds).

- `rpl_semi_sync_slave_enabled`

  Similar to `rpl_semi_sync_master_enabled`, but controls the replica plugin.

All `rpl_semi_sync_xxx` system variables are described at Section 17.1.4.2, “Replication Source Options and Variables”.

- Status variables enable semisynchronous replication monitoring. Some examples:

  - `Rpl_semi_sync_master_clients`

    The number of semisynchronous replicas.

  - `Rpl_semi_sync_master_status`

    Whether semisynchronous replication currently is operational on the source. The value is 1 if the plugin has been enabled and a commit acknowledgment has not occurred. It is 0 if the plugin is not enabled or the source has fallen back to asynchronous replication due to commit acknowledgment timeout.

  - `Rpl_semi_sync_master_no_tx`

    The number of commits that were not acknowledged successfully by a replica.

  - `Rpl_semi_sync_master_yes_tx`

    The number of commits that were acknowledged successfully by a replica.

  - `Rpl_semi_sync_slave_status`

    Whether semisynchronous replication currently is operational on the replica. This is 1 if the plugin has been enabled and the replication I/O thread is running, 0 otherwise.

  All `Rpl_semi_sync_xxx` status variables are described at Section 5.1.9, “Server Status Variables”.

The system and status variables are available only if the appropriate source or replica plugin has been installed with `INSTALL PLUGIN`.

17.3.9.2 Semisynchronous Replication Installation and Configuration

Semisynchronous replication is implemented using plugins, so the plugins must be installed into the server to make them available. After a plugin has been installed, you control it by means of the system variables associated with it. These system variables are unavailable until the associated plugin has been installed.

This section describes how to install the semisynchronous replication plugins. For general information about installing plugins, see Section 5.5.1, “Installing and Uninstalling Plugins”.

To use semisynchronous replication, the following requirements must be satisfied:

- The capability of installing plugins requires a MySQL server that supports dynamic loading. To verify this, check that the value of the `have_dynamic_loading` system variable is `YES`. Binary distributions should support dynamic loading.
- Replication must already be working, see Section 17.1, “Replication Configuration”.

To set up semisynchronous replication, use the following instructions. The `INSTALL PLUGIN`, `SET GLOBAL`, `STOP SLAVE`, and `START SLAVE` statements mentioned here require the `SUPER` privilege.

MySQL distributions include semisynchronous replication plugin files for the source side and the replica side.

To be usable by a source or replica server, the appropriate plugin library file must be located in the MySQL plugin directory (the directory named by the `plugin_dir` system variable). If necessary, configure the plugin directory location by setting the value of `plugin_dir` at server startup.

The plugin library file base names are `semisync_master` and `semisync_slave`. The file name suffix differs per platform (for example, `.so` for Unix and Unix-like systems, `.dll` for Windows).

The source plugin library file must be present in the plugin directory of the source server. The replica plugin library file must be present in the plugin directory of each replica server.

To load the plugins, use the `INSTALL PLUGIN` statement on the source and on each replica that is to be semisynchronous, adjusting the `.so` suffix for your platform as necessary.

On the source:

```
INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';
```

On each replica:

```
INSTALL PLUGIN rpl_semi_sync_slave SONAME 'semisync_slave.so';
```

If an attempt to install a plugin results in an error on Linux similar to that shown here, you must install `libimf`:

```
mysql> INSTALL PLUGIN rpl_semi_sync_master SONAME 'semisync_master.so';
ERROR 1126 (HY000): Can't open shared library
'/usr/local/mysql/lib/plugin/semisync_master.so'
(errno: 22 libimf.so: cannot open shared object file:
No such file or directory)
```

You can obtain `libimf` from https://dev.mysql.com/downloads/os-linux.html.

To see which plugins are installed, use the `SHOW PLUGINS` statement, or query the `INFORMATION_SCHEMA.PLUGINS` table.

To verify plugin installation, examine the `INFORMATION_SCHEMA.PLUGINS` table or use the `SHOW PLUGINS` statement (see Section 5.5.2, “Obtaining Server Plugin Information”). For example:

```
mysql> SELECT PLUGIN_NAME, PLUGIN_STATUS
       FROM INFORMATION_SCHEMA.PLUGINS
       WHERE PLUGIN_NAME LIKE '%semi%';
+----------------------+---------------+
| PLUGIN_NAME          | PLUGIN_STATUS |
+----------------------+---------------+
| rpl_semi_sync_master | ACTIVE        |
+----------------------+---------------+
```

If the plugin fails to initialize, check the server error log for diagnostic messages.

After a semisynchronous replication plugin has been installed, it is disabled by default. The plugins must be enabled both on the source side and the replica side to enable semisynchronous replication. If only one side is enabled, replication is asynchronous.

To control whether an installed plugin is enabled, set the appropriate system variables. You can set these variables at runtime using `SET GLOBAL`, or at server startup on the command line or in an option file.

At runtime, these source-side system variables are available:

```
SET GLOBAL rpl_semi_sync_master_enabled = {0|1};
SET GLOBAL rpl_semi_sync_master_timeout = N;
```

On the replica side, this system variable is available:

```
SET GLOBAL rpl_semi_sync_slave_enabled = {0|1};
```

For `rpl_semi_sync_master_enabled` or `rpl_semi_sync_slave_enabled`, the value should be 1 to enable semisynchronous replication or 0 to disable it. By default, these variables are set to 0.

For `rpl_semi_sync_master_timeout`, the value *`N`* is given in milliseconds. The default value is 10000 (10 seconds).

If you enable semisynchronous replication on a replica at runtime, you must also start the replication I/O thread (stopping it first if it is already running) to cause the replica to connect to the source and register as a semisynchronous replica:

```
STOP SLAVE IO_THREAD;
START SLAVE IO_THREAD;
```

If the I/O thread is already running and you do not restart it, the replica continues to use asynchronous replication.

At server startup, the variables that control semisynchronous replication can be set as command-line options or in an option file. A setting listed in an option file takes effect each time the server starts. For example, you can set the variables in `my.cnf` files on the source and replica sides as follows.

On the source:

```
[mysqld]
rpl_semi_sync_master_enabled=1
rpl_semi_sync_master_timeout=1000 # 1 second
```

On each replica:

```
[mysqld]
rpl_semi_sync_slave_enabled=1
```

17.3.9.3 Semisynchronous Replication Monitoring

The plugins for the semisynchronous replication capability expose several system and status variables that you can examine to determine its configuration and operational state.

The system variable reflect how semisynchronous replication is configured. To check their values, use `SHOW VARIABLES`:

```
mysql> SHOW VARIABLES LIKE 'rpl_semi_sync%';
```

The status variables enable you to monitor the operation of semisynchronous replication. To check their values, use `SHOW STATUS`:

```
mysql> SHOW STATUS LIKE 'Rpl_semi_sync%';
```

When the source switches between asynchronous or semisynchronous replication due to commit-blocking timeout or a replica catching up, it sets the value of the `Rpl_semi_sync_master_status` status variable appropriately. Automatic fallback from semisynchronous to asynchronous replication on the source means that it is possible for the `rpl_semi_sync_master_enabled` system variable to have a value of 1 on the source side even when semisynchronous replication is in fact not operational at the moment. You can monitor the `Rpl_semi_sync_master_status` status variable to determine whether the source currently is using asynchronous or semisynchronous replication.

To see how many semisynchronous replicas are connected, check `Rpl_semi_sync_master_clients`.

The number of commits that have been acknowledged successfully or unsuccessfully by replicas are indicated by the `Rpl_semi_sync_master_yes_tx` and `Rpl_semi_sync_master_no_tx` variables.

On the replica side, `Rpl_semi_sync_slave_status` indicates whether semisynchronous replication currently is operational.

17.3.10 Delayed Replication

MySQL 5.6 supports delayed replication such that a replica server deliberately lags behind the source by at least a specified amount of time. The default delay is 0 seconds. Use the `MASTER_DELAY` option for `CHANGE MASTER TO` to set the delay to *`N`* seconds:

```
CHANGE MASTER TO MASTER_DELAY = N;
```

An event received from the source is not executed until at least *`N`* seconds later than its execution on the source. The exceptions are that there is no delay for format description events or log file rotation events, which affect only the internal state of the SQL thread.

Delayed replication can be used for several purposes:

- To protect against user mistakes on the source. A DBA can roll back a delayed replica to the time just before the disaster.
- To test how the system behaves when there is a lag. For example, in an application, a lag might be caused by a heavy load on the replica. However, it can be difficult to generate this load level. Delayed replication can simulate the lag without having to simulate the load. It can also be used to debug conditions related to a lagging replica.
- To inspect what the database looked like long ago, without having to reload a backup. For example, if the delay is one week and the DBA needs to see what the database looked like before the last few days' worth of development, the delayed replica can be inspected.

`START SLAVE` and `STOP SLAVE` take effect immediately and ignore any delay. `RESET SLAVE` resets the delay to 0.

`SHOW SLAVE STATUS` has three fields that provide information about the delay:

- `SQL_Delay`: A nonnegative integer indicating the number of seconds that the replica must lag the source.
- `SQL_Remaining_Delay`: When `Slave_SQL_Running_State` is `Waiting until MASTER_DELAY seconds after master executed event`, this field contains an integer indicating the number of seconds left of the delay. At other times, this field is `NULL`.
- `Slave_SQL_Running_State`: A string indicating the state of the SQL thread (analogous to `Slave_IO_State`). The value is identical to the `State` value of the SQL thread as displayed by `SHOW PROCESSLIST`.

When the replication SQL thread is waiting for the delay to elapse before executing an event, `SHOW PROCESSLIST` displays its `State` value as `Waiting until MASTER_DELAY seconds after master executed event`.

The `relay-log.info` file now contains the delay value, so the file format has changed. See Section 17.2.2.2, “Replication Metadata Repositories”. In particular, the first line of the file now indicates how many lines are in the file. If you downgrade a replica server to a version older than MySQL 5.6, the older server does not read the file correctly. To address this, modify the file in a text editor to delete the initial line containing the number of lines.

17.4 Replication Notes and Tips

17.4.1 Replication Features and Issues

The following sections provide information about what is supported and what is not in MySQL replication, and about specific issues and situations that may occur when replicating certain statements.

Statement-based replication depends on compatibility at the SQL level between the source and replica. In other words, successful SBR requires that any SQL features used be supported by both the source and the replica servers. For example, if you use a feature on the replication source server that exists in

MySQL 5.6 but was removed in MySQL 5.7, errors occur if you replicate to a replica that uses MySQL 5.7. Such incompatibilities also can occur within a release series when using pre-production releases of MySQL.

For this reason, use Generally Available (GA) releases of MySQL for statement-based replication in a production setting, since we do not introduce new SQL statements or change their behavior within a given release series once that series reaches GA release status.

If you are planning to use statement-based replication between MySQL 5.6 and a previous MySQL release series, it is also a good idea to consult the edition of the *MySQL Reference Manual* corresponding to the earlier release series for information regarding the replication characteristics of that series.

With MySQL's statement-based replication, there may be issues with replicating stored routines or triggers. You can avoid these issues by using MySQL's row-based replication instead. For a detailed list of issues, see Section 20.7, “Stored Program Binary Logging”. For more information about row-based logging and row-based replication, see Section 5.4.4.1, “Binary Logging Formats”, and Section 17.1.2, “Replication Formats”.

For additional information specific to replication and `InnoDB`, see Section 14.19, “InnoDB and MySQL Replication”. For information relating to replication with NDB Cluster, see Section 18.6, “NDB Cluster Replication”.

17.4.1.1 Replication and AUTO_INCREMENT

Statement-based replication of `AUTO_INCREMENT`, `LAST_INSERT_ID()`, and `TIMESTAMP` values is done correctly, subject to the following exceptions:

- When using statement-based replication prior to MySQL 5.6.10, `AUTO_INCREMENT` columns in tables on the replica must match the same columns on the source; that is, `AUTO_INCREMENT` columns must be replicated to `AUTO_INCREMENT` columns. (Bug #12669186)

- A statement invoking a trigger or function that causes an update to an `AUTO_INCREMENT` column is not replicated correctly using statement-based replication. In MySQL 5.6, such statements are marked as unsafe. (Bug #45677)

- An `INSERT` into a table that has a composite primary key that includes an `AUTO_INCREMENT` column that is not the first column of this composite key is not safe for statement-based logging or replication. Beginning with MySQL 5.6.6, such statements are marked as unsafe. (Bug #11754117, Bug #45670)

  This issue does not affect tables using the `InnoDB` storage engine, since an `InnoDB` table with an AUTO_INCREMENT column requires at least one key where the auto-increment column is the only or leftmost column.

- Adding an `AUTO_INCREMENT` column to a table with `ALTER TABLE` might not produce the same ordering of the rows on the replica and the source. This occurs because the order in which the rows are numbered depends on the specific storage engine used for the table and the order in which the rows were inserted. If it is important to have the same order on the source and replica, the rows must be ordered before assigning an `AUTO_INCREMENT` number. Assuming that you want to add an `AUTO_INCREMENT` column to a table `t1` that has columns `col1` and `col2`, the following statements produce a new table `t2` identical to `t1` but with an `AUTO_INCREMENT` column:

```
CREATE TABLE t2 LIKE t1;
ALTER TABLE t2 ADD id INT AUTO_INCREMENT PRIMARY KEY;
INSERT INTO t2 SELECT * FROM t1 ORDER BY col1, col2;
```

Important

To guarantee the same ordering on both source and replica, the `ORDER BY` clause must name *all* columns of `t1`.

The instructions just given are subject to the limitations of `CREATE TABLE ... LIKE`: Foreign key definitions are ignored, as are the `DATA DIRECTORY` and `INDEX DIRECTORY` table options. If a table definition includes any of those characteristics, create `t2` using a `CREATE TABLE` statement that is identical to the one used to create `t1`, but with the addition of the `AUTO_INCREMENT` column.

Regardless of the method used to create and populate the copy having the `AUTO_INCREMENT` column, the final step is to drop the original table and then rename the copy:

```
DROP t1;
ALTER TABLE t2 RENAME t1;
```

See also Section B.3.6.1, “Problems with ALTER TABLE”.

17.4.1.2 Replication and BLACKHOLE Tables

The `BLACKHOLE` storage engine accepts data but discards it and does not store it. When performing binary logging, all inserts to such tables are always logged, regardless of the logging format in use. Updates and deletes are handled differently depending on whether statement based or row based logging is in use. With the statement based logging format, all statements affecting `BLACKHOLE` tables are logged, but their effects ignored. When using row-based logging, updates and deletes to such tables are simply skipped—they are not written to the binary log. In MySQL 5.6.12 and later, a warning is logged whenever this occurs (Bug #13004581)

For this reason we recommend when you replicate to tables using the `BLACKHOLE` storage engine that you have the `binlog_format` server variable set to `STATEMENT`, and not to either `ROW` or `MIXED`.

17.4.1.3 Replication and Character Sets

The following applies to replication between MySQL servers that use different character sets:

- If the source has databases with a character set different from the global `character_set_server` value, you should design your `CREATE TABLE` statements so that they do not implicitly rely on the database default character set. A good workaround is to state the character set and collation explicitly in `CREATE TABLE` statements.

17.4.1.4 Replication and CHECKSUM TABLE

`CHECKSUM TABLE` returns a checksum that is calculated row by row, using a method that depends on the table row storage format. The storage format is not guaranteed to remain the same between MySQL versions, so the checksum value might change following an upgrade.

17.4.1.5 Replication of CREATE SERVER, ALTER SERVER, and DROP SERVER

In MySQL 5.6, the statements `CREATE SERVER`, `ALTER SERVER`, and `DROP SERVER` are not written to the binary log, regardless of the binary logging format that is in use.

17.4.1.6 Replication of CREATE ... IF NOT EXISTS Statements

MySQL applies these rules when various `CREATE ... IF NOT EXISTS` statements are replicated:

- Every `CREATE DATABASE IF NOT EXISTS` statement is replicated, whether or not the database already exists on the source.
- Similarly, every `CREATE TABLE IF NOT EXISTS` statement without a `SELECT` is replicated, whether or not the table already exists on the source. This includes `CREATE TABLE IF NOT EXISTS ... LIKE`. Replication of `CREATE TABLE IF NOT EXISTS ... SELECT` follows somewhat different rules; see Section 17.4.1.7, “Replication of CREATE TABLE ... SELECT Statements”, for more information.
- `CREATE EVENT IF NOT EXISTS` is always replicated in MySQL 5.6, whether or not the event named in the statement already exists on the source.

See also Bug #45574.

17.4.1.7 Replication of CREATE TABLE ... SELECT Statements

MySQL applies these rules when `CREATE TABLE ... SELECT` statements are replicated:

- `CREATE TABLE ... SELECT` always performs an implicit commit (Section 13.3.3, “Statements That Cause an Implicit Commit”).
- If the destination table does not exist, logging occurs as follows. It does not matter whether `IF NOT EXISTS` is present.
  - `STATEMENT` or `MIXED` format: The statement is logged as written.
  - `ROW` format: The statement is logged as a `CREATE TABLE` statement followed by a series of insert-row events.
- If the statement fails, nothing is logged. This includes the case that the destination table exists and `IF NOT EXISTS` is not given.
- If the destination table exists and `IF NOT EXISTS` is given, MySQL ignores the statement completely; nothing is inserted or logged.

MySQL 5.6 does not allow a `CREATE TABLE ... SELECT` statement to make any changes in tables other than the table that is created by the statement. Some older versions of MySQL permitted these statements to do so; this means that, when using replication between a MySQL 5.6 or later replica and a source running a previous version of MySQL, a `CREATE TABLE ... SELECT` statement causing changes in other tables on the source fails on the replica, causing replication to stop. To prevent this from happening, you should use row-based replication, rewrite the offending statement before running it on the source, or upgrade the source to MySQL 5.6. (If you choose to upgrade the source, keep in mind that such a `CREATE TABLE ... SELECT` statement fails following the upgrade unless it is rewritten to remove any side effects on other tables.)

17.4.1.8 Replication of CURRENT_USER()

The following statements support use of the `CURRENT_USER()` function to take the place of the name of, and possibly the host for, an affected user or a definer:

- `DROP USER`
- `RENAME USER`
- `GRANT`
- `REVOKE`
- `CREATE FUNCTION`
- `CREATE PROCEDURE`
- `CREATE TRIGGER`
- `CREATE EVENT`
- `CREATE VIEW`
- `ALTER EVENT`
- `ALTER VIEW`
- `SET PASSWORD`

When binary logging is enabled and `CURRENT_USER()` or `CURRENT_USER` is used as the definer in any of these statements, MySQL Server ensures that the statement is applied to the same user on both

the source and the replica when the statement is replicated. In some cases, such as statements that change passwords, the function reference is expanded before it is written to the binary log, so that the statement includes the user name. For all other cases, the name of the current user on the source is replicated to the replica as metadata, and the replica applies the statement to the current user named in the metadata, rather than to the current user on the replica.

17.4.1.9 Replication with Differing Table Definitions on Source and Replica

Source and target tables for replication do not have to be identical. A table on the source can have more or fewer columns than the replica's copy of the table. In addition, corresponding table columns on the source and the replica can use different data types, subject to certain conditions.

Note

Replication between tables which are partitioned differently from one another is not supported. See Section 17.4.1.23, “Replication and Partitioning”.

In all cases where the source and target tables do not have identical definitions, the database and table names must be the same on both the source and the replica. Additional conditions are discussed, with examples, in the following two sections.

Replication with More Columns on Source or Replica

You can replicate a table from the source to the replica such that the source and replica copies of the table have differing numbers of columns, subject to the following conditions:

- Columns common to both versions of the table must be defined in the same order on the source and the replica.

  (This is true even if both tables have the same number of columns.)

- Columns common to both versions of the table must be defined before any additional columns.

  This means that executing an `ALTER TABLE` statement on the replica where a new column is inserted into the table within the range of columns common to both tables causes replication to fail, as shown in the following example:

  Suppose that a table `t`, existing on the source and the replica, is defined by the following `CREATE TABLE` statement:

  ```
  CREATE TABLE t (
      c1 INT,
      c2 INT,
      c3 INT
  );
  ```

  Suppose that the `ALTER TABLE` statement shown here is executed on the replica:

  ```
  ALTER TABLE t ADD COLUMN cnew1 INT AFTER c3;
  ```

  The previous `ALTER TABLE` is permitted on the replica because the columns `c1`, `c2`, and `c3` that are common to both versions of table `t` remain grouped together in both versions of the table, before any columns that differ.

  However, the following `ALTER TABLE` statement cannot be executed on the replica without causing replication to break:

  ```
  ALTER TABLE t ADD COLUMN cnew2 INT AFTER c2;
  ```

  Replication fails after execution on the replica of the `ALTER TABLE` statement just shown, because the new column `cnew2` comes between columns common to both versions of `t`.

- Each “extra” column in the version of the table having more columns must have a default value.

A column's default value is determined by a number of factors, including its type, whether it is defined with a `DEFAULT` option, whether it is declared as `NULL`, and the server SQL mode in effect at the time of its creation; for more information, see Section 11.5, “Data Type Default Values”).

In addition, when the replica's copy of the table has more columns than the source's copy, each column common to the tables must use the same data type in both tables.

Examples. The following examples illustrate some valid and invalid table definitions:

More columns on the source. The following table definitions are valid and replicate correctly:

```
source> CREATE TABLE t1 (c1 INT, c2 INT, c3 INT);
replica>  CREATE TABLE t1 (c1 INT, c2 INT);
```

The following table definitions would raise an error because the definitions of the columns common to both versions of the table are in a different order on the replica than they are on the source:

```
source> CREATE TABLE t1 (c1 INT, c2 INT, c3 INT);
replica>  CREATE TABLE t1 (c2 INT, c1 INT);
```

The following table definitions would also raise an error because the definition of the extra column on the source appears before the definitions of the columns common to both versions of the table:

```
source> CREATE TABLE t1 (c3 INT, c1 INT, c2 INT);
replica>  CREATE TABLE t1 (c1 INT, c2 INT);
```

More columns on the replica. The following table definitions are valid and replicate correctly:

```
source> CREATE TABLE t1 (c1 INT, c2 INT);
replica>  CREATE TABLE t1 (c1 INT, c2 INT, c3 INT);
```

The following definitions raise an error because the columns common to both versions of the table are not defined in the same order on both the source and the replica:

```
source> CREATE TABLE t1 (c1 INT, c2 INT);
replica>  CREATE TABLE t1 (c2 INT, c1 INT, c3 INT);
```

The following table definitions also raise an error because the definition for the extra column in the replica's version of the table appears before the definitions for the columns which are common to both versions of the table:

```
source> CREATE TABLE t1 (c1 INT, c2 INT);
replica>  CREATE TABLE t1 (c3 INT, c1 INT, c2 INT);
```

The following table definitions fail because the replica's version of the table has additional columns compared to the source's version, and the two versions of the table use different data types for the common column `c2`:

```
source> CREATE TABLE t1 (c1 INT, c2 BIGINT);
replica>  CREATE TABLE t1 (c1 INT, c2 INT, c3 INT);
```

Replication of Columns Having Different Data Types

Corresponding columns on the source's and the replica's copies of the same table ideally should have the same data type. However, beginning with MySQL 5.1.21, this is not always strictly enforced, as long as certain conditions are met.

All other things being equal, it is always possible to replicate from a column of a given data type to another column of the same type and same size or width, where applicable, or larger. For example, you can replicate from a `CHAR(10)` column to another `CHAR(10)`, or from a `CHAR(10)` column to a `CHAR(25)` column without any problems. In certain cases, it also possible to replicate from a column having one data type (on the source) to a column having a different data type (on the replica); when the data type of the source's version of the column is promoted to a type that is the same size or larger on the replica, this is known as *attribute promotion*.

Attribute promotion can be used with both statement-based and row-based replication, and is not dependent on the storage engine used by either the source or the replica. However, the choice of logging format does have an effect on the type conversions that are permitted; the particulars are discussed later in this section.

Important

Whether you use statement-based or row-based replication, the replica's copy of the table cannot contain more columns than the source's copy if you wish to employ attribute promotion.

Statement-based replication. When using statement-based replication, a simple rule of thumb to follow is, “If the statement run on the source would also execute successfully on the replica, it should also replicate successfully”. In other words, if the statement uses a value that is compatible with the type of a given column on the replica, the statement can be replicated. For example, you can insert any value that fits in a `TINYINT` column into a `BIGINT` column as well; it follows that, even if you change the type of a `TINYINT` column in the replica's copy of a table to `BIGINT`, any insert into that column on the source that succeeds should also succeed on the replica, since it is impossible to have a legal `TINYINT` value that is large enough to exceed a `BIGINT` column.

Prior to MySQL 5.6.10, when using statement-based replication, `AUTO_INCREMENT` columns were required to be the same on both the source and the replica; otherwise, updates could be applied to the wrong table on the replica. (Bug #12669186)

Row-based replication: attribute promotion and demotion. Row-based replication in MySQL 5.6 supports attribute promotion and demotion between smaller data types and larger types. It is also possible to specify whether or not to permit lossy (truncated) or non-lossy conversions of demoted column values, as explained later in this section.

Lossy and non-lossy conversions. In the event that the target type cannot represent the value being inserted, a decision must be made on how to handle the conversion. If we permit the conversion but truncate (or otherwise modify) the source value to achieve a “fit” in the target column, we make what is known as a *lossy conversion*. A conversion which does not require truncation or similar modifications to fit the source column value in the target column is a *non-lossy* conversion.

Type conversion modes (slave_type_conversions variable). The setting of the `slave_type_conversions` global server variable controls the type conversion mode used on the replica. This variable takes a set of values from the following table, which shows the effects of each mode on the replica's type-conversion behavior:

Mode	Effect
`ALL_LOSSY`	In this mode, type conversions that would mean loss of information are permitted. This does not imply that non-lossy conversions are permitted, merely that only cases requiring either lossy conversions or no conversion at all are permitted; for example, enabling *only* this mode permits an `INT` column to be converted to `TINYINT` (a lossy conversion), but not a `TINYINT` column to an `INT` column (non-lossy). Attempting the latter conversion in this case would cause replication to stop with an error on the replica.
`ALL_NON_LOSSY`	This mode permits conversions that do not require truncation or other special handling of the source value; that is, it permits conversions where the target type has a wider range than the source type. Setting this mode has no bearing on whether lossy conversions are permitted; this is controlled with the `ALL_LOSSY` mode. If only

Mode	Effect
	ALL_NON_LOSSY is set, but not ALL_LOSSY, then attempting a conversion that would result in the loss of data (such as INT to TINYINT, or CHAR(25) to VARCHAR(20)) causes the replica to stop with an error.
ALL_LOSSY,ALL_NON_LOSSY	When this mode is set, all supported type conversions are permitted, whether or not they are lossy conversions.
ALL_SIGNED	Treat promoted integer types as signed values (the default behavior).
ALL_UNSIGNED	Treat promoted integer types as unsigned values.
ALL_SIGNED,ALL_UNSIGNED	Treat promoted integer types as signed if possible, otherwise as unsigned.
[*empty*]	When slave_type_conversions is not set, no attribute promotion or demotion is permitted; this means that all columns in the source and target tables must be of the same types. This mode is the default.

When an integer type is promoted, its signedness is not preserved. By default, the replica treats all such values as signed. Beginning with MySQL 5.6.13, you can control this behavior using ALL_SIGNED, ALL_UNSIGNED, or both. (Bug#15831300) ALL_SIGNED tells the replica to treat all promoted integer types as signed; ALL_UNSIGNED instructs it to treat these as unsigned. Specifying both causes the replica to treat the value as signed if possible, otherwise to treat it as unsigned; the order in which they are listed is not significant. Neither ALL_SIGNED nor ALL_UNSIGNED has any effect if at least one of ALL_LOSSY or ALL_NONLOSSY is not also used.

Changing the type conversion mode requires restarting the replica with the new slave_type_conversions setting.

Supported conversions. Supported conversions between different but similar data types are shown in the following list:

- Between any of the integer types TINYINT, SMALLINT, MEDIUMINT, INT, and BIGINT.

  This includes conversions between the signed and unsigned versions of these types.

  Lossy conversions are made by truncating the source value to the maximum (or minimum) permitted by the target column. For ensuring non-lossy conversions when going from unsigned to signed types, the target column must be large enough to accommodate the range of values in the source column. For example, you can demote TINYINT UNSIGNED non-lossily to SMALLINT, but not to TINYINT.

- Between any of the decimal types DECIMAL, FLOAT, DOUBLE, and NUMERIC.

  FLOAT to DOUBLE is a non-lossy conversion; DOUBLE to FLOAT can only be handled lossily. A conversion from DECIMAL(*M*,*D*) to DECIMAL(*M'*,*D'*) where *D'* >= *D* and (*M'*-*D'*) >= (*M*-*D*) is non-lossy; for any case where *M'* < *M*, *D'* < *D*, or both, only a lossy conversion can be made.

  For any of the decimal types, if a value to be stored cannot be fit in the target type, the value is rounded down according to the rounding rules defined for the server elsewhere in the documentation. See Section 12.21.4, “Rounding Behavior”, for information about how this is done for decimal types.

- Between any of the string types CHAR, VARCHAR, and TEXT, including conversions between different widths.

  Conversion of a CHAR, VARCHAR, or TEXT to a CHAR, VARCHAR, or TEXT column the same size or larger is never lossy. Lossy conversion is handled by inserting only the first *N* characters of the string on the replica, where *N* is the width of the target column.

Important

Replication between columns using different character sets is not supported.

- Between any of the binary data types `BINARY`, `VARBINARY`, and `BLOB`, including conversions between different widths.

  Conversion of a `BINARY`, `VARBINARY`, or `BLOB` to a `BINARY`, `VARBINARY`, or `BLOB` column the same size or larger is never lossy. Lossy conversion is handled by inserting only the first *`N`* bytes of the string on the replica, where *`N`* is the width of the target column.

- Between any 2 `BIT` columns of any 2 sizes.

  When inserting a value from a `BIT(`*`M`*`)` column into a `BIT(`*`M'`*`)` column, where *`M' > M`*, the most significant bits of the `BIT(`*`M'`*`)` columns are cleared (set to zero) and the *`M`* bits of the `BIT(`*`M`*`)` value are set as the least significant bits of the `BIT(`*`M'`*`)` column.

  When inserting a value from a source `BIT(`*`M`*`)` column into a target `BIT(`*`M'`*`)` column, where *`M' < M`*, the maximum possible value for the `BIT(`*`M'`*`)` column is assigned; in other words, an "all-set" value is assigned to the target column.

Conversions between types not in the previous list are not permitted.

17.4.1.10 Replication and DIRECTORY Table Options

If a `DATA DIRECTORY` or `INDEX DIRECTORY` table option is used in a `CREATE TABLE` statement on the source server, the table option is also used on the replica. This can cause problems if no corresponding directory exists in the replica's host file system or if it exists but is not accessible to the replica server. This can be overridden by using the `NO_DIR_IN_CREATE` server SQL mode on the replica, which causes the replica to ignore the `DATA DIRECTORY` and `INDEX DIRECTORY` table options when replicating `CREATE TABLE` statements. The result is that `MyISAM` data and index files are created in the table's database directory.

For more information, see Section 5.1.10, "Server SQL Modes".

17.4.1.11 Replication of DROP ... IF EXISTS Statements

The `DROP DATABASE IF EXISTS`, `DROP TABLE IF EXISTS`, and `DROP VIEW IF EXISTS` statements are always replicated, even if the database, table, or view to be dropped does not exist on the source. This is to ensure that the object to be dropped no longer exists on either the source or the replica, once the replica has caught up with the source.

`DROP ... IF EXISTS` statements for stored programs (stored procedures and functions, triggers, and events) are also replicated, even if the stored program to be dropped does not exist on the source.

17.4.1.12 Replication and Floating-Point Values

With statement-based replication, values are converted from decimal to binary. Because conversions between decimal and binary representations of them may be approximate, comparisons involving floating-point values are inexact. This is true for operations that use floating-point values explicitly, or that use values that are converted to floating-point implicitly. Comparisons of floating-point values might yield different results on source and replica servers due to differences in computer architecture, the compiler used to build MySQL, and so forth. See Section 12.3, "Type Conversion in Expression Evaluation", and Section B.3.4.8, "Problems with Floating-Point Values".

17.4.1.13 Replication and FLUSH

Some forms of the `FLUSH` statement are not logged because they could cause problems if replicated to a replica: `FLUSH LOGS` and `FLUSH TABLES WITH READ LOCK`. For a syntax example, see

Section 13.7.6.3, “FLUSH Statement”. The `FLUSH TABLES`, `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` statements are written to the binary log and thus replicated to replicas. This is not normally a problem because these statements do not modify table data.

However, this behavior can cause difficulties under certain circumstances. If you replicate the privilege tables in the `mysql` database and update those tables directly without using `GRANT`, you must issue a `FLUSH PRIVILEGES` on the replicas to put the new privileges into effect. In addition, if you use `FLUSH TABLES` when renaming a `MyISAM` table that is part of a `MERGE` table, you must issue `FLUSH TABLES` manually on the replicas. These statements are written to the binary log unless you specify `NO_WRITE_TO_BINLOG` or its alias `LOCAL`.

17.4.1.14 Replication and Fractional Seconds Support

MySQL 5.6.4 and up permits fractional seconds for `TIME`, `DATETIME`, and `TIMESTAMP` values, with up to microseconds (6 digits) precision. See Section 11.2.7, “Fractional Seconds in Time Values”.

There may be problems replicating from a source server that understands fractional seconds to an older replica that does not:

- For `CREATE TABLE` statements containing columns that have an *`fsp`* (fractional seconds precision) value greater than 0, replication fails due to parser errors.
- Statements that use temporal data types with an *`fsp`* value of 0 works with statement-based logging but not row-based logging. In the latter case, the data types have binary formats and type codes on the source that differ from those on the replica.
- Some expression results differ on source and replica. Examples: On the source, the `timestamp` system variable returns a value that includes a microseconds fractional part; on the replica, it returns an integer. On the source, functions that return a result that includes the current time (such as `CURTIME()`, `SYSDATE()`, or `UTC_TIMESTAMP()`) interpret an argument as an *`fsp`* value and the return value includes a fractional seconds part of that many digits. On the replica, these functions permit an argument but ignore it.

17.4.1.15 Replication and System Functions

Certain functions do not replicate well under some conditions:

- The `USER()`, `CURRENT_USER()` (or `CURRENT_USER`), `UUID()`, `VERSION()`, and `LOAD_FILE()` functions are replicated without change and thus do not work reliably on the replica unless row-based replication is enabled. (See Section 17.1.2, “Replication Formats”.)

  `USER()` and `CURRENT_USER()` are automatically replicated using row-based replication when using `MIXED` mode, and generate a warning in `STATEMENT` mode. (See also Section 17.4.1.8, “Replication of CURRENT_USER()”.) This is also true for `VERSION()` and `RAND()`.

- For `NOW()`, the binary log includes the timestamp. This means that the value *as returned by the call to this function on the source* is replicated to the replica. To avoid unexpected results when replicating between MySQL servers in different time zones, set the time zone on both source and replica. For more information, see Section 17.4.1.31, “Replication and Time Zones”.

  To explain the potential problems when replicating between servers which are in different time zones, suppose that the source is located in New York, the replica is located in Stockholm, and both servers are using local time. Suppose further that, on the source, you create a table `mytable`, perform an `INSERT` statement on this table, and then select from the table, as shown here:

```
mysql> CREATE TABLE mytable (mycol TEXT);
Query OK, 0 rows affected (0.06 sec)

mysql> INSERT INTO mytable VALUES ( NOW() );
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SELECT * FROM mytable;
+---------------------+
| mycol               |
+---------------------+
| 2009-09-01 12:00:00 |
+---------------------+
1 row in set (0.00 sec)
```

Local time in Stockholm is 6 hours later than in New York; so, if you issue `SELECT NOW()` on the replica at that exact same instant, the value `2009-09-01 18:00:00` is returned. For this reason, if you select from the replica's copy of `mytable` after the `CREATE TABLE` and `INSERT` statements just shown have been replicated, you might expect `mycol` to contain the value `2009-09-01 18:00:00`. However, this is not the case; when you select from the replica's copy of `mytable`, you obtain exactly the same result as on the source:

```
mysql> SELECT * FROM mytable;
+---------------------+
| mycol               |
+---------------------+
| 2009-09-01 12:00:00 |
+---------------------+
1 row in set (0.00 sec)
```

Unlike `NOW()`, the `SYSDATE()` function is not replication-safe because it is not affected by `SET TIMESTAMP` statements in the binary log and is nondeterministic if statement-based logging is used. This is not a problem if row-based logging is used.

An alternative is to use the `--sysdate-is-now` option to cause `SYSDATE()` to be an alias for `NOW()`. This must be done on the source and the replica to work correctly. In such cases, a warning is still issued by this function, but can safely be ignored as long as `--sysdate-is-now` is used on both the source and the replica.

`SYSDATE()` is automatically replicated using row-based replication when using `MIXED` mode, and generates a warning in `STATEMENT` mode.

See also Section 17.4.1.31, “Replication and Time Zones”.

- *The following restriction applies to statement-based replication only, not to row-based replication.* The `GET_LOCK()`, `RELEASE_LOCK()`, `IS_FREE_LOCK()`, and `IS_USED_LOCK()` functions that handle user-level locks are replicated without the replica knowing the concurrency context on the source. Therefore, these functions should not be used to insert into a source table because the content on the replica would differ. For example, do not issue a statement such as `INSERT INTO mytable VALUES(GET_LOCK(...))`. These functions are automatically replicated using row-based replication when using `MIXED` mode, and generate a warning in `STATEMENT` mode.

As a workaround for the preceding limitations when statement-based replication is in effect, you can use the strategy of saving the problematic function result in a user variable and referring to the variable in a later statement. For example, the following single-row `INSERT` is problematic due to the reference to the `UUID()` function:

```
INSERT INTO t VALUES(UUID());
```

To work around the problem, do this instead:

```
SET @my_uuid = UUID();
INSERT INTO t VALUES(@my_uuid);
```

That sequence of statements replicates because the value of `@my_uuid` is stored in the binary log as a user-variable event prior to the `INSERT` statement and is available for use in the `INSERT`.

The same idea applies to multiple-row inserts, but is more cumbersome to use. For a two-row insert, you can do this:

```
SET @my_uuid1 = UUID(); @my_uuid2 = UUID();
```

```
INSERT INTO t VALUES(@my_uuid1),(@my_uuid2);
```

However, if the number of rows is large or unknown, the workaround is difficult or impracticable. For example, you cannot convert the following statement to one in which a given individual user variable is associated with each row:

```
INSERT INTO t2 SELECT UUID(), * FROM t1;
```

Within a stored function, `RAND()` replicates correctly as long as it is invoked only once during the execution of the function. (You can consider the function execution timestamp and random number seed as implicit inputs that are identical on the source and replica.)

The `FOUND_ROWS()` and `ROW_COUNT()` functions are not replicated reliably using statement-based replication. A workaround is to store the result of the function call in a user variable, and then use that in the `INSERT` statement. For example, if you wish to store the result in a table named `mytable`, you might normally do so like this:

```
SELECT SQL_CALC_FOUND_ROWS FROM mytable LIMIT 1;
INSERT INTO mytable VALUES( FOUND_ROWS() );
```

However, if you are replicating `mytable`, you should use `SELECT ... INTO`, and then store the variable in the table, like this:

```
SELECT SQL_CALC_FOUND_ROWS INTO @found_rows FROM mytable LIMIT 1;
INSERT INTO mytable VALUES(@found_rows);
```

In this way, the user variable is replicated as part of the context, and applied on the replica correctly.

These functions are automatically replicated using row-based replication when using `MIXED` mode, and generate a warning in `STATEMENT` mode. (Bug #12092, Bug #30244)

Prior to MySQL 5.6.15, the value of `LAST_INSERT_ID()` was not replicated correctly if any filtering options such as `--replicate-ignore-db` and `--replicate-do-table` were enabled on the replica. (Bug #17234370, BUG# 69861)

17.4.1.16 Replication of Invoked Features

Replication of invoked features such as user-defined functions (UDFs) and stored programs (stored procedures and functions, triggers, and events) provides the following characteristics:

- The effects of the feature are always replicated.
- The following statements are replicated using statement-based replication:
  - `CREATE EVENT`
  - `ALTER EVENT`
  - `DROP EVENT`
  - `CREATE PROCEDURE`
  - `DROP PROCEDURE`
  - `CREATE FUNCTION`
  - `DROP FUNCTION`
  - `CREATE TRIGGER`
  - `DROP TRIGGER`

  However, the *effects* of features created, modified, or dropped using these statements are replicated using row-based replication.

Note

Attempting to replicate invoked features using statement-based replication produces the warning `Statement is not safe to log in statement format`. For example, trying to replicate a UDF with statement-based replication generates this warning because it currently cannot be determined by the MySQL server whether the UDF is deterministic. If you are absolutely certain that the invoked feature's effects are deterministic, you can safely disregard such warnings.

- In the case of `CREATE EVENT` and `ALTER EVENT`:
  - The status of the event is set to `SLAVESIDE_DISABLED` on the replica regardless of the state specified (this does not apply to `DROP EVENT`).
  - The source on which the event was created is identified on the replica by its server ID. The `ORIGINATOR` column in `INFORMATION_SCHEMA.EVENTS` and the `originator` column in `mysql.event` store this information. See Section 21.8, "The INFORMATION_SCHEMA EVENTS Table", and Section 13.7.5.19, "SHOW EVENTS Statement", for more information.
- The feature implementation resides on the replica in a renewable state so that if the source fails, the replica can be used as the source without loss of event processing.

To determine whether there are any scheduled events on a MySQL server that were created on a different server (that was acting as a replication source server), query the `INFORMATION_SCHEMA.EVENTS` table in a manner similar to what is shown here:

```
SELECT EVENT_SCHEMA, EVENT_NAME
    FROM INFORMATION_SCHEMA.EVENTS
    WHERE STATUS = 'SLAVESIDE_DISABLED';
```

Alternatively, you can use the `SHOW EVENTS` statement, like this:

```
SHOW EVENTS
    WHERE STATUS = 'SLAVESIDE_DISABLED';
```

When promoting a replica that has such events to a replication source server, you must enable each event using `ALTER EVENT event_name ENABLE`, where *`event_name`* is the name of the event.

If more than one source was involved in creating events on this replica, and you wish to identify events that were created only on a given source having the server ID *`source_id`*, modify the previous query on the `EVENTS` table to include the `ORIGINATOR` column, as shown here:

```
SELECT EVENT_SCHEMA, EVENT_NAME, ORIGINATOR
    FROM INFORMATION_SCHEMA.EVENTS
    WHERE STATUS = 'SLAVESIDE_DISABLED'
    AND   ORIGINATOR = 'source_id'
```

You can employ `ORIGINATOR` with the `SHOW EVENTS` statement in a similar fashion:

```
SHOW EVENTS
    WHERE STATUS = 'SLAVESIDE_DISABLED'
    AND   ORIGINATOR = 'source_id'
```

Before enabling events that were replicated from the source, you should disable the MySQL Event Scheduler on the replica (using a statement such as `SET GLOBAL event_scheduler = OFF;`), run any necessary `ALTER EVENT` statements, restart the server, then re-enable the Event Scheduler on the replica afterward (using a statement such as `SET GLOBAL event_scheduler = ON;`)-

If you later demote the new source back to being a replica, you must disable manually all events enabled by the `ALTER EVENT` statements. You can do this by storing in a separate table the event names from the `SELECT` statement shown previously, or using `ALTER EVENT` statements to rename the events with a common prefix such as `replicated_` to identify them.

If you rename the events, then when demoting this server back to being a replica, you can identify the events by querying the `EVENTS` table, as shown here:

```
SELECT CONCAT(EVENT_SCHEMA, '.', EVENT_NAME) AS 'Db.Event'
      FROM INFORMATION_SCHEMA.EVENTS
      WHERE INSTR(EVENT_NAME, 'replicated_') = 1;
```

17.4.1.17 Replication and LIMIT

Statement-based replication of `LIMIT` clauses in `DELETE`, `UPDATE`, and `INSERT ... SELECT` statements is unsafe since the order of the rows affected is not defined. (Such statements can be replicated correctly with statement-based replication only if they also contain an `ORDER BY` clause.) When such a statement is encountered:

- When using `STATEMENT` mode, a warning that the statement is not safe for statement-based replication is now issued.

  When using `STATEMENT` mode, warnings are issued for DML statements containing `LIMIT` even when they also have an `ORDER BY` clause (and so are made deterministic). This is a known issue. (Bug #42851)

- When using `MIXED` mode, the statement is now automatically replicated using row-based mode.

17.4.1.18 Replication and LOAD DATA

`LOAD DATA` is considered unsafe for statement-based logging (see Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”). When `binlog_format=MIXED` is set, the statement is logged in row-based format. When `binlog_format=STATEMENT` is set, note that `LOAD DATA` does not generate a warning, unlike other unsafe statements.

When `mysqlbinlog` reads log events for `LOAD DATA` statements logged in statement-based format, a generated local file is created in a temporary directory. These temporary files are not automatically removed by `mysqlbinlog` or any other MySQL program. If you do use `LOAD DATA` statements with statement-based binary logging, you should delete the temporary files yourself after you no longer need the statement log. For more information, see Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”.

17.4.1.19 Replication and max_allowed_packet

`max_allowed_packet` sets an upper limit on the size of any single message between the MySQL server and clients, including replicas. If you are replicating large column values (such as might be found in `TEXT` or `BLOB` columns) and `max_allowed_packet` is too small on the source, the source fails with an error, and the replica shuts down the I/O thread. If `max_allowed_packet` is too small on the replica, this also causes the replica to stop the I/O thread.

Row-based replication currently sends all columns and column values for updated rows from the source to the replica, including values of columns that were not actually changed by the update. This means that, when you are replicating large column values using row-based replication, you must take care to set `max_allowed_packet` large enough to accommodate the largest row in any table to be replicated, even if you are replicating updates only, or you are inserting only relatively small values.

17.4.1.20 Replication and MEMORY Tables

When a replication source server shuts down and restarts, its `MEMORY` tables become empty. To replicate this effect to replicas, the first time that the source uses a given `MEMORY` table after startup, it logs an event that notifies replicas that the table must be emptied by writing a `DELETE` statement for that table to the binary log. The statement is always logged in statement format, even if the binary logging format is set to `ROW`, and it is written even if `read_only` or `super_read_only` mode is set on the server. Note that the replica still has outdated data in a `MEMORY` table during the interval between the source's restart and its first use of the table. To avoid this interval when a direct query to the

replica could return stale data, you can set the `init_file` system variable to name a file containing statements that populate the `MEMORY` table on the source at startup.

When a replica server shuts down and restarts, its `MEMORY` tables become empty. This causes the replica to be out of synchrony with the source and may lead to other failures or cause the replica to stop:

- Row-format updates and deletes received from the source may fail with `Can't find record in 'memory_table'`.
- Statements such as `INSERT INTO ... SELECT FROM memory_table` may insert a different set of rows on the source and replica.

The replica also writes a `DELETE` statement to its own binary log, which is passed on to any downstream replicas, causing them to empty their own `MEMORY` tables.

The safe way to restart a replica that is replicating `MEMORY` tables is to first drop or delete all rows from the `MEMORY` tables on the source and wait until those changes have replicated to the replica. Then it is safe to restart the replica.

An alternative restart method may apply in some cases. When `binlog_format=ROW`, you can prevent the replica from stopping if you set `slave_exec_mode=IDEMPOTENT` before you start the replica again. This allows the replica to continue to replicate, but its `MEMORY` tables still differ from those on the source. This is acceptable if the application logic is such that the contents of `MEMORY` tables can be safely lost (for example, if the `MEMORY` tables are used for caching). `slave_exec_mode=IDEMPOTENT` applies globally to all tables, so it may hide other replication errors in non-`MEMORY` tables.

(The method just described is not applicable in NDB Cluster, where `slave_exec_mode` is always `IDEMPOTENT`, and cannot be changed.)

The size of `MEMORY` tables is limited by the value of the `max_heap_table_size` system variable, which is not replicated (see Section 17.4.1.35, “Replication and Variables”). A change in `max_heap_table_size` takes effect for `MEMORY` tables that are created or updated using `ALTER TABLE ... ENGINE = MEMORY` or `TRUNCATE TABLE` following the change, or for all `MEMORY` tables following a server restart. If you increase the value of this variable on the source without doing so on the replica, it becomes possible for a table on the source to grow larger than its counterpart on the replica, leading to inserts that succeed on the source but fail on the replica with `Table is full` errors. This is a known issue (Bug #48666). In such cases, you must set the global value of `max_heap_table_size` on the replica as well as on the source, then restart replication. It is also recommended that you restart both the source and replica MySQL servers, to insure that the new value takes complete (global) effect on each of them.

See Section 15.3, “The MEMORY Storage Engine” for more information about `MEMORY` tables.

17.4.1.21 Replication of the mysql System Database

Data modification statements made to tables in the `mysql` database are replicated according to the value of `binlog_format`; if this value is `MIXED`, these statements are replicated using row-based format. However, statements that would normally update this information indirectly—such `GRANT`, `REVOKE`, and statements manipulating triggers, stored routines, and views—are replicated to replicas using statement-based replication.

17.4.1.22 Replication and the Query Optimizer

It is possible for the data on the source and replica to become different if a statement is written in such a way that the data modification is nondeterministic; that is, left up the query optimizer. (In general, this is not a good practice, even outside of replication.) Examples of nondeterministic statements include `DELETE` or `UPDATE` statements that use `LIMIT` with no `ORDER BY` clause; see Section 17.4.1.17, “Replication and LIMIT”, for a detailed discussion of these.

17.4.1.23 Replication and Partitioning

Replication is supported between partitioned tables as long as they use the same partitioning scheme and otherwise have the same structure except where an exception is specifically allowed (see Section 17.4.1.9, “Replication with Differing Table Definitions on Source and Replica”).

Replication between tables having different partitioning is generally not supported. This because statements (such as `ALTER TABLE ... DROP PARTITION`) acting directly on partitions in such cases may produce different results on source and replica. In the case where a table is partitioned on the source but not on the replica, any statements operating on partitions on the source's copy of the replica fail on the replica. When the replica's copy of the table is partitioned but the source's copy is not, statements acting on partitions cannot be run on the source without causing errors there.

Due to these dangers of causing replication to fail entirely (on account of failed statements) and of inconsistencies (when the result of a partition-level SQL statement produces different results on source and replica), we recommend that insure that the partitioning of any tables to be replicated from the source is matched by the replica's versions of these tables.

17.4.1.24 Replication and REPAIR TABLE

When used on a corrupted or otherwise damaged table, it is possible for the `REPAIR TABLE` statement to delete rows that cannot be recovered. However, any such modifications of table data performed by this statement are not replicated, which can cause source and replica to lose synchronization. For this reason, in the event that a table on the source becomes damaged and you use `REPAIR TABLE` to repair it, you should first stop replication (if it is still running) before using `REPAIR TABLE`, then afterward compare the source's and replica's copies of the table and be prepared to correct any discrepancies manually, before restarting replication.

17.4.1.25 Replication and Reserved Words

You can encounter problems when you attempt to replicate from an older source to a newer replica and you make use of identifiers on the source that are reserved words in the newer MySQL version running on the replica. An example of this is using a table column named `range` on a 5.0 source that is replicating to a 5.1 or higher replica because `RANGE` is a reserved word beginning in MySQL 5.1. Replication can fail in such cases with Error 1064 `You have an error in your SQL syntax...`, *even if a database or table named using the reserved word or a table having a column named using the reserved word is excluded from replication*. This is due to the fact that each SQL event must be parsed by the replica prior to execution, so that the replica knows which database object or objects would be affected; only after the event is parsed can the replica apply any filtering rules defined by `--replicate-do-db`, `--replicate-do-table`, `--replicate-ignore-db`, and `--replicate-ignore-table`.

To work around the problem of database, table, or column names on the source which would be regarded as reserved words by the replica, do one of the following:

- Use one or more `ALTER TABLE` statements on the source to change the names of any database objects where these names would be considered reserved words on the replica, and change any SQL statements that use the old names to use the new names instead.
- In any SQL statements using these database object names, write the names as quoted identifiers using backtick characters (`` ` ``).

For listings of reserved words by MySQL version, see Reserved Words, in the *MySQL Server Version Reference*. For identifier quoting rules, see Section 9.2, “Schema Object Names”.

17.4.1.26 Replication and Source or Replica Shutdowns

It is safe to shut down a replication source server and restart it later. When a replica loses its connection to the source, the replica tries to reconnect immediately and retries periodically if that

fails. The default is to retry every 60 seconds. This may be changed with the `CHANGE MASTER TO` statement. A replica also is able to deal with network connectivity outages. However, the replica notices the network outage only after receiving no data from the source for `slave_net_timeout` seconds. If your outages are short, you may want to decrease `slave_net_timeout`. See Section 5.1.7, “Server System Variables”.

An unclean shutdown (for example, a crash) on the source side can result in the source's binary log having a final position less than the most recent position read by the replica, due to the source's binary log file not being flushed. This can cause the replica not to be able to replicate when the source comes back up. Setting `sync_binlog=1` in the source `my.cnf` file helps to minimize this problem because it causes the source to flush its binary log more frequently. For the greatest possible durability and consistency in a replication setup using `InnoDB` with transactions, you should also set `innodb_flush_log_at_trx_commit=1`. With this setting, the contents of the `InnoDB` redo log buffer are written out to the log file at each transaction commit and the log file is flushed to disk. Note that the durability of transactions is still not guaranteed with this setting, because operating systems or disk hardware may tell `mysqld` that the flush-to-disk operation has taken place, even though it has not.

Shutting down a replica cleanly is safe because it keeps track of where it left off. However, be careful that the replica does not have temporary tables open; see Section 17.4.1.29, “Replication and Temporary Tables”. Unclean shutdowns might produce problems, especially if the disk cache was not flushed to disk before the problem occurred:

- For transactions, the replica commits and then updates `relay-log.info`. If an unexpected exit occurs between these two operations, relay log processing proceeds further than the information file indicates and the replica re-executes the events from the last transaction in the relay log after it has been restarted.
- A similar problem can occur if the replica updates `relay-log.info` but the server host crashes before the write has been flushed to disk. To minimize the chance of this occurring, set `sync_relay_log_info=1` in the replica `my.cnf` file. The default value of `sync_relay_log_info` is 0, which does not cause writes to be forced to disk; the server relies on the operating system to flush the file from time to time.

The fault tolerance of your system for these types of problems is greatly increased if you have a good uninterruptible power supply.

17.4.1.27 Replica Errors During Replication

If a statement produces the same error (identical error code) on both the source and the replica, the error is logged, but replication continues.

If a statement produces different errors on the source and the replica, the replication SQL thread terminates, and the replica writes a message to its error log and waits for the database administrator to decide what to do about the error. This includes the case that a statement produces an error on the source or the replica, but not both. To address the issue, connect to the replica manually and determine the cause of the problem. `SHOW SLAVE STATUS` is useful for this. Then fix the problem and run `START SLAVE`. For example, you might need to create a nonexistent table before you can start the replica again.

Note

If a temporary error is recorded in the replica's error log, you do not necessarily have to take any action suggested in the quoted error message. Temporary errors should be handled by the client retrying the transaction. For example, if the replica SQL thread records a temporary error relating to a deadlock, you do not need to restart the transaction manually on the replica, unless the replication SQL thread subsequently terminates with a nontemporary error message.

If this error code validation behavior is not desirable, some or all errors can be masked out (ignored) with the `--slave-skip-errors` option.

For nontransactional storage engines such as `MyISAM`, it is possible to have a statement that only partially updates a table and returns an error code. This can happen, for example, on a multiple-row insert that has one row violating a key constraint, or if a long update statement is killed after updating some of the rows. If that happens on the source, the replica expects execution of the statement to result in the same error code. If it does not, the replication SQL thread stops as described previously.

If you are replicating between tables that use different storage engines on the source and replica, keep in mind that the same statement might produce a different error when run against one version of the table, but not the other, or might cause an error for one version of the table, but not the other. For example, since `MyISAM` ignores foreign key constraints, an `INSERT` or `UPDATE` statement accessing an `InnoDB` table on the source might cause a foreign key violation but the same statement performed on a `MyISAM` version of the same table on the replica would produce no such error, causing replication to stop.

17.4.1.28 Replication and Server SQL Mode

Using different server SQL mode settings on the source and the replica may cause the same `INSERT` statements to be handled differently on the source and the replica, leading the source and replica to diverge. For best results, you should always use the same server SQL mode on the source and on the replica. This advice applies whether you are using statement-based or row-based replication.

If you are replicating partitioned tables, using different SQL modes on the source and the replica is likely to cause issues. At a minimum, this is likely to cause the distribution of data among partitions to be different in the source's and replica's copies of a given table. It may also cause inserts into partitioned tables that succeed on the source to fail on the replica.

For more information, see Section 5.1.10, “Server SQL Modes”.

17.4.1.29 Replication and Temporary Tables

The discussion in the following paragraphs does not apply when `binlog_format=ROW` because, in that case, temporary tables are not replicated; this means that there are never any temporary tables on the replica to be lost in the event of an unplanned shutdown by the replica. The remainder of this section applies only when using statement-based or mixed-format replication. Loss of replicated temporary tables on the replica can be an issue, whenever `binlog_format` is `STATEMENT` or `MIXED`, for statements involving temporary tables that can be logged safely using statement-based format. For more information about row-based replication and temporary tables, see Row-based logging of temporary tables.

Safe replica shutdown when using temporary tables. Temporary tables are replicated except in the case where you stop the replica server (not just the replica threads) and you have replicated temporary tables that are open for use in updates that have not yet been executed on the replica. If you stop the replica server, the temporary tables needed by those updates are no longer available when the replica is restarted. To avoid this problem, do not shut down the replica while it has temporary tables open. Instead, use the following procedure:

1. Issue a `STOP SLAVE SQL_THREAD` statement.
2. Use `SHOW STATUS` to check the value of the `Slave_open_temp_tables` variable.
3. If the value is not 0, restart the replication SQL thread with `START SLAVE SQL_THREAD` and repeat the procedure later.
4. When the value is 0, issue a `mysqladmin shutdown` command to stop the replica.

Temporary tables and replication options. By default, all temporary tables are replicated; this happens whether or not there are any matching `--replicate-do-db`, `--replicate-do-table`, or `--replicate-wild-do-table` options in effect. However, the `--replicate-ignore-table` and `--replicate-wild-ignore-table` options are honored for temporary tables. The exception is that to enable correct removal of temporary tables at the end of a session, a replica always replicates

a `DROP TEMPORARY TABLE IF EXISTS` statement, regardless of any exclusion rules that would normally apply for the specified table.

A recommended practice when using statement-based or mixed-format replication is to designate a prefix for exclusive use in naming temporary tables that you do not want replicated, then employ a `--replicate-wild-ignore-table` option to match that prefix. For example, you might give all such tables names beginning with `norep` (such as `norepmytable`, `norepyourtable`, and so on), then use `--replicate-wild-ignore-table=norep%` to prevent them from being replicated.

17.4.1.30 Replication Retries and Timeouts

The global system variable `slave_transaction_retries` affects replication as follows: If the replication SQL thread fails to execute a transaction because of an `InnoDB` deadlock or because it exceeded the `InnoDB` `innodb_lock_wait_timeout` value, or the `NDB` `TransactionDeadlockDetectionTimeout` or `TransactionInactiveTimeout` value, the replica automatically retries the transaction `slave_transaction_retries` times before stopping with an error. The default value is 10. The total retry count can be seen in the output of `SHOW STATUS`; see Section 5.1.9, “Server Status Variables”.

17.4.1.31 Replication and Time Zones

By default, source and replica servers assume that they are in the same time zone. If you are replicating between servers in different time zones, the time zone must be set on both source and replica. Otherwise, statements depending on the local time on the source are not replicated properly, such as statements that use the `NOW()` or `FROM_UNIXTIME()` functions.

Verify that your combination of settings for the system time zone (`system_time_zone`), server current time zone (the global value of `time_zone`), and per-session time zones (the session value of `time_zone`) on the source and replica is producing the correct results. In particular, if the `time_zone` system variable is set to the value `SYSTEM`, indicating that the server time zone is the same as the system time zone, this can cause the source and replica to apply different time zones. For example, a source could write the following statement in the binary log:

```
SET @@session.time_zone='SYSTEM';
```

If this source and its replica have a different setting for their system time zones, this statement can produce unexpected results on the replica, even if the replica's global `time_zone` value has been set to match the source's. For an explanation of MySQL Server's time zone settings, and how to change them, see Section 5.1.13, “MySQL Server Time Zone Support”.

See also Section 17.4.1.15, “Replication and System Functions”.

17.4.1.32 Replication and Transactions

Mixing transactional and nontransactional statements within the same transaction. In general, you should avoid transactions that update both transactional and nontransactional tables in a replication environment. You should also avoid using any statement that accesses both transactional (or temporary) and nontransactional tables and writes to any of them.

The MySQL server uses these rules for binary logging:

- If the initial statements in a transaction are nontransactional, they are written to the binary log immediately. The remaining statements in the transaction are cached and not written to the binary log until the transaction is committed. (If the transaction is rolled back, the cached statements are written to the binary log only if they make nontransactional changes that cannot be rolled back. Otherwise, they are discarded.)
- For statement-based logging, logging of nontransactional statements is affected by the `binlog_direct_non_transactional_updates` system variable. When this variable is `OFF` (the default), logging is as just described. When this variable is `ON`, logging occurs immediately for nontransactional statements occurring anywhere in the transaction (not just initial nontransactional

statements). Other statements are kept in the transaction cache and logged when the transaction commits. `binlog_direct_non_transactional_updates` has no effect for row-format or mixed-format binary logging.

Transactional, nontransactional, and mixed statements.
To apply those rules, the server considers a statement nontransactional if it changes only nontransactional tables, and transactional if it changes only transactional tables. In MySQL 5.6, a statement that references both nontransactional and transactional tables and updates *any* of the tables involved, is considered a “mixed” statement. (In previous MySQL release series, a statement that changed both nontransactional and transactional tables was considered mixed.) Mixed statements, like transactional statements, are cached and logged when the transaction commits.

A mixed statement that updates a transactional table is considered unsafe if the statement also performs either of the following actions:

- Updates or reads a temporary table
- Reads a nontransactional table and the transaction isolation level is less than REPEATABLE_READ

A mixed statement following the update of a transactional table within a transaction is considered unsafe if it performs either of the following actions:

- Updates any table and reads from any temporary table
- Updates a nontransactional table and `binlog_direct_non_transactional_updates` is OFF

For more information, see Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”.

Note

A mixed statement is unrelated to mixed binary logging format.

In situations where transactions mix updates to transactional and nontransactional tables, the order of statements in the binary log is correct, and all needed statements are written to the binary log even in case of a `ROLLBACK`. However, when a second connection updates the nontransactional table before the first connection transaction is complete, statements can be logged out of order because the second connection update is written immediately after it is performed, regardless of the state of the transaction being performed by the first connection.

Using different storage engines on source and replica. It is possible to replicate transactional tables on the source using nontransactional tables on the replica. For example, you can replicate an `InnoDB` source table as a `MyISAM` replica table. However, if you do this, there are problems if the replica is stopped in the middle of a `BEGIN` ... `COMMIT` block because the replica restarts at the beginning of the `BEGIN` block.

In MySQL 5.6, it is also safe to replicate transactions from `MyISAM` tables on the source to transactional tables, such as tables that use the `InnoDB` storage engine, on the replica. In such cases, an `AUTOCOMMIT=1` statement issued on the source is replicated, thus enforcing `AUTOCOMMIT` mode on the replica.

When the storage engine type of the replica is nontransactional, transactions on the source that mix updates of transactional and nontransactional tables should be avoided because they can cause inconsistency of the data between the source transactional table and the replica nontransactional table. That is, such transactions can lead to behavior specific to the source's storage engine with the possible effect of replication going out of synchrony. MySQL does not issue a warning about this currently, so extra care should be taken when replicating transactional tables from the source to nontransactional tables on the replicas.

Changing the binary logging format within transactions. The `binlog_format` system variable is read-only as long as a transaction is in progress. (Bug #47863)

Every transaction (including `autocommit` transactions) is recorded in the binary log as though it starts with a `BEGIN` statement, and ends with either a `COMMIT` or a `ROLLBACK` statement. In MySQL 5.6, this true is even for statements affecting tables that use a nontransactional storage engine (such as `MyISAM`).

17.4.1.33 Replication and Triggers

With statement-based replication, triggers executed on the source also execute on the replica. With row-based replication, triggers executed on the source do not execute on the replica. Instead, the row changes on the source resulting from trigger execution are replicated and applied on the replica.

This behavior is by design. If under row-based replication the replica applied the triggers as well as the row changes caused by them, the changes would in effect be applied twice on the replica, leading to different data on the source and the replica.

If you want triggers to execute on both the source and the replica, perhaps because you have different triggers on the source and replica, you must use statement-based replication. However, to enable replica-side triggers, it is not necessary to use statement-based replication exclusively. It is sufficient to switch to statement-based replication only for those statements where you want this effect, and to use row-based replication the rest of the time.

A statement invoking a trigger (or function) that causes an update to an `AUTO_INCREMENT` column is not replicated correctly using statement-based replication. MySQL 5.6 marks such statements as unsafe. (Bug #45677)

17.4.1.34 Replication and TRUNCATE TABLE

`TRUNCATE TABLE` is normally regarded as a DML statement, and so would be expected to be logged and replicated using row-based format when the binary logging mode is `ROW` or `MIXED`. However this caused issues when logging or replicating, in `STATEMENT` or `MIXED` mode, tables that used transactional storage engines such as `InnoDB` when the transaction isolation level was `READ COMMITTED` or `READ UNCOMMITTED`, which precludes statement-based logging.

`TRUNCATE TABLE` is treated for purposes of logging and replication as DDL rather than DML so that it can be logged and replicated as a statement. However, the effects of the statement as applicable to `InnoDB` and other transactional tables on replicas still follow the rules described in Section 13.1.33, “TRUNCATE TABLE Statement” governing such tables. (Bug #36763)

17.4.1.35 Replication and Variables

System variables are not replicated correctly when using `STATEMENT` mode, except for the following variables when they are used with session scope:

- `auto_increment_increment`
- `auto_increment_offset`
- `character_set_client`
- `character_set_connection`
- `character_set_database`
- `character_set_server`
- `collation_connection`
- `collation_database`
- `collation_server`
- `foreign_key_checks`

- identity
- last_insert_id
- lc_time_names
- pseudo_thread_id
- sql_auto_is_null
- time_zone
- timestamp
- unique_checks

When MIXED mode is used, the variables in the preceding list, when used with session scope, cause a switch from statement-based to row-based logging. See Section 5.4.4.3, “Mixed Binary Logging Format”.

sql_mode is also replicated except for the NO_DIR_IN_CREATE mode; the replica always preserves its own value for NO_DIR_IN_CREATE, regardless of changes to it on the source. This is true for all replication formats.

However, when mysqlbinlog parses a SET @@sql_mode = *mode* statement, the full *mode* value, including NO_DIR_IN_CREATE, is passed to the receiving server. For this reason, replication of such a statement may not be safe when STATEMENT mode is in use.

The default_storage_engine and storage_engine system variables are not replicated, regardless of the logging mode; this is intended to facilitate replication between different storage engines.

The read_only system variable is not replicated. In addition, the enabling this variable has different effects with regard to temporary tables, table locking, and the SET PASSWORD statement in different MySQL versions.

The max_heap_table_size system variable is not replicated. Increasing the value of this variable on the source without doing so on the replica can lead eventually to Table is full errors on the replica when trying to execute INSERT statements on a MEMORY table on the source that is thus permitted to grow larger than its counterpart on the replica. For more information, see Section 17.4.1.20, “Replication and MEMORY Tables”.

In statement-based replication, session variables are not replicated properly when used in statements that update tables. For example, the following sequence of statements do not insert the same data on the source and the replica:

```
SET max_join_size=1000;
INSERT INTO mytable VALUES(@@max_join_size);
```

This does not apply to the common sequence:

```
SET time_zone=...;
INSERT INTO mytable VALUES(CONVERT_TZ(..., ..., @@time_zone));
```

Replication of session variables is not a problem when row-based replication is being used, in which case, session variables are always replicated safely. See Section 17.1.2, “Replication Formats”.

In MySQL 5.6, the following session variables are written to the binary log and honored by the replica when parsing the binary log, regardless of the logging format:

- sql_mode
- foreign_key_checks

- `unique_checks`
- `character_set_client`
- `collation_connection`
- `collation_database`
- `collation_server`
- `sql_auto_is_null`

Important

Even though session variables relating to character sets and collations are written to the binary log, replication between different character sets is not supported.

To help reduce possible confusion, we recommend that you always use the same setting for the `lower_case_table_names` system variable on both source and replica, especially when you are running MySQL on platforms with case-sensitive file systems.

Note

In previous versions of MySQL, when a case-sensitive file system was in use, setting this variable to 1 on the replica and to a different value on the source could lead to replication failure. This issue is fixed in MySQL 5.6.1. (Bug #37656)

17.4.1.36 Replication and Views

Views are always replicated to replicas. Views are filtered by their own name, not by the tables they refer to. This means that a view can be replicated to the replica even if the view contains a table that would normally be filtered out by `replication-ignore-table` rules. Care should therefore be taken to ensure that views do not replicate table data that would normally be filtered for security reasons.

Replication from a table to a samed-named view is supported using statement-based logging, but not when using row-based logging. In MySQL 5.6.11 and later, trying to do so when row-based logging is in effect causes an error. (Bug #11752707, Bug #43975)

17.4.2 Replication Compatibility Between MySQL Versions

MySQL supports replication from one release series to the next higher release series. For example, you can replicate from a source running MySQL 5.6 to a replica running MySQL 5.7, from a source running MySQL 5.7 to a replica running MySQL 8.0, and so on. However, you may encounter difficulties when replicating from an older source to a newer replica if the source uses statements or relies on behavior no longer supported in the version of MySQL used on the replica. For example, foreign key names longer than 64 characters are no longer supported from MySQL 8.0.

The use of more than two MySQL Server versions is not supported in replication setups involving multiple sources, regardless of the number of source or replica MySQL servers. This restriction applies not only to release series, but to version numbers within the same release series as well. For example, if you are using a chained or circular replication setup, you cannot use MySQL 5.6.39, MySQL 5.6.40, and MySQL 5.6.41 concurrently, although you could use any two of these releases together.

Important

It is strongly recommended to use the most recent release available within a given MySQL release series because replication (and other) capabilities are

> continually being improved. It is also recommended to upgrade sources and replicas that use early releases of a release series of MySQL to GA (production) releases when the latter become available for that release series.

Replication from newer sources to older replicas may be possible, but is generally not supported. This is due to a number of factors:

- **Binary log format changes.** The binary log format can change between major releases. While we attempt to maintain backward compatibility, this is not always possible. For example, the binary log format implemented in MySQL 5.0 changed considerably from that used in previous versions, especially with regard to handling of character sets, `LOAD DATA`, and time zones. This means that replication from a MySQL 5.0 (or later) source to a MySQL 4.1 (or earlier) replica is generally not supported.

  This also has significant implications for upgrading replication servers; see Section 17.4.3, “Upgrading a Replication Setup”, for more information.

- **Use of row-based replication.** Row-based replication was implemented in MySQL 5.1.5, so you cannot replicate using row-based replication from any MySQL 5.6 or later source to a replica older than MySQL 5.1.5.

  For more information about row-based replication, see Section 17.1.2, “Replication Formats”.

- **SQL incompatibilities.** You cannot replicate from a newer source to an older replica using statement-based replication if the statements to be replicated use SQL features available on the source but not on the replica.

  However, if both the source and the replica support row-based replication, and there are no data definition statements to be replicated that depend on SQL features found on the source but not on the replica, you can use row-based replication to replicate the effects of data modification statements even if the DDL run on the source is not supported on the replica.

- **Important Variables in MySQL 5.6.** Features have been added to MySQL 5.6 which need to be disabled when replicating to earlier MySQL versions. To avoid incompatibilities, set the following variables on the MySQL 5.6 source:

  - `binlog_checksum=NONE`
  - `binlog_row_image=FULL`
  - `binlog_rows_query_log_events=OFF`
  - `log_bin_use_v1_row_events=1` (NDB Cluster only)
  - `gtid_mode=OFF`

For more information on potential replication issues, see Section 17.4.1, “Replication Features and Issues”.

17.4.3 Upgrading a Replication Setup

When you upgrade servers that participate in a replication setup, the procedure for upgrading depends on the current server versions and the version to which you are upgrading. This section provides information about how upgrading affects replication. For general information about upgrading MySQL, see Section 2.11, “Upgrading MySQL”

When you upgrade a source to 5.6 from an earlier MySQL release series, you should first ensure that all the replicas of this source are using the same 5.6.x release. If this is not the case, you should first upgrade the replicas. To upgrade each replica, shut it down, upgrade it to the appropriate 5.6.x version, restart it, and restart replication. Relay logs created by the replica after the upgrade are in 5.6 format.

Changes affecting operations in strict SQL mode may result in replication failure on an updated replica. For example, as of MySQL 5.6.13, the server restricts insertion of a `DEFAULT` value of 0 for temporal data types in strict mode (`STRICT_TRANS_TABLES` or `STRICT_ALL_TABLES`). A resulting incompatibility for replication if you use statement-based logging (`binlog_format=STATEMENT`) is that if a replica is upgraded, a source which has not been upgraded executes statements without error that may fail on the replica and replication stops. To deal with this, stop all new statements on the source and wait until the replicas catch up. Then upgrade the replicas. Alternatively, if you cannot stop new statements, temporarily change to row-based logging on the source (`binlog_format=ROW`) and wait until all replicas have processed all binary logs produced up to the point of this change. Then upgrade the replicas.

After the replicas have been upgraded, shut down the source, upgrade it to the same 5.6.x release as the replicas, and restart it. If you had temporarily changed the source to row-based logging, change it back to statement-based logging. The 5.6 source is able to read the old binary logs written prior to the upgrade and to send them to the 5.6 replicas. The replicas recognize the old format and handle it properly. Binary logs created by the source subsequent to the upgrade are in 5.6 format. These too are recognized by the 5.6 replicas.

In other words, when upgrading to MySQL 5.6, the replicas must be MySQL 5.6 before you can upgrade the source to 5.6. Note that downgrading from 5.6 to older versions does not work so simply: You must ensure that any 5.6 binary log or relay log has been fully processed, so that you can remove it before proceeding with the downgrade.

Some upgrades may require that you drop and re-create database objects when you move from one MySQL series to the next. For example, collation changes might require that table indexes be rebuilt. Such operations, if necessary, are detailed at Section 2.11.3, “Changes in MySQL 5.6”. It is safest to perform these operations separately on the replicas and the source, and to disable replication of these operations from the source to the replica. To achieve this, use the following procedure:

1. Stop all the replicas and upgrade them. Restart them with the `--skip-slave-start` option so that they do not connect to the source. Perform any table repair or rebuilding operations needed to re-create database objects, such as use of `REPAIR TABLE` or `ALTER TABLE`, or dumping and reloading tables or triggers.

2. Disable the binary log on the source. To do this without restarting the source, execute a `SET sql_log_bin = OFF` statement. Alternatively, stop the source and restart it without the `--log-bin` option. If you restart the source, you might also want to disallow client connections. For example, if all clients connect using TCP/IP, enable the `skip_networking` system variable when you restart the source.

3. With the binary log disabled, perform any table repair or rebuilding operations needed to re-create database objects. The binary log must be disabled during this step to prevent these operations from being logged and sent to the replicas later.

4. Re-enable the binary log on the source. If you set `sql_log_bin` to `OFF` earlier, execute a `SET sql_log_bin = ON` statement. If you restarted the source to disable the binary log, restart it with `--log-bin`, and without enabling the `skip_networking` system variable so that clients and replicas can connect.

5. Restart the replicas, this time without the `--skip-slave-start` option.

Replication with global transaction identifiers was introduced in MySQL 5.6.7. If you are upgrading an existing replication setup from a version of MySQL that does not support GTIDs to a version that does, you should not enable GTIDs on either the source or the replica before making sure that the setup meets all the requirements for GTID-based replication. See Section 17.1.3.2, “Setting Up Replication Using GTIDs”, which contains information about converting existing replication setups to use GTID-based replication.

When the server is running with global transaction identifiers (GTIDs) enabled (`gtid_mode=ON`), do not enable binary logging by `mysql_upgrade`.

It is not recommended to load a dump file when GTIDs are enabled on the server (`gtid_mode=ON`), if your dump file includes system tables. `mysqldump` issues DML instructions for the system tables which use the non-transactional MyISAM storage engine, and this combination is not permitted when GTIDs are enabled. Also be aware that loading a dump file from a server with GTIDs enabled, into another server with GTIDs enabled, causes different transaction identifiers to be generated.

17.4.4 Troubleshooting Replication

If you have followed the instructions but your replication setup is not working, the first thing to do is *check the error log for messages*. Many users have lost time by not doing this soon enough after encountering problems.

If you cannot tell from the error log what the problem was, try the following techniques:

- Verify that the source has binary logging enabled by issuing a `SHOW MASTER STATUS` statement. If logging is enabled, `Position` is nonzero. If binary logging is not enabled, verify that you are running the source with the `--log-bin` option.
- Verify that the `server_id` system variable was set at startup on both the source and replica and that the ID value is unique on each server.
- Verify that the replica is running. Use `SHOW SLAVE STATUS` to check whether the `Slave_IO_Running` and `Slave_SQL_Running` values are both `Yes`. If not, verify the options that were used when starting the replica server. For example, `--skip-slave-start` prevents the replication threads from starting until you issue a `START SLAVE` statement.
- If the replica is running, check whether it established a connection to the source. Use `SHOW PROCESSLIST`, find the replication I/O and SQL threads and check their `State` column to see what they display. See Section 17.2.1, “Replication Threads”. If the I/O thread state says `Connecting to master`, check the following:
  - Verify the privileges for the user being used for replication on the source.
  - Check that the host name of the source is correct and that you are using the correct port to connect to the source. The port used for replication is the same as used for client network communication (the default is `3306`). For the host name, ensure that the name resolves to the correct IP address.
  - Check the configuration file to see whether the `skip_networking` system variable has been enabled on the source or replica to disable networking. If so, comment the setting or remove it.
  - If the source has a firewall or IP filtering configuration, ensure that the network port being used for MySQL is not being filtered.
  - Check that you can reach the source by using `ping` or `traceroute`/`tracert` to reach the host.
- If the replica was running previously but has stopped, the reason usually is that some statement that succeeded on the source failed on the replica. This should never happen if you have taken a proper snapshot of the source, and never modified the data on the replica outside of the replica thread. If the replica stops unexpectedly, it is a bug or you have encountered one of the known replication limitations described in Section 17.4.1, “Replication Features and Issues”. If it is a bug, see Section 17.4.5, “How to Report Replication Bugs or Problems”, for instructions on how to report it.
- If a statement that succeeded on the source refuses to run on the replica, try the following procedure if it is not feasible to do a full database resynchronization by deleting the replica's databases and copying a new snapshot from the source:
  1. Determine whether the affected table on the replica is different from the source table. Try to understand how this happened. Then make the replica's table identical to the source's and run `START SLAVE`.

2. If the preceding step does not work or does not apply, try to understand whether it would be safe to make the update manually (if needed) and then ignore the next statement from the source.

3. If you decide that the replica can skip the next statement from the source, issue the following statements:

```
mysql> SET GLOBAL sql_slave_skip_counter = N;
mysql> START SLAVE;
```

   The value of *`N`* should be 1 if the next statement from the source does not use `AUTO_INCREMENT` or `LAST_INSERT_ID()`. Otherwise, the value should be 2. The reason for using a value of 2 for statements that use `AUTO_INCREMENT` or `LAST_INSERT_ID()` is that they take two events in the binary log of the source.

   See also Section 13.4.2.4, “SET GLOBAL sql_slave_skip_counter Statement”.

4. If you are sure that the replica started out perfectly synchronized with the source, and that no one has updated the tables involved outside of the replication SQL thread, then presumably the discrepancy is the result of a bug. If you are running the most recent version of MySQL, please report the problem. If you are running an older version, try upgrading to the latest production release to determine whether the problem persists.

17.4.5 How to Report Replication Bugs or Problems

When you have determined that there is no user error involved, and replication still either does not work at all or is unstable, it is time to send us a bug report. We need to obtain as much information as possible from you to be able to track down the bug. Please spend some time and effort in preparing a good bug report.

If you have a repeatable test case that demonstrates the bug, please enter it into our bugs database using the instructions given in Section 1.6, “How to Report Bugs or Problems”. If you have a “phantom” problem (one that you cannot duplicate at will), use the following procedure:

1. Verify that no user error is involved. For example, if you update the replica outside of the replication SQL thread, the data goes out of synchrony, and you can have unique key violations on updates. In this case, the replication SQL thread stops and waits for you to clean up the tables manually to bring them into synchrony. *This is not a replication problem. It is a problem of outside interference causing replication to fail.*

2. Run the replica with the `--log-slave-updates` and `--log-bin` options. These options cause the replica to log the updates that it receives from the source into its own binary logs.

3. Save all evidence before resetting the replication state. If we have no information or only sketchy information, it becomes difficult or impossible for us to track down the problem. The evidence you should collect is:

   - All binary log files from the source
   - All relay log files from the replica
   - The output of `SHOW MASTER STATUS` from the source at the time you discovered the problem
   - The output of `SHOW SLAVE STATUS` from the replica at the time you discovered the problem
   - Error logs from the source and the replica

4. Use `mysqlbinlog` to examine the binary logs. The following should be helpful to find the problem statement. *`log_file`* and *`log_pos`* are the `Master_Log_File` and `Read_Master_Log_Pos` values from `SHOW SLAVE STATUS`.

```
shell> mysqlbinlog --start-position=log_pos log_file | head
```

After you have collected the evidence for the problem, try to isolate it as a separate test case first. Then enter the problem with as much information as possible into our bugs database using the instructions at Section 1.6, “How to Report Bugs or Problems”.

Chapter 18 MySQL NDB Cluster 7.3 and NDB Cluster 7.4

Table of Contents

18.1 NDB Cluster Overview 2380
18.1.1 NDB Cluster Core Concepts 2382
18.1.2 NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions 2384
18.1.3 NDB Cluster Hardware, Software, and Networking Requirements 2387
18.1.4 What is New in MySQL NDB Cluster 2388
18.1.5 NDB: Added, Deprecated, and Removed Options, Variables, and Parameters 2393
18.1.6 MySQL Server Using InnoDB Compared with NDB Cluster 2397
18.1.7 Known Limitations of NDB Cluster 2400
18.2 NDB Cluster Installation 2411
18.2.1 Installation of NDB Cluster on Linux 2413
18.2.2 Installing NDB Cluster on Windows 2419
18.2.3 Initial Configuration of NDB Cluster 2428
18.2.4 Initial Startup of NDB Cluster 2430
18.2.5 NDB Cluster Example with Tables and Data 2431
18.2.6 Safe Shutdown and Restart of NDB Cluster 2434
18.2.7 Upgrading and Downgrading NDB Cluster 2435
18.2.8 The NDB Cluster Auto-Installer (No longer supported) 2438
18.3 Configuration of NDB Cluster 2449
18.3.1 Quick Test Setup of NDB Cluster 2449
18.3.2 Overview of NDB Cluster Configuration Parameters, Options, and Variables 2451
18.3.3 NDB Cluster Configuration Files 2468
18.3.4 Using High-Speed Interconnects with NDB Cluster 2608
18.4 NDB Cluster Programs 2609
18.4.1 `ndbd` — The NDB Cluster Data Node Daemon 2609
18.4.2 `ndbinfo_select_all` — Select From ndbinfo Tables 2616
18.4.3 `ndbmtd` — The NDB Cluster Data Node Daemon (Multi-Threaded) 2618
18.4.4 `ndb_mgmd` — The NDB Cluster Management Server Daemon 2619
18.4.5 `ndb_mgm` — The NDB Cluster Management Client 2626
18.4.6 `ndb_blob_tool` — Check and Repair BLOB and TEXT columns of NDB Cluster Tables 2628
18.4.7 `ndb_config` — Extract NDB Cluster Configuration Information 2630
18.4.8 `ndb_cpcd` — Automate Testing for NDB Development 2639
18.4.9 `ndb_delete_all` — Delete All Rows from an NDB Table 2639
18.4.10 `ndb_desc` — Describe NDB Tables 2639
18.4.11 `ndb_drop_index` — Drop Index from an NDB Table 2644
18.4.12 `ndb_drop_table` — Drop an NDB Table 2645
18.4.13 `ndb_error_reporter` — NDB Error-Reporting Utility 2645
18.4.14 `ndb_index_stat` — NDB Index Statistics Utility 2647
18.4.15 `ndb_move_data` — NDB Data Copy Utility 2652
18.4.16 `ndb_print_backup_file` — Print NDB Backup File Contents 2654
18.4.17 `ndb_print_file` — Print NDB Disk Data File Contents 2654
18.4.18 `ndb_print_frag_file` — Print NDB Fragment List File Contents 2655
18.4.19 `ndb_print_schema_file` — Print NDB Schema File Contents 2656
18.4.20 `ndb_print_sys_file` — Print NDB System File Contents 2656
18.4.21 `ndb_redo_log_reader` — Check and Print Content of Cluster Redo Log 2656
18.4.22 `ndb_restore` — Restore an NDB Cluster Backup 2659
18.4.23 `ndb_select_all` — Print Rows from an NDB Table 2680
18.4.24 `ndb_select_count` — Print Row Counts for NDB Tables 2683
18.4.25 `ndb_setup.py` — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED) 2684
18.4.26 `ndb_show_tables` — Display List of NDB Tables 2687
18.4.27 `ndb_size.pl` — NDBCLUSTER Size Requirement Estimator 2688

18.4.28 ndb_waiter — Wait for NDB Cluster to Reach a Given Status 2691
18.4.29 Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs 2694
18.5 Management of NDB Cluster 2699
18.5.1 Commands in the NDB Cluster Management Client 2699
18.5.2 NDB Cluster Log Messages 2703
18.5.3 Event Reports Generated in NDB Cluster 2717
18.5.4 Summary of NDB Cluster Start Phases 2728
18.5.5 Performing a Rolling Restart of an NDB Cluster 2729
18.5.6 NDB Cluster Single User Mode 2731
18.5.7 Adding NDB Cluster Data Nodes Online 2732
18.5.8 Online Backup of NDB Cluster 2743
18.5.9 MySQL Server Usage for NDB Cluster 2747
18.5.10 NDB Cluster Disk Data Tables 2748
18.5.11 Online Operations with ALTER TABLE in NDB Cluster 2756
18.5.12 Distributed Privileges Using Shared Grant Tables 2759
18.5.13 NDB API Statistics Counters and Variables 2762
18.5.14 ndbinfo: The NDB Cluster Information Database 2772
18.5.15 Quick Reference: NDB Cluster SQL Statements 2812
18.5.16 NDB Cluster Security Issues 2817
18.6 NDB Cluster Replication 2824
18.6.1 NDB Cluster Replication: Abbreviations and Symbols 2826
18.6.2 General Requirements for NDB Cluster Replication 2826
18.6.3 Known Issues in NDB Cluster Replication 2827
18.6.4 NDB Cluster Replication Schema and Tables 2834
18.6.5 Preparing the NDB Cluster for Replication 2836
18.6.6 Starting NDB Cluster Replication (Single Replication Channel) 2838
18.6.7 Using Two Replication Channels for NDB Cluster Replication 2839
18.6.8 Implementing Failover with NDB Cluster Replication 2840
18.6.9 NDB Cluster Backups With NDB Cluster Replication 2842
18.6.10 NDB Cluster Replication: Bidirectional and Circular Replication 2848
18.6.11 NDB Cluster Replication Conflict Resolution 2851
18.7 NDB Cluster Release Notes 2865

MySQL *NDB Cluster* is a high-availability, high-redundancy version of MySQL adapted for the distributed computing environment. The most recent NDB Cluster release series uses version 8 of the NDB storage engine (also known as NDBCLUSTER) to enable running several computers with MySQL servers and other software in a cluster. NDB Cluster 8.0, now available as a General Availability (GA) release beginning with version 8.0.19, incorporates version 8.0 of the NDB storage engine. NDB Cluster 7.6 and NDB Cluster 7.5, still available as GA releases, use versions 7.6 and 7.5 of NDB, respectively. Previous GA releases still available for use in production, NDB Cluster 7.4 and NDB Cluster 7.3, incorporate NDB versions 7.4 and 7.3, respectively. NDB Cluster 7.2, which uses version 7.2 of the NDB storage engine, is a past GA release that is no longer supported for new deployments; legacy NDB 7.2 users are encouraged to upgrade to NDB 7.6 or NDB 8.0. *NDB 7.1 and older release series are no longer supported or maintained*.

Support for the NDB storage engine is not included in standard MySQL Server 5.6 binaries built by Oracle. Instead, users of NDB Cluster binaries from Oracle should upgrade to the most recent binary release of NDB Cluster for supported platforms—these include RPMs that should work with most Linux distributions. NDB Cluster users who build from source should use the sources provided for NDB Cluster. (Locations where the sources can be obtained are listed later in this section.)

This chapter contains information about NDB Cluster 7.3 releases through 5.6.51-ndb-7.3.32 as well as NDB Cluster 7.4 releases through 5.6.51-ndb-7.4.31. The latest available rleases of these are 5.6.50-ndb-7.3.31 and 5.6.50-ndb-7.4.30, respectively. NDB Cluster 8.0 is now available as a General Availability release, and recommended for new deployments; for information about NDB Cluster 8.0, see What is New in NDB Cluster. NDB Cluster 7.6, 7.5, 7.4, and 7.3 are previous GA releases still

supported in production. For similar information about NDB Cluster 7.6 and 7.5, see What is New in NDB Cluster.

Supported Platforms. NDB Cluster is currently available and supported on a number of platforms. For exact levels of support available for on specific combinations of operating system versions, operating system distributions, and hardware platforms, please refer to https://www.mysql.com/support/supportedplatforms/cluster.html.

Availability. NDB Cluster binary and source packages are available for supported platforms from https://dev.mysql.com/downloads/cluster/.

NDB Cluster release numbers. NDB Cluster follows a somewhat different release pattern from the mainline MySQL Server 5.6 series of releases. In this *Manual* and other MySQL documentation, we identify these and later NDB Cluster releases employing a version number that begins with “NDB”. This version number is that of the `NDBCLUSTER` storage engine used in the release, and not of the MySQL server version on which the NDB Cluster release is based.

Version strings used in NDB Cluster software. The version string displayed by NDB Cluster programs uses this format:

```
mysql-mysql_server_version-ndb-ndb_engine_version
```

`mysql_server_version` represents the version of the MySQL Server on which the NDB Cluster release is based. For all NDB Cluster 7.3 and current NDB Cluster 7.4 releases, this is “5.6”. *`ndb_engine_version`* is the version of the `NDB` storage engine used by this release of the NDB Cluster software. You can see this format used in the `mysql` client, as shown here:

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.6.51-ndb-7.4.31 Source distribution

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SELECT VERSION()\G
*************************** 1. row ***************************
VERSION(): 5.6.51-ndb-7.4.31
1 row in set (0.00 sec)
```

This version string is also displayed in the output of the `SHOW` command in the `ndb_mgm` client:

```
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=1    @10.0.10.6  (5.6.51-ndb-7.4.31, Nodegroup: 0, *)
id=2    @10.0.10.8  (5.6.51-ndb-7.4.31, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=3    @10.0.10.2  (5.6.51-ndb-7.4.31)

[mysqld(API)]   2 node(s)
id=4    @10.0.10.10  (5.6.51-ndb-7.4.31)
id=5 (not connected, accepting connect from any host)
```

The version string identifies the mainline MySQL version from which the NDB Cluster release was branched and the version of the `NDB` storage engine used. For example, the full version string for NDB 7.4.4 (the first NDB Cluster 7.4 GA release) is `mysql-5.6.23-ndb-7.4.4`. From this we can determine the following:

- Since the portion of the version string preceding `-ndb-` is the base MySQL Server version, this means that NDB 7.4.4 derives from MySQL 5.6.23, and contains all feature enhancements and bug fixes from MySQL 5.6 up to and including MySQL 5.6.23.

- Since the portion of the version string following `-ndb-` represents the version number of the `NDB` (or `NDBCLUSTER`) storage engine, NDB 7.4.4 uses version 7.4.4 of the `NDBCLUSTER` storage engine.

New NDB Cluster releases are numbered according to updates in the `NDB` storage engine, and do not necessarily correspond in a one-to-one fashion with mainline MySQL Server releases. For example, NDB 7.4.4 (as previously noted) is based on MySQL 5.6.23, while NDB 7.4.3 was based on MySQL 5.6.22 (version string: `mysql-5.6.22-ndb-7.4.3`).

Compatibility with standard MySQL 5.6 releases. While many standard MySQL schemas and applications can work using NDB Cluster, it is also true that unmodified applications and database schemas may be slightly incompatible or have suboptimal performance when run using NDB Cluster (see Section 18.1.7, "Known Limitations of NDB Cluster"). Most of these issues can be overcome, but this also means that you are very unlikely to be able to switch an existing application datastore—that currently uses, for example, `MyISAM` or `InnoDB`—to use the `NDB` storage engine without allowing for the possibility of changes in schemas, queries, and applications. In addition, the MySQL Server and NDB Cluster codebases diverge considerably, so that the standard `mysqld` cannot function as a drop-in replacement for the version of `mysqld` supplied with NDB Cluster.

NDB Cluster development source trees. NDB Cluster development trees can also be accessed from https://github.com/mysql/mysql-server.

The NDB Cluster development sources maintained at https://github.com/mysql/mysql-server are licensed under the GPL. For information about obtaining MySQL sources using Git and building them yourself, see Section 2.9.5, "Installing MySQL Using a Development Source Tree".

Note

As with MySQL Server 5.6, NDB Cluster 7.3 and NDB Cluster 7.4 releases are built using `CMake`.

NDB Cluster 8.0 is available beginning with NDB 8.0.19 as a General Availability release, and is recommended for new deployments; see What is New in NDB Cluster, for more information. NDB Cluster 7.6 and 7.5 are previous GA releases still supported in production; for information about NDB Cluster 7.6, see What is New in NDB Cluster 7.6. For similar information about NDB Cluster 7.5, see What is New in NDB Cluster 7.5. NDB Cluster 7.4 and 7.3 are previous GA releases still supported in production, although we recommend that new deployments for production use NDB Cluster 8.0.

This chapter represents a work in progress, and its contents are subject to revision as NDB Cluster continues to evolve. Additional information regarding NDB Cluster can be found on the MySQL website at http://www.mysql.com/products/cluster/.

Additional Resources. More information about NDB Cluster can be found in the following places:

- For answers to some commonly asked questions about NDB Cluster, see Section A.10, "MySQL 5.6 FAQ: NDB Cluster".
- The NDB Cluster Forum: https://forums.mysql.com/list.php?25.
- Many NDB Cluster users and developers blog about their experiences with NDB Cluster, and make feeds of these available through PlanetMySQL.

18.1 NDB Cluster Overview

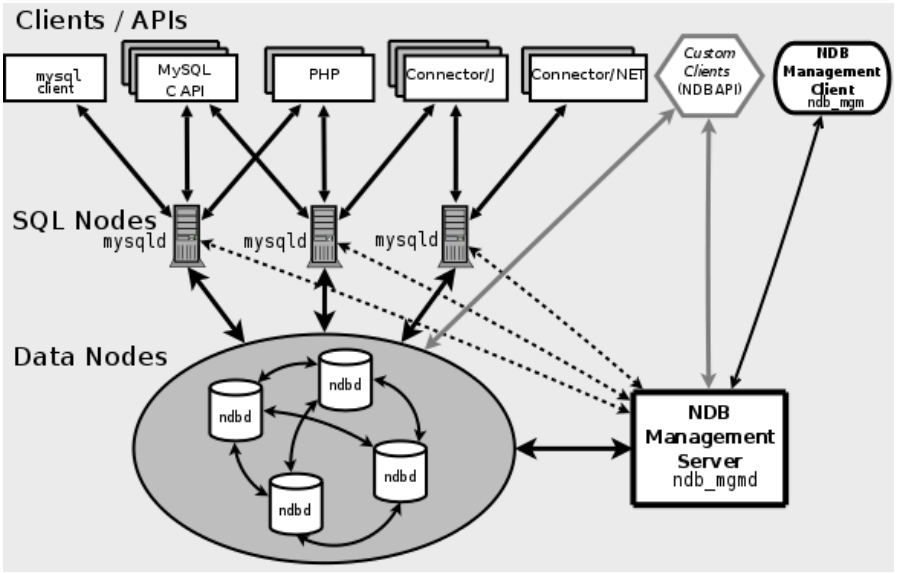
NDB Cluster is a technology that enables clustering of in-memory databases in a shared-nothing system. The shared-nothing architecture enables the system to work with very inexpensive hardware, and with a minimum of specific requirements for hardware or software.

NDB Cluster is designed not to have any single point of failure. In a shared-nothing system, each component is expected to have its own memory and disk, and the use of shared storage mechanisms such as network shares, network file systems, and SANs is not recommended or supported.

NDB Cluster integrates the standard MySQL server with an in-memory clustered storage engine called NDB (which stands for "*N*etwork *D*ata*B*ase"). In our documentation, the term NDB refers to the part of the setup that is specific to the storage engine, whereas "MySQL NDB Cluster" refers to the combination of one or more MySQL servers with the NDB storage engine.

An NDB Cluster consists of a set of computers, known as *hosts*, each running one or more processes. These processes, known as *nodes*, may include MySQL servers (for access to NDB data), data nodes (for storage of the data), one or more management servers, and possibly other specialized data access programs. The relationship of these components in an NDB Cluster is shown here:

Figure 18.1 NDB Cluster Components



All these programs work together to form an NDB Cluster (see Section 18.4, "NDB Cluster Programs". When data is stored by the NDB storage engine, the tables (and table data) are stored in the data nodes. Such tables are directly accessible from all other MySQL servers (SQL nodes) in the cluster. Thus, in a payroll application storing data in a cluster, if one application updates the salary of an employee, all other MySQL servers that query this data can see this change immediately.

Although an NDB Cluster SQL node uses the mysqld server daemon, it differs in a number of critical respects from the mysqld binary supplied with the MySQL 5.6 distributions, and the two versions of mysqld are not interchangeable.

In addition, a MySQL server that is not connected to an NDB Cluster cannot use the NDB storage engine and cannot access any NDB Cluster data.

The data stored in the data nodes for NDB Cluster can be mirrored; the cluster can handle failures of individual data nodes with no other impact than that a small number of transactions are aborted due to losing the transaction state. Because transactional applications are expected to handle transaction failure, this should not be a source of problems.

Individual nodes can be stopped and restarted, and can then rejoin the system (cluster). Rolling restarts (in which all nodes are restarted in turn) are used in making configuration changes and software upgrades (see Section 18.5.5, "Performing a Rolling Restart of an NDB Cluster"). Rolling restarts are also used as part of the process of adding new data nodes online (see Section 18.5.7, "Adding NDB Cluster Data Nodes Online"). For more information about data nodes, how they are organized in an NDB Cluster, and how they handle and store NDB Cluster data, see Section 18.1.2, "NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions".

Backing up and restoring NDB Cluster databases can be done using the NDB-native functionality found in the NDB Cluster management client and the ndb_restore program included in the NDB Cluster distribution. For more information, see Section 18.5.8, "Online Backup of NDB Cluster", and Section 18.4.22, "ndb_restore — Restore an NDB Cluster Backup". You can also use the

standard MySQL functionality provided for this purpose in `mysqldump` and the MySQL server. See Section 4.5.4, “`mysqldump` — A Database Backup Program”, for more information.

NDB Cluster nodes can employ different transport mechanisms for inter-node communications; TCP/IP over standard 100 Mbps or faster Ethernet hardware is used in most real-world deployments.

18.1.1 NDB Cluster Core Concepts

`NDBCLUSTER` (also known as `NDB`) is an in-memory storage engine offering high-availability and data-persistence features.

The `NDBCLUSTER` storage engine can be configured with a range of failover and load-balancing options, but it is easiest to start with the storage engine at the cluster level. NDB Cluster's `NDB` storage engine contains a complete set of data, dependent only on other data within the cluster itself.

The “Cluster” portion of NDB Cluster is configured independently of the MySQL servers. In an NDB Cluster, each part of the cluster is considered to be a *node*.

Note

In many contexts, the term “node” is used to indicate a computer, but when discussing NDB Cluster it means a *process*. It is possible to run multiple nodes on a single computer; for a computer on which one or more cluster nodes are being run we use the term *cluster host*.

There are three types of cluster nodes, and in a minimal NDB Cluster configuration, there must be at least three nodes, one of each of these types:

- *Management node*: The role of this type of node is to manage the other nodes within the NDB Cluster, performing such functions as providing configuration data, starting and stopping nodes, and running backups. Because this node type manages the configuration of the other nodes, a node of this type should be started first, before any other node. A management node is started with the command `ndb_mgmd`.

- *Data node*: This type of node stores cluster data. There are as many data nodes as there are fragment replicas, times the number of fragments (see Section 18.1.2, “NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions”). For example, with two fragment replicas, each having two fragments, you need four data nodes. One fragment replica is sufficient for data storage, but provides no redundancy; therefore, it is recommended to have two (or more) fragment replicas to provide redundancy, and thus high availability. A data node is started with the command `ndbd` (see Section 18.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”) or `ndbmtd` (see Section 18.4.3, “`ndbmtd` — The NDB Cluster Data Node Daemon (Multi-Threaded)”).

  NDB Cluster tables are normally stored completely in memory rather than on disk (this is why we refer to NDB Cluster as an *in-memory* database). However, some NDB Cluster data can be stored on disk; see Section 18.5.10, “NDB Cluster Disk Data Tables”, for more information.

- *SQL node*: This is a node that accesses the cluster data. In the case of NDB Cluster, an SQL node is a traditional MySQL server that uses the `NDBCLUSTER` storage engine. An SQL node is a `mysqld` process started with the `--ndbcluster` and `--ndb-connectstring` options, which are explained elsewhere in this chapter, possibly with additional MySQL server options as well.

  An SQL node is actually just a specialized type of *API node*, which designates any application which accesses NDB Cluster data. Another example of an API node is the `ndb_restore` utility that is used to restore a cluster backup. It is possible to write such applications using the NDB API. For basic information about the NDB API, see Getting Started with the NDB API.

Important

It is not realistic to expect to employ a three-node setup in a production environment. Such a configuration provides no redundancy; to benefit from NDB

Cluster's high-availability features, you must use multiple data and SQL nodes. The use of multiple management nodes is also highly recommended.

For a brief introduction to the relationships between nodes, node groups, fragment replicas, and partitions in NDB Cluster, see Section 18.1.2, “NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions”.

Configuration of a cluster involves configuring each individual node in the cluster and setting up individual communication links between nodes. NDB Cluster is currently designed with the intention that data nodes are homogeneous in terms of processor power, memory space, and bandwidth. In addition, to provide a single point of configuration, all configuration data for the cluster as a whole is located in one configuration file.

The management server manages the cluster configuration file and the cluster log. Each node in the cluster retrieves the configuration data from the management server, and so requires a way to determine where the management server resides. When interesting events occur in the data nodes, the nodes transfer information about these events to the management server, which then writes the information to the cluster log.

In addition, there can be any number of cluster client processes or applications. These include standard MySQL clients, `NDB`-specific API programs, and management clients. These are described in the next few paragraphs.

Standard MySQL clients. NDB Cluster can be used with existing MySQL applications written in PHP, Perl, C, C++, Java, Python, Ruby, and so on. Such client applications send SQL statements to and receive responses from MySQL servers acting as NDB Cluster SQL nodes in much the same way that they interact with standalone MySQL servers.

MySQL clients using an NDB Cluster as a data source can be modified to take advantage of the ability to connect with multiple MySQL servers to achieve load balancing and failover. For example, Java clients using Connector/J 5.0.6 and later can use `jdbc:mysql:loadbalance://` URLs (improved in Connector/J 5.1.7) to achieve load balancing transparently; for more information about using Connector/J with NDB Cluster, see Using Connector/J with NDB Cluster.

NDB client programs. Client programs can be written that access NDB Cluster data directly from the `NDBCLUSTER` storage engine, bypassing any MySQL Servers that may be connected to the cluster, using the *NDB API*, a high-level C++ API. Such applications may be useful for specialized purposes where an SQL interface to the data is not needed. For more information, see The NDB API.

`NDB`-specific Java applications can also be written for NDB Cluster using the *NDB Cluster Connector for Java*. This NDB Cluster Connector includes *ClusterJ*, a high-level database API similar to object-relational mapping persistence frameworks such as Hibernate and JPA that connect directly to `NDBCLUSTER`, and so does not require access to a MySQL Server. See Java and NDB Cluster, and The ClusterJ API and Data Object Model, for more information.

Management clients. These clients connect to the management server and provide commands for starting and stopping nodes gracefully, starting and stopping message tracing (debug versions only), showing node versions and status, starting and stopping backups, and so on. An example of this type of program is the `ndb_mgm` management client supplied with NDB Cluster (see Section 18.4.5, “`ndb_mgm` — The NDB Cluster Management Client”). Such applications can be written using the *MGM API*, a C-language API that communicates directly with one or more NDB Cluster management servers. For more information, see The MGM API.

Oracle also makes available MySQL Cluster Manager, which provides an advanced command-line interface simplifying many complex NDB Cluster management tasks, such restarting an NDB Cluster with a large number of nodes. The MySQL Cluster Manager client also supports commands for getting and setting the values of most node configuration parameters as well as `mysqld` server options and variables relating to NDB Cluster. See MySQL™ Cluster Manager 1.4.8 User Manual, for more information.

Event logs. NDB Cluster logs events by category (startup, shutdown, errors, checkpoints, and so on), priority, and severity. A complete listing of all reportable events may be found in Section 18.5.3, “Event Reports Generated in NDB Cluster”. Event logs are of the two types listed here:

- *Cluster log*: Keeps a record of all desired reportable events for the cluster as a whole.
- *Node log*: A separate log which is also kept for each individual node.

Note

Under normal circumstances, it is necessary and sufficient to keep and examine only the cluster log. The node logs need be consulted only for application development and debugging purposes.

Checkpoint. Generally speaking, when data is saved to disk, it is said that a *checkpoint* has been reached. More specific to NDB Cluster, a checkpoint is a point in time where all committed transactions are stored on disk. With regard to the `NDB` storage engine, there are two types of checkpoints which work together to ensure that a consistent view of the cluster's data is maintained. These are shown in the following list:

- *Local Checkpoint (LCP)*: This is a checkpoint that is specific to a single node; however, LCPs take place for all nodes in the cluster more or less concurrently. An LCP involves saving all of a node's data to disk, and so usually occurs every few minutes. The precise interval varies, and depends upon the amount of data stored by the node, the level of cluster activity, and other factors.
- *Global Checkpoint (GCP)*: A GCP occurs every few seconds, when transactions for all nodes are synchronized and the redo-log is flushed to disk.

For more information about the files and directories created by local checkpoints and global checkpoints, see NDB Cluster Data Node File System Directory.

18.1.2 NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions

This section discusses the manner in which NDB Cluster divides and duplicates data for storage.

A number of concepts central to an understanding of this topic are discussed in the next few paragraphs.

Data node. An `ndbd` or `ndbmtd` process, which stores one or more *fragment replicas*—that is, copies of the *partitions* (discussed later in this section) assigned to the node group of which the node is a member.

Each data node should be located on a separate computer. While it is also possible to host multiple data node processes on a single computer, such a configuration is not usually recommended.

It is common for the terms “node” and “data node” to be used interchangeably when referring to an `ndbd` or `ndbmtd` process; where mentioned, management nodes (`ndb_mgmd` processes) and SQL nodes (`mysqld` processes) are specified as such in this discussion.

Node group. A node group consists of one or more nodes, and stores partitions, or sets of *fragment replicas* (see next item).

The number of node groups in an NDB Cluster is not directly configurable; it is a function of the number of data nodes and of the number of fragment replicas (`NoOfReplicas` configuration parameter), as shown here:

```
[# of node groups] = [# of data nodes] / NoOfReplicas
```

Thus, an NDB Cluster with 4 data nodes has 4 node groups if `NoOfReplicas` is set to 1 in the `config.ini` file, 2 node groups if `NoOfReplicas` is set to 2, and 1 node group if `NoOfReplicas` is set to 4. Fragment replicas are discussed later in this section; for more information about `NoOfReplicas`, see Section 18.3.3.6, “Defining NDB Cluster Data Nodes”.

Note

All node groups in an NDB Cluster must have the same number of data nodes.

You can add new node groups (and thus new data nodes) online, to a running NDB Cluster; see Section 18.5.7, “Adding NDB Cluster Data Nodes Online”, for more information.

Partition. This is a portion of the data stored by the cluster. Each node is responsible for keeping at least one copy of any partitions assigned to it (that is, at least one fragment replica) available to the cluster.

The number of partitions used by default by NDB Cluster depends on the number of data nodes and the number of LDM threads in use by the data nodes, as shown here:

```
[# of partitions] = [# of data nodes] * [# of LDM threads]
```

When using data nodes running `ndbmtd`, the number of LDM threads is controlled by the setting for `MaxNoOfExecutionThreads`. When using `ndbd` there is a single LDM thread, which means that there are as many cluster partitions as nodes participating in the cluster. This is also the case when using `ndbmtd` with `MaxNoOfExecutionThreads` set to 3 or less. (You should be aware that the number of LDM threads increases with the value of this parameter, but not in a strictly linear fashion, and that there are additional constraints on setting it; see the description of `MaxNoOfExecutionThreads` for more information.)

NDB and user-defined partitioning. NDB Cluster normally partitions `NDBCLUSTER` tables automatically. However, it is also possible to employ user-defined partitioning with `NDBCLUSTER` tables. This is subject to the following limitations:

1. Only the `KEY` and `LINEAR KEY` partitioning schemes are supported in production with `NDB` tables.

2. The maximum number of partitions that may be defined explicitly for any `NDB` table is `8 * [number of LDM threads] * [number of node groups]`, the number of node groups in an NDB Cluster being determined as discussed previously in this section. When running `ndbd` for data node processes, setting the number of LDM threads has no effect (since `ThreadConfig` applies only to `ndbmtd`); in such cases, this value can be treated as though it were equal to 1 for purposes of performing this calculation.

   See Section 18.4.3, “`ndbmtd` — The NDB Cluster Data Node Daemon (Multi-Threaded)”, for more information.

For more information relating to NDB Cluster and user-defined partitioning, see Section 18.1.7, “Known Limitations of NDB Cluster”, and Section 19.6.2, “Partitioning Limitations Relating to Storage Engines”.

Fragment Replica. This is a copy of a cluster partition. Each node in a node group stores a replica. Also sometimes known as a *partition replica*. The number of fragment replicas is equal to the number of nodes per node group.

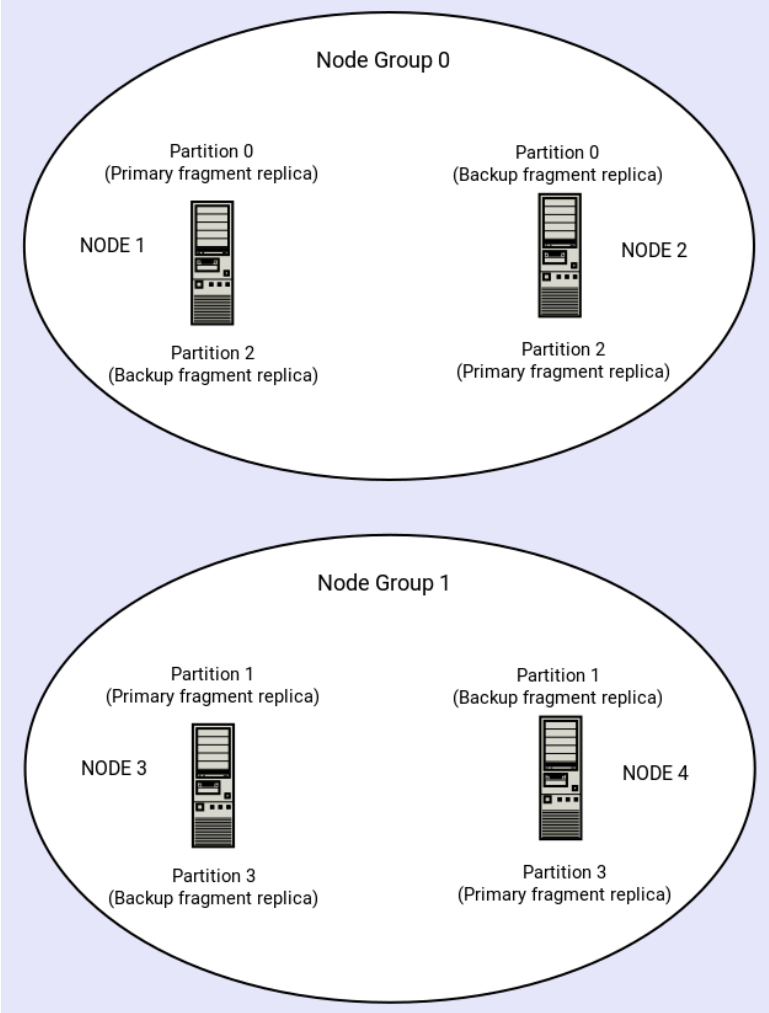
A fragment replica belongs entirely to a single node; a node can (and usually does) store several fragment replicas.

The following diagram illustrates an NDB Cluster with four data nodes running `ndbd`, arranged in two node groups of two nodes each; nodes 1 and 2 belong to node group 0, and nodes 3 and 4 belong to node group 1.

Note

Only data nodes are shown here; although a working NDB Cluster requires an `ndb_mgmd` process for cluster management and at least one SQL node to access the data stored by the cluster, these have been omitted from the figure for clarity.

Figure 18.2 NDB Cluster with Two Node Groups

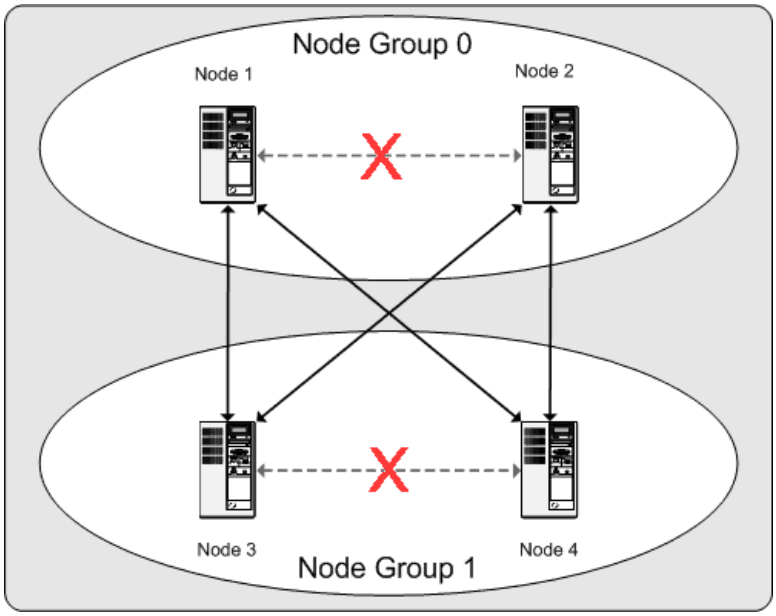


The data stored by the cluster is divided into four partitions, numbered 0, 1, 2, and 3. Each partition is stored—in multiple copies—on the same node group. Partitions are stored on alternate node groups as follows:

- Partition 0 is stored on node group 0; a *primary fragment replica* (primary copy) is stored on node 1, and a *backup fragment replica* (backup copy of the partition) is stored on node 2.
- Partition 1 is stored on the other node group (node group 1); this partition's primary fragment replica is on node 3, and its backup fragment replica is on node 4.
- Partition 2 is stored on node group 0. However, the placing of its two fragment replicas is reversed from that of Partition 0; for Partition 2, the primary fragment replica is stored on node 2, and the backup on node 1.
- Partition 3 is stored on node group 1, and the placement of its two fragment replicas are reversed from those of partition 1. That is, its primary fragment replica is located on node 4, with the backup on node 3.

What this means regarding the continued operation of an NDB Cluster is this: so long as each node group participating in the cluster has at least one node operating, the cluster has a complete copy of all data and remains viable. This is illustrated in the next diagram.

Figure 18.3 Nodes Required for a 2x2 NDB Cluster



In this example, the cluster consists of two node groups each consisting of two data nodes. Each data node is running an instance of `ndbd`. Any combination of at least one node from node group 0 and at least one node from node group 1 is sufficient to keep the cluster “alive”. However, if both nodes from a single node group fail, the combination consisting of the remaining two nodes in the other node group is not sufficient. In this situation, the cluster has lost an entire partition and so can no longer provide access to a complete set of all NDB Cluster data.

18.1.3 NDB Cluster Hardware, Software, and Networking Requirements

One of the strengths of NDB Cluster is that it can be run on commodity hardware and has no unusual requirements in this regard, other than for large amounts of RAM, due to the fact that all live data storage is done in memory. (It is possible to reduce this requirement using Disk Data tables—see Section 18.5.10, “NDB Cluster Disk Data Tables”, for more information about these.) Naturally, multiple and faster CPUs can enhance performance. Memory requirements for other NDB Cluster processes are relatively small.

The software requirements for NDB Cluster are also modest. Host operating systems do not require any unusual modules, services, applications, or configuration to support NDB Cluster. For supported operating systems, a standard installation should be sufficient. The MySQL software requirements are simple: all that is needed is a production release of NDB Cluster. It is not strictly necessary to compile MySQL yourself merely to be able to use NDB Cluster. We assume that you are using the binaries appropriate to your platform, available from the NDB Cluster software downloads page at https://dev.mysql.com/downloads/cluster/.

For communication between nodes, NDB Cluster supports TCP/IP networking in any standard topology, and the minimum expected for each host is a standard 100 Mbps Ethernet card, plus a switch, hub, or router to provide network connectivity for the cluster as a whole. We strongly recommend that an NDB Cluster be run on its own subnet which is not shared with machines not forming part of the cluster for the following reasons:

- **Security.** Communications between NDB Cluster nodes are not encrypted or shielded in any way. The only means of protecting transmissions within an NDB Cluster is to run your NDB Cluster on a protected network. If you intend to use NDB Cluster for Web applications, the cluster should definitely reside behind your firewall and not in your network's De-Militarized Zone (DMZ) or elsewhere.

  See Section 18.5.16.1, “NDB Cluster Security and Networking Issues”, for more information.

- **Efficiency.** Setting up an NDB Cluster on a private or protected network enables the cluster to make exclusive use of bandwidth between cluster hosts. Using a separate switch for your NDB Cluster not only helps protect against unauthorized access to NDB Cluster data, it also ensures that NDB Cluster nodes are shielded from interference caused by transmissions between other computers on the network. For enhanced reliability, you can use dual switches and dual cards to remove the network as a single point of failure; many device drivers support failover for such communication links.

Network communication and latency. NDB Cluster requires communication between data nodes and API nodes (including SQL nodes), as well as between data nodes and other data nodes, to execute queries and updates. Communication latency between these processes can directly affect the observed performance and latency of user queries. In addition, to maintain consistency and service despite the silent failure of nodes, NDB Cluster uses heartbeating and timeout mechanisms which treat an extended loss of communication from a node as node failure. This can lead to reduced redundancy. Recall that, to maintain data consistency, an NDB Cluster shuts down when the last node in a node group fails. Thus, to avoid increasing the risk of a forced shutdown, breaks in communication between nodes should be avoided wherever possible.

The failure of a data or API node results in the abort of all uncommitted transactions involving the failed node. Data node recovery requires synchronization of the failed node's data from a surviving data node, and re-establishment of disk-based redo and checkpoint logs, before the data node returns to service. This recovery can take some time, during which the Cluster operates with reduced redundancy.

Heartbeating relies on timely generation of heartbeat signals by all nodes. This may not be possible if the node is overloaded, has insufficient machine CPU due to sharing with other programs, or is experiencing delays due to swapping. If heartbeat generation is sufficiently delayed, other nodes treat the node that is slow to respond as failed.

This treatment of a slow node as a failed one may or may not be desirable in some circumstances, depending on the impact of the node's slowed operation on the rest of the cluster. When setting timeout values such as `HeartbeatIntervalDbDb` and `HeartbeatIntervalDbApi` for NDB Cluster, care must be taken care to achieve quick detection, failover, and return to service, while avoiding potentially expensive false positives.

Where communication latencies between data nodes are expected to be higher than would be expected in a LAN environment (on the order of 100 µs), timeout parameters must be increased to ensure that any allowed periods of latency periods are well within configured timeouts. Increasing timeouts in this way has a corresponding effect on the worst-case time to detect failure and therefore time to service recovery.

LAN environments can typically be configured with stable low latency, and such that they can provide redundancy with fast failover. Individual link failures can be recovered from with minimal and controlled latency visible at the TCP level (where NDB Cluster normally operates). WAN environments may offer a range of latencies, as well as redundancy with slower failover times. Individual link failures may require route changes to propagate before end-to-end connectivity is restored. At the TCP level this can appear as large latencies on individual channels. The worst-case observed TCP latency in these scenarios is related to the worst-case time for the IP layer to reroute around the failures.

18.1.4 What is New in MySQL NDB Cluster

This section lists changes in the implementation of NDB Cluster in MySQL NDB Cluster 7.3 and NDB Cluster 7.4, as compared to earlier releases. Changes and features most likely to be of interest in NDB 7.3 are shown in the following list:

- NDB Cluster 7.3 is based on MySQL 5.6. For more information about new features in MySQL Server 5.6, see Section 1.3, “What Is New in MySQL 5.6”.
- NDB Cluster 7.3 supports foreign key constraints on tables. See Section 1.7.3.2, “FOREIGN KEY Constraints”, and Section 13.1.17.5, “FOREIGN KEY Constraints”, for more information.

- NDB Cluster 7.3 provides support for Node.js using the MySQL NoSQL Connector for JavaScript. See MySQL NoSQL Connector for JavaScript, for more information.

Changes and features in NDB Cluster 7.4 that are most likely to be of interest are shown in the following list:

- NDB Cluster 7.4 is based on MySQL 5.6 (For more information about new features in MySQL Server 5.6, see Section 1.3, “What Is New in MySQL 5.6”)
- NDB Cluster Replication conflict detection and resolution enhancements, including extensions to conflict exceptions tables (see Section 18.6.11, “NDB Cluster Replication Conflict Resolution”)
- Improvements in the management of circular (“active-active”) replication; primary/secondary assignment with `ndb_slave_conflict_role`
- Per-fragment memory usage reporting in the `memory_per_fragment` table
- A number of performance improvements, including the following enhancements:
  - Faster initial allocation of memory
  - Increased parallelization of local checkpoints (LCPs now support 32 fragments rather than 2)

    A group of configuration parameters (`MaxDiskWriteSpeed`, `MaxDiskWriteSpeedOtherNoderestart`, `MaxDiskWriteSpeedOwnRestart`) introduced in this version provides improved control over disk writes during LCPs

    Information about recent disk writes is available in the `disk_write_speed_base`, `disk_write_speed_aggregate`, and `disk_write_speed_aggregate_node` tables added to the `ndbinfo` database in the this version
  - Faster times for restoring an NDB Cluster from backup
  - Optimization of the NDB receive thread
- Improved error and other reporting during node restarts

This section contains information about NDB Cluster 7.3 releases through 5.6.51-ndb-7.3.32 as well as NDB Cluster 7.4 releases through 5.6.51-ndb-7.4.31 as compared to earlier release series. NDB Cluster 8.0 is available as a General Availability (GA) release, beginning with NDB 8.0.19; see What is New in NDB Cluster, for more information about new features and other changes in NDB 8.0. NDB Cluster 7.6 and 7.5 are previous GA releases still supported in production; for more information, see MySQL NDB Cluster 7.5 and NDB Cluster 7.6. NDB Cluster 7.4 and 7.3 are previous GA releases still supported in production, although we recommend that new deployments for production use NDB Cluster 8.0; see Section 18.1.4.2, “What is New in NDB Cluster 7.4”, and Section 18.1.4.1, “What is New in NDB Cluster 7.3”, for more information.

18.1.4.1 What is New in NDB Cluster 7.3

The following improvements to NDB Cluster have been made in NDB Cluster 7.3:

- **Based on MySQL Server 5.6.** NDB Cluster 7.3 is based on MySQL Server 5.6, so that NDB Cluster users can benefit from MySQL 5.6's improvements in scalability and performance monitoring. As with MySQL 5.6, NDB Cluster 7.3 uses `CMake` for configuring and building from source. For more information about changes and improvements in MySQL 5.6, see Section 1.3, “What Is New in MySQL 5.6”.
- **Foreign keys.** Tables created using the `NDB` storage engine version 7.3.0 and later provide support for foreign key constraints. (This includes all NDB Cluster 7.3 releases.) For general information about how MySQL 5.6 and NDB Cluster 7.3 handle foreign keys, see Section 1.7.3.2, “FOREIGN KEY Constraints”. For syntax and related information, see Section 13.1.17, “CREATE TABLE Statement”, and Section 13.1.17.5, “FOREIGN KEY Constraints”.

- **Node.js support.** NDB Cluster 7.3 also supports applications written in JavaScript using Node.js. The MySQL Connector for JavaScript includes adapters for direct access to the `NDB` storage engine and as well as for the MySQL Server. Applications using this Connector are typically event-driven and use a domain object model similar in many ways to that employed by ClusterJ. For more information, see MySQL NoSQL Connector for JavaScript.

- **End of ClusterJPA support.** ClusterJPA is no longer supported beginning with NDB 7.3.18; its source code and binary have been removed from the NDB Cluster distribution.

- **ndb_restore—NDBT dependency removal.** The dependency of `ndb_restore` on the `NDBT` library has been removed as of NDB 7.3.25. This library is used internally for development, and is not required for normal use; its inclusion in this program could lead to unwanted issues when testing.

  The principal effect of this change for users is that `ndb_restore` no longer prints `NDBT_ProgramExit - status` following completion of a run. Applications that depend upon such behavior should be updated to reflect the change when upgrading.

- **Auto-Installer deprecation and removal.** The MySQL NDB Cluster Auto-Installer web-based installation tool (`ndb_setup.py`) is deprecated in NDB 7.3.31, and is removed in NDB 7.3.32 and later. It is no longer supported.

- **ndbmemcache deprecation and removal.** `ndbmemcache` is no longer supported. `ndbmemcache` was deprecated in NDB 7.3.31, and removed in NDB 7.3.32.

- **Node.js support removed.** Beginning with the NDB Cluster 7.3.31 release, support for Node.js by NDB 7.3 has been removed.

  Support for Node.js by NDB Cluster is maintained in NDB 8.0 only.

NDB Cluster 7.3 is also supported by MySQL Cluster Manager, which provides an advanced command-line interface that can simplify many complex NDB Cluster management tasks. See MySQL™ Cluster Manager 1.4.8 User Manual, for more information.

18.1.4.2 What is New in NDB Cluster 7.4

The following improvements to NDB Cluster have been made in NDB Cluster 7.4:

- **Conflict detection and resolution enhancements.** A reserved column name namespace `NDB$` is now employed for exceptions table metacolumns, allowing an arbitrary subset of main table columns to be recorded, even if they are not part of the original table's primary key.

  Recording the complete original primary key is no longer required, due to the fact that matching against exceptions table columns is now done by name and type only. It is now also possible for you to record values of columns which not are part of the main table's primary key in the exceptions table.

  Read conflict detection is now possible. All rows read by the conflicting transaction are flagged, and logged in the exceptions table. Rows inserted in the same transaction are not included among the rows read or logged. This read tracking depends on the fragment replica having an exclusive read lock which requires setting `ndb_log_exclusive_reads` in advance. See Read conflict detection and resolution, for more information and examples.

  Existing exceptions tables remain supported. For more information, see Section 18.6.11, “NDB Cluster Replication Conflict Resolution”.

- **Circular (“active-active”) replication improvements.** When using a circular or “active-active” NDB Cluster Replication topology, you can assign one of the roles of primary of secondary to a given NDB Cluster using the `ndb_slave_conflict_role` server system variable, which can be employed when failing over from an NDB Cluster acting as primary, or when using conflict detection and resolution with `NDB$EPOCH2()` and `NDB$EPOCH2_TRANS()` (NDB 7.4.2 and later), which support delete-delete conflict handling.

See the description of the `ndb_slave_conflict_role` variable, as well as NDB$EPOCH2(), for more information. See also Section 18.6.11, “NDB Cluster Replication Conflict Resolution”.

- **Per-fragment memory usage reporting.** You can now obtain data about memory usage by individual NDB Cluster fragments from the `memory_per_fragment` view, added in NDB 7.4.1 to the `ndbinfo` information database. For more information, see Section 18.5.14.17, “The ndbinfo memory_per_fragment Table”.

- **Node restart improvements.** NDB Cluster 7.4 includes a number of improvements which decrease the time needed for data nodes to be restarted. These are described in the following list:

  - Memory allocated that is allocated on node startup cannot be used until it has been touched, which causes the operating system to set aside the actual physical memory required. In previous versions of NDB Cluster, the process of touching each page of memory that was allocated was singlethreaded, which made it relatively time-consuming. This process has now been reimplimented with multithreading. In tests with 16 threads, touch times on the order of 3 times shorter than with a single thread were observed.

  - Increased parallelization of local checkpoints; in NDB Cluster 7.4, LCPs now support 32 fragments rather than 2 as before. This greatly increases utilization of CPU power that would otherwise go unused, and can make LCPs faster by up to a factor of 10; this speedup in turn can greatly improve node restart times.

    The degree of parallelization used for the node copy phase during node and system restarts can be controlled in NDB 7.4.3 and later by setting the `MaxParallelCopyInstances` data node configuration parameter to a nonzero value.

  - Reporting on disk writes is provided by new `ndbinfo` tables `disk_write_speed_base`, `disk_write_speed_aggregate`, and `disk_write_speed_aggregate_node`, which provide information about the speed of disk writes for each LDM thread that is in use.

    This release also adds the data node configuration parameters `MinDiskWriteSpeed`, `MaxDiskWriteSpeed`, `MaxDiskWriteSpeedOtherNodeRestart`, and `MaxDiskWriteSpeedOwnRestart` to control write speeds for LCPs and backups when the present node, another node, or no node is currently restarting.

    These changes are intended to supersede configuration of disk writes using the `DiskCheckpointSpeed` and `DiskCheckpointSpeedInRestart` configuration parameters. These 2 parameters have now been deprecated, and are subject to removal in a future NDB Cluster release.

  - Faster times for restoring an NDB Cluster from backup have been obtained by replacing delayed signals found at a point which was found to be critical to performance with normal (undelayed) signals. The elimination or replacement of these unnecessary delayed signals should noticeably reduce the amount of time required to back up an NDB Cluster, or to restore an NDB Cluster from backup.

  - Several internal methods relating to the `NDB` receive thread have been optimized, to increase the efficiency of SQL processing by `NDB`. The receiver thread at time may have to process several million received records per second, so it is critical that it not perform unnecessary work or waste resources when retrieving records from NDB Cluster data nodes.

- **Improved reporting of NDB Cluster restarts and start phases.** The `restart_info` table (included in the `ndbinfo` information database beginning with NDB 7.4.2) provides current status and timing information about node and system restarts.

  Reporting and logging of NDB Cluster start phases also provides more frequent and specific printouts during startup than previously. See Section 18.5.4, “Summary of NDB Cluster Start Phases”, for more information.

- **NDB API: new Event API.** NDB 7.4.3 introduces an epoch-driven Event API that supercedes the earlier GCI-based model. The new version of the API also simplifies error detection and handling. These changes are realized in the NDB API by implementing a number of new methods for `Ndb` and `NdbEventOperation`, deprecating several other methods of both classes, and adding new type values to `Event::TableEvent`.

  The event handling methods added to `Ndb` in NDB 7.4.3 are `pollEvents2()`, `nextEvent2()`, `getHighestQueuedEpoch()`, and `getNextEventOpInEpoch2()`. The `Ndb` methods `pollEvents()`, `nextEvent()`, `getLatestGCI()`, `getGCIEventOperations()`, `isConsistent()`, and `isConsistentGCI()` are deprecated beginning with the same release.

  NDB 7.4.3 adds the `NdbEventOperation` event handling methods `getEventType2()`, `getEpoch()`, `isEmptyEpoch()`, and `isErrorEpoch`; it obsoletes `getEventType()`, `getGCI()`, `getLatestGCI()`, `isOverrun()`, `hasError()`, and `clearError()`.

  While some (but not all) of the new methods are direct replacements for deprecated methods, not all of the deprecated methods map to new ones. The Event Class, provides information as to which old methods correspond to new ones.

  Error handling using the new API is no longer handled using dedicated `hasError()` and `clearError()` methods, which are now deprecated (and thus subject to removal in a future release of NDB Cluster). To support this change, the list of `TableEvent` types now includes the values `TE_EMPTY` (empty epoch), `TE_INCONSISTENT` (inconsistent epoch), and `TE_OUT_OF_MEMORY` (inconsistent data).

  Improvements in event buffer management have also been made by implementing new `get_eventbuffer_free_percent()`, `set_eventbuffer_free_percent()`, and `get_event_buffer_memory_usage()` methods. Memory buffer usage can now be represented in application code using `Ndb::EventBufferMemoryUsage`. The `ndb_eventbuffer_free_percent` system variable, also implemented in NDB Cluster 7.4, makes it possible for event buffer memory usage to be checked from MySQL client applications.

  For more information, see the detailed descriptions for the `Ndb` and `NdbEventOperation` methods listed. See also Event::TableEvent, as well as Ndb::EventBufferMemoryUsage.

- **Per-fragment operations information.** In NDB 7.4.3 and later, counts of various types of operations on a given fragment or fragment replica can obtained easily using the `operations_per_fragment` table in the `ndbinfo` information database. This includes read, write, update, and delete operations, as well as scan and index operations performed by these. Information about operations refused, and about rows scanned and returned from a given fragment replica, is also shown in `operations_per_fragment`. This table also provides information about interpreted programs used as attribute values, and values returned by them.

- **--ndb-log-fail-terminate option.** Beginning with NDB 7.4.28, you can cause the SQL node to terminate whenever it is unable to log all row events fully. This can be done by starting `mysqld` with the `--ndb-log-fail-terminate` option.

- **NDB programs—NDBT dependency removal.** The dependency of a number of `NDB` utility programs on the `NDBT` library has been removed. This library is used internally for development, and

is not required for normal use; its inclusion in these programs could lead to unwanted issues when testing.

Affected programs are listed here, along with the NDB versions in which the dependency was removed:

- ndb_restore, in NDB 7.4.24
- ndb_show_tables, in NDB 7.4.28
- ndb_waiter, in NDB 7.4.28

The principal effect of this change for users is that these programs no longer print NDBT_ProgramExit - *status* following completion of a run. Applications that depend upon such behavior should be updated to reflect the change when upgrading to the indicated versions.

- **End of ClusterJPA support.** ClusterJPA is no longer supported beginning with NDB 7.4.16; its source code and binary have been removed from the NDB Cluster distribution.
- **Auto-Installer deprecation and removal.** The MySQL NDB Cluster Auto-Installer web-based installation tool (ndb_setup.py) is deprecated in NDB 7.4.30, and is removed in NDB 7.4.31 and later. It is no longer supported.
- **ndbmemcache deprecation and removal.** ndbmemcache is no longer supported. ndbmemcache has been deprecated beginning with NDB 7.4.3, and is scheduled for removal in NDB 7.4.31.
- **Node.js support removed.** Beginning with the NDB Cluster 7.4.30 release, support for Node.js by NDB 7.4 has been removed.

  Support for Node.js by NDB Cluster is maintained in NDB 8.0 only.

NDB Cluster 7.4 is also supported by MySQL Cluster Manager, which provides an advanced command-line interface that can simplify many complex NDB Cluster management tasks. See MySQL™ Cluster Manager 1.4.8 User Manual, for more information.

18.1.5 NDB: Added, Deprecated, and Removed Options, Variables, and Parameters

18.1.5.1 Options, Variables, and Parameters Added, Deprecated or Removed in NDB 7.3

- Node Configuration Parameters Introduced in NDB 7.3
- Node Configuration Parameters Deprecated in NDB 7.3
- Node Configuration Parameters Removed in NDB 7.3
- MySQL Server Options and Variables Introduced in NDB 7.3
- MySQL Server Options and Variables Deprecated in NDB 7.3
- MySQL Server Options and Variables Removed in NDB 7.3

The next few sections contain information about NDB node configuration parameters and NDB-specific mysqld options and variables that have been added to, deprecated in, or removed from NDB 7.3.

Node Configuration Parameters Introduced in NDB 7.3

The following node configuration parameters have been added in NDB 7.3.

- ConnectBackoffMaxTime: Specifies longest time in milliseconds (~100ms resolution) to allow between connection attempts to any given data node by this API node. Excludes time elapsed while

connection attempts are ongoing, which in worst case can take several seconds. Disable by setting to 0. If no data nodes are currently connected to this API node, StartConnectBackoffMaxTime is used instead. Added in NDB 7.3.7.

- `DiskPageBufferEntries`: Memory to allocate in DiskPageBufferMemory; very large disk transactions may require increasing this value. Added in NDB 7.3.8.
- `HeartbeatIntervalMgmdMgmd`: Time between management-node-to-management-node heartbeats; connection between management nodes is considered lost after 3 missed heartbeats. Added in NDB 7.3.3.
- `LcpScanProgressTimeout`: Maximum time that local checkpoint fragment scan can be stalled before node is shut down to ensure systemwide LCP progress. Use 0 to disable. Added in NDB 7.3.3.
- `RestartSubscriberConnectTimeout`: Amount of time for data node to wait for subscribing API nodes to connect. Set to 0 to disable timeout, which is always resolved to nearest full second. Added in NDB 7.3.6.
- `StartConnectBackoffMaxTime`: Same as ConnectBackoffMaxTime except that this parameter is used in its place if no data nodes are connected to this API node. Added in NDB 7.3.7.
- `TimeBetweenGlobalCheckpointsTimeout`: Minimum timeout for group commit of transactions to disk. Added in NDB 7.3.9.

Node Configuration Parameters Deprecated in NDB 7.3

The following node configuration parameters have been deprecated in NDB 7.3.

- `ReservedSendBufferMemory`: This parameter is present in NDB code but is not enabled. Deprecated as of NDB 7.3.1.

Node Configuration Parameters Removed in NDB 7.3

No node configuration parameters have been removed from NDB 7.3.

MySQL Server Options and Variables Introduced in NDB 7.3

The following `mysqld` system variables, status variables, and options have been added in NDB 7.3.

- `Ndb_last_commit_epoch_server`: Epoch most recently committed by NDB. Added in NDB 7.3.8.
- `Ndb_last_commit_epoch_session`: Epoch most recently committed by this NDB client. Added in NDB 7.3.8.
- `Ndb_slave_max_replicated_epoch`: Most recently committed NDB epoch on this replica. When this value is greater than or equal to Ndb_conflict_last_conflict_epoch, no conflicts have yet been detected. Added in NDB 7.3.8.
- `create_old_temporals`: Use pre-5.6.4 storage format for temporal types when creating tables. Intended for use in replication and upgrades/downgrades between NDB 7.2 and NDB 7.3/7.4. Added in NDB 7.3.10.
- `ndb_eventbuffer_max_alloc`: Maximum memory that can be allocated for buffering events by NDB API. Defaults to 0 (no limit). Added in NDB 7.3.3.
- `ndb_recv_thread_activation_threshold`: Activation threshold when receive thread takes over polling of cluster connection (measured in concurrently active threads). Added in NDB 7.3.1.
- `ndb_recv_thread_cpu_mask`: CPU mask for locking receiver threads to specific CPUs; specified as hexadecimal. See documentation for details. Added in NDB 7.3.1.

- `ndb_show_foreign_key_mock_tables`: Show mock tables used to support foreign_key_checks=0. Added in NDB 7.3.2.

MySQL Server Options and Variables Deprecated in NDB 7.3

The following `mysqld` system variables, status variables, and options have been deprecated in NDB 7.3.

- `create_old_temporals`: Use pre-5.6.4 storage format for temporal types when creating tables. Intended for use in replication and upgrades/downgrades between NDB 7.2 and NDB 7.3/7.4. Deprecated as of NDB 7.3.10.

MySQL Server Options and Variables Removed in NDB 7.3

The following `mysqld` system variables, status variables, and options have been removed in NDB 7.3.

- `ndb_index_stat_cache_entries`: Granularity of statistics by determining number of starting and ending keys. Removed in NDB 7.3.5.
- `ndb_index_stat_update_freq`: How often to query data nodes rather than statistics cache. Removed in NDB 7.3.5.

18.1.5.2 Options, Variables, and Parameters Added, Deprecated or Removed in NDB 7.4

- Node Configuration Parameters Introduced in NDB 7.4
- Node Configuration Parameters Deprecated in NDB 7.4
- Node Configuration Parameters Removed in NDB 7.4
- MySQL Server Options and Variables Introduced in NDB 7.4
- MySQL Server Options and Variables Deprecated in NDB 7.4
- MySQL Server Options and Variables Removed in NDB 7.4

The next few sections contain information about `NDB` node configuration parameters and NDB-specific `mysqld` options and variables that have been added to, deprecated in, or removed from NDB 7.4.

Node Configuration Parameters Introduced in NDB 7.4

The following node configuration parameters have been added in NDB 7.4.

- `ApiVerbose`: Enable NDB API debugging; for NDB development. Added in NDB 7.4.12.
- `BackupDiskWriteSpeedPct`: Sets percentage of data node's allocated maximum write speed (MaxDiskWriteSpeed) to reserve for LCPs when starting backup. Added in NDB 7.4.8.
- `ConnectBackoffMaxTime`: Specifies longest time in milliseconds (~100ms resolution) to allow between connection attempts to any given data node by this API node. Excludes time elapsed while connection attempts are ongoing, which in worst case can take several seconds. Disable by setting to 0. If no data nodes are currently connected to this API node, StartConnectBackoffMaxTime is used instead. Added in NDB 7.4.2.
- `DiskPageBufferEntries`: Memory to allocate in DiskPageBufferMemory; very large disk transactions may require increasing this value. Added in NDB 7.4.3.
- `MaxDiskWriteSpeed`: Maximum number of bytes per second that can be written by LCP and backup when no restarts are ongoing. Added in NDB 7.4.1.
- `MaxDiskWriteSpeedOtherNodeRestart`: Maximum number of bytes per second that can be written by LCP and backup when another node is restarting. Added in NDB 7.4.1.

- `MaxDiskWriteSpeedOwnRestart`: Maximum number of bytes per second that can be written by LCP and backup when this node is restarting. Added in NDB 7.4.1.
- `MaxParallelCopyInstances`: Number of parallel copies during node restarts. Default is 0, which uses number of LDMs on both nodes, to maximum of 16. Added in NDB 7.4.3.
- `MinDiskWriteSpeed`: Minimum number of bytes per second that can be written by LCP and backup. Added in NDB 7.4.1.
- `SchedulerResponsiveness`: Set NDB scheduler response optimization 0-10; higher values provide better response time but lower throughput. Added in NDB 7.4.9.
- `StartConnectBackoffMaxTime`: Same as ConnectBackoffMaxTime except that this parameter is used in its place if no data nodes are connected to this API node. Added in NDB 7.4.2.
- `TimeBetweenGlobalCheckpointsTimeout`: Minimum timeout for group commit of transactions to disk. Added in NDB 7.4.5.

Node Configuration Parameters Deprecated in NDB 7.4

The following node configuration parameters have been deprecated in NDB 7.4.

- `BackupMemory`: Total memory allocated for backups per node (in bytes). Deprecated as of NDB 7.4.8.
- `DiskCheckpointSpeed`: Bytes allowed to be written by checkpoint, per second. Deprecated as of NDB 7.4.1.
- `DiskCheckpointSpeedInRestart`: Bytes allowed to be written by checkpoint during restart, per second. Deprecated as of NDB 7.4.1.

Node Configuration Parameters Removed in NDB 7.4

No node configuration parameters have been removed from NDB 7.4.

MySQL Server Options and Variables Introduced in NDB 7.4

The following `mysqld` system variables, status variables, and options have been added in NDB 7.4.

- `Ndb_conflict_fn_epoch2`: Number of rows that have been found in conflict by NDB$EPOCH2() conflict detection function. Added in NDB 7.4.2.
- `Ndb_conflict_fn_epoch2_trans`: Number of rows that have been found in conflict by NDB $EPOCH2_TRANS() conflict detection function. Added in NDB 7.4.2.
- `Ndb_conflict_fn_max_del_win`: Number of times that conflict resolution based on outcome of NDB$MAX_DELETE_WIN() has been applied. Added in NDB 7.4.1.
- `Ndb_conflict_last_conflict_epoch`: Most recent NDB epoch on this replica in which some conflict was detected. Added in NDB 7.4.2.
- `Ndb_conflict_last_stable_epoch`: Number of rows found to be in conflict by transactional conflict function. Added in NDB 7.4.2.
- `Ndb_conflict_reflected_op_discard_count`: Number of reflected operations that were not applied due error during execution. Added in NDB 7.4.2.
- `Ndb_conflict_reflected_op_prepare_count`: Number of reflected operations received that have been prepared for execution. Added in NDB 7.4.2.
- `Ndb_conflict_refresh_op_count`: Number of refresh operations that have been prepared. Added in NDB 7.4.2.

- `Ndb_conflict_trans_row_conflict_count`: Number of rows found in conflict by transactional conflict function. Includes any rows included in or dependent on conflicting transactions. Added in NDB 7.4.2.
- `Ndb_epoch_delete_delete_count`: Number of delete-delete conflicts detected (delete operation is applied, but row does not exist). Added in NDB 7.4.2.
- `Ndb_last_commit_epoch_server`: Epoch most recently committed by NDB. Added in NDB 7.4.1.
- `Ndb_last_commit_epoch_session`: Epoch most recently committed by this NDB client. Added in NDB 7.4.1.
- `Ndb_slave_max_replicated_epoch`: Most recently committed NDB epoch on this replica. When this value is greater than or equal to Ndb_conflict_last_conflict_epoch, no conflicts have yet been detected. Added in NDB 7.4.1.
- `create_old_temporals`: Use pre-5.6.4 storage format for temporal types when creating tables. Intended for use in replication and upgrades/downgrades between NDB 7.2 and NDB 7.3/7.4. Added in NDB 7.4.7.
- `ndb-log-exclusive-reads`: Log primary key reads with exclusive locks; allow conflict resolution based on read conflicts. Added in NDB 7.4.1.
- `ndb-log-fail-terminate`: Terminate mysqld process if complete logging of all found row events is not possible. Added in NDB 7.4.28.
- `ndb-log-update-minimal`: Log updates in minimal format. Added in NDB 7.4.16.
- `ndb_clear_apply_status`: Causes RESET SLAVE/RESET REPLICA to clear all rows from ndb_apply_status table; ON by default. Added in NDB 7.4.9.
- `ndb_eventbuffer_free_percent`: Percentage of free memory that should be available in event buffer before resumption of buffering, after reaching limit set by ndb_eventbuffer_max_alloc. Added in NDB 7.4.3.
- `ndb_log_exclusive_reads`: Log primary key reads with exclusive locks; allow conflict resolution based on read conflicts. Added in NDB 7.4.1.
- `ndb_slave_conflict_role`: Role for replica to play in conflict detection and resolution. Value is one of PRIMARY, SECONDARY, PASS, or NONE (default). Can be changed only when replication SQL thread is stopped. See documentation for further information. Added in NDB 7.4.1.

MySQL Server Options and Variables Deprecated in NDB 7.4

The following `mysqld` system variables, status variables, and options have been deprecated in NDB 7.4.

- `create_old_temporals`: Use pre-5.6.4 storage format for temporal types when creating tables. Intended for use in replication and upgrades/downgrades between NDB 7.2 and NDB 7.3/7.4. Deprecated as of NDB 7.4.7.

MySQL Server Options and Variables Removed in NDB 7.4

No system variables, status variables, or options have been removed from NDB 7.4.

18.1.6 MySQL Server Using InnoDB Compared with NDB Cluster

MySQL Server offers a number of choices in storage engines. Since both `NDB` and `InnoDB` can serve as transactional MySQL storage engines, users of MySQL Server sometimes become interested in NDB Cluster. They see `NDB` as a possible alternative or upgrade to the default `InnoDB` storage engine in MySQL 5.6. While `NDB` and `InnoDB` share common characteristics, there are differences in architecture and implementation, so that some existing MySQL Server applications and usage scenarios can be a good fit for NDB Cluster, but not all of them.

In this section, we discuss and compare some characteristics of the NDB storage engine used by NDB Cluster 7.3 and 7.4 with InnoDB used in MySQL 5.6. The next few sections provide a technical comparison. In many instances, decisions about when and where to use NDB Cluster must be made on a case-by-case basis, taking all factors into consideration. While it is beyond the scope of this documentation to provide specifics for every conceivable usage scenario, we also attempt to offer some very general guidance on the relative suitability of some common types of applications for NDB as opposed to InnoDB back ends.

NDB Cluster 7.3 and 7.4 use a mysqld based on MySQL 5.6, including support for InnoDB 1.1. While it is possible to use InnoDB tables with NDB Cluster, such tables are not clustered. It is also not possible to use programs or libraries from an NDB Cluster 7.3 or 7.4 distribution with MySQL Server 5.6, or the reverse.

While it is also true that some types of common business applications can be run either on NDB Cluster or on MySQL Server (most likely using the InnoDB storage engine), there are some important architectural and implementation differences. Section 18.1.6.1, “Differences Between the NDB and InnoDB Storage Engines”, provides a summary of the these differences. Due to the differences, some usage scenarios are clearly more suitable for one engine or the other; see Section 18.1.6.2, “NDB and InnoDB Workloads”. This in turn has an impact on the types of applications that better suited for use with NDB or InnoDB. See Section 18.1.6.3, “NDB and InnoDB Feature Usage Summary”, for a comparison of the relative suitability of each for use in common types of database applications.

For information about the relative characteristics of the NDB and MEMORY storage engines, see When to Use MEMORY or NDB Cluster.

See Chapter 15, *Alternative Storage Engines*, for additional information about MySQL storage engines.

18.1.6.1 Differences Between the NDB and InnoDB Storage Engines

The NDB storage engine is implemented using a distributed, shared-nothing architecture, which causes it to behave differently from InnoDB in a number of ways. For those unaccustomed to working with NDB, unexpected behaviors can arise due to its distributed nature with regard to transactions, foreign keys, table limits, and other characteristics. These are shown in the following table:

Table 18.1 Feature differences between InnoDB and NDB storage engines.

Feature	InnoDB 1.1	NDB 7.3, NDB 7.4
MySQL Server Version	5.6	5.6
InnoDB Version	InnoDB 5.6.51	InnoDB 5.6.51
NDB Cluster Version	N/A	NDB 7.3.32, 7.4.31
Storage Limits	64TB	3TB (Practical upper limit based on 48 data nodes with 64GB RAM each; can be increased with disk-based data and BLOBs)
Foreign Keys	Yes	Yes
Transactions	All standard types	READ COMMITTED
MVCC	Yes	No
Data Compression	Yes	No (NDB checkpoint and backup files can be compressed)
Large Row Support (> 14K)	Supported for VARBINARY, VARCHAR, BLOB, and TEXT columns	Supported for BLOB and TEXT columns only (Using these types to store very large amounts of data can lower NDB performance)

Feature	InnoDB 1.1	NDB 7.3, NDB 7.4
Replication Support	Asynchronous and semisynchronous replication using MySQL Replication	Automatic synchronous replication within an NDB Cluster; asynchronous replication between NDB Clusters, using MySQL Replication (Semisynchronous replication is not supported)
Scaleout for Read Operations	Yes (MySQL Replication)	Yes (Automatic partitioning in NDB Cluster; NDB Cluster Replication)
Scaleout for Write Operations	Requires application-level partitioning (sharding)	Yes (Automatic partitioning in NDB Cluster is transparent to applications)
High Availability (HA)	Requires additional software	Yes (Designed for 99.999% uptime)
Node Failure Recovery and Failover	Requires additional software	Automatic (Key element in NDB architecture)
Time for Node Failure Recovery	30 seconds or longer	Typically < 1 second
Real-Time Performance	No	Yes
In-Memory Tables	No	Yes (Some data can optionally be stored on disk; both in-memory and disk data storage are durable)
NoSQL Access to Storage Engine	Yes	Yes (Multiple APIs, including Memcached, Node.js/JavaScript, Java, JPA, C++, and HTTP/REST)
Concurrent and Parallel Writes	Not supported	Up to 48 writers, optimized for concurrent writes
Conflict Detection and Resolution (Multiple Replication sources)	No	Yes
Hash Indexes	No	Yes
Online Addition of Nodes	Read-only replicas using MySQL Replication	Yes (all node types)
Online Upgrades	No	Yes
Online Schema Modifications	Yes, as part of MySQL 5.6	Yes

18.1.6.2 NDB and InnoDB Workloads

NDB Cluster has a range of unique attributes that make it ideal to serve applications requiring high availability, fast failover, high throughput, and low latency. Due to its distributed architecture and multi-node implementation, NDB Cluster also has specific constraints that may keep some workloads from performing well. A number of major differences in behavior between the NDB and InnoDB storage engines with regard to some common types of database-driven application workloads are shown in the following table::

Table 18.2 Differences between the InnoDB and NDB storage engines, common types of database-driven application workloads

Workload	InnoDB	NDB Cluster (NDB)
High-Volume OLTP Applications	Yes	Yes

Workload	InnoDB	NDB Cluster (NDB)
DSS Applications (data marts, analytics)	Yes	Limited (Join operations across OLTP datasets not exceeding 3TB in size)
Custom Applications	Yes	Yes
Packaged Applications	Yes	Limited (should be mostly primary key access); NDB 7.3 and 7.4 support foreign keys
In-Network Telecoms Applications (HLR, HSS, SDP)	No	Yes
Session Management and Caching	Yes	Yes
E-Commerce Applications	Yes	Yes
User Profile Management, AAA Protocol	Yes	Yes

18.1.6.3 NDB and InnoDB Feature Usage Summary

When comparing application feature requirements to the capabilities of InnoDB with NDB, some are clearly more compatible with one storage engine than the other.

The following table lists supported application features according to the storage engine to which each feature is typically better suited.

Table 18.3 Supported application features according to the storage engine to which each feature is typically better suited

Preferred application requirements for InnoDB	Preferred application requirements for NDB
• Foreign keys **Note** NDB Cluster 7.4 and 7.4 support foreign keys • Full table scans • Very large databases, rows, or transactions • Transactions other than READ COMMITTED	• Write scaling • 99.999% uptime • Online addition of nodes and online schema operations • Multiple SQL and NoSQL APIs (see NDB Cluster APIs: Overview and Concepts) • Real-time performance • Limited use of BLOB columns • Foreign keys are supported, although their use may have an impact on performance at high throughput

18.1.7 Known Limitations of NDB Cluster

In the sections that follow, we discuss known limitations in current releases of NDB Cluster as compared with the features available when using the MyISAM and InnoDB storage engines. If you check the “Cluster” category in the MySQL bugs database at http://bugs.mysql.com, you can find known bugs in the following categories under “MySQL Server:” in the MySQL bugs database at http://bugs.mysql.com, which we intend to correct in upcoming releases of NDB Cluster:

- NDB Cluster

- Cluster Direct API (NDBAPI)
- Cluster Disk Data
- Cluster Replication
- ClusterJ

This information is intended to be complete with respect to the conditions just set forth. You can report any discrepancies that you encounter to the MySQL bugs database using the instructions given in Section 1.6, “How to Report Bugs or Problems”. If we do not plan to fix the problem in NDB Cluster 7.3 or 7.4, we add it to the list.

See Section 18.1.7.11, “Previous NDB Cluster Issues Resolved in NDB Cluster 7.3” for a list of issues in earlier versions of NDB Cluster that have been resolved in NDB 7.3 or 7.4.

Note

Limitations and other issues specific to NDB Cluster Replication are described in Section 18.6.3, “Known Issues in NDB Cluster Replication”.

18.1.7.1 Noncompliance with SQL Syntax in NDB Cluster

Some SQL statements relating to certain MySQL features produce errors when used with `NDB` tables, as described in the following list:

- **Temporary tables.** Temporary tables are not supported. Trying either to create a temporary table that uses the `NDB` storage engine or to alter an existing temporary table to use `NDB` fails with the error `Table storage engine 'ndbcluster' does not support the create option 'TEMPORARY'`.
- **Indexes and keys in NDB tables.** Keys and indexes on NDB Cluster tables are subject to the following limitations:
  - **Column width.** Attempting to create an index on an `NDB` table column whose width is greater than 3072 bytes succeeds, but only the first 3072 bytes are actually used for the index. In such cases, a warning `Specified key was too long; max key length is 3072 bytes` is issued, and a `SHOW CREATE TABLE` statement shows the length of the index as 3072.
  - **TEXT and BLOB columns.** You cannot create indexes on `NDB` table columns that use any of the `TEXT` or `BLOB` data types.
  - **FULLTEXT indexes.** The `NDB` storage engine does not support `FULLTEXT` indexes, which are possible for `MyISAM` and (MySQL 5.6.4 and later) `InnoDB` tables only.

    However, you can create indexes on `VARCHAR` columns of `NDB` tables.
  - **USING HASH keys and NULL.** Using nullable columns in unique keys and primary keys means that queries using these columns are handled as full table scans. To work around this issue, make the column `NOT NULL`, or re-create the index without the `USING HASH` option.
  - **Prefixes.** There are no prefix indexes; only entire columns can be indexed. (The size of an `NDB` column index is always the same as the width of the column in bytes, up to and including 3072 bytes, as described earlier in this section. Also see Section 18.1.7.6, “Unsupported or Missing Features in NDB Cluster”, for additional information.)
  - **BIT columns.** A `BIT` column cannot be a primary key, unique key, or index, nor can it be part of a composite primary key, unique key, or index.
  - **AUTO_INCREMENT columns.** Like other MySQL storage engines, the `NDB` storage engine can handle a maximum of one `AUTO_INCREMENT` column per table, and this column must be indexed. However, in the case of an NDB table with no explicit primary key, an `AUTO_INCREMENT`

column is automatically defined and used as a “hidden” primary key. For this reason, you cannot create an `NDB` table having an `AUTO_INCREMENT` column and no explicit primary key.

- **Restrictions on foreign keys.** Support for foreign key constraints in NDB Cluster 7.3 and 7.4 is comparable to that provided by `InnoDB`, subject to the following restrictions:

  - Every column referenced as a foreign key requires an explicit unique key, if it is not the table's primary key.

  - `ON UPDATE CASCADE` is not supported when the reference is to the parent table's primary key.

    This is because an update of a primary key is implemented as a delete of the old row (containing the old primary key) plus an insert of the new row (with a new primary key). This is not visible to the `NDB` kernel, which views these two rows as being the same, and thus has no way of knowing that this update should be cascaded.

  - As of NDB 7.3.25 and NDB 7.4.24: `ON DELETE CASCADE` is not supported where the child table contains one or more columns of any of the `TEXT` or `BLOB` types. (Bug #89511, Bug #27484882)

  - `SET DEFAULT` is not supported. (Also not supported by `InnoDB`.)

  - The `NO ACTION` keywords are accepted but treated as `RESTRICT`. (Also the same as with `InnoDB`.)

  - Prior to NDB 7.3.5, when creating a table with foreign key referencing an index in another table, it sometimes appeared possible to create the foreign key even if the order of the columns in the indexes did not match, due to the fact that an appropriate error was not always returned internally. A partial fix for this issue in NDB 7.3.5 improves the error used internally to work in most cases; however, it is still possible for this situation to occur in the event that the parent index is a unique index. (Bug #18094360)

  - In NDB 7.3, when adding or dropping a foreign key using `ALTER TABLE`, the parent table's metadata is not updated, which makes it possible subsequently to execute `ALTER TABLE` statements on the parent that should be invalid. This issue also affects NDB 7.4 releases prior to 7.4.15. To work around this issue, execute `SHOW CREATE TABLE` on the parent table immediately after adding or dropping the foreign key; this forces the parent's metadata to be reloaded.

    This issue is fixed in NDB 7.4.15 and later. (Bug #82989, Bug #24666177)

  For more information, see Section 13.1.17.5, “FOREIGN KEY Constraints”, and Section 1.7.3.2, “FOREIGN KEY Constraints”.

- **NDB Cluster and geometry data types.**
  Geometry data types (`WKT` and `WKB`) are supported for `NDB` tables. However, spatial indexes are not supported.

- **Character sets and binary log files.** Currently, the `ndb_apply_status` and `ndb_binlog_index` tables are created using the `latin1` (ASCII) character set. Because names of binary logs are recorded in this table, binary log files named using non-Latin characters are not referenced correctly in these tables. This is a known issue, which we are working to fix. (Bug #50226)

  To work around this problem, use only Latin-1 characters when naming binary log files or setting any the `--basedir`, `--log-bin`, or `--log-bin-index` options.

- **Creating NDB tables with user-defined partitioning.** Support for user-defined partitioning in NDB Cluster is restricted to [`LINEAR`] `KEY` partitioning. Using any other partitioning type with `ENGINE=NDB` or `ENGINE=NDBCLUSTER` in a `CREATE TABLE` statement results in an error.

  It is possible to override this restriction, but doing so is not supported for use in production settings. For details, see User-defined partitioning and the NDB storage engine (NDB Cluster).

Default partitioning scheme. All NDB Cluster tables are by default partitioned by `KEY` using the table's primary key as the partitioning key. If no primary key is explicitly set for the table, the "hidden" primary key automatically created by the `NDB` storage engine is used instead. For additional discussion of these and related issues, see Section 19.2.5, "KEY Partitioning".

`CREATE TABLE` and `ALTER TABLE` statements that would cause a user-partitioned `NDBCLUSTER` table not to meet either or both of the following two requirements are not permitted, and fail with an error:

1. The table must have an explicit primary key.

2. All columns listed in the table's partitioning expression must be part of the primary key.

Exception. If a user-partitioned `NDBCLUSTER` table is created using an empty column-list (that is, using `PARTITION BY [LINEAR] KEY()`), then no explicit primary key is required.

Maximum number of partitions for NDBCLUSTER tables. The maximum number of partitions that can defined for a `NDBCLUSTER` table when employing user-defined partitioning is 8 per node group. (See Section 18.1.2, "NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions", for more information about NDB Cluster node groups.

DROP PARTITION not supported. It is not possible to drop partitions from `NDB` tables using `ALTER TABLE ... DROP PARTITION`. The other partitioning extensions to `ALTER TABLE`—`ADD PARTITION`, `REORGANIZE PARTITION`, and `COALESCE PARTITION`—are supported for NDB tables, but use copying and so are not optimized. See Section 19.3.1, "Management of RANGE and LIST Partitions" and Section 13.1.7, "ALTER TABLE Statement".

- **Row-based replication.**
  When using row-based replication with NDB Cluster, binary logging cannot be disabled. That is, the `NDB` storage engine ignores the value of `sql_log_bin`. (Bug #16680)

18.1.7.2 Limits and Differences of NDB Cluster from Standard MySQL Limits

In this section, we list limits found in NDB Cluster that either differ from limits found in, or that are not found in, standard MySQL.

Memory usage and recovery. Memory consumed when data is inserted into an `NDB` table is not automatically recovered when deleted, as it is with other storage engines. Instead, the following rules hold true:

- A `DELETE` statement on an `NDB` table makes the memory formerly used by the deleted rows available for re-use by inserts on the same table only. However, this memory can be made available for general re-use by performing `OPTIMIZE TABLE`.

  A rolling restart of the cluster also frees any memory used by deleted rows. See Section 18.5.5, "Performing a Rolling Restart of an NDB Cluster".

- A `DROP TABLE` or `TRUNCATE TABLE` operation on an `NDB` table frees the memory that was used by this table for re-use by any `NDB` table, either by the same table or by another `NDB` table.

  Note

  Recall that `TRUNCATE TABLE` drops and re-creates the table. See Section 13.1.33, "TRUNCATE TABLE Statement".

- **Limits imposed by the cluster's configuration.**
  A number of hard limits exist which are configurable, but available main memory in the cluster sets limits. See the complete list of configuration parameters in Section 18.3.3, "NDB Cluster Configuration Files". Most configuration parameters can be upgraded online. These hard limits include:

- Database memory size and index memory size (`DataMemory` and `IndexMemory`, respectively).

  `DataMemory` is allocated as 32KB pages. As each `DataMemory` page is used, it is assigned to a specific table; once allocated, this memory cannot be freed except by dropping the table.

  See Section 18.3.3.6, “Defining NDB Cluster Data Nodes”, for more information.

- The maximum number of operations that can be performed per transaction is set using the configuration parameters `MaxNoOfConcurrentOperations` and `MaxNoOfLocalOperations`.

  Note

  Bulk loading, `TRUNCATE TABLE`, and `ALTER TABLE` are handled as special cases by running multiple transactions, and so are not subject to this limitation.

- Different limits related to tables and indexes. For example, the maximum number of ordered indexes in the cluster is determined by `MaxNoOfOrderedIndexes`, and the maximum number of ordered indexes per table is 16.

- **Node and data object maximums.** The following limits apply to numbers of cluster nodes and metadata objects:

  - The maximum number of data nodes is 48.

    A data node must have a node ID in the range of 1 to 48, inclusive. (Management and API nodes may use node IDs in the range 1 to 255, inclusive.)

  - The total maximum number of nodes in an NDB Cluster is 255. This number includes all SQL nodes (MySQL Servers), API nodes (applications accessing the cluster other than MySQL servers), data nodes, and management servers.

  - The maximum number of metadata objects in current versions of NDB Cluster is 20320. This limit is hard-coded.

  See Section 18.1.7.11, “Previous NDB Cluster Issues Resolved in NDB Cluster 7.3”, for more information.

18.1.7.3 Limits Relating to Transaction Handling in NDB Cluster

A number of limitations exist in NDB Cluster with regard to the handling of transactions. These include the following:

- **Transaction isolation level.** The `NDBCLUSTER` storage engine supports only the `READ COMMITTED` transaction isolation level. (`InnoDB`, for example, supports `READ COMMITTED`, `READ UNCOMMITTED`, `REPEATABLE READ`, and `SERIALIZABLE`.) You should keep in mind that `NDB` implements `READ COMMITTED` on a per-row basis; when a read request arrives at the data node storing the row, what is returned is the last committed version of the row at that time.

  Uncommitted data is never returned, but when a transaction modifying a number of rows commits concurrently with a transaction reading the same rows, the transaction performing the read can observe “before” values, “after” values, or both, for different rows among these, due to the fact that a given row read request can be processed either before or after the commit of the other transaction.

  To ensure that a given transaction reads only before or after values, you can impose row locks using `SELECT ... LOCK IN SHARE MODE`. In such cases, the lock is held until the owning transaction is committed. Using row locks can also cause the following issues:

  - Increased frequency of lock wait timeout errors, and reduced concurrency

- Increased transaction processing overhead due to reads requiring a commit phase
- Possibility of exhausting the available number of concurrent locks, which is limited by `MaxNoOfConcurrentOperations`

`NDB` uses `READ COMMITTED` for all reads unless a modifier such as `LOCK IN SHARE MODE` or `FOR UPDATE` is used. `LOCK IN SHARE MODE` causes shared row locks to be used; `FOR UPDATE` causes exclusive row locks to be used. Unique key reads have their locks upgraded automatically by `NDB` to ensure a self-consistent read; `BLOB` reads also employ extra locking for consistency.

See Section 18.5.8.4, “NDB Cluster Backup Troubleshooting”, for information on how NDB Cluster's implementation of transaction isolation level can affect backup and restoration of `NDB` databases.

- **Unique key lookups and transaction isolation.** Unique indexes are implemented in `NDB` using a hidden index table which is maintained internally. When a user-created `NDB` table is accessed using a unique index, the hidden index table is first read to find the primary key that is then used to read the user-created table. To avoid modification of the index during this double-read operation, the row found in the hidden index table is locked. When a row referenced by a unique index in the user-created `NDB` table is updated, the hidden index table is subject to an exclusive lock by the transaction in which the update is performed. This means that any read operation on the same (user-created) `NDB` table must wait for the update to complete. This is true even when the transaction level of the read operation is `READ COMMITTED`.

  One workaround which can be used to bypass potentially blocking reads is to force the SQL node to ignore the unique index when performing the read. This can be done by using the `IGNORE INDEX` index hint as part of the `SELECT` statement reading the table (see Section 8.9.3, “Index Hints”). Because the MySQL server creates a shadowing ordered index for every unique index created in `NDB`, this lets the ordered index be read instead, and avoids unique index access locking. The resulting read is as consistent as a committed read by primary key, returning the last committed value at the time the row is read.

  Reading via an ordered index makes less efficient use of resources in the cluster, and may have higher latency.

  It is also possible to avoid using the unique index for access by querying for ranges rather than for unique values.

- **Transactions and BLOB or TEXT columns.** `NDBCLUSTER` stores only part of a column value that uses any of MySQL's `BLOB` or `TEXT` data types in the table visible to MySQL; the remainder of the `BLOB` or `TEXT` is stored in a separate internal table that is not accessible to MySQL. This gives rise to two related issues of which you should be aware whenever executing `SELECT` statements on tables that contain columns of these types:

  1. For any `SELECT` from an NDB Cluster table: If the `SELECT` includes a `BLOB` or `TEXT` column, the `READ COMMITTED` transaction isolation level is converted to a read with read lock. This is done to guarantee consistency.

  2. For any `SELECT` which uses a unique key lookup to retrieve any columns that use any of the `BLOB` or `TEXT` data types and that is executed within a transaction, a shared read lock is held on the table for the duration of the transaction—that is, until the transaction is either committed or aborted.

     This issue does not occur for queries that use index or table scans, even against `NDB` tables having `BLOB` or `TEXT` columns.

     For example, consider the table `t` defined by the following `CREATE TABLE` statement:

     ```
     CREATE TABLE t (
         a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
         b INT NOT NULL,
     ```

```
    c INT NOT NULL,
    d TEXT,
    INDEX i(b),
    UNIQUE KEY u(c)
) ENGINE = NDB,
```

The following query on `t` causes a shared read lock, because it uses a unique key lookup:

```
SELECT * FROM t WHERE c = 1;
```

However, none of the four queries shown here causes a shared read lock:

```
SELECT * FROM t WHERE b = 1;

SELECT * FROM t WHERE d = '1';

SELECT * FROM t;

SELECT b,c WHERE a = 1;
```

This is because, of these four queries, the first uses an index scan, the second and third use table scans, and the fourth, while using a primary key lookup, does not retrieve the value of any `BLOB` or `TEXT` columns.

You can help minimize issues with shared read locks by avoiding queries that use unique key lookups that retrieve `BLOB` or `TEXT` columns, or, in cases where such queries are not avoidable, by committing transactions as soon as possible afterward.

- **Rollbacks.** There are no partial transactions, and no partial rollbacks of transactions. A duplicate key or similar error causes the entire transaction to be rolled back.

  This behavior differs from that of other transactional storage engines such as `InnoDB` that may roll back individual statements.

- **Transactions and memory usage.**
  As noted elsewhere in this chapter, NDB Cluster does not handle large transactions well; it is better to perform a number of small transactions with a few operations each than to attempt a single large transaction containing a great many operations. Among other considerations, large transactions require very large amounts of memory. Because of this, the transactional behavior of a number of MySQL statements is affected as described in the following list:

  - `TRUNCATE TABLE` is not transactional when used on `NDB` tables. If a `TRUNCATE TABLE` fails to empty the table, then it must be re-run until it is successful.

  - `DELETE FROM` (even with no `WHERE` clause) *is* transactional. For tables containing a great many rows, you may find that performance is improved by using several `DELETE FROM ... LIMIT ...` statements to “chunk” the delete operation. If your objective is to empty the table, then you may wish to use `TRUNCATE TABLE` instead.

  - **LOAD DATA statements.** `LOAD DATA` is not transactional when used on `NDB` tables.

    Important

    When executing a `LOAD DATA` statement, the `NDB` engine performs commits at irregular intervals that enable better utilization of the communication network. It is not possible to know ahead of time when such commits take place.

  - **ALTER TABLE and transactions.** When copying an `NDB` table as part of an `ALTER TABLE`, the creation of the copy is nontransactional. (In any case, this operation is rolled back when the copy is deleted.)

- **Transactions and the COUNT() function.** When using NDB Cluster Replication, it is not possible to guarantee the transactional consistency of the `COUNT()` function on the replica. In other words, when performing on the source a series of statements (`INSERT`, `DELETE`, or both) that changes the number of rows in a table within a single transaction, executing `SELECT COUNT(*) FROM table` queries on the replica may yield intermediate results. This is due to the fact that `SELECT COUNT(...)` may perform dirty reads, and is not a bug in the `NDB` storage engine. (See Bug #31321 for more information.)

18.1.7.4 NDB Cluster Error Handling

Starting, stopping, or restarting a node may give rise to temporary errors causing some transactions to fail. These include the following cases:

- **Temporary errors.** When first starting a node, it is possible that you may see Error 1204 `Temporary failure, distribution changed` and similar temporary errors.
- **Errors due to node failure.** The stopping or failure of any data node can result in a number of different node failure errors. (However, there should be no aborted transactions when performing a planned shutdown of the cluster.)

In either of these cases, any errors that are generated must be handled within the application. This should be done by retrying the transaction.

See also Section 18.1.7.2, “Limits and Differences of NDB Cluster from Standard MySQL Limits”.

18.1.7.5 Limits Associated with Database Objects in NDB Cluster

Some database objects such as tables and indexes have different limitations when using the `NDBCLUSTER` storage engine:

- **Database and table names.** When using the `NDB` storage engine, the maximum allowed length both for database names and for table names is 63 characters.

  In NDB 7.3.8 and later, a statement using a database name or table name longer than this limit fails with an appropriate error. (Bug #19550973)

- **Number of database objects.** The maximum number of *all* `NDB` database objects in a single NDB Cluster—including databases, tables, and indexes—is limited to 20320.
- **Attributes per table.** The maximum number of attributes (that is, columns and indexes) that can belong to a given table is 512.
- **Attributes per key.** The maximum number of attributes per key is 32.
- **Row size.** The maximum permitted size of any one row is 14000 bytes.

  Each `BLOB` or `TEXT` column contributes 256 + 8 = 264 bytes to this total; see String Type Storage Requirements, for more information relating to these types.

  In addition, the maximum offset for a fixed-width column of an `NDB` table is 8188 bytes; attempting to create a table that violates this limitation fails with NDB error 851 `Maximum offset for fixed-size columns exceeded`. For memory-based columns, you can work around this limitation by using a variable-width column type such as `VARCHAR` or defining the column as `COLUMN_FORMAT=DYNAMIC`; this does not work with columns stored on disk. For disk-based columns, you may be able to do so by reordering one or more of the table's disk-based columns such that the combined width of all but the disk-based column defined last in the `CREATE TABLE` statement used to create the table does not exceed 8188 bytes, less any possible rounding performed for some data types such as `CHAR` or `VARCHAR`; otherwise it is necessary to use memory-based storage for one or more of the offending column or columns instead.

- **BIT column storage per table.** The maximum combined width for all `BIT` columns used in a given `NDB` table is 4096.

- **FIXED column storage.** NDB Cluster supports a maximum of 16 GB per fragment of data in `FIXED` columns.

18.1.7.6 Unsupported or Missing Features in NDB Cluster

A number of features supported by other storage engines are not supported for `NDB` tables. Trying to use any of these features in NDB Cluster does not cause errors in or of itself; however, errors may occur in applications that expects the features to be supported or enforced. Statements referencing such features, even if effectively ignored by `NDB`, must be syntactically and otherwise valid.

- **Index prefixes.** Prefixes on indexes are not supported for `NDB` tables. If a prefix is used as part of an index specification in a statement such as `CREATE TABLE`, `ALTER TABLE`, or `CREATE INDEX`, the prefix is not created by `NDB`.

  A statement containing an index prefix, and creating or modifying an `NDB` table, must still be syntactically valid. For example, the following statement always fails with Error 1089 `Incorrect prefix key; the used key part isn't a string, the used length is longer than the key part, or the storage engine doesn't support unique prefix keys`, regardless of storage engine:

  ```
  CREATE TABLE t1 (
      c1 INT NOT NULL,
      c2 VARCHAR(100),
      INDEX i1 (c2(500))
  );
  ```

  This happens on account of the SQL syntax rule that no index may have a prefix larger than itself.

- **Savepoints and rollbacks.** Savepoints and rollbacks to savepoints are ignored as in `MyISAM`.

- **Durability of commits.** There are no durable commits on disk. Commits are replicated, but there is no guarantee that logs are flushed to disk on commit.

- **Replication.** Statement-based replication is not supported. Use `--binlog-format=ROW` (or `--binlog-format=MIXED`) when setting up cluster replication. See Section 18.6, “NDB Cluster Replication”, for more information.

  Replication using global transaction identifiers (GTIDs) is not compatible with NDB Cluster, and is not supported in NDB Cluster 7.3 or NDB Cluster 7.4. Do not enable GTIDs when using the `NDB` storage engine, as this is very likely to cause problems up to and including failure of NDB Cluster Replication.

  Semisynchronous replication is not supported in NDB Cluster.

  When replicating between clusters, it is possible to use IPv6 addresses between SQL nodes in different clusters, but all connections within a given cluster must use IPv4 addressing. For more information, see NDB Cluster Replication and IPv6.

Note

See Section 18.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”, for more information relating to limitations on transaction handling in `NDB`.

18.1.7.7 Limitations Relating to Performance in NDB Cluster

The following performance issues are specific to or especially pronounced in NDB Cluster:

- **Range scans.** There are query performance issues due to sequential access to the `NDB` storage engine; it is also relatively more expensive to do many range scans than it is with either `MyISAM` or `InnoDB`.

- **Reliability of Records in range.** The `Records in range` statistic is available but is not completely tested or officially supported. This may result in nonoptimal query plans in some cases.

If necessary, you can employ `USE INDEX` or `FORCE INDEX` to alter the execution plan. See Section 8.9.3, “Index Hints”, for more information on how to do this.

- **Unique hash indexes.** Unique hash indexes created with `USING HASH` cannot be used for accessing a table if `NULL` is given as part of the key.

18.1.7.8 Issues Exclusive to NDB Cluster

The following are limitations specific to the `NDB` storage engine:

- **Machine architecture.** All machines used in the cluster must have the same architecture. That is, all machines hosting nodes must be either big-endian or little-endian, and you cannot use a mixture of both. For example, you cannot have a management node running on a PowerPC which directs a data node that is running on an x86 machine. This restriction does not apply to machines simply running `mysql` or other clients that may be accessing the cluster's SQL nodes.

- **Binary logging.**
  NDB Cluster has the following limitations or restrictions with regard to binary logging:

  - `sql_log_bin` has no effect on data operations; however, it is supported for schema operations.
  - NDB Cluster cannot produce a binary log for tables having `BLOB` columns but no primary key.
  - Only the following schema operations are logged in a cluster binary log which is *not* on the `mysqld` executing the statement:
    - `CREATE TABLE`
    - `ALTER TABLE`
    - `DROP TABLE`
    - `CREATE DATABASE` / `CREATE SCHEMA`
    - `DROP DATABASE` / `DROP SCHEMA`
    - `CREATE TABLESPACE`
    - `ALTER TABLESPACE`
    - `DROP TABLESPACE`
    - `CREATE LOGFILE GROUP`
    - `ALTER LOGFILE GROUP`
    - `DROP LOGFILE GROUP`

- **Schema operations.** Schema operations (DDL statements) are rejected while any data node restarts. Schema operations are also not supported while performing an online upgrade or downgrade.

- **Number of fragment replicas.** The number of fragment replicas, as determined by the `NoOfReplicas` data node configuration parameter, is the number of copies of all data stored by NDB Cluster. Setting this parameter to 1 means there is only a single copy; in this case, no redundancy is provided, and the loss of a data node entails loss of data. To guarantee redundancy, and thus preservation of data even if a data node fails, set this parameter to 2, which is the default and recommended value in production.

  Setting `NoOfReplicas` to a value greater than 2 is possible (to a maximum of 4) but unnecessary to guard against loss of data. In addition, *values greater than 2 for this parameter are not supported in production*.

See also Section 18.1.7.10, “Limitations Relating to Multiple NDB Cluster Nodes”.

18.1.7.9 Limitations Relating to NDB Cluster Disk Data Storage

Disk Data object maximums and minimums. Disk data objects are subject to the following maximums and minimums:

- Maximum number of tablespaces: 2^{32} (4294967296)
- Maximum number of data files per tablespace: 2^{16} (65536)
- The minimum and maximum possible sizes of extents for tablespace data files are 32K and 2G, respectively. See Section 13.1.18, “CREATE TABLESPACE Statement”, for more information.

In addition, when working with NDB Disk Data tables, you should be aware of the following issues regarding data files and extents:

- Data files use `DataMemory`. Usage is the same as for in-memory data.
- Data files use file descriptors. It is important to keep in mind that data files are always open, which means the file descriptors are always in use and cannot be re-used for other system tasks.
- Extents require sufficient `DiskPageBufferMemory`; you must reserve enough for this parameter to account for all memory used by all extents (number of extents times size of extents).

Disk Data tables and diskless mode. Use of Disk Data tables is not supported when running the cluster in diskless mode.

18.1.7.10 Limitations Relating to Multiple NDB Cluster Nodes

Multiple SQL nodes.
The following are issues relating to the use of multiple MySQL servers as NDB Cluster SQL nodes, and are specific to the `NDBCLUSTER` storage engine:

- **Stored programs not distributed.** Stored procedures, stored functions, triggers, and scheduled events are all supported by tables using the `NDB` storage engine, but these do *not* propagate automatically between MySQL Servers acting as Cluster SQL nodes, and must be re-created separately on each SQL node. See Stored Programs in NDB Cluster.
- **No distributed table locks.** A `LOCK TABLES` works only for the SQL node on which the lock is issued; no other SQL node in the cluster “sees” this lock. This is also true for a lock issued by any statement that locks tables as part of its operations. (See next item for an example.)
- **ALTER TABLE operations.** `ALTER TABLE` is not fully locking when running multiple MySQL servers (SQL nodes). (As discussed in the previous item, NDB Cluster does not support distributed table locks.)

Multiple management nodes.
When using multiple management servers:

- If any of the management servers are running on the same host, you must give nodes explicit IDs in connection strings because automatic allocation of node IDs does not work across multiple management servers on the same host. This is not required if every management server resides on a different host.
- When a management server starts, it first checks for any other management server in the same NDB Cluster, and upon successful connection to the other management server uses its configuration data. This means that the management server `--reload` and `--initial` startup options are ignored unless the management server is the only one running. It also means that, when performing a rolling restart of an NDB Cluster with multiple management nodes, the management server reads its own configuration file if (and only if) it is the only management server running in this NDB Cluster. See Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”, for more information.

Multiple network addresses. Multiple network addresses per data node are not supported. Use of these is liable to cause problems: In the event of a data node failure, an SQL node waits for confirmation that the data node went down but never receives it because another route to that data node remains open. This can effectively make the cluster inoperable.

Note

It is possible to use multiple network hardware *interfaces* (such as Ethernet cards) for a single data node, but these must be bound to the same address. This also means that it not possible to use more than one `[tcp]` section per connection in the `config.ini` file. See Section 18.3.3.9, “NDB Cluster TCP/IP Connections”, for more information.

18.1.7.11 Previous NDB Cluster Issues Resolved in NDB Cluster 7.3

A number of limitations and related issues that existed in earlier versions of NDB Cluster have been resolved in NDB Cluster 7.3. These are described briefly in the following list:

- **Support for foreign keys.** Foreign key constraints are now supported for `NDB` tables, similar to how these are supported by the `InnoDB` storage engine.

  Note

  Unlike the case with user-partitioned `InnoDB` tables, foreign keys are supported for `NDB` tables that are partitioned by `KEY` or `LINEAR KEY`.

  Section 1.7.3.2, “FOREIGN KEY Constraints”, provides more information about foreign key support in MySQL. For more information about the syntax supported by MySQL for foreign keys, see Section 13.1.17.5, “FOREIGN KEY Constraints”.

18.2 NDB Cluster Installation

This section describes the basics for planning, installing, configuring, and running an NDB Cluster. Whereas the examples in Section 18.3, “Configuration of NDB Cluster” provide more in-depth information on a variety of clustering options and configuration, the result of following the guidelines and procedures outlined here should be a usable NDB Cluster which meets the *minimum* requirements for availability and safeguarding of data.

For information about upgrading or downgrading an NDB Cluster between release versions, see Section 18.2.7, “Upgrading and Downgrading NDB Cluster”.

This section covers hardware and software requirements; networking issues; installation of NDB Cluster; basic configuration issues; starting, stopping, and restarting the cluster; loading of a sample database; and performing queries.

Assumptions. The following sections make a number of assumptions regarding the cluster's physical and network configuration. These assumptions are discussed in the next few paragraphs.

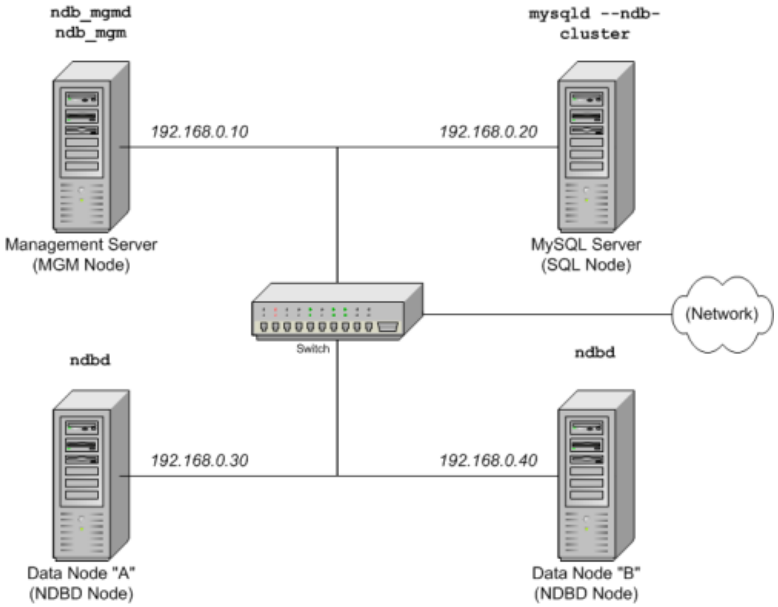
Cluster nodes and host computers. The cluster consists of four nodes, each on a separate host computer, and each with a fixed network address on a typical Ethernet network as shown here:

Table 18.4 Network addresses of nodes in example cluster

Node	IP Address
Management node (`mgmd`)	198.51.100.10
SQL node (`mysqld`)	198.51.100.20
Data node "A" (`ndbd`)	198.51.100.30
Data node "B" (`ndbd`)	198.51.100.40

This setup is also shown in the following diagram:

Figure 18.4 NDB Cluster Multi-Computer Setup



Network addressing. In the interest of simplicity (and reliability), this *How-To* uses only numeric IP addresses. However, if DNS resolution is available on your network, it is possible to use host names in lieu of IP addresses in configuring Cluster. Alternatively, you can use the `hosts` file (typically `/etc/hosts` for Linux and other Unix-like operating systems, `C:\WINDOWS\system32\drivers\etc\hosts` on Windows, or your operating system's equivalent) for providing a means to do host lookup if such is available.

Potential hosts file issues. A common problem when trying to use host names for Cluster nodes arises because of the way in which some operating systems (including some Linux distributions) set up the system's own host name in the `/etc/hosts` during installation. Consider two machines with the host names `ndb1` and `ndb2`, both in the `cluster` network domain. Red Hat Linux (including some derivatives such as CentOS and Fedora) places the following entries in these machines' `/etc/hosts` files:

```
#  ndb1 /etc/hosts:
127.0.0.1   ndb1.cluster ndb1 localhost.localdomain localhost
```

```
#  ndb2 /etc/hosts:
127.0.0.1   ndb2.cluster ndb2 localhost.localdomain localhost
```

SUSE Linux (including OpenSUSE) places these entries in the machines' `/etc/hosts` files:

```
#  ndb1 /etc/hosts:
127.0.0.1       localhost
127.0.0.2       ndb1.cluster ndb1
```

```
#  ndb2 /etc/hosts:
127.0.0.1       localhost
127.0.0.2       ndb2.cluster ndb2
```

In both instances, `ndb1` routes `ndb1.cluster` to a loopback IP address, but gets a public IP address from DNS for `ndb2.cluster`, while `ndb2` routes `ndb2.cluster` to a loopback address and obtains a public address for `ndb1.cluster`. The result is that each data node connects to the management server, but cannot tell when any other data nodes have connected, and so the data nodes appear to hang while starting.

Caution

You cannot mix `localhost` and other host names or IP addresses in `config.ini`. For these reasons, the solution in such cases (other than to use

IP addresses for *all* `config.ini` `HostName` entries) is to remove the fully qualified host names from `/etc/hosts` and use these in `config.ini` for all cluster hosts.

Host computer type. Each host computer in our installation scenario is an Intel-based desktop PC running a supported operating system installed to disk in a standard configuration, and running no unnecessary services. The core operating system with standard TCP/IP networking capabilities should be sufficient. Also for the sake of simplicity, we also assume that the file systems on all hosts are set up identically. In the event that they are not, you should adapt these instructions accordingly.

Network hardware. Standard 100 Mbps or 1 gigabit Ethernet cards are installed on each machine, along with the proper drivers for the cards, and that all four hosts are connected through a standard-issue Ethernet networking appliance such as a switch. (All machines should use network cards with the same throughput. That is, all four machines in the cluster should have 100 Mbps cards *or* all four machines should have 1 Gbps cards.) NDB Cluster works in a 100 Mbps network; however, gigabit Ethernet provides better performance.

Important

NDB Cluster is *not* intended for use in a network for which throughput is less than 100 Mbps or which experiences a high degree of latency. For this reason (among others), attempting to run an NDB Cluster over a wide area network such as the Internet is not likely to be successful, and is not supported in production.

Sample data. We use the `world` database which is available for download from the MySQL website (see https://dev.mysql.com/doc/index-other.html). We assume that each machine has sufficient memory for running the operating system, required NDB Cluster processes, and (on the data nodes) storing the database.

For general information about installing MySQL, see Chapter 2, *Installing and Upgrading MySQL*. For information about installation of NDB Cluster on Linux and other Unix-like operating systems, see Section 18.2.1, “Installation of NDB Cluster on Linux”. For information about installation of NDB Cluster on Windows operating systems, see Section 18.2.2, “Installing NDB Cluster on Windows”.

For general information about NDB Cluster hardware, software, and networking requirements, see Section 18.1.3, “NDB Cluster Hardware, Software, and Networking Requirements”.

18.2.1 Installation of NDB Cluster on Linux

This section covers installation methods for NDB Cluster on Linux and other Unix-like operating systems. While the next few sections refer to a Linux operating system, the instructions and procedures given there should be easily adaptable to other supported Unix-like platforms. For manual installation and setup instructions specific to Windows systems, see Section 18.2.2, “Installing NDB Cluster on Windows”.

Each NDB Cluster host computer must have the correct executable programs installed. A host running an SQL node must have installed on it a MySQL Server binary (`mysqld`). Management nodes require the management server daemon (`ndb_mgmd`); data nodes require the data node daemon (`ndbd` or `ndbmtd`). It is not necessary to install the MySQL Server binary on management node hosts and data node hosts. It is recommended that you also install the management client (`ndb_mgm`) on the management server host.

Installation of NDB Cluster on Linux can be done using precompiled binaries from Oracle (downloaded as a .tar.gz archive), with RPM packages (also available from Oracle), or from source code. All three of these installation methods are described in the section that follow.

Regardless of the method used, it is still necessary following installation of the NDB Cluster binaries to create configuration files for all cluster nodes, before you can start the cluster. See Section 18.2.3, “Initial Configuration of NDB Cluster”.

18.2.1.1 Installing an NDB Cluster Binary Release on Linux

This section covers the steps necessary to install the correct executables for each type of Cluster node from precompiled binaries supplied by Oracle.

For setting up a cluster using precompiled binaries, the first step in the installation process for each cluster host is to download the binary archive from the NDB Cluster downloads page. (For the most recent 64-bit NDB 7.4 release, this is `mysql-cluster-gpl-7.4.30-linux-glibc2.12-x86_64.tar.gz`.) We assume that you have placed this file in each machine's `/var/tmp` directory.

If you require a custom binary, see Section 2.9.5, “Installing MySQL Using a Development Source Tree”.

Note

After completing the installation, do not yet start any of the binaries. We show you how to do so following the configuration of the nodes (see Section 18.2.3, “Initial Configuration of NDB Cluster”).

SQL nodes. On each of the machines designated to host SQL nodes, perform the following steps as the system `root` user:

1. Check your `/etc/passwd` and `/etc/group` files (or use whatever tools are provided by your operating system for managing users and groups) to see whether there is already a `mysql` group and `mysql` user on the system. Some OS distributions create these as part of the operating system installation process. If they are not already present, create a new `mysql` user group, and then add a `mysql` user to this group:

   ```
   shell> groupadd mysql
   shell> useradd -g mysql -s /bin/false mysql
   ```

   The syntax for `useradd` and `groupadd` may differ slightly on different versions of Unix, or they may have different names such as `adduser` and `addgroup`.

2. Change location to the directory containing the downloaded file, unpack the archive, and create a symbolic link named `mysql` to the `mysql` directory.

   Note

   The actual file and directory names vary according to the NDB Cluster version number.

   ```
   shell> cd /var/tmp
   shell> tar -C /usr/local -xzvf mysql-cluster-gpl-7.4.30-linux-glibc2.12-x86_64.tar.gz
   shell> ln -s /usr/local/mysql-cluster-gpl-7.4.30-linux-glibc2.12-x86_64 /usr/local/mysql
   ```

3. Change location to the `mysql` directory and run the supplied script for creating the system databases:

   ```
   shell> cd mysql
   shell> scripts/mysql_install_db --user=mysql
   ```

4. Set the necessary permissions for the MySQL server and data directories:

   ```
   shell> chown -R root .
   shell> chown -R mysql data
   shell> chgrp -R mysql .
   ```

5. Copy the MySQL startup script to the appropriate directory, make it executable, and set it to start when the operating system is booted up:

   ```
   shell> cp support-files/mysql.server /etc/rc.d/init.d/
   shell> chmod +x /etc/rc.d/init.d/mysql.server
   shell> chkconfig --add mysql.server
   ```

(The startup scripts directory may vary depending on your operating system and version—for example, in some Linux distributions, it is `/etc/init.d`.)

Here we use Red Hat's `chkconfig` for creating links to the startup scripts; use whatever means is appropriate for this purpose on your platform, such as `update-rc.d` on Debian.

Remember that the preceding steps must be repeated on each machine where an SQL node is to reside.

Data nodes. Installation of the data nodes does not require the `mysqld` binary. Only the NDB Cluster data node executable `ndbd` (single-threaded) or `ndbmtd` (multithreaded) is required. These binaries can also be found in the `.tar.gz` archive. Again, we assume that you have placed this archive in `/var/tmp`.

As system `root` (that is, after using `sudo`, `su root`, or your system's equivalent for temporarily assuming the system administrator account's privileges), perform the following steps to install the data node binaries on the data node hosts:

1. Change location to the `/var/tmp` directory, and extract the `ndbd` and `ndbmtd` binaries from the archive into a suitable directory such as `/usr/local/bin`:

```
shell> cd /var/tmp
shell> tar -zxvf mysql-cluster-gpl-7.4.30-linux-glibc2.12-x86_64.tar.gz
shell> cd mysql-cluster-gpl-7.4.30-linux-glibc2.12-x86_64
shell> cp bin/ndbd /usr/local/bin/ndbd
shell> cp bin/ndbmtd /usr/local/bin/ndbmtd
```

   (You can safely delete the directory created by unpacking the downloaded archive, and the files it contains, from `/var/tmp` once `ndb_mgm` and `ndb_mgmd` have been copied to the executables directory.)

2. Change location to the directory into which you copied the files, and then make both of them executable:

```
shell> cd /usr/local/bin
shell> chmod +x ndb*
```

The preceding steps should be repeated on each data node host.

Although only one of the data node executables is required to run an NDB Cluster data node, we have shown you how to install both `ndbd` and `ndbmtd` in the preceding instructions. We recommend that you do this when installing or upgrading NDB Cluster, even if you plan to use only one of them, since this saves time and trouble in the event that you later decide to change from one to the other.

Note

The data directory on each machine hosting a data node is `/usr/local/mysql/data`. This piece of information is essential when configuring the management node. (See Section 18.2.3, “Initial Configuration of NDB Cluster”.)

Management nodes. Installation of the management node does not require the `mysqld` binary. Only the NDB Cluster management server (`ndb_mgmd`) is required; you most likely want to install the management client (`ndb_mgm`) as well. Both of these binaries also be found in the `.tar.gz` archive. Again, we assume that you have placed this archive in `/var/tmp`.

As system `root`, perform the following steps to install `ndb_mgmd` and `ndb_mgm` on the management node host:

1. Change location to the `/var/tmp` directory, and extract the `ndb_mgm` and `ndb_mgmd` from the archive into a suitable directory such as `/usr/local/bin`:

```
shell> cd /var/tmp
```

```
shell> tar -zxvf mysql-cluster-gpl-7.4.30-linux-glibc2.12-x86_64.tar.gz
shell> cd mysql-cluster-gpl-7.4.30-linux-glibc2.12-x86_64
shell> cp bin/ndb_mgm* /usr/local/bin
```

(You can safely delete the directory created by unpacking the downloaded archive, and the files it contains, from `/var/tmp` once `ndb_mgm` and `ndb_mgmd` have been copied to the executables directory.)

2. Change location to the directory into which you copied the files, and then make both of them executable:

```
shell> cd /usr/local/bin
shell> chmod +x ndb_mgm*
```

In Section 18.2.3, “Initial Configuration of NDB Cluster”, we create configuration files for all of the nodes in our example NDB Cluster.

18.2.1.2 Installing NDB Cluster from RPM

This section covers the steps necessary to install the correct executables for each type of NDB Cluster node using RPM packages supplied by Oracle.

RPMs are available for both 32-bit and 64-bit Linux platforms. The filenames for these RPMs use the following pattern:

```
MySQL-Cluster-component-producttype-ndbversion.distribution.architecture.rpm

component:= {server | client [| other]}

producttype:= {gpl | advanced}

ndbversion:= major.minor.release

distribution:= {sles10 | rhel5 | el6}

architecture:= {i386 | x86_64}
```

The *`component`* can be `server` or `client`. (Other values are possible, but since only the `server` and `client` components are required for a working NDB Cluster installation, we do not discuss them here.) The *`producttype`* for Community RPMs downloaded from https://dev.mysql.com/downloads/cluster/ is always `gpl`; `advanced` is used to indicate commercial releases. *`ndbversion`* represents the three-part `NDB` storage engine version number in 7.3.*x* or 7.4.*x* format. The *`distribution`* can be one of `sles11` (SUSE Enterprise Linux 11), `rhel5` (Oracle Linux 5, Red Hat Enterprise Linux 4 and 5), or `el6` (Oracle Linux 6, Red Hat Enterprise Linux 6) The *`architecture`* is `i386` for 32-bit RPMs and `x86_64` for 64-bit versions.

For an NDB Cluster, one and possibly two RPMs are required:

- The `server` RPM (for example, `MySQL-Cluster-server-gpl-7.3.31-1.sles11.i386.rpm` or `MySQL-Cluster-server-gpl-7.4.30-1.sles11.i386.rpm`), which supplies the core files needed to run a MySQL Server with `NDBCLUSTER` storage engine support (that is, as an NDB Cluster SQL node) as well as all NDB Cluster executables, including the management node, data node, and `ndb_mgm` client binaries. This RPM is always required for installing NDB Cluster.

- If you do not have your own client application capable of administering a MySQL server, you should also obtain and install the `client` RPM (for example, `MySQL-Cluster-client-gpl-7.3.31-1.sles11.i386.rpm` or `MySQL-Cluster-client-gpl-7.4.30-1.sles11.i386.rpm`), which supplies the `mysql` client

The NDB Cluster version number in the RPM file names (shown here as `7.3.31` or `7.4.30`, depending on whether you are installing NDB Cluster 7.3 or NDB Cluster 7.4) can vary according to the version which you are actually using. *It is very important that all of the Cluster RPMs to be installed have the same version number*. The *`architecture`* designation should also be appropriate to the

machine on which the RPM is to be installed; in particular, you should keep in mind that 64-bit RPMs cannot be used with 32-bit operating systems.

Data nodes. On a computer that is to host a cluster data node it is necessary to install only the `server` RPM. To do so, copy this RPM to the data node host, and run the following command as the system root user, replacing the name shown for the RPM as necessary to match that of the RPM downloaded from the MySQL website:

```
shell> rpm -Uhv MySQL-Cluster-server-gpl-7.3.31-1.sles11.i386.rpm
```

or

```
shell> rpm -Uhv MySQL-Cluster-server-gpl-7.4.30-1.sles11.i386.rpm
```

Although this installs all NDB Cluster binaries, only the program `ndbd` or `ndbmtd` (both in `/usr/sbin`) is actually needed to run an NDB Cluster data node.

SQL nodes. On each machine to be used for hosting a cluster SQL node, install the `server` RPM by executing the following command as the system root user, replacing the name shown for the RPM as necessary to match the name of the RPM downloaded from the MySQL website:

```
shell> rpm -Uhv MySQL-Cluster-server-gpl-7.3.31-1.sles11.i386.rpm
```

or

```
shell> rpm -Uhv MySQL-Cluster-server-gpl-7.4.30-1.sles11.i386.rpm
```

This installs the MySQL server binary (`mysqld`) with `NDB` storage engine support in the `/usr/sbin` directory, as well as all needed MySQL Server support files. It also installs the `mysql.server` and `mysqld_safe` startup scripts (in `/usr/share/mysql` and `/usr/bin`, respectively). The RPM installer should take care of general configuration issues (such as creating the `mysql` user and group, if needed) automatically.

To administer the SQL node (MySQL server), you should also install the `client` RPM, as shown here:

```
shell> rpm -Uhv MySQL-Cluster-client-gpl-7.3.31-1.sles11.i386.rpm
```

or

```
shell> rpm -Uhv MySQL-Cluster-client-gpl-7.4.30-1.sles11.i386.rpm
```

This installs the `mysql` client program.

Management nodes. To install the NDB Cluster management server, it is necessary only to use the `server` RPM. Copy this RPM to the computer intended to host the management node, and then install it by running the following command as the system root user (replace the name shown for the RPM as necessary to match that of the `server` RPM downloaded from the MySQL website):

```
shell> rpm -Uhv MySQL-Cluster-server-gpl-7.3.31-1.sles11.i386.rpm
```

or

```
shell> rpm -Uhv MySQL-Cluster-server-gpl-7.4.30-1.sles11.i386.rpm
```

Although this RPM installs many other files, only the management server binary `ndb_mgmd` (in the `/usr/sbin` directory) is actually required for running a management node. The `server` RPM also installs `ndb_mgm`, the `NDB` management client.

See Section 2.5.5, “Installing MySQL on Linux Using RPM Packages from Oracle”, for general information about installing MySQL using RPMs supplied by Oracle.

After installing from RPM, you still need to configure the cluster as discussed in Section 18.2.3, “Initial Configuration of NDB Cluster”.

18.2.1.3 Installing NDB Cluster Using .deb Files

The section provides information about installing NDB Cluster on Debian and related Linux distributions such Ubuntu using the `.deb` files supplied by Oracle for this purpose.

Oracle provides `.deb` installer files for NDB Cluster 7.3 and NDB Cluster 7.4 for 32-bit and 64-bit platforms. For a Debian-based system, only a single installer file is necessary. This file is named using the pattern shown here, according to the applicable NDB Cluster version, Debian version, and architecture:

```
mysql-cluster-gpl-ndbver-debiandebianver-arch.deb
```

Here, *`ndbver`* is the 3-part `NDB` engine version number, *`debianver`* is the major version of Debian (`6.0` or `7`), and *`arch`* is one of `i686` or `x86_64`. In the examples that follow, we assume you wish to install NDB 7.4.9 on a 64-bit Debian 7 system; in this case, the installer file is named `mysql-cluster-gpl-7.4.9-debian7-x86_64.deb`.

Once you have downloaded the appropriate `.deb` file, you can install it from the command line using `dpkg`, like this:

```
shell> dpkg -i mysql-cluster-gpl-7.4.9-debian7-i686.deb
```

You can also remove it using `dpkg` as shown here:

```
shell> dpkg -r mysql
```

The installer file should also be compatible with most graphical package managers that work with `.deb` files, such as `GDebi` for the Gnome desktop.

The `.deb` file installs NDB Cluster under `/opt/mysql/server-version/`, where *`version`* is the 2-part release series version for the included MySQL server. For both NDB Cluster 7.3 and NDB Cluster 7.4, this is always `5.6`. The directory layout is the same as that for the generic Linux binary distribution (see Table 2.3, “MySQL Installation Layout for Generic Unix/Linux Binary Package”), with the exception that startup scripts and configuration files are found in `support-files` instead of `share`. All NDB Cluster executables, such as `ndb_mgm`, `ndbd`, and `ndb_mgmd`, are placed in the `bin` directory.

18.2.1.4 Building NDB Cluster from Source on Linux

This section provides information about compiling NDB Cluster on Linux and other Unix-like platforms. Building NDB Cluster from source is similar to building the standard MySQL Server, although it differs in a few key respects discussed here. For general information about building MySQL from source, see Section 2.9, “Installing MySQL from Source”. For information about compiling NDB Cluster on Windows platforms, see Section 18.2.2.2, “Compiling and Installing NDB Cluster from Source on Windows”.

Building NDB Cluster requires using the NDB Cluster sources. These are available from the NDB Cluster downloads page at https://dev.mysql.com/downloads/cluster/. The archived source file should have a name similar to `mysql-cluster-gpl-7.3.31.tar.gz` (NDB Cluster 7.3) or `mysql-cluster-gpl-7.4.30.tar.gz` (NDB Cluster 7.4). You can also obtain NDB Cluster sources from GitHub at https://github.com/mysql/mysql-server/tree/cluster-7.3 (NDB 7.3) and https://github.com/mysql/mysql-server/tree/cluster-7.4 (NDB 7.4). *Building NDB Cluster 7.3 or 7.4 from standard MySQL Server 5.6 sources is not supported*.

The `WITH_NDBCLUSTER_STORAGE_ENGINE` option for `CMake` causes the binaries for the management nodes, data nodes, and other NDB Cluster programs to be built; it also causes `mysqld` to be compiled with `NDB` storage engine support. This option (or its alias `WITH_NDBCLUSTER`) is required when building NDB Cluster.

Important

In NDB Cluster 7.3 and later, the `WITH_NDB_JAVA` option is enabled by default. This means that, by default, if `CMake` cannot find the location of Java on your

> system, the configuration process fails; if you do not wish to enable Java and ClusterJ support, you must indicate this explicitly by configuring the build using `-DWITH_NDB_JAVA=OFF`. Use `WITH_CLASSPATH` to provide the Java classpath if needed.

For more information about `CMake` options specific to building NDB Cluster, see Options for Compiling NDB Cluster.

After you have run `make && make install` (or your system's equivalent), the result is similar to what is obtained by unpacking a precompiled binary to the same location.

Management nodes. When building from source and running the default `make install`, the management server and management client binaries (`ndb_mgmd` and `ndb_mgm`) can be found in `/usr/local/mysql/bin`. Only `ndb_mgmd` is required to be present on a management node host; however, it is also a good idea to have `ndb_mgm` present on the same host machine. Neither of these executables requires a specific location on the host machine's file system.

Data nodes. The only executable required on a data node host is the data node binary `ndbd` or `ndbmtd`. (`mysqld`, for example, does not have to be present on the host machine.) By default, when building from source, this file is placed in the directory `/usr/local/mysql/bin`. For installing on multiple data node hosts, only `ndbd` or `ndbmtd` need be copied to the other host machine or machines. (This assumes that all data node hosts use the same architecture and operating system; otherwise you may need to compile separately for each different platform.) The data node binary need not be in any particular location on the host's file system, as long as the location is known.

When compiling NDB Cluster from source, no special options are required for building multithreaded data node binaries. Configuring the build with `NDB` storage engine support causes `ndbmtd` to be built automatically; `make install` places the `ndbmtd` binary in the installation `bin` directory along with `mysqld`, `ndbd`, and `ndb_mgm`.

SQL nodes. If you compile MySQL with clustering support, and perform the default installation (using `make install` as the system `root` user), `mysqld` is placed in `/usr/local/mysql/bin`. Follow the steps given in Section 2.9, “Installing MySQL from Source” to make `mysqld` ready for use. If you want to run multiple SQL nodes, you can use a copy of the same `mysqld` executable and its associated support files on several machines. The easiest way to do this is to copy the entire `/usr/local/mysql` directory and all directories and files contained within it to the other SQL node host or hosts, then repeat the steps from Section 2.9, “Installing MySQL from Source” on each machine. If you configure the build with a nondefault `PREFIX` option, you must adjust the directory accordingly.

In Section 18.2.3, “Initial Configuration of NDB Cluster”, we create configuration files for all of the nodes in our example NDB Cluster.

18.2.2 Installing NDB Cluster on Windows

This section describes installation procedures for NDB Cluster on Windows hosts. NDB Cluster 7.3 and NDB Cluster 7.4 binaries for Windows can be obtained from https://dev.mysql.com/downloads/cluster/. For information about installing NDB Cluster on Windows from a binary release provided by Oracle, see Section 18.2.2.1, “Installing NDB Cluster on Windows from a Binary Release”.

It is also possible to compile and install NDB Cluster from source on Windows using Microsoft Visual Studio. For more information, see Section 18.2.2.2, “Compiling and Installing NDB Cluster from Source on Windows”.

18.2.2.1 Installing NDB Cluster on Windows from a Binary Release

This section describes a basic installation of NDB Cluster on Windows using a binary “no-install” NDB Cluster release provided by Oracle, using the same 4-node setup outlined in the beginning of this section (see Section 18.2, “NDB Cluster Installation”), as shown in the following table:

Table 18.5 Network addresses of nodes in example cluster

Node	IP Address
Management node (mgmd)	198.51.100.10
SQL node (mysqld)	198.51.100.20
Data node "A" (ndbd)	198.51.100.30
Data node "B" (ndbd)	198.51.100.40

As on other platforms, the NDB Cluster host computer running an SQL node must have installed on it a MySQL Server binary (mysqld.exe). You should also have the MySQL client (mysql.exe) on this host. For management nodes and data nodes, it is not necessary to install the MySQL Server binary; however, each management node requires the management server daemon (ndb_mgmd.exe); each data node requires the data node daemon (ndbd.exe or ndbmtd.exe). For this example, we refer to ndbd.exe as the data node executable, but you can install ndbmtd.exe, the multithreaded version of this program, instead, in exactly the same way. You should also install the management client (ndb_mgm.exe) on the management server host. This section covers the steps necessary to install the correct Windows binaries for each type of NDB Cluster node.

Note

As with other Windows programs, NDB Cluster executables are named with the .exe file extension. However, it is not necessary to include the .exe extension when invoking these programs from the command line. Therefore, we often simply refer to these programs in this documentation as mysqld, mysql, ndb_mgmd, and so on. You should understand that, whether we refer (for example) to mysqld or mysqld.exe, either name means the same thing (the MySQL Server program).

For setting up an NDB Cluster using Oracles's no-install binaries, the first step in the installation process is to download the latest NDB Cluster Windows ZIP binary archive from https://dev.mysql.com/downloads/cluster/. This archive has a filename of the mysql-cluster-gpl-*ver*-win*arch*.zip, where *ver* is the NDB storage engine version (such as 7.4.30), and *arch* is the architecture (32 for 32-bit binaries, and 64 for 64-bit binaries). For example, the NDB Cluster 7.4.30 archive for 64-bit Windows systems is named mysql-cluster-gpl-7.4.30-win64.zip.

You can run 32-bit NDB Cluster binaries on both 32-bit and 64-bit versions of Windows; however, 64-bit NDB Cluster binaries can be used only on 64-bit versions of Windows. If you are using a 32-bit version of Windows on a computer that has a 64-bit CPU, then you must use the 32-bit NDB Cluster binaries.

To minimize the number of files that need to be downloaded from the Internet or copied between machines, we start with the computer where you intend to run the SQL node.

SQL node. We assume that you have placed a copy of the archive in the directory C:\Documents and Settings*username*\My Documents\Downloads on the computer having the IP address 198.51.100.20, where *username* is the name of the current user. (You can obtain this name using ECHO %USERNAME% on the command line.) To install and run NDB Cluster executables as Windows services, this user should be a member of the Administrators group.

Extract all the files from the archive. The Extraction Wizard integrated with Windows Explorer is adequate for this task. (If you use a different archive program, be sure that it extracts all files and directories from the archive, and that it preserves the archive's directory structure.) When you are asked for a destination directory, enter C:\, which causes the Extraction Wizard to extract the archive to the directory C:\mysql-cluster-gpl-*ver*-win*arch*. Rename this directory to C:\mysql.

It is possible to install the NDB Cluster binaries to directories other than C:\mysql\bin; however, if you do so, you must modify the paths shown in this procedure accordingly. In particular, if the MySQL Server (SQL node) binary is installed to a location other than C:\mysql or C:\Program Files\MySQL\MySQL Server 5.6, or if the SQL node's data directory is in a location other than C:

`\mysql\data` or `C:\Program Files\MySQL\MySQL Server 5.6\data`, extra configuration options must be used on the command line or added to the `my.ini` or `my.cnf` file when starting the SQL node. For more information about configuring a MySQL Server to run in a nonstandard location, see Section 2.3.4, "Installing MySQL on Microsoft Windows Using a `noinstall` ZIP Archive".

For a MySQL Server with NDB Cluster support to run as part of an NDB Cluster, it must be started with the options `--ndbcluster` and `--ndb-connectstring`. While you can specify these options on the command line, it is usually more convenient to place them in an option file. To do this, create a new text file in Notepad or another text editor. Enter the following configuration information into this file:

```
[mysqld]
# Options for mysqld process:
ndbcluster                      # run NDB storage engine
ndb-connectstring=198.51.100.10  # location of management server
```

You can add other options used by this MySQL Server if desired (see Section 2.3.4.2, "Creating an Option File"), but the file must contain the options shown, at a minimum. Save this file as `C:\mysql\my.ini`. This completes the installation and setup for the SQL node.

Data nodes. An NDB Cluster data node on a Windows host requires only a single executable, one of either `ndbd.exe` or `ndbmtd.exe`. For this example, we assume that you are using `ndbd.exe`, but the same instructions apply when using `ndbmtd.exe`. On each computer where you wish to run a data node (the computers having the IP addresses 198.51.100.30 and 198.51.100.40), create the directories `C:\mysql`, `C:\mysql\bin`, and `C:\mysql\cluster-data`; then, on the computer where you downloaded and extracted the `no-install` archive, locate `ndbd.exe` in the `C:\mysql\bin` directory. Copy this file to the `C:\mysql\bin` directory on each of the two data node hosts.

To function as part of an NDB Cluster, each data node must be given the address or hostname of the management server. You can supply this information on the command line using the `--ndb-connectstring` or `-c` option when starting each data node process. However, it is usually preferable to put this information in an option file. To do this, create a new text file in Notepad or another text editor and enter the following text:

```
[mysql_cluster]
# Options for data node process:
ndb-connectstring=198.51.100.10  # location of management server
```

Save this file as `C:\mysql\my.ini` on the data node host. Create another text file containing the same information and save it on as `C:mysql\my.ini` on the other data node host, or copy the my.ini file from the first data node host to the second one, making sure to place the copy in the second data node's `C:\mysql` directory. Both data node hosts are now ready to be used in the NDB Cluster, which leaves only the management node to be installed and configured.

Management node. The only executable program required on a computer used for hosting an NDB Cluster management node is the management server program `ndb_mgmd.exe`. However, in order to administer the NDB Cluster once it has been started, you should also install the NDB Cluster management client program `ndb_mgm.exe` on the same machine as the management server. Locate these two programs on the machine where you downloaded and extracted the `no-install` archive; this should be the directory `C:\mysql\bin` on the SQL node host. Create the directory `C:\mysql\bin` on the computer having the IP address 198.51.100.10, then copy both programs to this directory.

You should now create two configuration files for use by `ndb_mgmd.exe`:

1. A local configuration file to supply configuration data specific to the management node itself. Typically, this file needs only to supply the location of the NDB Cluster global configuration file (see item 2).

   To create this file, start a new text file in Notepad or another text editor, and enter the following information:

   ```
   [mysql_cluster]
   # Options for management node process
   ```

```
config-file=C:/mysql/bin/config.ini
```

Save this file as the text file `C:\mysql\bin\my.ini`.

2. A global configuration file from which the management node can obtain configuration information governing the NDB Cluster as a whole. At a minimum, this file must contain a section for each node in the NDB Cluster, and the IP addresses or hostnames for the management node and all data nodes (`HostName` configuration parameter). It is also advisable to include the following additional information:

   - The IP address or hostname of any SQL nodes
   - The data memory and index memory allocated to each data node (`DataMemory` and `IndexMemory` configuration parameters)
   - The number of replicas, using the `NoOfReplicas` configuration parameter (see Section 18.1.2, “NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions”)
   - The directory where each data node stores it data and log file, and the directory where the management node keeps its log files (in both cases, the `DataDir` configuration parameter)

   Create a new text file using a text editor such as Notepad, and input the following information:

```
[ndbd default]
# Options affecting ndbd processes on all data nodes:
NoOfReplicas=2                      # Number of fragment replicas
DataDir=C:/mysql/cluster-data       # Directory for each data node's data files
                                    # Forward slashes used in directory path,
                                    # rather than backslashes. This is correct;
                                    # see Important note in text
DataMemory=80M    # Memory allocated to data storage
IndexMemory=18M   # Memory allocated to index storage
                  # For DataMemory and IndexMemory, we have used the
                  # default values. Since the "world" database takes up
                  # only about 500KB, this should be more than enough for
                  # this example Cluster setup.

[ndb_mgmd]
# Management process options:
HostName=198.51.100.10              # Hostname or IP address of management node
DataDir=C:/mysql/bin/cluster-logs   # Directory for management node log files

[ndbd]
# Options for data node "A":
                                # (one [ndbd] section per data node)
HostName=198.51.100.30          # Hostname or IP address

[ndbd]
# Options for data node "B":
HostName=198.51.100.40          # Hostname or IP address

[mysqld]
# SQL node options:
HostName=198.51.100.20          # Hostname or IP address
```

Save this file as the text file `C:\mysql\bin\config.ini`.

Important

A single backslash character (`\`) cannot be used when specifying directory paths in program options or configuration files used by NDB Cluster on Windows. Instead, you must either escape each backslash character with a second backslash (`\\`), or replace the backslash with a forward slash character (`/`). For example, the following line from the `[ndb_mgmd]` section of an NDB Cluster `config.ini` file does not work:

```
DataDir=C:\mysql\bin\cluster-logs
```

Instead, you may use either of the following:

```
DataDir=C:\\mysql\\bin\\cluster-logs  # Escaped backslashes
```

```
DataDir=C:/mysql/bin/cluster-logs     # Forward slashes
```

For reasons of brevity and legibility, we recommend that you use forward slashes in directory paths used in NDB Cluster program options and configuration files on Windows.

18.2.2.2 Compiling and Installing NDB Cluster from Source on Windows

Oracle provides precompiled NDB Cluster binaries for Windows which should be adequate for most users. However, if you wish, it is also possible to compile NDB Cluster for Windows from source code. The procedure for doing this is almost identical to the procedure used to compile the standard MySQL Server binaries for Windows, and uses the same tools. However, there are two major differences:

- Building NDB Cluster requires using the NDB Cluster sources. These are available from the NDB Cluster downloads page at https://dev.mysql.com/downloads/cluster/. The archived source file should have a name similar to `mysql-cluster-gpl-7.3.31.tar.gz` (NDB Cluster 7.3) or `mysql-cluster-gpl-7.4.30.tar.gz` (NDB Cluster 7.4). You can also obtain NDB Cluster sources from GitHub at https://github.com/mysql/mysql-server/tree/cluster-7.3 (NDB 7.3) and https://github.com/mysql/mysql-server/tree/cluster-7.4 (NDB 7.4). *Building NDB Cluster 7.3 or 7.4 from standard MySQL Server 5.6 sources is not supported*.

- You must configure the build using the `WITH_NDBCLUSTER_STORAGE_ENGINE` or `WITH_NDBCLUSTER` option in addition to any other build options you wish to use with `CMake`. (`WITH_NDBCLUSTER` is supported as an alias for `WITH_NDBCLUSTER_STORAGE_ENGINE`, and works in exactly the same way.)

Important

In NDB Cluster 7.3 and later, the `WITH_NDB_JAVA` option is enabled by default. This means that, by default, if `CMake` cannot find the location of Java on your system, the configuration process fails; if you do not wish to enable Java and ClusterJ support, you must indicate this explicitly by configuring the build using `-DWITH_NDB_JAVA=OFF`. (Bug #12379735) Use `WITH_CLASSPATH` to provide the Java classpath if needed.

For more information about `CMake` options specific to building NDB Cluster, see Options for Compiling NDB Cluster.

Once the build process is complete, you can create a Zip archive containing the compiled binaries; Section 2.9.4, “Installing MySQL Using a Standard Source Distribution” provides the commands needed to perform this task on Windows systems. The NDB Cluster binaries can be found in the `bin` directory of the resulting archive, which is equivalent to the `no-install` archive, and which can be installed and configured in the same manner. For more information, see Section 18.2.2.1, “Installing NDB Cluster on Windows from a Binary Release”.

18.2.2.3 Initial Startup of NDB Cluster on Windows

Once the NDB Cluster executables and needed configuration files are in place, performing an initial start of the cluster is simply a matter of starting the NDB Cluster executables for all nodes in the cluster. Each cluster node process must be started separately, and on the host computer where it resides. The management node should be started first, followed by the data nodes, and then finally by any SQL nodes.

1. On the management node host, issue the following command from the command line to start the management node process. The output should appear similar to what is shown here:

```
C:\mysql\bin> ndb_mgmd
2010-06-23 07:53:34 [MgmtSrvr] INFO -- NDB Cluster Management Server. mysql-5.6.51-ndb-7.4.31
2010-06-23 07:53:34 [MgmtSrvr] INFO -- Reading cluster configuration from 'config.ini'
```

The management node process continues to print logging output to the console. This is normal, because the management node is not running as a Windows service. (If you have used NDB Cluster on a Unix-like platform such as Linux, you may notice that the management node's default behavior in this regard on Windows is effectively the opposite of its behavior on Unix systems, where it runs by default as a Unix daemon process. This behavior is also true of NDB Cluster data node processes running on Windows.) For this reason, do not close the window in which `ndb_mgmd.exe` is running; doing so kills the management node process. (See Section 18.2.2.4, "Installing NDB Cluster Processes as Windows Services", where we show how to install and run NDB Cluster processes as Windows services.)

The required `-f` option tells the management node where to find the global configuration file (`config.ini`). The long form of this option is `--config-file`.

Important

An NDB Cluster management node caches the configuration data that it reads from `config.ini`; once it has created a configuration cache, it ignores the `config.ini` file on subsequent starts unless forced to do otherwise. This means that, if the management node fails to start due to an error in this file, you must make the management node re-read `config.ini` after you have corrected any errors in it. You can do this by starting `ndb_mgmd.exe` with the `--reload` or `--initial` option on the command line. Either of these options works to refresh the configuration cache.

It is not necessary or advisable to use either of these options in the management node's `my.ini` file.

For additional information about options which can be used with `ndb_mgmd`, see Section 18.4.4, "`ndb_mgmd` — The NDB Cluster Management Server Daemon", as well as Section 18.4.29, "Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs".

2. On each of the data node hosts, run the command shown here to start the data node processes:

```
C:\mysql\bin> ndbd
2010-06-23 07:53:46 [ndbd] INFO -- Configuration fetched from 'localhost:1186', generation: 1
```

In each case, the first line of output from the data node process should resemble what is shown in the preceding example, and is followed by additional lines of logging output. As with the management node process, this is normal, because the data node is not running as a Windows service. For this reason, do not close the console window in which the data node process is running; doing so kills `ndbd.exe`. (For more information, see Section 18.2.2.4, "Installing NDB Cluster Processes as Windows Services".)

3. Do not start the SQL node yet; it cannot connect to the cluster until the data nodes have finished starting, which may take some time. Instead, in a new console window on the management node host, start the NDB Cluster management client `ndb_mgm.exe`, which should be in `C:\mysql\bin` on the management node host. (Do not try to re-use the console window where `ndb_mgmd.exe` is running by typing **CTRL**+**C**, as this kills the management node.) The resulting output should look like this:

```
C:\mysql\bin> ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm>
```

When the prompt `ndb_mgm>` appears, this indicates that the management client is ready to receive NDB Cluster management commands. You can observe the status of the data nodes as they start

by entering `ALL STATUS` at the management client prompt. This command causes a running report of the data nodes's startup sequence, which should look something like this:

```
ndb_mgm> ALL STATUS
Connected to Management Server at: localhost:1186
Node 2: starting (Last completed phase 3) (mysql-5.6.51-ndb-7.4.31)
Node 3: starting (Last completed phase 3) (mysql-5.6.51-ndb-7.4.31)

Node 2: starting (Last completed phase 4) (mysql-5.6.51-ndb-7.4.31)
Node 3: starting (Last completed phase 4) (mysql-5.6.51-ndb-7.4.31)

Node 2: Started (version 7.4.31)
Node 3: Started (version 7.4.31)

ndb_mgm>
```

Note

Commands issued in the management client are not case-sensitive; we use uppercase as the canonical form of these commands, but you are not required to observe this convention when inputting them into the `ndb_mgm` client. For more information, see Section 18.5.1, “Commands in the NDB Cluster Management Client”.

The output produced by `ALL STATUS` is likely to vary from what is shown here, according to the speed at which the data nodes are able to start, the release version number of the NDB Cluster software you are using, and other factors. What is significant is that, when you see that both data nodes have started, you are ready to start the SQL node.

You can leave `ndb_mgm.exe` running; it has no negative impact on the performance of the NDB Cluster, and we use it in the next step to verify that the SQL node is connected to the cluster after you have started it.

4. On the computer designated as the SQL node host, open a console window and navigate to the directory where you unpacked the NDB Cluster binaries (if you are following our example, this is `C:\mysql\bin`).

   Start the SQL node by invoking `mysqld.exe` from the command line, as shown here:

   ```
   C:\mysql\bin> mysqld --console
   ```

   The `--console` option causes logging information to be written to the console, which can be helpful in the event of problems. (Once you are satisfied that the SQL node is running in a satisfactory manner, you can stop it and restart it out without the `--console` option, so that logging is performed normally.)

   In the console window where the management client (`ndb_mgm.exe`) is running on the management node host, enter the `SHOW` command, which should produce output similar to what is shown here:

   ```
   ndb_mgm> SHOW
   Connected to Management Server at: localhost:1186
   Cluster Configuration
   ---------------------
   [ndbd(NDB)]     2 node(s)
   id=2    @198.51.100.30  (Version: 5.6.51-ndb-7.4.31, Nodegroup: 0, *)
   id=3    @198.51.100.40  (Version: 5.6.51-ndb-7.4.31, Nodegroup: 0)

   [ndb_mgmd(MGM)] 1 node(s)
   id=1    @198.51.100.10  (Version: 5.6.51-ndb-7.4.31)

   [mysqld(API)]   1 node(s)
   ```

```
id=4    @198.51.100.20  (Version: 5.6.51-ndb-7.4.31)
```

You can also verify that the SQL node is connected to the NDB Cluster in the `mysql` client (`mysql.exe`) using the `SHOW ENGINE NDB STATUS` statement.

You should now be ready to work with database objects and data using NDB Cluster's `NDBCLUSTER` storage engine. See Section 18.2.5, “NDB Cluster Example with Tables and Data”, for more information and examples.

You can also install `ndb_mgmd.exe`, `ndbd.exe`, and `ndbmtd.exe` as Windows services. For information on how to do this, see Section 18.2.2.4, “Installing NDB Cluster Processes as Windows Services”).

18.2.2.4 Installing NDB Cluster Processes as Windows Services

Once you are satisfied that NDB Cluster is running as desired, you can install the management nodes and data nodes as Windows services, so that these processes are started and stopped automatically whenever Windows is started or stopped. This also makes it possible to control these processes from the command line with the appropriate `SC START` and `SC STOP` commands, or using the Windows graphical `Services` utility. `NET START` and `NET STOP` commands can also be used.

Installing programs as Windows services usually must be done using an account that has Administrator rights on the system.

To install the management node as a service on Windows, invoke `ndb_mgmd.exe` from the command line on the machine hosting the management node, using the `--install` option, as shown here:

```
C:\> C:\mysql\bin\ndb_mgmd.exe --install
Installing service 'NDB Cluster Management Server'
  as '"C:\mysql\bin\ndbd.exe" "--service=ndb_mgmd"'
Service successfully installed.
```

Important

When installing an NDB Cluster program as a Windows service, you should always specify the complete path; otherwise the service installation may fail with the error `The system cannot find the file specified`.

The `--install` option must be used first, ahead of any other options that might be specified for `ndb_mgmd.exe`. However, it is preferable to specify such options in an options file instead. If your options file is not in one of the default locations as shown in the output of `ndb_mgmd.exe --help`, you can specify the location using the `--config-file` option.

Now you should be able to start and stop the management server like this:

```
C:\> SC START ndb_mgmd

C:\> SC STOP ndb_mgmd
```

Note

If using `NET` commands, you can also start or stop the management server as a Windows service using the descriptive name, as shown here:

```
C:\> NET START 'NDB Cluster Management Server'
The NDB Cluster Management Server service is starting.
The NDB Cluster Management Server service was started successfully.

C:\> NET STOP  'NDB Cluster Management Server'
The NDB Cluster Management Server service is stopping..
The NDB Cluster Management Server service was stopped successfully.
```

It is usually simpler to specify a short service name or to permit the default service name to be used when installing the service, and then reference that name when starting or stopping the service. To

specify a service name other than `ndb_mgmd`, append it to the `--install` option, as shown in this example:

```
C:\> C:\mysql\bin\ndb_mgmd.exe --install=mgmd1
Installing service 'NDB Cluster Management Server'
  as '"C:\mysql\bin\ndb_mgmd.exe" "--service=mgmd1"'
Service successfully installed.
```

Now you should be able to start or stop the service using the name you have specified, like this:

```
C:\> SC START mgmd1

C:\> SC STOP mgmd1
```

To remove the management node service, use `SC DELETE service_name`:

```
C:\> SC DELETE mgmd1
```

Alternatively, invoke `ndb_mgmd.exe` with the `--remove` option, as shown here:

```
C:\> C:\mysql\bin\ndb_mgmd.exe --remove
Removing service 'NDB Cluster Management Server'
Service successfully removed.
```

If you installed the service using a service name other than the default, pass the service name as the value of the `ndb_mgmd.exe --remove` option, like this:

```
C:\> C:\mysql\bin\ndb_mgmd.exe --remove=mgmd1
Removing service 'mgmd1'
Service successfully removed.
```

Installation of an NDB Cluster data node process as a Windows service can be done in a similar fashion, using the `--install` option for `ndbd.exe` (or `ndbmtd.exe`), as shown here:

```
C:\> C:\mysql\bin\ndbd.exe --install
Installing service 'NDB Cluster Data Node Daemon' as '"C:\mysql\bin\ndbd.exe" "--service=ndbd"'
Service successfully installed.
```

Now you can start or stop the data node as shown in the following example:

```
C:\> SC START ndbd

C:\> SC STOP ndbd
```

To remove the data node service, use `SC DELETE service_name`:

```
C:\> SC DELETE ndbd
```

Alternatively, invoke `ndbd.exe` with the `--remove` option, as shown here:

```
C:\> C:\mysql\bin\ndbd.exe --remove
Removing service 'NDB Cluster Data Node Daemon'
Service successfully removed.
```

As with `ndb_mgmd.exe` (and `mysqld.exe`), when installing `ndbd.exe` as a Windows service, you can also specify a name for the service as the value of `--install`, and then use it when starting or stopping the service, like this:

```
C:\> C:\mysql\bin\ndbd.exe --install=dnode1
Installing service 'dnode1' as '"C:\mysql\bin\ndbd.exe" "--service=dnode1"'
Service successfully installed.

C:\> SC START dnode1

C:\> SC STOP dnode1
```

If you specified a service name when installing the data node service, you can use this name when removing it as well, as shown here:

```
C:\> SC DELETE dnode1
```

Alternatively, you can pass the service name as the value of the `ndbd.exe --remove` option, as shown here:

```
C:\> C:\mysql\bin\ndbd.exe --remove=dnode1
Removing service 'dnode1'
Service successfully removed.
```

Installation of the SQL node as a Windows service, starting the service, stopping the service, and removing the service are done in a similar fashion, using `mysqld --install`, `SC START`, `SC STOP`, and `SC DELETE` (or `mysqld --remove`). `NET` commands can also be used to start or stop a service. For additional information, see Section 2.3.4.7, “Starting MySQL as a Windows Service”.

18.2.3 Initial Configuration of NDB Cluster

In this section, we discuss manual configuration of an installed NDB Cluster by creating and editing configuration files.

NDB Cluster (NDB versions 7.3 and later) also provides a GUI installer which can be used to perform the configuration without the need to edit text files in a separate application. For more information, see Section 18.2.8, “The NDB Cluster Auto-Installer (No longer supported)”.

For our four-node, four-host NDB Cluster (see Cluster nodes and host computers), it is necessary to write four configuration files, one per node host.

- Each data node or SQL node requires a `my.cnf` file that provides two pieces of information: a *connection string* that tells the node where to find the management node, and a line telling the MySQL server on this host (the machine hosting the data node) to enable the `NDBCLUSTER` storage engine.

  For more information on connection strings, see Section 18.3.3.3, “NDB Cluster Connection Strings”.

- The management node needs a `config.ini` file telling it how many fragment replicas to maintain, how much memory to allocate for data and indexes on each data node, where to find the data nodes, where to save data to disk on each data node, and where to find any SQL nodes.

Configuring the data nodes and SQL nodes. The `my.cnf` file needed for the data nodes is fairly simple. The configuration file should be located in the `/etc` directory and can be edited using any text editor. (Create the file if it does not exist.) For example:

```
shell> vi /etc/my.cnf
```

Note

We show `vi` being used here to create the file, but any text editor should work just as well.

For each data node and SQL node in our example setup, `my.cnf` should look like this:

```
[mysqld]
# Options for mysqld process:
ndbcluster                      # run NDB storage engine

[mysql_cluster]
# Options for NDB Cluster processes:
ndb-connectstring=198.51.100.10  # location of management server
```

After entering the preceding information, save this file and exit the text editor. Do this for the machines hosting data node “A”, data node “B”, and the SQL node.

Important

Once you have started a `mysqld` process with the `ndbcluster` and `ndb-connectstring` parameters in the `[mysqld]` and `[mysql_cluster]`

sections of the `my.cnf` file as shown previously, you cannot execute any `CREATE TABLE` or `ALTER TABLE` statements without having actually started the cluster. Otherwise, these statements fails with an error. This is by design.

Configuring the management node. The first step in configuring the management node is to create the directory in which the configuration file can be found and then to create the file itself. For example (running as `root`):

```
shell> mkdir /var/lib/mysql-cluster
shell> cd /var/lib/mysql-cluster
shell> vi config.ini
```

For our representative setup, the `config.ini` file should read as follows:

```
[ndbd default]
# Options affecting ndbd processes on all data nodes:
NoOfReplicas=2    # Number of fragment replicas
DataMemory=80M    # How much memory to allocate for data storage
IndexMemory=18M   # How much memory to allocate for index storage
                  # For DataMemory and IndexMemory, we have used the
                  # default values. Since the "world" database takes up
                  # only about 500KB, this should be more than enough for
                  # this example NDB Cluster setup.
ServerPort=2202   # This the default value; however, you can use any
                  # port that is free for all the hosts in the cluster
                  # Note1: It is recommended that you do not specify the port
                  # number at all and simply allow the default value to be used
                  # instead
                  # Note2: The port was formerly specified using the PortNumber
                  # TCP parameter; this parameter is no longer available in NDB
                  # Cluster 7.5.

[ndb_mgmd]
# Management process options:
HostName=198.51.100.10          # Hostname or IP address of management node
DataDir=/var/lib/mysql-cluster  # Directory for management node log files

[ndbd]
# Options for data node "A":
                                # (one [ndbd] section per data node)
HostName=198.51.100.30          # Hostname or IP address
NodeId=2                        # Node ID for this data node
DataDir=/usr/local/mysql/data   # Directory for this data node's data files

[ndbd]
# Options for data node "B":
HostName=198.51.100.40          # Hostname or IP address
NodeId=3                        # Node ID for this data node
DataDir=/usr/local/mysql/data   # Directory for this data node's data files

[mysqld]
# SQL node options:
HostName=198.51.100.20          # Hostname or IP address
                                # (additional mysqld connections can be
                                # specified for this node for various
                                # purposes such as running ndb_restore)
```

Note

The `world` database can be downloaded from https://dev.mysql.com/doc/index-other.html.

After all the configuration files have been created and these minimal options have been specified, you are ready to proceed with starting the cluster and verifying that all processes are running. We discuss how this is done in Section 18.2.4, “Initial Startup of NDB Cluster”.

For more detailed information about the available NDB Cluster configuration parameters and their uses, see Section 18.3.3, “NDB Cluster Configuration Files”, and Section 18.3, “Configuration of

NDB Cluster". For configuration of NDB Cluster as relates to making backups, see Section 18.5.8.3, "Configuration for NDB Cluster Backups".

Note

The default port for Cluster management nodes is 1186; the default port for data nodes is 2202. However, the cluster can automatically allocate ports for data nodes from those that are already free.

18.2.4 Initial Startup of NDB Cluster

Starting the cluster is not very difficult after it has been configured. Each cluster node process must be started separately, and on the host where it resides. The management node should be started first, followed by the data nodes, and then finally by any SQL nodes:

1. On the management host, issue the following command from the system shell to start the management node process:

   ```
   shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
   ```

   The first time that it is started, `ndb_mgmd` must be told where to find its configuration file, using the `-f` or `--config-file` option. (See Section 18.4.4, "ndb_mgmd — The NDB Cluster Management Server Daemon", for details.)

   For additional options which can be used with `ndb_mgmd`, see Section 18.4.29, "Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs".

2. On each of the data node hosts, run this command to start the `ndbd` process:

   ```
   shell> ndbd
   ```

3. If you used RPM files to install MySQL on the cluster host where the SQL node is to reside, you can (and should) use the supplied startup script to start the MySQL server process on the SQL node.

If all has gone well, and the cluster has been set up correctly, the cluster should now be operational. You can test this by invoking the `ndb_mgm` management node client. The output should look like that shown here, although you might see some slight differences in the output depending upon the exact version of MySQL that you are using:

```
shell> ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=2    @198.51.100.30  (Version: 5.6.51-ndb-7.4.31, Nodegroup: 0, *)
id=3    @198.51.100.40  (Version: 5.6.51-ndb-7.4.31, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @198.51.100.10  (Version: 5.6.51-ndb-7.4.31)

[mysqld(API)]   1 node(s)
id=4    @198.51.100.20  (Version: 5.6.51-ndb-7.4.31)
```

The SQL node is referenced here as `[mysqld(API)]`, which reflects the fact that the `mysqld` process is acting as an NDB Cluster API node.

Note

The IP address shown for a given NDB Cluster SQL or other API node in the output of `SHOW` is the address used by the SQL or API node to connect to the cluster data nodes, and not to any management node.

You should now be ready to work with databases, tables, and data in NDB Cluster. See Section 18.2.5, “NDB Cluster Example with Tables and Data”, for a brief discussion.

18.2.5 NDB Cluster Example with Tables and Data

Note

The information in this section applies to NDB Cluster running on both Unix and Windows platforms.

Working with database tables and data in NDB Cluster is not much different from doing so in standard MySQL. There are two key points to keep in mind:

- For a table to be replicated in the cluster, it must use the `NDBCLUSTER` storage engine. To specify this, use the `ENGINE=NDBCLUSTER` or `ENGINE=NDB` option when creating the table:

  ```
  CREATE TABLE tbl_name (col_name column_definitions) ENGINE=NDBCLUSTER;
  ```

  Alternatively, for an existing table that uses a different storage engine, use `ALTER TABLE` to change the table to use `NDBCLUSTER`:

  ```
  ALTER TABLE tbl_name ENGINE=NDBCLUSTER;
  ```

- Every `NDBCLUSTER` table has a primary key. If no primary key is defined by the user when a table is created, the `NDBCLUSTER` storage engine automatically generates a hidden one. Such a key takes up space just as does any other table index. (It is not uncommon to encounter problems due to insufficient memory for accommodating these automatically created indexes.)

If you are importing tables from an existing database using the output of `mysqldump`, you can open the SQL script in a text editor and add the `ENGINE` option to any table creation statements, or replace any existing `ENGINE` options. Suppose that you have the `world` sample database on another MySQL server that does not support NDB Cluster, and you want to export the `City` table:

```
shell> mysqldump --add-drop-table world City > city_table.sql
```

The resulting `city_table.sql` file contains this table creation statement (and the `INSERT` statements necessary to import the table data):

```
DROP TABLE IF EXISTS `City`;
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY  (`ID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;

INSERT INTO `City` VALUES (1,'Kabul','AFG','Kabol',1780000);
INSERT INTO `City` VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO `City` VALUES (3,'Herat','AFG','Herat',186800);
(remaining INSERT statements omitted)
```

You need to make sure that MySQL uses the `NDBCLUSTER` storage engine for this table. There are two ways that this can be accomplished. One of these is to modify the table definition *before* importing it into the Cluster database. Using the `City` table as an example, modify the `ENGINE` option of the definition as follows:

```
DROP TABLE IF EXISTS `City`;
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
```

```
  PRIMARY KEY  (`ID`)
) ENGINE=NDBCLUSTER DEFAULT CHARSET=latin1;

INSERT INTO `City` VALUES (1,'Kabul','AFG','Kabol',1780000);
INSERT INTO `City` VALUES (2,'Qandahar','AFG','Qandahar',237500);
INSERT INTO `City` VALUES (3,'Herat','AFG','Herat',186800);
(remaining INSERT statements omitted)
```

This must be done for the definition of each table that is to be part of the clustered database. The easiest way to accomplish this is to do a search-and-replace on the file that contains the definitions and replace all instances of `TYPE=engine_name` or `ENGINE=engine_name` with `ENGINE=NDBCLUSTER`. If you do not want to modify the file, you can use the unmodified file to create the tables, and then use `ALTER TABLE` to change their storage engine. The particulars are given later in this section.

Assuming that you have already created a database named `world` on the SQL node of the cluster, you can then use the `mysql` command-line client to read `city_table.sql`, and create and populate the corresponding table in the usual manner:

```
shell> mysql world < city_table.sql
```

It is very important to keep in mind that the preceding command must be executed on the host where the SQL node is running (in this case, on the machine with the IP address `198.51.100.20`).

To create a copy of the entire `world` database on the SQL node, use `mysqldump` on the noncluster server to export the database to a file named `world.sql` (for example, in the `/tmp` directory). Then modify the table definitions as just described and import the file into the SQL node of the cluster like this:

```
shell> mysql world < /tmp/world.sql
```

If you save the file to a different location, adjust the preceding instructions accordingly.

Running `SELECT` queries on the SQL node is no different from running them on any other instance of a MySQL server. To run queries from the command line, you first need to log in to the MySQL Monitor in the usual way (specify the `root` password at the `Enter password:` prompt):

```
shell> mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.6.51-ndb-7.4.31

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

We simply use the MySQL server's `root` account and assume that you have followed the standard security precautions for installing a MySQL server, including setting a strong `root` password. For more information, see Section 2.10.4, “Securing the Initial MySQL Accounts”.

It is worth taking into account that Cluster nodes do *not* make use of the MySQL privilege system when accessing one another. Setting or changing MySQL user accounts (including the `root` account) effects only applications that access the SQL node, not interaction between nodes. See Section 18.5.16.2, “NDB Cluster and MySQL Privileges”, for more information.

If you did not modify the `ENGINE` clauses in the table definitions prior to importing the SQL script, you should run the following statements at this point:

```
mysql> USE world;
mysql> ALTER TABLE City ENGINE=NDBCLUSTER;
mysql> ALTER TABLE Country ENGINE=NDBCLUSTER;
mysql> ALTER TABLE CountryLanguage ENGINE=NDBCLUSTER;
```

Selecting a database and running a `SELECT` query against a table in that database is also accomplished in the usual manner, as is exiting the MySQL Monitor:

```
mysql> USE world;
mysql> SELECT Name, Population FROM City ORDER BY Population DESC LIMIT 5;
+-----------+------------+
| Name      | Population |
+-----------+------------+
| Bombay    |   10500000 |
| Seoul     |    9981619 |
| São Paulo |    9968485 |
| Shanghai  |    9696300 |
| Jakarta   |    9604900 |
+-----------+------------+
5 rows in set (0.34 sec)

mysql> \q
Bye

shell>
```

Applications that use MySQL can employ standard APIs to access `NDB` tables. It is important to remember that your application must access the SQL node, and not the management or data nodes. This brief example shows how we might execute the `SELECT` statement just shown by using the PHP 5.X `mysqli` extension running on a Web server elsewhere on the network:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
  "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <meta http-equiv="Content-Type"
          content="text/html; charset=iso-8859-1">
  <title>SIMPLE mysqli SELECT</title>
</head>
<body>
<?php
  # connect to SQL node:
  $link = new mysqli('198.51.100.20', 'root', 'root_password', 'world');
  # parameters for mysqli constructor are:
  #   host, user, password, database

  if( mysqli_connect_errno() )
    die("Connect failed: " . mysqli_connect_error());

  $query = "SELECT Name, Population
            FROM City
            ORDER BY Population DESC
            LIMIT 5";

  # if no errors...
  if( $result = $link->query($query) )
  {
?>
<table border="1" width="40%" cellpadding="4" cellspacing ="1">
  <tbody>
  <tr>
    <th width="10%">City</th>
    <th>Population</th>
  </tr>
<?
    # then display the results...
    while($row = $result->fetch_object())
      printf("<tr>\n  <td align=\"center\">%s</td><td>%d</td>\n</tr>\n",
              $row->Name, $row->Population);
?>
  </tbody
</table>
<?
  # ...and verify the number of rows that were retrieved
    printf("<p>Affected rows: %d</p>\n", $link->affected_rows);
  }
  else
    # otherwise, tell us what went wrong
    echo mysqli_error();
```

```
  # free the result set and the mysqli connection object
  $result->close();
  $link->close();
?>
</body>
</html>
```

We assume that the process running on the Web server can reach the IP address of the SQL node.

In a similar fashion, you can use the MySQL C API, Perl-DBI, Python-mysql, or MySQL Connectors to perform the tasks of data definition and manipulation just as you would normally with MySQL.

18.2.6 Safe Shutdown and Restart of NDB Cluster

To shut down the cluster, enter the following command in a shell on the machine hosting the management node:

```
shell> ndb_mgm -e shutdown
```

The `-e` option here is used to pass a command to the `ndb_mgm` client from the shell. (See Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”, for more information about this option.) The command causes the `ndb_mgm`, `ndb_mgmd`, and any `ndbd` or `ndbmtd` processes to terminate gracefully. Any SQL nodes can be terminated using `mysqladmin shutdown` and other means. On Windows platforms, assuming that you have installed the SQL node as a Windows service, you can use `SC STOP service_name` or `NET STOP service_name`.

To restart the cluster on Unix platforms, run these commands:

- On the management host (`198.51.100.10` in our example setup):

  ```
  shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
  ```

- On each of the data node hosts (`198.51.100.30` and `198.51.100.40`):

  ```
  shell> ndbd
  ```

- Use the `ndb_mgm` client to verify that both data nodes have started successfully.

- On the SQL host (`198.51.100.20`):

  ```
  shell> mysqld_safe &
  ```

On Windows platforms, assuming that you have installed all NDB Cluster processes as Windows services using the default service names (see Section 18.2.2.4, “Installing NDB Cluster Processes as Windows Services”), you can restart the cluster as follows:

- On the management host (`198.51.100.10` in our example setup), execute the following command:

  ```
  C:\> SC START ndb_mgmd
  ```

- On each of the data node hosts (`198.51.100.30` and `198.51.100.40`), execute the following command:

  ```
  C:\> SC START ndbd
  ```

- On the management node host, use the `ndb_mgm` client to verify that the management node and both data nodes have started successfully (see Section 18.2.2.3, “Initial Startup of NDB Cluster on Windows”).

- On the SQL node host (`198.51.100.20`), execute the following command:

  ```
  C:\> SC START mysql
  ```

In a production setting, it is usually not desirable to shut down the cluster completely. In many cases, even when making configuration changes, or performing upgrades to the cluster hardware or software (or both), which require shutting down individual host machines, it is possible to do so without shutting down the cluster as a whole by performing a *rolling restart* of the cluster. For more information about doing this, see Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”.

18.2.7 Upgrading and Downgrading NDB Cluster

This section provides information about NDB Cluster software and table file compatibility between different NDB Cluster 7.3 releases with regard to performing upgrades and downgrades as well as compatibility matrices and notes. You should already be familiar with installing and configuring NDB Cluster prior to attempting an upgrade or downgrade. See Section 18.3, “Configuration of NDB Cluster”.

Schema operations, including SQL DDL statements, cannot be performed while any data nodes are restarting, and thus during an online upgrade or downgrade of the cluster. For other information regarding the rolling restart procedure used to perform an online upgrade, see Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”.

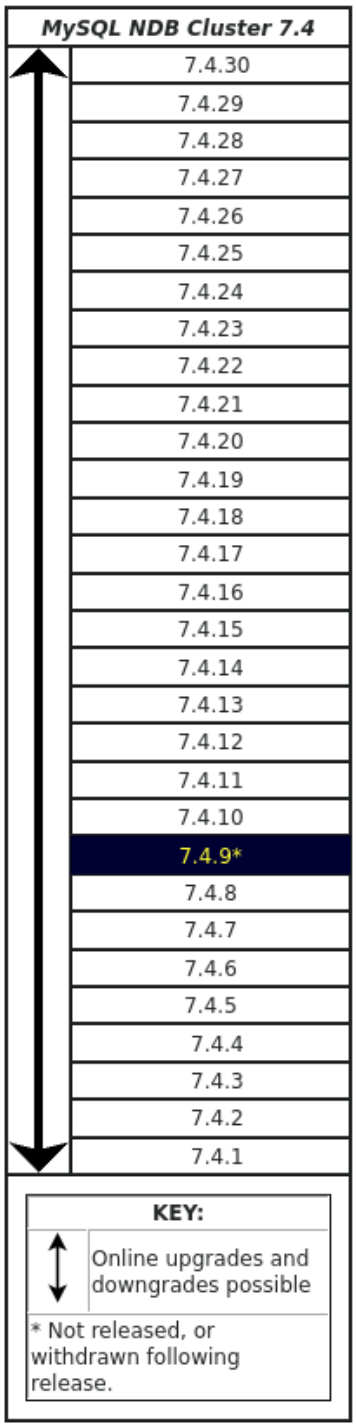
Important

Only compatibility between MySQL versions with regard to `NDBCLUSTER` is taken into account in this section, and there are likely other issues to be considered. *As with any other MySQL software upgrade or downgrade, you are strongly encouraged to review the relevant portions of the MySQL Manual for the MySQL versions from which and to which you intend to migrate, before attempting an upgrade or downgrade of the NDB Cluster software*. See Section 2.11, “Upgrading MySQL”.

The tables shown here provide information on NDB Cluster upgrade and downgrade compatibility among different releases of NDB Cluster 7.3 and of NDB Cluster 7.4, respectively. Additional notes about upgrades and downgrades to, from, or within the NDB Cluster 7.3 and NDB Cluster 7.4 release series can be found following the tables.

Upgrades and Downgrades, NDB Cluster 7.4

Figure 18.5 NDB Cluster Upgrade and Downgrade Compatibility, MySQL NDB Cluster 7.4



Version support. NDB Cluster 7.3 GA releases (7.3.2 and later)are supported for upgrades to NDB Cluster 7.4 (7.4.4 and later):

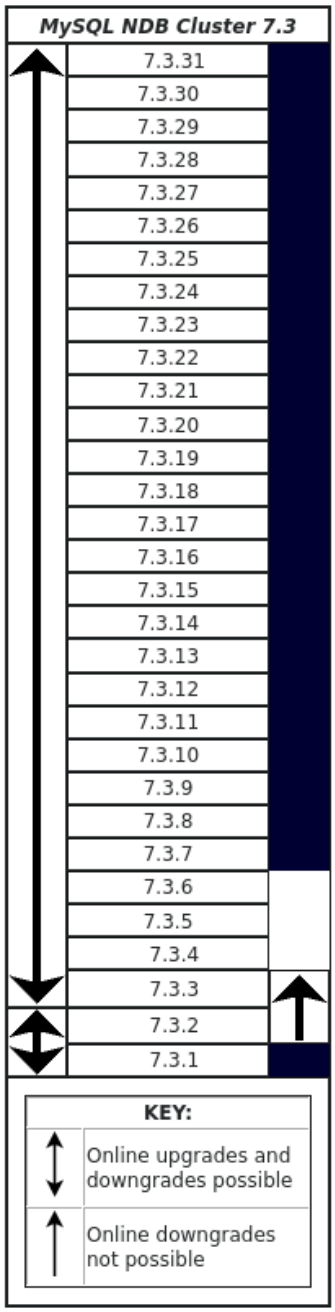
NDB 7.4.10 Replacement Release. Shortly after the release of NDB 7.4.9, a regression was discovered that adversely affected node and system restarts (Bug #22582233). This issue was known to affect NDB 7.4.8 as well. NDB 7.4.10—incorporating a fix for this regression, but otherwise identical to NDB 7.4.9—was released shortly thereafter as a replacement. *Users of the NDB 7.4 series are advised to bypass the 7.4.8 and 7.4.9 releases and to upgrade directly to NDB 7.4.10 (or later)*.

Known Issues—NDB 7.4

- Prior to NDB 7.4.4, when upgrading from NDB 7.3 to NDB 7.4, the first new data node binary to be started caused the master node (still running NDB 7.3) to fail, then itself failed. (Bug #20608889)
- Prior to NDB 7.4.3, `mysql_upgrade` failed to drop and recreate `ndbinfo`. (Bug #74863, Bug #20031425) In addition, when running `mysql_upgrade` on an NDB Cluster SQL node, the expected drop of the `performance_schema` database on this node was instead performed on all SQL nodes connected to the cluster. (Bug #200328691)

Upgrades and Downgrades, NDB Cluster 7.3

Figure 18.6 NDB Cluster Upgrade and Downgrade Compatibility, MySQL NDB Cluster 7.3



Known Issues—NDB 7.3

- Prior to NDB 7.3.8, `mysql_upgrade` failed to drop and recreate `ndbinfo`. (Bug #74863, Bug #20031425) In addition, when running `mysql_upgrade` on an NDB Cluster SQL node, the expected drop of the `performance_schema` database on this node was instead performed on all SQL nodes connected to the cluster. (Bug #200328691)
- NDB API, ClusterJ, and other applications used with recent releases of NDB Cluster 6.3 and later should continue to work with NDB 7.3.2 and later without rewriting or recompiling.
- It is not possible to downgrade online to NDB 7.3.2 or earlier from NDB 7.3.3 or later. Online upgrades from NDB 7.3.2 to later NDB Cluster 7.3 releases are supported.

18.2.8 The NDB Cluster Auto-Installer (No longer supported)

Note

This feature has been removed from NDB Cluster, and is no longer supported. See Section 18.1.4, “What is New in MySQL NDB Cluster”, for more information.

This section describes the web-based graphical configuration installer included as part of the NDB Cluster distribution beginning with NDB 7.3.1. Topics discussed include an overview of the installer and its parts, software and other requirements for running the installer, navigating the GUI, and using the installer to set up and start or stop an NDB Cluster on one or more host computers.

The NDB Cluster Auto-Installer is made up of two components. The front end is a GUI client implemented as a Web page that loads and runs in a standard Web browser such as Firefox or Microsoft Internet Explorer. The back end is a server process (`ndb_setup.py`) that runs on the local machine or on another host to which you have access.

These two components (client and server) communicate with each other using standard HTTP requests and responses. The back end can manage NDB Cluster software programs on any host where the back end user has granted access. If the NDB Cluster software is on a different host, the back end relies on SSH for access.

18.2.8.1 NDB Cluster Auto-Installer Requirements

This section provides information on supported operating platforms and software, required software, and other prerequisites for running the NDB Cluster Auto-Installer.

Supported platforms. The NDB Cluster Auto-Installer is available with most NDB Cluster 7.3 and later distributions for recent versions of Linux, Windows, Solaris, and macOS. For more detailed information about platform support for NDB Cluster and the NDB Cluster Auto-Installer, see https://www.mysql.com/support/supportedplatforms/cluster.html.

Supported Web browsers. The Web-based installer is supported with recent versions of Firefox and Microsoft Internet Explorer. It should also work with recent versions of Opera, Safari, and Chrome, although we have not thoroughly tested for compability with these browsers.

Required software—server. The following software must be installed on the host where the Auto-Installer is run:

- **Python 2.6 or higher.** The Auto-Installer requires the Python interpreter and standard libraries. If these are not already installed on the system, you may be able to add them using the system's package manager. Otherwise, they can be downloaded from http://python.org/download/.
- **Paramiko 1.7.7.1 or higher.** You can download this from http://www.lag.net/paramiko/ if it is not available from your system's package manager.

- **Pycrypto version 2.6 or higher.** This cryptography module is required by Paramiko. If it is not available using your system's package manage, you can download it from https://www.dlitz.net/software/pycrypto/.

All of the software in the preceding list is included in the Windows version of the configuration tool, and does not need to be installed separately.

Required software—remote hosts. The only software required for remote hosts where you wish to deploy NDB Cluster nodes is the SSH server, which is usually installed by default on Linux and Solaris systems. Several alternatives are available for Windows; for an overview of these, see http://en.wikipedia.org/wiki/Comparison_of_SSH_servers.

An additional requirement when using multiple hosts is that it is possible to authenticate to any of the remote hosts using SSH and the proper keys or user credentials, as discussed in the next few paragraphs:

Authentication and security. Three basic security or authentication mechanisms for remote access are available to the Auto-Installer, which we list and describe here:

- **SSH.** A secure shell connection is used to enable the back end to perform actions on remote hosts. For this reason, an SSH server must be running on the remote host. In addition, the operating system user running the installer must have access to the remote server, either with a user name and password, or by using public and private keys.

Important

You should never use the system `root` account for remote access, as this is extremely insecure. In addition, `mysqld` cannot normally be started by system `root`. For these and other reasons, you should provide SSH credentials for a regular user account on the target system, and not for system `root`. For more information about this issue, see Section 6.1.5, “How to Run MySQL as a Normal User”.

- **HTTPS.** Remote communication between the Web browser front end and the back end is not encrypted by default, which means that information such as the user's SSH password is transmitted as cleartext that is readable to anyone. For communication from a remote client to be encrypted, the back end must have a certificate, and the front end must communicate with the back end using HTTPS rather than HTTP. Enabling HTTPS is accomplished most easily through issuing a self-signed certificate. Once the certificate is issued, you must make sure that it is used. You can do this by starting `ndb_setup.py` from the command line with the `--use-https` and `--cert-file` options.

- **Certificate-based authentication.** The back end `ndb_setup.py` process can execute commands on the local host as well as remote hosts. This means that anyone connecting to the back end can take charge of how commands are executed. To reject unwanted connections to the back end, a certificate may be required for authentication of the client. In this case, a certificate must be issued by the user, installed in the browser, and made available to the back end for authentication purposes. You can enact this requirement (together with or in place of password or key authentication) by starting `ndb_setup.py` with the `--ca-certs-file` option.

There is no need or requirement for secure authentication when the client browser is running on the same host as the Auto-Installer back end.

See also Section 18.5.16, “NDB Cluster Security Issues”, which discusses security considerations to take into account when deploying NDB Cluster, as well as Chapter 6, *Security*, for more general MySQL security information.

18.2.8.2 Using the NDB Cluster Auto-Installer

The NDB Cluster Auto-Installer consists of several pages, each corresponding to a step in the process used to configure and deploy an NDB Cluster, and listed here:

- **Welcome**: Begin using the Auto-Installer by choosing either to configure a new NDB Cluster, or to continue configuring an existing one.
- **Define Cluster**: Set basic information about the cluster as a whole, such as name, hosts, and load type. Here you can also set the SSH authentication type for accessing remote hosts, if needed.
- **Define Hosts**: Identify the hosts where you intend to run NDB Cluster processes.
- **Define Processes**: Assign one or more processes of a given type or types to each cluster host.
- **Define Attributes**: Set configuration attributes for processes or types of processes.
- **Deploy Cluster**: Deploy the cluster with the configuration set previously; start and stop the deployed cluster.

The following sections describe in greater detail the purpose and function of each of these pages, in the order just listed.

Starting the NDB Cluster Auto-Installer

The Auto-Installer is provided together with the NDB Cluster software. (See Section 18.2, "NDB Cluster Installation".) The present section explains how to start the installer. You can do by invoking the `ndb_setup.py` executable.

Important

You should run the `ndb_setup.py` as a normal user; no special privileges are needed to do so. You should *not* run this program as the `mysql` user, or using the system `root` or Administrator account; doing so may cause the installation to fail.

`ndb_setup.py` is found in the `bin` within the NDB Cluster installation directory; a typical location might be `/usr/local/mysql/bin` on a Linux system or `C:\Program Files\MySQL\MySQL Server 5.6\bin` on a Windows system, but this can vary according to where the NDB Cluster software is installed on your system.

On Windows, you can also start the installer by running `setup.bat` in the NDB Cluster installation directory. When invoked from the command line, it accepts the same options as does `ndb_setup.py`.

`ndb_setup.py` can be started with any of several options that affect its operation, but it is usually sufficient to allow the default settings be used, in which case you can start `ndb_setup.py` by either of the following two methods:

1. Navigate to the NDB Cluster `bin` directory in a terminal and invoke it from the command line, without any additional arguments or options, like this:

   ```
   shell> ndb_setup
   ```

   This works regardless of operating platform.

2. Navigate to the NDB Cluster `bin` directory in a file browser (such Windows Explorer on Windows, or Konqueror, Dolphin, or Nautilus on Linux) and activate (usually by double-clicking) the **ndb_setup.py** file icon. This works on Windows, and should work with most common Linux desktops as well.

   On Windows, you can also navigate to the NDB Cluster installation directory and activate the **setup.bat** file icon.

In either case, once `ndb_setup.py` is invoked, the Auto-Installer's **Welcome** screen should open in the system's default Web browser.

In some cases, you may wish to use non-default settings for the installer, such as specifying a different port for the Auto-Installer's included Web server to run on, in which case you must invoke

`ndb_setup.py` with one or more startup options with values overriding the necessary defaults. The same startup options can be used on Windows systems with the `setup.bat` file supplied for such platforms in the NDB Cluster software distribution. This can be done using the command line, but if you want or need to start the installer from a desktop or file browser while emplying one or more of these options, it is also possible to create a script or batch file containing the proper invocation, then to double-click its file icon in the file browser to start the installer. (On Linux systems, you might also need to make the script file executable first.) For information about advanced startup options for the NDB Cluster Auto-Installer, see Section 18.4.25, “`ndb_setup.py` — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”.

NDB Cluster Auto-Installer Welcome Screen

The **Welcome** screen is loaded in the default browser when `ndb_setup.py` is invoked, as shown here:

Figure 18.7 The NDB Cluster Auto-Installer Welcome screen (Closeup)

This screen provides the following two choices for entering the installer, one of which must be selected to continue:

1. **Create New NDB Cluster**: Start the Auto-Installer with a completely new cluster to be set up and deployed.

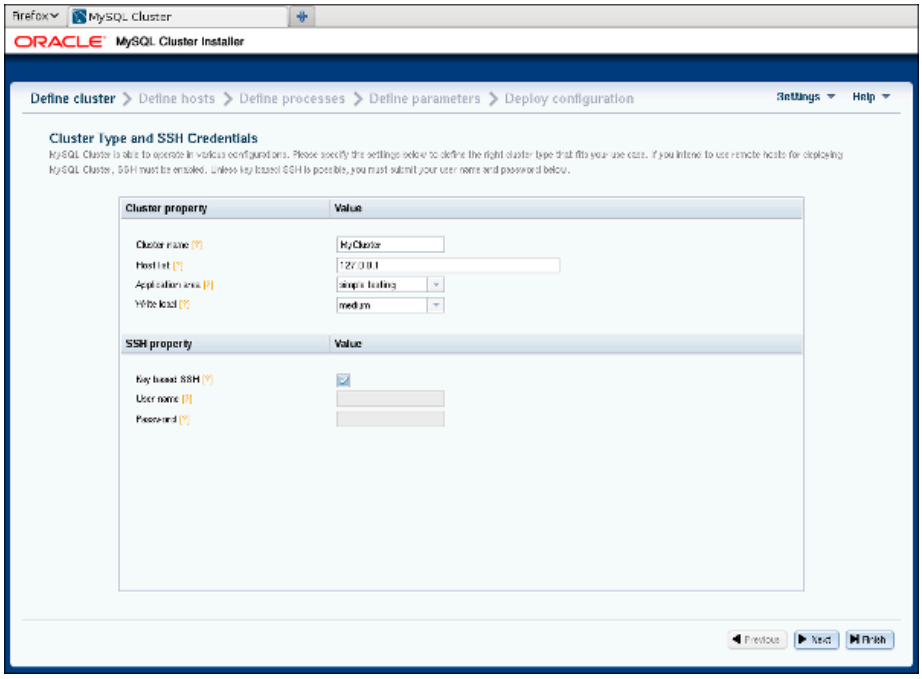
2. **Continue Previous Cluster Configuration**: Start the Auto-Installer at the same point where the previous session ended, with all previous settings preserved.

The second option requires that the browser be able to access its cookies from the previous session, as these provide the mechanism by which configuration and other information generated during a session is stored. In other words, to continue the previous session with the Auto-Installer, you must use the same web browser running on the same host as you did for the previous session.

NDB Cluster Auto-Installer Define Cluster Screen

The **Define Cluster** screen is the first screen to appear following the choice made in the **Welcome** screen, and is used for setting general properties of the cluster. The layout of the **Define Cluster** screen is shown here:

Figure 18.8 The NDB Cluster Auto-Installer Define Cluster screen



The **Define Cluster** screen allows you to set a number of general properties for the cluster, as described in this list:

- **Cluster name**: A name that identifies the cluster. The default is `MyCluster`.
- **Host list**: A comma-delimited list of one or more hosts where cluster processes should run. By default, this is `127.0.0.1`. If you add remote hosts to the list, you must be able to connect to them using the **SSH Credentials** supplied.
- **Application type**: Choose one of the following:

  1. **Simple testing**: Minimal resource usage for small-scale testing. This the default. *Not intended for production environments*.
  2. **Web**: Maximize performance for the given hardware.
  3. **Real-time**: Maximize performance while maximizing sensitivity to timeouts in order to minimize the time needed to detect failed cluster processes.

- **Write load**: Choose a level for the anticipated number of writes for the cluster as a whole. You can choose any one of the following levels:

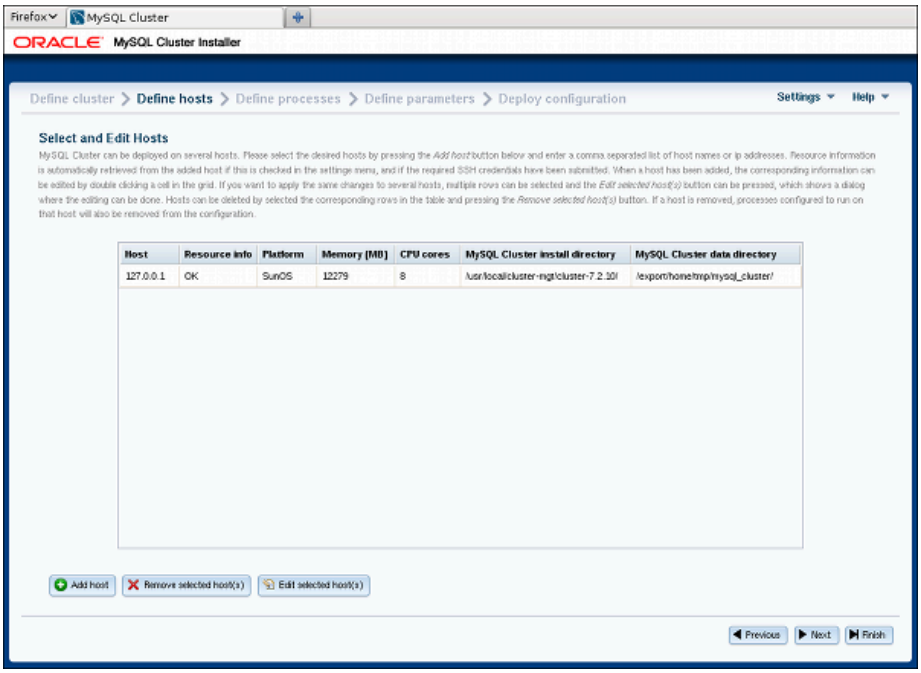
  1. **Low**: The expected load includes fewer than 100 write transactions for second.
  2. **Medium**: The expected load includes 100 to 1000 write transactions per second.
  3. **High**: The expected load includes more than 1000 write transactions per second.

- **SSH Credentials**: Choose **Key-Based SSH** or enter **User** and **Password** credentials. The SSH key or a user name with password is required for connecting to any remote hosts specified in the **Host list**. By default, **Key-Based SSH** is selected, and the **User** and **Password** fields are blank.

NDB Cluster Auto-Installer Define Hosts Screen

The **Define Hosts** screen, shown here, provides a means of viewing and specifying several key properties of each cluster host:

Figure 18.9 NDB Cluster Define Hosts screen



The hosts currently entered are displayed in the grid with various pieces of information. You can add hosts by clicking the **Add hosts** button and entering a list of one or more comma-separated host names, IP addresses, or both (as when editing the host list on the **Define Cluster** screen).

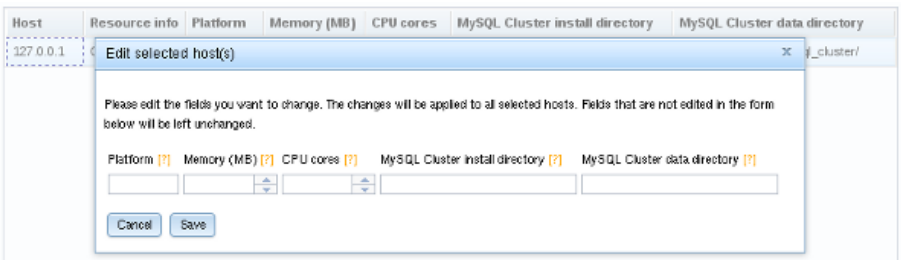
Similarly, you can remove one or more hosts using the button labelled **Remove selected host(s)**. When you remove a host in this fashion, any process which was configured for that host is also removed.

If **Automatically get resource information for new hosts** is checked in the **Settings** menu, the Auto-Installer attempts to retrieve the platform name, amount of memory, and number of CPU cores and to fill these in automatically. The status of this is displayed in the *Resource info* column. Fetching the information from remote hosts is not instantaneous and may take some time, particularly from remote hosts running Windows.

If the SSH user credentials on the *Define Cluster* screen are changed, the tool tries to refresh the hardware information from any hosts for which information is missing. However, if a given field has already been edited, the user-supplied information is *not* overwritten by any value fetched from that host.

The hardware resource information, platform name, installation directory, and data directory can be edited by the user by clicking the corresponding cell in the grid, by selecting one or more hosts and clicking the button labelled **Edit selected host(s)**. This causes a dialog box to appear, in which these fields can be edited, as shown here:

Figure 18.10 NDB Cluster Auto-Installer Edit Hosts dialog

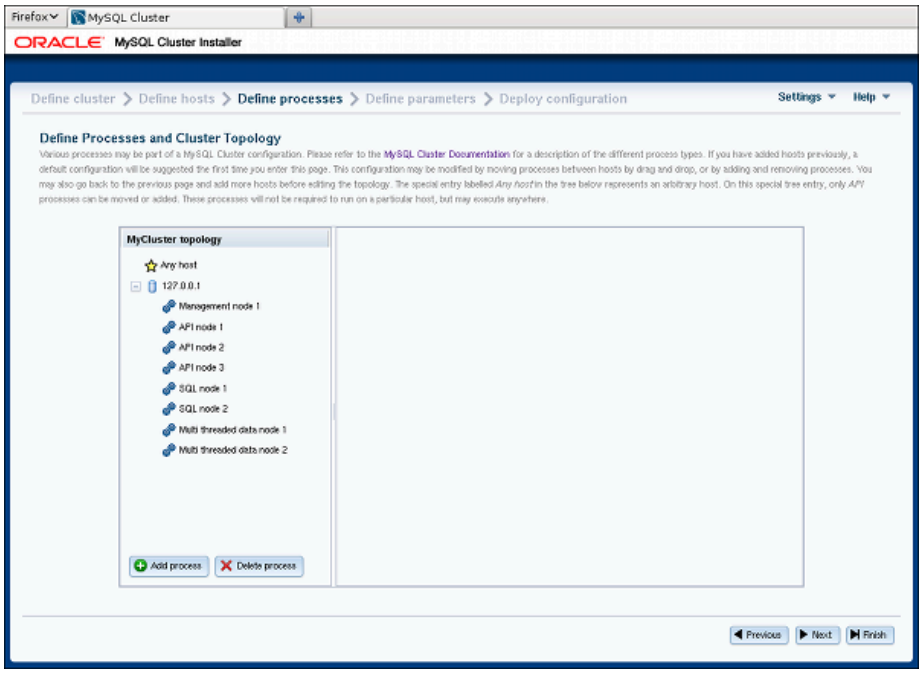


When more than one host is selected, any edited values are applied to all selected hosts.

NDB Cluster Auto-Installer Define Processes Screen

The **Define Processes** screen, shown here, provides a way to assign NDB Cluster processes (nodes) to cluster hosts:

Figure 18.11 NDB Cluster Auto-Installer Define Processes dialog



This screen contains a process tree showing cluster hosts and processes set up to run on each one, as well as a panel which displays information about the item currently selected in the tree.

When this screen is accessed for the first time for a given cluster, a default set of processes is defined for you, based on the number of hosts. If you later return to the **Define Hosts** screen, remove all hosts, and add new hosts, this also causes a new default set of processes to be defined.

NDB Cluster processes are of the following types:

- **Management node.** Performs administrative tasks such as stopping individual data nodes, querying node and cluster status, and making backups. Executable: `ndb_mgmd`.
- **Single-threaded data node.** Stores data and executes queries. Executable: `ndbd`.
- **Multi threaded data node.** Stores data and executes queries with multiple worker threads executing in parallel. Executable: `ndbmtd`.
- **SQL node.** MySQL server for executing SQL queries against `NDB`. Executable: `mysqld`.
- **API node.** A client accessing data in `NDB` by means of the NDB API or other low-level client API, rather than by using SQL. See MySQL NDB Cluster API Developer Guide, for more information.

For more information about process (node) types, see Section 18.1.1, “NDB Cluster Core Concepts”.

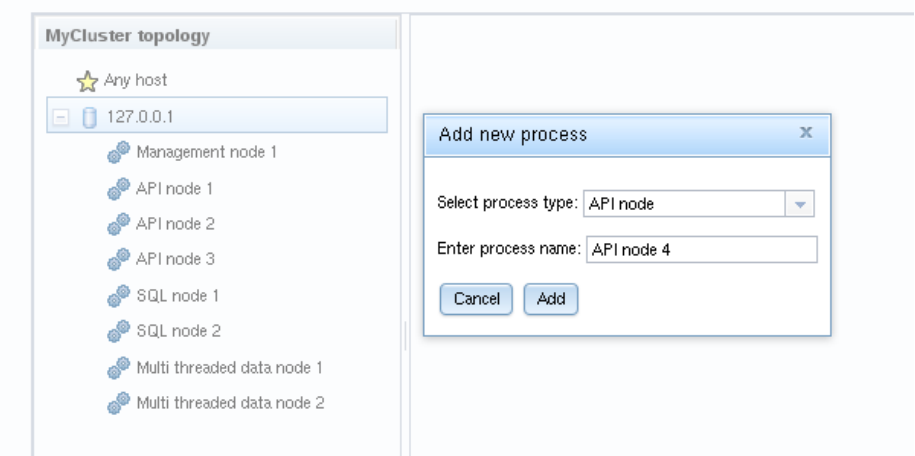
Processes shown in the tree are numbered sequentially by type, for each host—for example, `SQL node 1`, `SQL node 2`, and so on—to simplify identification.

Each management node, data node, or SQL process must be assigned to a specific host, and is not allowed to run on any other host. An API node *may* be assigned to a single host, but this is not

required. Instead, you can assign it to the special **Any host** entry which the tree also contains in addition to any other hosts, and which acts as a placeholder for processes that are allowed to run on any host. *Only API processes may use this **Any host** entry*.

Adding processes. To add a new process to a given host, either right-click that host's entry in the tree, then select the **Add process** popup when it appears, or select a host in the process tree, and press the **Add process** button below the process tree. Performing either of these actions opens the add process dialog, as shown here:

Figure 18.12 NDB Cluster Auto-Installer Add Process Dialog



Here you can select from among the available process types described earlier this section; you can also enter an arbitrary process name to take the place of the suggested value, if desired.

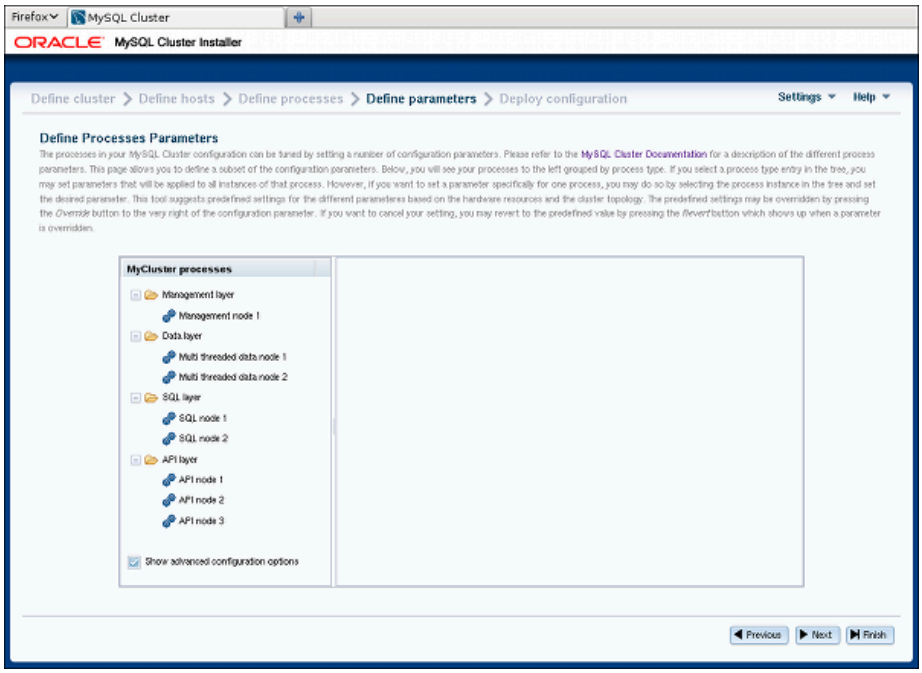
Removing processes. To delete a process, right-click on a process in the tree and select **delete process** from the pop up menu that appears, or select a process, then use the **delete process** button below the process tree.

When a process is selected in the process tree, information about that process is displayed in the information panel, where you can change the process name and possibly its type. *Important*: Currently, you can change a single-threaded data node (`ndbd`) to a multithreaded data node (`ndbmtd`), or the reverse, only; no other process type changes are allowed. If you want to make a change between any other process types, you must delete the original process first, then add a new process of the desired type.

NDB Cluster Auto-Installer Define Attributes Screen

This screen has a layout similar to that of the **Define Processes** screen, including a process tree. Unlike that screen's tree, the **Define Attributes** process tree is organized by process or node type, with single-threaded and multithreaded data nodes considered to be of the same type for this purpose, in groups labelled **Management Layer**, **Data Layer**, **SQL Layer**, and **API Layer**. An information panel displays information regarding the item currently selected. The **Define Attributes** screen is shown here:

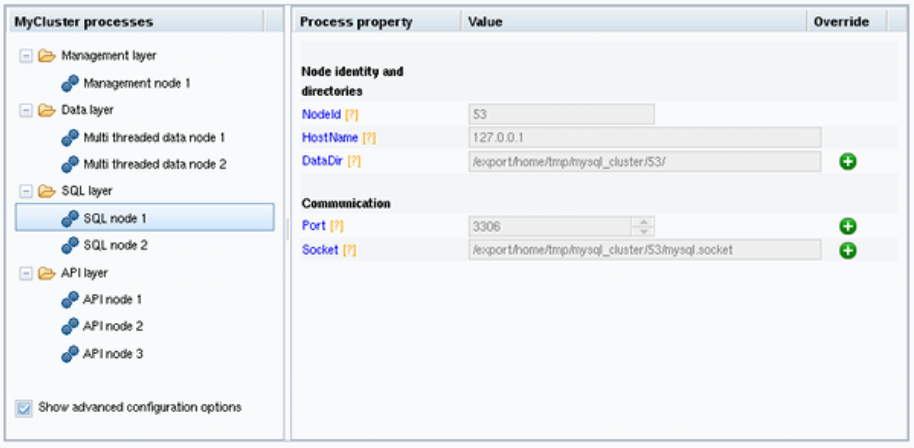
Figure 18.13 NDB Cluster Auto-Installer Define Attributes screen



The check box labelled **Show advanced configuration**, when checked, makes advanced options visible in the information pane. These options are set and used whether or not they are visible.

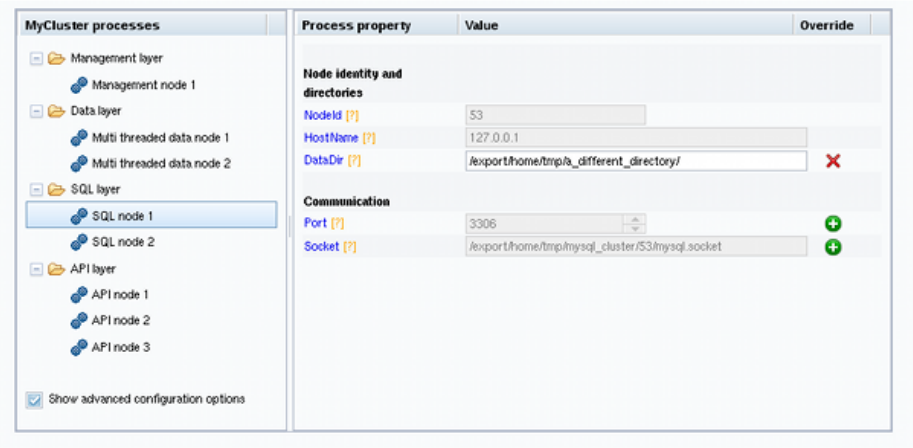
You can edit attributes for a single process by selecting that process from the tree, or for all processes of the same type in the cluster by selecting one of the **Layer** folders. A per-process value set for a given attribute overrides any per-group setting for that attribute that would otherwise apply to the process in question. An example of such an information panel (for an SQL process) is shown here:

Figure 18.14 Define Attributes Detail With SQL Process Attributes



For some of the attributes shown in the information panel, a button bearing a plus sign is displayed, which means that the value of the attribute can be overridden. This **+** button activates an input widget for the attribute, enabling you to change its value. When the value has been overridden, this button changes into a button showing an **X**, as shown here:

Figure 18.15 Define Attributes Detail, Overriding Attribute Default Value



Clicking the **X** button next to an attribute undoes any changes made to it; it immediately reverts to the predefined value.

All configuration attributes have predefined values calculated by the installer, based such factors as host name, node ID, node type, and so on. In most cases, these values may be left as they are. If you are not familiar with it already, it is highly recommended that you read the applicable documentation before making changes to any of the attribute values. To make finding this information easier, each attribute name shown in the information panel is linked to its description in the online NDB Cluster documentation.

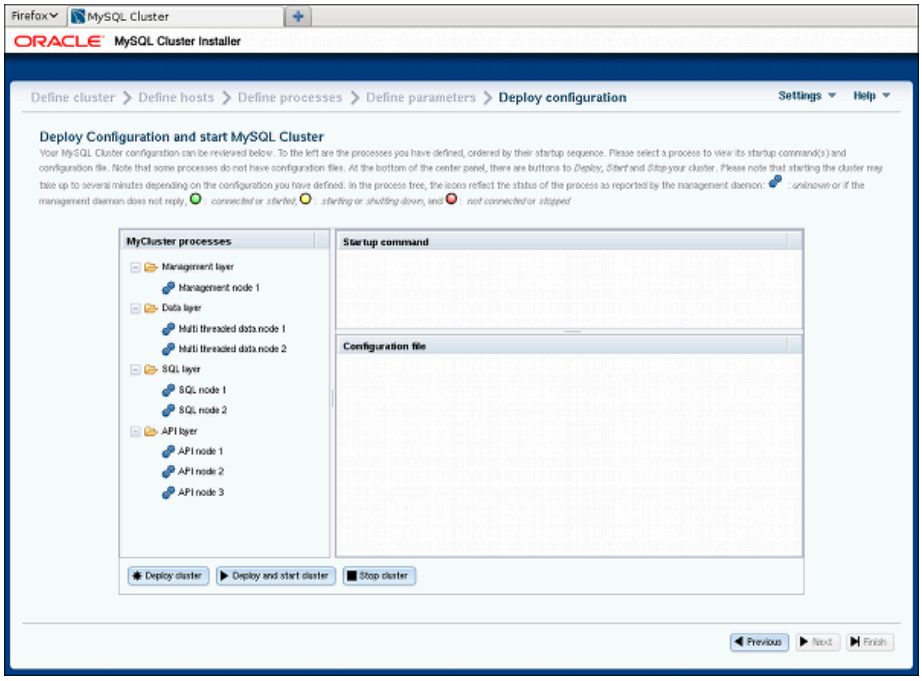
NDB Cluster Auto-Installer Deploy Cluster Screen

This screen allows you to perform the following tasks:

- Review process startup commands and configuration files to be applied
- Distribute configuration files by creating any necessary files and directories on all cluster hosts—that is, *deploy* the cluster as presently configured
- Start and stop the cluster

The **Deploy Cluster** screen is shown here:

Figure 18.16 NDB Cluster Auto-Installer Deploy Cluster Configuration screen



Like the **Define Attributes** screen, this screen features a process tree which is organized by process type. Next to each process in the tree is a status icon indicating the current status of the process: connected (`CONNECTED`), starting (`STARTING`), running (`STARTED`), stopping (`STOPPING`), or disconnected (`NO_CONTACT`). The icon shows green if the process is connected or running; yellow if it is starting or stopping; red if the process is stopped or cannot be contacted by the management server.

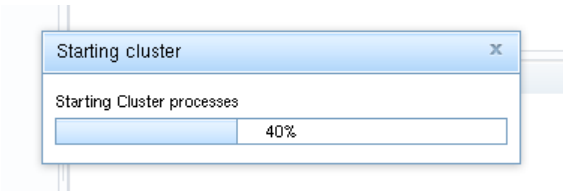
This screen also contains two information panels, one showing the startup command or commands needed to start the selected process. (For some processes, more than one command may be required—for example, if initialization is necessary.) The other panel shows the contents of the configuration file, if any, for the given process; currently, the management node process is only type of process having a configuration file. Other process types are configured using command-line parameters when starting the process, or by obtaining configuration information from the management nodes as needed in real time.

This screen also contains three buttons, labelled as and performing the functions described in the following list:

- **Deploy cluster**: Verify that the configuration is valid. Create any directories required on the cluster hosts, and distribute the configuration files onto the hosts. A progress bar shows how far the deployment has proceeded.
- **Start cluster**: The cluster is deployed as with **Deploy cluster**, after which all cluster processes are started in the correct order.

  Starting these processes may take some time. If the estimated time to completion is too large, the installer provides an opportunity to cancel or to continue of the startup procedure. A progress bar indicates the current status of the startup procedure, as shown here:

  Figure 18.17 Progress Bar With Status of Node Startup Process



The process status icons adjoining the process tree mentioned previously also update with the status of each process.

- **Stop cluster**: After the cluster has been started, you can stop it using this button. As with starting the cluster, cluster shutdown is not instantaneous, and may require some time complete. A progress bar, similar to that displayed during cluster startup, shows the approximate current status of the cluster shutdown procedure, as do the process status icons adjoining the process tree.

Prior to NDB 7.3.3, SQL nodes were started with all options employed on the command line. Beginning with NDB 7.3.3, the Auto-Installer generates a `my.cnf` file containing the appropriate options for each `mysqld` process in the cluster. (Bug #16994782)

18.3 Configuration of NDB Cluster

A MySQL server that is part of an NDB Cluster differs in one chief respect from a normal (nonclustered) MySQL server, in that it employs the `NDB` storage engine. This engine is also referred to sometimes as `NDBCLUSTER`, although `NDB` is preferred.

To avoid unnecessary allocation of resources, the server is configured by default with the `NDB` storage engine disabled. To enable `NDB`, you must modify the server's `my.cnf` configuration file, or start the server with the `--ndbcluster` option.

This MySQL server is a part of the cluster, so it also must know how to access a management node to obtain the cluster configuration data. The default behavior is to look for the management node on `localhost`. However, should you need to specify that its location is elsewhere, this can be done in `my.cnf`, or with the `mysql` client. Before the `NDB` storage engine can be used, at least one management node must be operational, as well as any desired data nodes.

For more information about `--ndbcluster` and other `mysqld` options specific to NDB Cluster, see MySQL Server Options for NDB Cluster.

For general information about installing NDB Cluster, see Section 18.2, “NDB Cluster Installation”.

18.3.1 Quick Test Setup of NDB Cluster

To familiarize you with the basics, we describe the simplest possible configuration for a functional NDB Cluster. After this, you should be able to design your desired setup from the information provided in the other relevant sections of this chapter.

First, you need to create a configuration directory such as `/var/lib/mysql-cluster`, by executing the following command as the system `root` user:

```
shell> mkdir /var/lib/mysql-cluster
```

In this directory, create a file named `config.ini` that contains the following information. Substitute appropriate values for `HostName` and `DataDir` as necessary for your system.

```
# file "config.ini" - showing minimal setup consisting of 1 data node,
# 1 management server, and 3 MySQL servers.
# The empty default sections are not required, and are shown only for
# the sake of completeness.
# Data nodes must provide a hostname but MySQL Servers are not required
# to do so.
# If you don't know the hostname for your machine, use localhost.
# The DataDir parameter also has a default value, but it is recommended to
# set it explicitly.
# Note: [db], [api], and [mgm] are aliases for [ndbd], [mysqld], and [ndb_mgmd],
# respectively. [db] is deprecated and should not be used in new installations.

[ndbd default]
NoOfReplicas= 1

[mysqld  default]
[ndb_mgmd default]
```

```
[tcp default]

[ndb_mgmd]
HostName= myhost.example.com

[ndbd]
HostName= myhost.example.com
DataDir= /var/lib/mysql-cluster

[mysqld]
[mysqld]
[mysqld]
```

You can now start the `ndb_mgmd` management server. By default, it attempts to read the `config.ini` file in its current working directory, so change location into the directory where the file is located and then invoke `ndb_mgmd`:

```
shell> cd /var/lib/mysql-cluster
shell> ndb_mgmd
```

Then start a single data node by running `ndbd`:

```
shell> ndbd
```

For command-line options which can be used when starting `ndbd`, see Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

By default, `ndbd` looks for the management server at `localhost` on port 1186.

Note

If you have installed MySQL from a binary tarball, you must specify the path of the `ndb_mgmd` and `ndbd` servers explicitly. (Normally, these are found in `/usr/local/mysql/bin`.)

Finally, change location to the MySQL data directory (usually `/var/lib/mysql` or `/usr/local/mysql/data`), and make sure that the `my.cnf` file contains the option necessary to enable the NDB storage engine:

```
[mysqld]
ndbcluster
```

You can now start the MySQL server as usual:

```
shell> mysqld_safe --user=mysql &
```

Wait a moment to make sure the MySQL server is running properly. If you see the notice `mysql ended`, check the server's `.err` file to find out what went wrong.

If all has gone well so far, you now can start using the cluster. Connect to the server and verify that the `NDBCLUSTER` storage engine is enabled:

```
shell> mysql
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.6.51

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SHOW ENGINES\G
...
*************************** 12. row ***************************
Engine: NDBCLUSTER
Support: YES
Comment: Clustered, fault-tolerant, memory-based tables
*************************** 13. row ***************************
Engine: NDB
Support: YES
Comment: Alias for NDBCLUSTER
...
```

The row numbers shown in the preceding example output may be different from those shown on your system, depending upon how your server is configured.

Try to create an `NDBCLUSTER` table:

```
shell> mysql
mysql> USE test;
Database changed

mysql> CREATE TABLE ctest (i INT) ENGINE=NDBCLUSTER;
Query OK, 0 rows affected (0.09 sec)

mysql> SHOW CREATE TABLE ctest \G
*************************** 1. row ***************************
       Table: ctest
Create Table: CREATE TABLE `ctest` (
  `i` int(11) default NULL
) ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

To check that your nodes were set up properly, start the management client:

```
shell> ndb_mgm
```

Use the `SHOW` command from within the management client to obtain a report on the cluster's status:

```
ndb_mgm> SHOW
Cluster Configuration
---------------------
[ndbd(NDB)]     1 node(s)
id=2    @127.0.0.1  (Version: 5.6.51-ndb-7.4.31, Nodegroup: 0, *)

[ndb_mgmd(MGM)] 1 node(s)
id=1    @127.0.0.1  (Version: 5.6.51-ndb-7.4.31)

[mysqld(API)]   3 node(s)
id=3    @127.0.0.1  (Version: 5.6.51-ndb-7.4.31)
id=4 (not connected, accepting connect from any host)
id=5 (not connected, accepting connect from any host)
```

At this point, you have successfully set up a working NDB Cluster. You can now store data in the cluster by using any table created with `ENGINE=NDBCLUSTER` or its alias `ENGINE=NDB`.

18.3.2 Overview of NDB Cluster Configuration Parameters, Options, and Variables

The next several sections provide summary tables of NDB Cluster node configuration parameters used in the `config.ini` file to govern various aspects of node behavior, as well as of options and variables read by `mysqld` from a `my.cnf` file or from the command line when run as an NDB Cluster process. Each of the node parameter tables lists the parameters for a given type (`ndbd`, `ndb_mgmd`, `mysqld`, `computer`, `tcp`, or `shm`). All tables include the data type for the parameter, option, or variable, as well as its default, mimimum, and maximum values as applicable.

Considerations when restarting nodes. For node parameters, these tables also indicate what type of restart is required (node restart or system restart)—and whether the restart must be done with `--initial`—to change the value of a given configuration parameter. When performing a node restart or an initial node restart, all of the cluster's data nodes must be restarted in turn (also referred to as a *rolling restart*). It is possible to update cluster configuration parameters marked as `node` online—that is, without shutting down the cluster—in this fashion. An initial node restart requires restarting each `ndbd` process with the `--initial` option.

A system restart requires a complete shutdown and restart of the entire cluster. An initial system restart requires taking a backup of the cluster, wiping the cluster file system after shutdown, and then restoring from the backup following the restart.

In any cluster restart, all of the cluster's management servers must be restarted for them to read the updated configuration parameter values.

Important

Values for numeric cluster parameters can generally be increased without any problems, although it is advisable to do so progressively, making such adjustments in relatively small increments. Many of these can be increased online, using a rolling restart.

However, decreasing the values of such parameters—whether this is done using a node restart, node initial restart, or even a complete system restart of the cluster—is not to be undertaken lightly; it is recommended that you do so only after careful planning and testing. This is especially true with regard to those parameters that relate to memory usage and disk space, such as `MaxNoOfTables`, `MaxNoOfOrderedIndexes`, and `MaxNoOfUniqueHashIndexes`. In addition, it is the generally the case that configuration parameters relating to memory and disk usage can be raised using a simple node restart, but they require an initial node restart to be lowered.

Because some of these parameters can be used for configuring more than one type of cluster node, they may appear in more than one of the tables.

Note

`4294967039` often appears as a maximum value in these tables. This value is defined in the `NDBCLUSTER` sources as `MAX_INT_RNIL` and is equal to `0xFFFFFEFF`, or $2^{32} - 2^8 - 1$.

18.3.2.1 NDB Cluster Data Node Configuration Parameters

The listings in this section provide information about parameters used in the `[ndbd]` or `[ndbd default]` sections of a `config.ini` file for configuring NDB Cluster data nodes. For detailed descriptions and other additional information about each of these parameters, see Section 18.3.3.6, “Defining NDB Cluster Data Nodes”.

These parameters also apply to `ndbmtd`, the multithreaded version of `ndbd`. For more information, see Section 18.4.3, “`ndbmtd` — The NDB Cluster Data Node Daemon (Multi-Threaded)”.

- `Arbitration`: How arbitration should be performed to avoid split-brain issues in event of node failure.
- `ArbitrationTimeout`: Maximum time (milliseconds) database partition waits for arbitration signal.
- `BackupDataBufferSize`: Default size of databuffer for backup (in bytes).
- `BackupDataDir`: Path to where to store backups. Note that string '/BACKUP' is always appended to this setting, so that *effective* default is FileSystemPath/BACKUP.
- `BackupDiskWriteSpeedPct`: Sets percentage of data node's allocated maximum write speed (MaxDiskWriteSpeed) to reserve for LCPs when starting backup.
- `BackupLogBufferSize`: Default size of log buffer for backup (in bytes).
- `BackupMaxWriteSize`: Maximum size of file system writes made by backup (in bytes).
- `BackupMemory`: Total memory allocated for backups per node (in bytes).
- `BackupReportFrequency`: Frequency of backup status reports during backup in seconds.
- `BackupWriteSize`: Default size of file system writes made by backup (in bytes).
- `BatchSizePerLocalScan`: Used to calculate number of lock records for scan with hold lock.

- `BuildIndexThreads`: Number of threads to use for building ordered indexes during system or node restart. Also applies when running ndb_restore --rebuild-indexes. Setting this parameter to 0 disables multithreaded building of ordered indexes.
- `CompressedBackup`: Use zlib to compress backups as they are written.
- `CompressedLCP`: Write compressed LCPs using zlib.
- `ConnectCheckIntervalDelay`: Time between data node connectivity check stages. Data node is considered suspect after 1 interval and dead after 2 intervals with no response.
- `CrashOnCorruptedTuple`: When enabled, forces node to shut down whenever it detects corrupted tuple.
- `DataDir`: Data directory for this node.
- `DataMemory`: Number of bytes on each data node allocated for storing data; subject to available system RAM and size of IndexMemory.
- `DefaultHashMapSize`: Set size (in buckets) to use for table hash maps. Three values are supported: 0, 240, and 3840..
- `DictTrace`: Enable DBDICT debugging; for NDB development.
- `DiskCheckpointSpeed`: Bytes allowed to be written by checkpoint, per second.
- `DiskCheckpointSpeedInRestart`: Bytes allowed to be written by checkpoint during restart, per second.
- `DiskIOThreadPool`: Number of unbound threads for file access, applies to disk data only.
- `Diskless`: Run without using disk.
- `DiskPageBufferEntries`: Memory to allocate in DiskPageBufferMemory; very large disk transactions may require increasing this value.
- `DiskPageBufferMemory`: Number of bytes on each data node allocated for disk page buffer cache.
- `DiskSyncSize`: Amount of data written to file before synch is forced.
- `EventLogBufferSize`: Size of circular buffer for NDB log events within data nodes.
- `ExecuteOnComputer`: String referencing earlier defined COMPUTER.
- `ExtraSendBufferMemory`: Memory to use for send buffers in addition to any allocated by TotalSendBufferMemory or SendBufferMemory. Default (0) allows up to 16MB.
- `FileSystemPath`: Path to directory where data node stores its data (directory must exist).
- `FileSystemPathDataFiles`: Path to directory where data node stores its Disk Data files. Default value is FilesystemPathDD, if set; otherwise, FilesystemPath is used if it is set; otherwise, value of DataDir is used.
- `FileSystemPathDD`: Path to directory where data node stores its Disk Data and undo files. Default value is FileSystemPath, if set; otherwise, value of DataDir is used.
- `FileSystemPathUndoFiles`: Path to directory where data node stores its undo files for Disk Data. Default value is FilesystemPathDD, if set; otherwise, FilesystemPath is used if it is set; otherwise, value of DataDir is used.
- `FragmentLogFileSize`: Size of each redo log file.
- `HeartbeatIntervalDbApi`: Time between API node-data node heartbeats. (API connection closed after 3 missed heartbeats).

- HeartbeatIntervalDbDb: Time between data node-to-data node heartbeats; data node considered dead after 3 missed heartbeats.
- HeartbeatOrder: Sets order in which data nodes check each others' heartbeats for determining whether given node is still active and connected to cluster. Must be zero for all data nodes or distinct nonzero values for all data nodes; see documentation for further guidance.
- HostName: Host name or IP address for this data node.
- Id: Number identifying data node. Now deprecated; use NodeId instead.
- IndexMemory: Number of bytes on each data node allocated for storing indexes; subject to available system RAM and size of DataMemory.
- IndexStatAutoCreate: Enable/disable automatic statistics collection when indexes are created.
- IndexStatAutoUpdate: Monitor indexes for changes and trigger automatic statistics updates.
- IndexStatSaveScale: Scaling factor used in determining size of stored index statistics.
- IndexStatSaveSize: Maximum size in bytes for saved statistics per index.
- IndexStatTriggerPct: Threshold percent change in DML operations for index statistics updates. Value is scaled down by IndexStatTriggerScale.
- IndexStatTriggerScale: Scale down IndexStatTriggerPct by this amount, multiplied by base 2 logarithm of index size, for large index. Set to 0 to disable scaling.
- IndexStatUpdateDelay: Minimum delay between automatic index statistics updates for given index. 0 means no delay.
- InitFragmentLogFiles: Initialize fragment logfiles (sparse/full).
- InitialLogFileGroup: Describes log file group that is created during initial start. See documentation for format.
- InitialNoOfOpenFiles: Initial number of files open per data node. (One thread is created per file).
- InitialTablespace: Describes tablespace that is created during initial start. See documentation for format.
- LateAlloc: Allocate memory after connection to management server has been established.
- LcpScanProgressTimeout: Maximum time that local checkpoint fragment scan can be stalled before node is shut down to ensure systemwide LCP progress. Use 0 to disable.
- LockExecuteThreadToCPU: Comma-delimited list of CPU IDs.
- LockMaintThreadsToCPU: CPU ID indicating which CPU runs maintenance threads.
- LockPagesInMainMemory: 0=disable locking, 1=lock after memory allocation, 2=lock before memory allocation.
- LogLevelCheckpoint: Log level of local and global checkpoint information printed to stdout.
- LogLevelCongestion: Level of congestion information printed to stdout.
- LogLevelConnection: Level of node connect/disconnect information printed to stdout.
- LogLevelError: Transporter, heartbeat errors printed to stdout.
- LogLevelInfo: Heartbeat and log information printed to stdout.
- LogLevelNodeRestart: Level of node restart and node failure information printed to stdout.

- `LogLevelShutdown`: Level of node shutdown information printed to stdout.
- `LogLevelStartup`: Level of node startup information printed to stdout.
- `LogLevelStatistic`: Level of transaction, operation, and transporter information printed to stdout.
- `LongMessageBuffer`: Number of bytes allocated on each data node for internal long messages.
- `MaxAllocate`: Maximum size of allocation to use when allocating memory for tables.
- `MaxBufferedEpochs`: Allowed numbered of epochs that subscribing node can lag behind (unprocessed epochs). Exceeding causes lagging subscribers to be disconnected.
- `MaxBufferedEpochBytes`: Total number of bytes allocated for buffering epochs.
- `MaxDiskWriteSpeed`: Maximum number of bytes per second that can be written by LCP and backup when no restarts are ongoing.
- `MaxDiskWriteSpeedOtherNodeRestart`: Maximum number of bytes per second that can be written by LCP and backup when another node is restarting.
- `MaxDiskWriteSpeedOwnRestart`: Maximum number of bytes per second that can be written by LCP and backup when this node is restarting.
- `MaxDMLOperationsPerTransaction`: Limit size of transaction; aborts transaction if it requires more than this many DML operations. Set to 0 to disable.
- `MaxLCPStartDelay`: Time in seconds that LCP polls for checkpoint mutex (to allow other data nodes to complete metadata synchronization), before putting itself in lock queue for parallel recovery of table data.
- `MaxNoOfAttributes`: Suggests total number of attributes stored in database (sum over all tables).
- `MaxNoOfConcurrentIndexOperations`: Total number of index operations that can execute simultaneously on one data node.
- `MaxNoOfConcurrentOperations`: Maximum number of operation records in transaction coordinator.
- `MaxNoOfConcurrentScans`: Maximum number of scans executing concurrently on data node.
- `MaxNoOfConcurrentSubOperations`: Maximum number of concurrent subscriber operations.
- `MaxNoOfConcurrentTransactions`: Maximum number of transactions executing concurrently on this data node, total number of transactions that can be executed concurrently is this value times number of data nodes in cluster.
- `MaxNoOfFiredTriggers`: Total number of triggers that can fire simultaneously on one data node.
- `MaxNoOfLocalOperations`: Maximum number of operation records defined on this data node.
- `MaxNoOfLocalScans`: Maximum number of fragment scans in parallel on this data node.
- `MaxNoOfOpenFiles`: Maximum number of files open per data node.(One thread is created per file).
- `MaxNoOfOrderedIndexes`: Total number of ordered indexes that can be defined in system.
- `MaxNoOfSavedMessages`: Maximum number of error messages to write in error log and maximum number of trace files to retain.
- `MaxNoOfSubscribers`: Maximum number of subscribers (default 0 = MaxNoOfTables * 2).
- `MaxNoOfSubscriptions`: Maximum number of subscriptions (default 0 = MaxNoOfTables).
- `MaxNoOfTables`: Suggests total number of NDB tables stored in database.
- `MaxNoOfTriggers`: Total number of triggers that can be defined in system.

- `MaxNoOfUniqueHashIndexes`: Total number of unique hash indexes that can be defined in system.
- `MaxParallelCopyInstances`: Number of parallel copies during node restarts. Default is 0, which uses number of LDMs on both nodes, to maximum of 16.
- `MaxParallelScansPerFragment`: Maximum number of parallel scans per fragment. Once this limit is reached, scans are serialized.
- `MaxStartFailRetries`: Maximum retries when data node fails on startup, requires StopOnError = 0. Setting to 0 causes start attempts to continue indefinitely.
- `MemReportFrequency`: Frequency of memory reports in seconds; 0 = report only when exceeding percentage limits.
- `MinDiskWriteSpeed`: Minimum number of bytes per second that can be written by LCP and backup.
- `MinFreePct`: Percentage of memory resources to keep in reserve for restarts.
- `NodeGroup`: Node group to which data node belongs; used only during initial start of cluster.
- `NodeId`: Number uniquely identifying data node among all nodes in cluster.
- `NoOfFragmentLogFiles`: Number of 16 MB redo log files in each of 4 file sets belonging to data node.
- `NoOfReplicas`: Number of copies of all data in database.
- `Numa`: (Linux only; requires libnuma) Controls NUMA support. Setting to 0 permits system to determine use of interleaving by data node process; 1 means that it is determined by data node.
- `ODirect`: Use O_DIRECT file reads and writes when possible.
- `RealtimeScheduler`: When true, data node threads are scheduled as real-time threads. Default is false.
- `RedoBuffer`: Number of bytes on each data node allocated for writing redo logs.
- `RedoOverCommitCounter`: When RedoOverCommitLimit has been exceeded this many times, transactions are aborted, and operations are handled as specified by DefaultOperationRedoProblemAction.
- `RedoOverCommitLimit`: Each time that flushing current redo buffer takes longer than this many seconds, number of times that this has happened is compared to RedoOverCommitCounter.
- `ReservedSendBufferMemory`: This parameter is present in NDB code but is not enabled.
- `RestartOnErrorInsert`: Control type of restart caused by inserting error (when StopOnError is enabled).
- `SchedulerExecutionTimer`: Number of microseconds to execute in scheduler before sending.
- `SchedulerResponsiveness`: Set NDB scheduler response optimization 0-10; higher values provide better response time but lower throughput.
- `SchedulerSpinTimer`: Number of microseconds to execute in scheduler before sleeping.
- `ServerPort`: Port used to set up transporter for incoming connections from API nodes.
- `SharedGlobalMemory`: Total number of bytes on each data node allocated for any use.
- `StartFailRetryDelay`: Delay in seconds after start failure prior to retry; requires StopOnError = 0.
- `StartFailureTimeout`: Milliseconds to wait before terminating. (0=Wait forever).

- `StartNoNodeGroupTimeout`: Time to wait for nodes without nodegroup before trying to start (0=forever).
- `StartPartialTimeout`: Milliseconds to wait before trying to start without all nodes. (0=Wait forever).
- `StartPartitionedTimeout`: Milliseconds to wait before trying to start partitioned. (0=Wait forever).
- `StartupStatusReportFrequency`: Frequency of status reports during startup.
- `StopOnError`: When set to 0, data node automatically restarts and recovers following node failures.
- `StringMemory`: Default size of string memory (0 to 100 = % of maximum, 101+ = actual bytes).
- `TcpBind_INADDR_ANY`: Bind IP_ADDR_ANY so that connections can be made from anywhere (for autogenerated connections).
- `TimeBetweenEpochs`: Time between epochs (synchronization used for replication).
- `TimeBetweenEpochsTimeout`: Timeout for time between epochs. Exceeding causes node shutdown.
- `TimeBetweenGlobalCheckpoints`: Time between group commits of transactions to disk.
- `TimeBetweenGlobalCheckpointsTimeout`: Minimum timeout for group commit of transactions to disk.
- `TimeBetweenInactiveTransactionAbortCheck`: Time between checks for inactive transactions.
- `TimeBetweenLocalCheckpoints`: Time between taking snapshots of database (expressed in base-2 logarithm of bytes).
- `TimeBetweenWatchDogCheck`: Time between execution checks inside data node.
- `TimeBetweenWatchDogCheckInitial`: Time between execution checks inside data node (early start phases when memory is allocated).
- `TotalSendBufferMemory`: Total memory to use for all transporter send buffers..
- `TransactionBufferMemory`: Dynamic buffer space (in bytes) for key and attribute data allocated for each data node.
- `TransactionDeadlockDetectionTimeout`: Time transaction can spend executing within data node. This is time that transaction coordinator waits for each data node participating in transaction to execute request. If data node takes more than this amount of time, transaction is aborted.
- `TransactionInactiveTimeout`: Milliseconds that application waits before executing another part of transaction. This is time transaction coordinator waits for application to execute or send another part (query, statement) of transaction. If application takes too much time, then transaction is aborted. Timeout = 0 means that application never times out.
- `TwoPassInitialNodeRestartCopy`: Copy data in 2 passes during initial node restart, which enables multithreaded building of ordered indexes for such restarts.
- `UndoDataBuffer`: Number of bytes on each data node allocated for writing data undo logs.
- `UndoIndexBuffer`: Number of bytes on each data node allocated for writing index undo logs.

The following parameters are specific to `ndbmtd`:

- `MaxNoOfExecutionThreads`: For ndbmtd only, specify maximum number of execution threads.
- `NoOfFragmentLogParts`: Number of redo log file groups belonging to this data node.

- `ThreadConfig`: Used for configuration of multithreaded data nodes (ndbmtd). Default is empty string; see documentation for syntax and other information.

18.3.2.2 NDB Cluster Management Node Configuration Parameters

The listing in this section provides information about parameters used in the `[ndb_mgmd]` or `[mgm]` section of a `config.ini` file for configuring NDB Cluster management nodes. For detailed descriptions and other additional information about each of these parameters, see Section 18.3.3.5, "Defining an NDB Cluster Management Server".

- `ArbitrationDelay`: When asked to arbitrate, arbitrator waits this long before voting (milliseconds).
- `ArbitrationRank`: If 0, then management node is not arbitrator. Kernel selects arbitrators in order 1, 2.
- `DataDir`: Data directory for this node.
- `ExecuteOnComputer`: String referencing earlier defined COMPUTER.
- `ExtraSendBufferMemory`: Memory to use for send buffers in addition to any allocated by TotalSendBufferMemory or SendBufferMemory. Default (0) allows up to 16MB.
- `HeartbeatIntervalMgmdMgmd`: Time between management-node-to-management-node heartbeats; connection between management nodes is considered lost after 3 missed heartbeats.
- `HeartbeatThreadPriority`: Set heartbeat thread policy and priority for management nodes; see manual for allowed values.
- `HostName`: Host name or IP address for this management node.
- `Id`: Number identifying management node. Now deprecated; use NodeId instead.
- `LogDestination`: Where to send log messages: console, system log, or specified log file.
- `MaxNoOfSavedEvents`: Not used.
- `NodeId`: Number uniquely identifying management node among all nodes in cluster.
- `PortNumber`: Port number to send commands to and fetch configuration from management server.
- `PortNumberStats`: Port number used to get statistical information from management server.
- `TotalSendBufferMemory`: Total memory to use for all transporter send buffers.
- `wan`: Use WAN TCP setting as default.

Note

After making changes in a management node's configuration, it is necessary to perform a rolling restart of the cluster for the new configuration to take effect. See Section 18.3.3.5, "Defining an NDB Cluster Management Server", for more information.

To add new management servers to a running NDB Cluster, it is also necessary perform a rolling restart of all cluster nodes after modifying any existing `config.ini` files. For more information about issues arising when using multiple management nodes, see Section 18.1.7.10, "Limitations Relating to Multiple NDB Cluster Nodes".

18.3.2.3 NDB Cluster SQL Node and API Node Configuration Parameters

The listing in this section provides information about parameters used in the `[mysqld]` and `[api]` sections of a `config.ini` file for configuring NDB Cluster SQL nodes and API nodes. For detailed descriptions and other additional information about each of these parameters, see Section 18.3.3.7, "Defining SQL and Other API Nodes in an NDB Cluster".

- `ApiVerbose`: Enable NDB API debugging; for NDB development.
- `ArbitrationDelay`: When asked to arbitrate, arbitrator waits this many milliseconds before voting.
- `ArbitrationRank`: If 0, then API node is not arbitrator. Kernel selects arbitrators in order 1, 2.
- `AutoReconnect`: Specifies whether an API node should reconnect fully when disconnected from cluster.
- `BatchByteSize`: Default batch size in bytes.
- `BatchSize`: Default batch size in number of records.
- `ConnectBackoffMaxTime`: Specifies longest time in milliseconds (~100ms resolution) to allow between connection attempts to any given data node by this API node. Excludes time elapsed while connection attempts are ongoing, which in worst case can take several seconds. Disable by setting to 0. If no data nodes are currently connected to this API node, StartConnectBackoffMaxTime is used instead.
- `ConnectionMap`: Specifies which data nodes to connect.
- `DefaultHashMapSize`: Set size (in buckets) to use for table hash maps. Three values are supported: 0, 240, and 3840.
- `DefaultOperationRedoProblemAction`: How operations are handled in event that RedoOverCommitCounter is exceeded.
- `ExecuteOnComputer`: String referencing earlier defined COMPUTER.
- `ExtraSendBufferMemory`: Memory to use for send buffers in addition to any allocated by TotalSendBufferMemory or SendBufferMemory. Default (0) allows up to 16MB.
- `HeartbeatThreadPriority`: Set heartbeat thread policy and priority for API nodes; see manual for allowed values.
- `HostName`: Host name or IP address for this SQL or API node.
- `Id`: Number identifying MySQL server or API node (Id). Now deprecated; use NodeId instead.
- `MaxScanBatchSize`: Maximum collective batch size for one scan.
- `NodeId`: Number uniquely identifying SQL node or API node among all nodes in cluster.
- `StartConnectBackoffMaxTime`: Same as ConnectBackoffMaxTime except that this parameter is used in its place if no data nodes are connected to this API node.
- `TotalSendBufferMemory`: Total memory to use for all transporter send buffers.
- `wan`: Use WAN TCP setting as default.

For a discussion of MySQL server options for NDB Cluster, see MySQL Server Options for NDB Cluster. For information about MySQL server system variables relating to NDB Cluster, see NDB Cluster System Variables.

Note

To add new SQL or API nodes to the configuration of a running NDB Cluster, it is necessary to perform a rolling restart of all cluster nodes after adding new `[mysqld]` or `[api]` sections to the `config.ini` file (or files, if you are using more than one management server). This must be done before the new SQL or API nodes can connect to the cluster.

It is *not* necessary to perform any restart of the cluster if new SQL or API nodes can employ previously unused API slots in the cluster configuration to connect to the cluster.

18.3.2.4 Other NDB Cluster Configuration Parameters

The listings in this section provide information about parameters used in the `[computer]`, `[tcp]`, and `[shm]` sections of a `config.ini` file for configuring NDB Cluster. For detailed descriptions and additional information about individual parameters, see Section 18.3.3.9, "NDB Cluster TCP/IP Connections", or Section 18.3.3.11, "NDB Cluster Shared-Memory Connections", as appropriate.

The following parameters apply to the `config.ini` file's `[computer]` section:

- `HostName`: Host name or IP address of this computer.
- `Id`: Unique identifier for this computer.

The following parameters apply to the `config.ini` file's `[tcp]` section:

- `Checksum`: If checksum is enabled, all signals between nodes are checked for errors.
- `Group`: Used for group proximity; smaller value is interpreted as being closer.
- `NodeId1`: ID of node (data node, API node, or management node) on one side of connection.
- `NodeId2`: ID of node (data node, API node, or management node) on one side of connection.
- `NodeIdServer`: Set server side of TCP connection.
- `OverloadLimit`: When more than this many unsent bytes are in send buffer, connection is considered overloaded.
- `PortNumber`: Port used for TCP transporter.
- `Proxy`: .
- `ReceiveBufferMemory`: Bytes of buffer for signals received by this node.
- `SendBufferMemory`: Bytes of TCP buffer for signals sent from this node.
- `SendSignalId`: Sends ID in each signal. Used in trace files. Defaults to true in debug builds.
- `TCP_MAXSEG_SIZE`: Value used for TCP_MAXSEG.
- `TCP_RCV_BUF_SIZE`: Value used for SO_RCVBUF.
- `TCP_SND_BUF_SIZE`: Value used for SO_SNDBUF.
- `TcpBind_INADDR_ANY`: Bind InAddrAny instead of host name for server part of connection.

The following parameters apply to the `config.ini` file's `[shm]` section:

- `Checksum`: If checksum is enabled, all signals between nodes are checked for errors.
- `Group`: Used for group proximity; smaller value is interpreted as being closer.
- `NodeId1`: ID of node (data node, API node, or management node) on one side of connection.
- `NodeId2`: ID of node (data node, API node, or management node) on one side of connection.
- `NodeIdServer`: Set server side of SHM connection.
- `OverloadLimit`: When more than this many unsent bytes are in send buffer, connection is considered overloaded.
- `PortNumber`: Port used for SHM transporter.
- `SendSignalId`: Sends ID in each signal. Used in trace files.
- `ShmKey`: Shared memory key; when set to 1, this is calculated by NDB.

- `ShmSize`: Size of shared memory segment.
- `Signum`: Signal number to be used for signalling.

18.3.2.5 NDB Cluster mysqld Option and Variable Reference

The following table provides a list of the command-line options, server and status variables applicable within `mysqld` when it is running as an SQL node in an NDB Cluster. For a table showing *all* command-line options, server and status variables available for use with `mysqld`, see Section 5.1.3, “Server Option, System Variable, and Status Variable Reference”.

- `Com_show_ndb_status`: Count of SHOW NDB STATUS statements.
- `create_old_temporals`: Use pre-5.6.4 storage format for temporal types when creating tables. Intended for use in replication and upgrades/downgrades between NDB 7.2 and NDB 7.3/7.4.
- `Handler_discover`: Number of times that tables have been discovered.
- `ndb-batch-size`: Size (in bytes) to use for NDB transaction batches.
- `ndb-blob-read-batch-bytes`: Specifies size in bytes that large BLOB reads should be batched into. 0 = no limit.
- `ndb-blob-write-batch-bytes`: Specifies size in bytes that large BLOB writes should be batched into. 0 = no limit.
- `ndb-cluster-connection-pool`: Number of connections to cluster used by MySQL.
- `ndb-connectstring`: Address of NDB management server distributing configuration information for this cluster.
- `ndb-deferred-constraints`: Specifies that constraint checks on unique indexes (where these are supported) should be deferred until commit time. Not normally needed or used; for testing purposes only.
- `ndb-distribution`: Default distribution for new tables in NDBCLUSTER (KEYHASH or LINHASH, default is KEYHASH).
- `ndb-log-apply-status`: Cause MySQL server acting as replica to log mysql.ndb_apply_status updates received from its immediate source in its own binary log, using its own server ID. Effective only if server is started with --ndbcluster option.
- `ndb-log-empty-epochs`: When enabled, causes epochs in which there were no changes to be written to ndb_apply_status and ndb_binlog_index tables, even when --log-slave-updates is enabled.
- `ndb-log-empty-update`: When enabled, causes updates that produced no changes to be written to ndb_apply_status and ndb_binlog_index tables, even when --log-slave-updates is enabled.
- `ndb-log-exclusive-reads`: Log primary key reads with exclusive locks; allow conflict resolution based on read conflicts.
- `ndb-log-fail-terminate`: Terminate mysqld process if complete logging of all found row events is not possible.
- `ndb-log-orig`: Log originating server id and epoch in mysql.ndb_binlog_index table.
- `ndb-log-transaction-id`: Write NDB transaction IDs in binary log. Requires --log-bin-v1-events=OFF.
- `ndb-log-update-as-write`: Toggles logging of updates on source between updates (OFF) and writes (ON).
- `ndb-mgmd-host`: Set host (and port, if desired) for connecting to management server.
- `ndb-nodeid`: NDB Cluster node ID for this MySQL server.

- `ndb-transid-mysql-connection-map`: Enable or disable ndb_transid_mysql_connection_map plugin; that is, enable or disable INFORMATION_SCHEMA table having that name.
- `ndb-wait-connected`: Time (in seconds) for MySQL server to wait for connection to cluster management and data nodes before accepting MySQL client connections.
- `ndb-wait-setup`: Time (in seconds) for MySQL server to wait for NDB engine setup to complete.
- `Ndb_api_adaptive_send_deferred_count`: Number of adaptive send calls not actually sent by this MySQL Server (SQL node).
- `Ndb_api_adaptive_send_deferred_count_slave`: Number of adaptive send calls not actually sent in this client session.
- `Ndb_api_adaptive_send_deferred_count_slave`: Number of adaptive send calls not actually sent by this replica.
- `Ndb_api_adaptive_send_forced_count`: Number of adaptive sends with forced-send set sent by this MySQL Server (SQL node).
- `Ndb_api_adaptive_send_forced_count_session`: Number of adaptive sends with forced-send set in this client session.
- `Ndb_api_adaptive_send_forced_count_slave`: Number of adaptive sends with forced-send set sent by this replica.
- `Ndb_api_adaptive_send_unforced_count`: Number of adaptive sends without forced-send sent by this MySQL Server (SQL node).
- `Ndb_api_adaptive_send_unforced_count_session`: Number of adaptive sends without forced-send in this client session.
- `Ndb_api_adaptive_send_unforced_count_slave`: Number of adaptive sends without forced-send sent by this replica.
- `Ndb_api_bytes_received_count`: Quantity of data (in bytes) received from data nodes by this MySQL Server (SQL node).
- `Ndb_api_bytes_received_count_session`: Quantity of data (in bytes) received from data nodes in this client session.
- `Ndb_api_bytes_received_count_slave`: Quantity of data (in bytes) received from data nodes by this replica.
- `Ndb_api_bytes_sent_count`: Quantity of data (in bytes) sent to data nodes by this MySQL Server (SQL node).
- `Ndb_api_bytes_sent_count_session`: Quantity of data (in bytes) sent to data nodes in this client session.
- `Ndb_api_bytes_sent_count_slave`: Qunatity of data (in bytes) sent to data nodes by this replica.
- `Ndb_api_event_bytes_count`: Number of bytes of events received by this MySQL Server (SQL node).
- `Ndb_api_event_bytes_count_injector`: Number of bytes of event data received by NDB binary log injector thread.
- `Ndb_api_event_data_count`: Number of row change events received by this MySQL Server (SQL node).
- `Ndb_api_event_data_count_injector`: Number of row change events received by NDB binary log injector thread.

- Ndb_api_event_nondata_count: Number of events received, other than row change events, by this MySQL Server (SQL node).
- Ndb_api_event_nondata_count_injector: Number of events received, other than row change events, by NDB binary log injector thread.
- Ndb_api_pk_op_count: Number of operations based on or using primary keys by this MySQL Server (SQL node).
- Ndb_api_pk_op_count_session: Number of operations based on or using primary keys in this client session.
- Ndb_api_pk_op_count_slave: Number of operations based on or using primary keys by this replica.
- Ndb_api_pruned_scan_count: Number of scans that have been pruned to one partition by this MySQL Server (SQL node).
- Ndb_api_pruned_scan_count_session: Number of scans that have been pruned to one partition in this client session.
- Ndb_api_pruned_scan_count_slave: Number of scans that have been pruned to one partition by this replica.
- Ndb_api_range_scan_count: Number of range scans that have been started by this MySQL Server (SQL node).
- Ndb_api_range_scan_count_session: Number of range scans that have been started in this client session.
- Ndb_api_range_scan_count_slave: Number of range scans that have been started by this replica.
- Ndb_api_read_row_count: Total number of rows that have been read by this MySQL Server (SQL node).
- Ndb_api_read_row_count_session: Total number of rows that have been read in this client session.
- Ndb_api_read_row_count_slave: Total number of rows that have been read by this replica.
- Ndb_api_scan_batch_count: Number of batches of rows received by this MySQL Server (SQL node).
- Ndb_api_scan_batch_count_session: Number of batches of rows received in this client session.
- Ndb_api_scan_batch_count_slave: Number of batches of rows received by this replica.
- Ndb_api_table_scan_count: Number of table scans that have been started, including scans of internal tables, by this MySQL Server (SQL node).
- Ndb_api_table_scan_count_session: Number of table scans that have been started, including scans of internal tables, in this client session.
- Ndb_api_table_scan_count_slave: Number of table scans that have been started, including scans of internal tables, by this replica.
- Ndb_api_trans_abort_count: Number of transactions aborted by this MySQL Server (SQL node).
- Ndb_api_trans_abort_count_session: Number of transactions aborted in this client session.
- Ndb_api_trans_abort_count_slave: Number of transactions aborted by this replica.

- Ndb_api_trans_close_count: Number of transactions aborted (may be greater than sum of TransCommitCount and TransAbortCount) by this MySQL Server (SQL node).
- Ndb_api_trans_close_count_session: Number of transactions aborted (may be greater than sum of TransCommitCount and TransAbortCount) in this client session.
- Ndb_api_trans_close_count_slave: Number of transactions aborted (may be greater than sum of TransCommitCount and TransAbortCount) by this replica.
- Ndb_api_trans_commit_count: Number of transactions committed by this MySQL Server (SQL node).
- Ndb_api_trans_commit_count_session: Number of transactions committed in this client session.
- Ndb_api_trans_commit_count_slave: Number of transactions committed by this replica.
- Ndb_api_trans_local_read_row_count: Total number of rows that have been read by this MySQL Server (SQL node).
- Ndb_api_trans_local_read_row_count_session: Total number of rows that have been read in this client session.
- Ndb_api_trans_local_read_row_count_slave: Total number of rows that have been read by this replica.
- Ndb_api_trans_start_count: Number of transactions started by this MySQL Server (SQL node).
- Ndb_api_trans_start_count_session: Number of transactions started in this client session.
- Ndb_api_trans_start_count_slave: Number of transactions started by this replica.
- Ndb_api_uk_op_count: Number of operations based on or using unique keys by this MySQL Server (SQL node).
- Ndb_api_uk_op_count_session: Number of operations based on or using unique keys in this client session.
- Ndb_api_uk_op_count_slave: Number of operations based on or using unique keys by this replica.
- Ndb_api_wait_exec_complete_count: Number of times thread has been blocked while waiting for operation execution to complete by this MySQL Server (SQL node).
- Ndb_api_wait_exec_complete_count_session: Number of times thread has been blocked while waiting for operation execution to complete in this client session.
- Ndb_api_wait_exec_complete_count_slave: Number of times thread has been blocked while waiting for operation execution to complete by this replica.
- Ndb_api_wait_meta_request_count: Number of times thread has been blocked waiting for metadata-based signal by this MySQL Server (SQL node).
- Ndb_api_wait_meta_request_count_session: Number of times thread has been blocked waiting for metadata-based signal in this client session.
- Ndb_api_wait_meta_request_count_slave: Number of times thread has been blocked waiting for metadata-based signal by this replica.
- Ndb_api_wait_nanos_count: Total time (in nanoseconds) spent waiting for some type of signal from data nodes by this MySQL Server (SQL node).
- Ndb_api_wait_nanos_count_session: Total time (in nanoseconds) spent waiting for some type of signal from data nodes in this client session.

- `Ndb_api_wait_nanos_count_slave`: Total time (in nanoseconds) spent waiting for some type of signal from data nodes by this replica.
- `Ndb_api_wait_scan_result_count`: Number of times thread has been blocked while waiting for scan-based signal by this MySQL Server (SQL node).
- `Ndb_api_wait_scan_result_count_session`: Number of times thread has been blocked while waiting for scan-based signal in this client session.
- `Ndb_api_wait_scan_result_count_slave`: Number of times thread has been blocked while waiting for scan-based signal by this replica.
- `ndb_autoincrement_prefetch_sz`: NDB auto-increment prefetch size.
- `ndb_cache_check_time`: Number of milliseconds between checks of cluster SQL nodes made by MySQL query cache.
- `ndb_clear_apply_status`: Causes RESET SLAVE/RESET REPLICA to clear all rows from ndb_apply_status table; ON by default.
- `Ndb_cluster_node_id`: Node ID of this server when acting as NDB Cluster SQL node.
- `Ndb_config_from_host`: NDB Cluster management server host name or IP address.
- `Ndb_config_from_port`: Port for connecting to NDB Cluster management server.
- `Ndb_conflict_fn_epoch`: Number of rows that have been found in conflict by NDB$EPOCH() conflict detection function.
- `Ndb_conflict_fn_epoch2`: Number of rows that have been found in conflict by NDB$EPOCH2() conflict detection function.
- `Ndb_conflict_fn_epoch2_trans`: Number of rows that have been found in conflict by NDB $EPOCH2_TRANS() conflict detection function.
- `Ndb_conflict_fn_epoch_trans`: Number of rows that have been found in conflict by NDB $EPOCH_TRANS() conflict detection function.
- `Ndb_conflict_fn_max`: Number of times that conflict resolution based on "greater timestamp wins" has been applied when server is part of an NDB Cluster involved in cluster replication.
- `Ndb_conflict_fn_max_del_win`: Number of times that conflict resolution based on outcome of NDB$MAX_DELETE_WIN() has been applied.
- `Ndb_conflict_fn_old`: Number of times that "same timestamp wins" conflict resolution has been applied when this server is part of an NDB Cluster involved in cluster replication.
- `Ndb_conflict_last_conflict_epoch`: Most recent NDB epoch on this replica in which some conflict was detected.
- `Ndb_conflict_last_stable_epoch`: Number of rows found to be in conflict by transactional conflict function.
- `Ndb_conflict_reflected_op_discard_count`: Number of reflected operations that were not applied due error during execution.
- `Ndb_conflict_reflected_op_prepare_count`: Number of reflected operations received that have been prepared for execution.
- `Ndb_conflict_refresh_op_count`: Number of refresh operations that have been prepared.
- `Ndb_conflict_trans_conflict_commit_count`: Number of epoch transactions committed after requiring transactional conflict handling.

- `Ndb_conflict_trans_detect_iter_count`: Number of internal iterations required to commit epoch transaction. Should be (slightly) greater than or equal to Ndb_conflict_trans_conflict_commit_count.
- `Ndb_conflict_trans_reject_count`: Number of transactions rejected after being found in conflict by transactional conflict function.
- `Ndb_conflict_trans_row_conflict_count`: Number of rows found in conflict by transactional conflict function. Includes any rows included in or dependent on conflicting transactions.
- `Ndb_conflict_trans_row_reject_count`: Total number of rows realigned after being found in conflict by transactional conflict function. Includes Ndb_conflict_trans_row_conflict_count and any rows included in or dependent on conflicting transactions.
- `ndb_deferred_constraints`: Specifies that constraint checks should be deferred (where these are supported). Not normally needed or used; for testing purposes only.
- `ndb_distribution`: Default distribution for new tables in NDBCLUSTER (KEYHASH or LINHASH, default is KEYHASH).
- `Ndb_epoch_delete_delete_count`: Number of delete-delete conflicts detected (delete operation is applied, but row does not exist).
- `ndb_eventbuffer_free_percent`: Percentage of free memory that should be available in event buffer before resumption of buffering, after reaching limit set by ndb_eventbuffer_max_alloc.
- `ndb_eventbuffer_max_alloc`: Maximum memory that can be allocated for buffering events by NDB API. Defaults to 0 (no limit).
- `Ndb_execute_count`: Number of round trips to NDB kernel made by operations.
- `ndb_extra_logging`: Controls logging of NDB Cluster schema, connection, and data distribution events in MySQL error log.
- `ndb_force_send`: Forces sending of buffers to NDB immediately, without waiting for other threads.
- `ndb_index_stat_cache_entries`: Granularity of statistics by determining number of starting and ending keys.
- `ndb_index_stat_enable`: Use NDB index statistics in query optimization.
- `ndb_index_stat_option`: Comma-separated list of tunable options for NDB index statistics; list should contain no spaces.
- `ndb_index_stat_update_freq`: How often to query data nodes rather than statistics cache.
- `ndb_join_pushdown`: Enables pushing down of joins to data nodes.
- `Ndb_last_commit_epoch_server`: Epoch most recently committed by NDB.
- `Ndb_last_commit_epoch_session`: Epoch most recently committed by this NDB client.
- `ndb_log_apply_status`: Whether or not MySQL server acting as replica logs mysql.ndb_apply_status updates received from its immediate source in its own binary log, using its own server ID.
- `ndb_log_bin`: Write updates to NDB tables in binary log. Effective only if binary logging is enabled with --log-bin.
- `ndb_log_binlog_index`: Insert mapping between epochs and binary log positions into ndb_binlog_index table. Defaults to ON. Effective only if binary logging is enabled.
- `ndb_log_empty_epochs`: When enabled, epochs in which there were no changes are written to ndb_apply_status and ndb_binlog_index tables, even when log_slave_updates is enabled.

- `ndb_log_empty_update`: When enabled, updates which produce no changes are written to ndb_apply_status and ndb_binlog_index tables, even when log_slave_updates is enabled.
- `ndb_log_exclusive_reads`: Log primary key reads with exclusive locks; allow conflict resolution based on read conflicts.
- `ndb_log_orig`: Whether id and epoch of originating server are recorded in mysql.ndb_binlog_index table. Set using --ndb-log-orig option when starting mysqld.
- `ndb_log_transaction_id`: Whether NDB transaction IDs are written into binary log (Read-only).
- `ndb-log-update-minimal`: Log updates in minimal format.
- `ndb-log-updated-only`: Log complete rows (ON) or updates only (OFF).
- `Ndb_number_of_data_nodes`: Number of data nodes in this NDB cluster; set only if server participates in cluster.
- `ndb-optimization-delay`: Number of milliseconds to wait between processing sets of rows by OPTIMIZE TABLE on NDB tables.
- `ndb_optimized_node_selection`: Determines how SQL node chooses cluster data node to use as transaction coordinator.
- `Ndb_pruned_scan_count`: Number of scans executed by NDB since cluster was last started where partition pruning could be used.
- `Ndb_pushed_queries_defined`: Number of joins that API nodes have attempted to push down to data nodes.
- `Ndb_pushed_queries_dropped`: Number of joins that API nodes have tried to push down, but failed.
- `Ndb_pushed_queries_executed`: Number of joins successfully pushed down and executed on data nodes.
- `Ndb_pushed_reads`: Number of reads executed on data nodes by pushed-down joins.
- `ndb_recv_thread_activation_threshold`: Activation threshold when receive thread takes over polling of cluster connection (measured in concurrently active threads).
- `ndb_recv_thread_cpu_mask`: CPU mask for locking receiver threads to specific CPUs; specified as hexadecimal. See documentation for details.
- `ndb_report_thresh_binlog_epoch_slip`: NDB 7.5.4 and later: Threshold for number of epochs completely buffered, but not yet consumed by binlog injector thread which when exceeded generates BUFFERED_EPOCHS_OVER_THRESHOLD event buffer status message; prior to NDB 7.5.4: Threshold for number of epochs to lag behind before reporting binary log status.
- `ndb_report_thresh_binlog_mem_usage`: Threshold for percentage of free memory remaining before reporting binary log status.
- `Ndb_scan_count`: Total number of scans executed by NDB since cluster was last started.
- `ndb_show_foreign_key_mock_tables`: Show mock tables used to support foreign_key_checks=0.
- `ndb_slave_conflict_role`: Role for replica to play in conflict detection and resolution. Value is one of PRIMARY, SECONDARY, PASS, or NONE (default). Can be changed only when replication SQL thread is stopped. See documentation for further information.
- `Ndb_slave_max_replicated_epoch`: Most recently committed NDB epoch on this replica. When this value is greater than or equal to Ndb_conflict_last_conflict_epoch, no conflicts have yet been detected.

- `ndb_table_no_logging`: NDB tables created when this setting is enabled are not checkpointed to disk (although table schema files are created). Setting in effect when table is created with or altered to use NDBCLUSTER persists for table's lifetime.
- `ndb_table_temporary`: NDB tables are not persistent on disk: no schema files are created and tables are not logged.
- `ndb_use_copying_alter_table`: Use copying ALTER TABLE operations in NDB Cluster.
- `ndb_use_exact_count`: Use exact row count when planning queries.
- `ndb_use_transactions`: Forces NDB to use a count of records during SELECT COUNT(*) query planning to speed up this type of query.
- `ndb_version`: Shows build and NDB engine version as an integer.
- `ndb_version_string`: Shows build information including NDB engine version in ndb-x.y.z format.
- `ndbcluster`: Enable NDB Cluster (if this version of MySQL supports it). Disabled by `--skip-ndbcluster`.
- `ndbinfo_database`: Name used for NDB information database; read only.
- `ndbinfo_max_bytes`: Used for debugging only.
- `ndbinfo_max_rows`: Used for debugging only.
- `ndbinfo_offline`: Put ndbinfo database into offline mode, in which no rows are returned from tables or views.
- `ndbinfo_show_hidden`: Whether to show ndbinfo internal base tables in mysql client; default is OFF.
- `ndbinfo_table_prefix`: Prefix to use for naming ndbinfo internal base tables; read only.
- `ndbinfo_version`: ndbinfo engine version; read only.
- `server_id_bits`: Number of least significant bits in server_id actually used for identifying server, permitting NDB API applications to store application data in most significant bits. server_id must be less than 2 to power of this value.
- `skip-ndbcluster`: Disable NDB Cluster storage engine.
- `slave_allow_batching`: Turns update batching on and off for replica.
- `transaction_allow_batching`: Allows batching of statements within one transaction. Disable AUTOCOMMIT to use.

18.3.3 NDB Cluster Configuration Files

Configuring NDB Cluster requires working with two files:

- `my.cnf`: Specifies options for all NDB Cluster executables. This file, with which you should be familiar with from previous work with MySQL, must be accessible by each executable running in the cluster.
- `config.ini`: This file, sometimes known as the *global configuration file*, is read only by the NDB Cluster management server, which then distributes the information contained therein to all processes participating in the cluster. `config.ini` contains a description of each node involved in the cluster. This includes configuration parameters for data nodes and configuration parameters for connections between all nodes in the cluster. For a quick reference to the sections that can appear in this file, and what sorts of configuration parameters may be placed in each section, see Sections of the `config.ini` File.

Caching of configuration data. In NDB Cluster 7.3 and later, `NDB` uses *stateful configuration*. Rather than reading the global configuration file every time the management server is restarted, the management server caches the configuration the first time it is started, and thereafter, the global configuration file is read only when one of the following conditions is true:

- **The management server is started using the --initial option.** When `--initial` is used, the global configuration file is re-read, any existing cache files are deleted, and the management server creates a new configuration cache.
- **The management server is started using the --reload option.** The `--reload` option causes the management server to compare its cache with the global configuration file. If they differ, the management server creates a new configuration cache; any existing configuration cache is preserved, but not used. If the management server's cache and the global configuration file contain the same configuration data, then the existing cache is used, and no new cache is created.
- **The management server is started using --config-cache=FALSE.** This disables `--config-cache` (enabled by default), and can be used to force the management server to bypass configuration caching altogether. In this case, the management server ignores any configuration files that may be present, always reading its configuration data from the `config.ini` file instead.
- **No configuration cache is found.** In this case, the management server reads the global configuration file and creates a cache containing the same configuration data as found in the file.

Configuration cache files. The management server by default creates configuration cache files in a directory named `mysql-cluster` in the MySQL installation directory. (If you build NDB Cluster from source on a Unix system, the default location is `/usr/local/mysql-cluster`.) This can be overridden at runtime by starting the management server with the `--configdir` option. Configuration cache files are binary files named according to the pattern `ndb_node_id_config.bin.seq_id`, where *`node_id`* is the management server's node ID in the cluster, and *`seq_id`* is a cache idenitifer. Cache files are numbered sequentially using *`seq_id`*, in the order in which they are created. The management server uses the latest cache file as determined by the *`seq_id`*.

Note

It is possible to roll back to a previous configuration by deleting later configuration cache files, or by renaming an earlier cache file so that it has a higher *`seq_id`*. However, since configuration cache files are written in a binary format, you should not attempt to edit their contents by hand.

For more information about the `--configdir`, `--config-cache`, `--initial`, and `--reload` options for the NDB Cluster management server, see Section 18.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”.

We are continuously making improvements in Cluster configuration and attempting to simplify this process. Although we strive to maintain backward compatibility, there may be times when introduce an incompatible change. In such cases we try to let Cluster users know in advance if a change is not backward compatible. If you find such a change and we have not documented it, please report it in the MySQL bugs database using the instructions given in Section 1.6, “How to Report Bugs or Problems”.

18.3.3.1 NDB Cluster Configuration: Basic Example

To support NDB Cluster, you must update `my.cnf` as shown in the following example. You may also specify these parameters on the command line when invoking the executables.

Note

The options shown here should not be confused with those that are used in `config.ini` global configuration files. Global configuration options are discussed later in this section.

```
# my.cnf
```

```
# example additions to my.cnf for NDB Cluster
# (valid in MySQL 5.6)

# enable ndbcluster storage engine, and provide connection string for
# management server host (default port is 1186)
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com


# provide connection string for management server host (default port: 1186)
[ndbd]
connect-string=ndb_mgmd.mysql.com

# provide connection string for management server host (default port: 1186)
[ndb_mgm]
connect-string=ndb_mgmd.mysql.com

# provide location of cluster configuration file
[ndb_mgmd]
config-file=/etc/config.ini
```

(For more information on connection strings, see Section 18.3.3.3, “NDB Cluster Connection Strings”.)

```
# my.cnf
# example additions to my.cnf for NDB Cluster
# (works on all versions)

# enable ndbcluster storage engine, and provide connection string for management
# server host to the default port 1186
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

Important

Once you have started a `mysqld` process with the `NDBCLUSTER` and `ndb-connectstring` parameters in the `[mysqld]` in the `my.cnf` file as shown previously, you cannot execute any `CREATE TABLE` or `ALTER TABLE` statements without having actually started the cluster. Otherwise, these statements fail with an error. *This is by design*.

You may also use a separate `[mysql_cluster]` section in the cluster `my.cnf` file for settings to be read and used by all executables:

```
# cluster-specific settings
[mysql_cluster]
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

For additional `NDB` variables that can be set in the `my.cnf` file, see NDB Cluster System Variables.

The NDB Cluster global configuration file is by convention named `config.ini` (but this is not required). If needed, it is read by `ndb_mgmd` at startup and can be placed in any location that can be read by it. The location and name of the configuration are specified using `--config-file=`*`path_name`* with `ndb_mgmd` on the command line. This option has no default value, and is ignored if `ndb_mgmd` uses the configuration cache.

The global configuration file for NDB Cluster uses INI format, which consists of sections preceded by section headings (surrounded by square brackets), followed by the appropriate parameter names and values. One deviation from the standard INI format is that the parameter name and value can be separated by a colon (`:`) as well as the equal sign (`=`); however, the equal sign is preferred. Another deviation is that sections are not uniquely identified by section name. Instead, unique sections (such as two different nodes of the same type) are identified by a unique ID specified as a parameter within the section.

Default values are defined for most parameters, and can also be specified in `config.ini`. To create a default value section, simply add the word `default` to the section name. For example, an `[ndbd]`

section contains parameters that apply to a particular data node, whereas an `[ndbd default]` section contains parameters that apply to all data nodes. Suppose that all data nodes should use the same data memory size. To configure them all, create an `[ndbd default]` section that contains a `DataMemory` line to specify the data memory size.

If used, the `[ndbd default]` section must precede any `[ndbd]` sections in the configuration file. This is also true for `default` sections of any other type.

Note

In some older releases of NDB Cluster, there was no default value for `NoOfReplicas`, which always had to be specified explicitly in the `[ndbd default]` section. Although this parameter now has a default value of 2, which is the recommended setting in most common usage scenarios, it is still recommended practice to set this parameter explicitly.

The global configuration file must define the computers and nodes involved in the cluster and on which computers these nodes are located. An example of a simple configuration file for a cluster consisting of one management server, two data nodes and two MySQL servers is shown here:

```
# file "config.ini" - 2 data nodes and 2 SQL nodes
# This file is placed in the startup directory of ndb_mgmd (the
# management server)
# The first MySQL Server can be started from any host. The second
# can be started only on the host mysqld_5.mysql.com

[ndbd default]
NoOfReplicas= 2
DataDir= /var/lib/mysql-cluster

[ndb_mgmd]
Hostname= ndb_mgmd.mysql.com
DataDir= /var/lib/mysql-cluster

[ndbd]
HostName= ndbd_2.mysql.com

[ndbd]
HostName= ndbd_3.mysql.com

[mysqld]
[mysqld]
HostName= mysqld_5.mysql.com
```

Note

The preceding example is intended as a minimal starting configuration for purposes of familiarization with NDB Cluster, and is almost certain not to be sufficient for production settings. See Section 18.3.3.2, “Recommended Starting Configuration for NDB Cluster”, which provides a more complete example starting configuration.

Each node has its own section in the `config.ini` file. For example, this cluster has two data nodes, so the preceding configuration file contains two `[ndbd]` sections defining these nodes.

Note

Do not place comments on the same line as a section heading in the `config.ini` file; this causes the management server not to start because it cannot parse the configuration file in such cases.

Sections of the config.ini File

There are six different sections that you can use in the `config.ini` configuration file, as described in the following list:

- `[computer]`: Defines cluster hosts. This is not required to configure a viable NDB Cluster, but be may used as a convenience when setting up a large cluster. See Section 18.3.3.4, “Defining Computers in an NDB Cluster”, for more information.
- `[ndbd]`: Defines a cluster data node (`ndbd` process). See Section 18.3.3.6, “Defining NDB Cluster Data Nodes”, for details.
- `[mysqld]`: Defines the cluster's MySQL server nodes (also called SQL or API nodes). For a discussion of SQL node configuration, see Section 18.3.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”.
- `[mgm]` or `[ndb_mgmd]`: Defines a cluster management server (MGM) node. For information concerning the configuration of management nodes, see Section 18.3.3.5, “Defining an NDB Cluster Management Server”.
- `[tcp]`: Defines a TCP/IP connection between cluster nodes, with TCP/IP being the default transport protocol. Normally, `[tcp]` or `[tcp default]` sections are not required to set up an NDB Cluster, as the cluster handles this automatically; however, it may be necessary in some situations to override the defaults provided by the cluster. See Section 18.3.3.9, “NDB Cluster TCP/IP Connections”, for information about available TCP/IP configuration parameters and how to use them. (You may also find Section 18.3.3.10, “NDB Cluster TCP/IP Connections Using Direct Connections” to be of interest in some cases.)
- `[shm]`: Defines shared-memory connections between nodes. In MySQL 5.6, it is enabled by default, but should still be considered experimental. For a discussion of SHM interconnects, see Section 18.3.3.11, “NDB Cluster Shared-Memory Connections”.
- `[sci]`: Defines Scalable Coherent Interface connections between cluster data nodes. Not supported in NDB 7.3 or 7.4.

You can define `default` values for each section. If used, a `default` section should come before any other sections of that type. For example, an `[ndbd default]` section should appear in the configuration file before any `[ndbd]` sections.

NDB Cluster parameter names are case-insensitive, unless specified in MySQL Server `my.cnf` or `my.ini` files.

18.3.3.2 Recommended Starting Configuration for NDB Cluster

Achieving the best performance from an NDB Cluster depends on a number of factors including the following:

- NDB Cluster software version
- Numbers of data nodes and SQL nodes
- Hardware
- Operating system
- Amount of data to be stored
- Size and type of load under which the cluster is to operate

Therefore, obtaining an optimum configuration is likely to be an iterative process, the outcome of which can vary widely with the specifics of each NDB Cluster deployment. Changes in configuration are also likely to be indicated when changes are made in the platform on which the cluster is run, or in applications that use the NDB Cluster's data. For these reasons, it is not possible to offer a single configuration that is ideal for all usage scenarios. However, in this section, we provide a recommended base configuration.

Starting config.ini file. The following `config.ini` file is a recommended starting point for configuring a cluster running NDB Cluster 7.3 or later:

```
# TCP PARAMETERS

[tcp default]
SendBufferMemory=2M
ReceiveBufferMemory=2M

# Increasing the sizes of these two buffers beyond the default values
# helps prevent bottlenecks due to slow disk I/O.

# MANAGEMENT NODE PARAMETERS

[ndb_mgmd default]
DataDir=path/to/management/server/data/directory

# It is possible to use a different data directory for each management
# server, but for ease of administration it is preferable to be
# consistent.

[ndb_mgmd]
HostName=management-server-A-hostname
# NodeId=management-server-A-nodeid

[ndb_mgmd]
HostName=management-server-B-hostname
# NodeId=management-server-B-nodeid

# Using two2 management servers helps guarantee that there is always an
# arbitrator in the event of network partitioning, and so is recommended for
# high availability. Each management server must be identified by a HostName.
# You may for the sake of convenience specify a NodeId for any management
# server, although one is allocated for it automatically; if you do so, it must
# be in the range 1-255 inclusive and must be unique among all IDs specified for
# cluster nodes.

# DATA NODE PARAMETERS

[ndbd default]
NoOfReplicas=2

# Using two fragment replicas is recommended to guarantee availability of data;
# using only one fragment replica does not provide any redundancy, which means
# that the failure of a single data node causes the entire cluster to shut down.
# We do not recommend using more than two fragment replicas, since 2 is
# sufficient to provide high availability, and we do not currently test with
# greater values for this parameter.

LockPagesInMainMemory=1

# On Linux and Solaris systems, setting this parameter locks data node processes
# into memory. Doing so prevents them from swapping to disk, which can severely
# degrade cluster performance.

DataMemory=3072M
IndexMemory=384M

# The values provided for DataMemory and IndexMemory assume 4 GB RAM per data
# node. However, for best results, you should first calculate the memory that
# would be used based on the data you actually plan to store (you may find the
# ndb_size.pl utility helpful in estimating this), then allow an extra 20%
# over the calculated values. Naturally, you should ensure that each data node
# host has at least as much physical memory as the sum of these two values.

# ODirect=1

# Enabling this parameter causes NDBCLUSTER to try using O_DIRECT writes for
local checkpoints and redo logs; this can reduce load on CPUs. We recommend
doing so when using NDB Cluster on systems running Linux kernel 2.6 or later.

NoOfFragmentLogFiles=300
```

```
DataDir=path/to/data/node/data/directory
MaxNoOfConcurrentOperations=100000

SchedulerSpinTimer=400
SchedulerExecutionTimer=100
RealTimeScheduler=1
# Setting these parameters allows you to take advantage of real-time scheduling
# of NDB threads to achieve increased throughput when using ndbd. They
# are not needed when using ndbmtd; in particular, you should not set
# RealTimeScheduler for ndbmtd data nodes.

TimeBetweenGlobalCheckpoints=1000
TimeBetweenEpochs=200
DiskCheckpointSpeed=10M
DiskCheckpointSpeedInRestart=100M
# The two parameters just listed are deprecated in NDB 7.4.1 and later, where
# setting either or both of them has no effect; see
# Section 18.3.3.6, “Defining NDB Cluster Data Nodes”,
# for more information
RedoBuffer=32M

# CompressedLCP=1
# CompressedBackup=1
# Enabling CompressedLCP and CompressedBackup causes, respectively, local
# checkpoint files and backup files to be compressed, which can result in a
# space savings of up to 50% over noncompressed LCPs and backups.

# MaxNoOfLocalScans=64
MaxNoOfTables=1024
MaxNoOfOrderedIndexes=256

[ndbd]
HostName=data-node-A-hostname
# NodeId=data-node-A-nodeid

LockExecuteThreadToCPU=1
LockMaintThreadsToCPU=0
# On systems with multiple CPUs, these parameters can be used to lock NDBCLUSTER
# threads to specific CPUs

[ndbd]
HostName=data-node-B-hostname
# NodeId=data-node-B-nodeid

LockExecuteThreadToCPU=1
LockMaintThreadsToCPU=0

# You must have an [ndbd] section for every data node in the cluster; each of
# these sections must include a HostName. Each section may optionally include a
# NodeId for convenience, but in most cases, it is sufficient to allow the
# cluster to allocate node IDs dynamically. If you do specify the node ID for a
# data node, it must be in the range 1 to 48 inclusive and must be unique among
# all IDs specified for cluster nodes.

# SQL NODE / API NODE PARAMETERS

[mysqld]
# HostName=sql-node-A-hostname
# NodeId=sql-node-A-nodeid

[mysqld]

[mysqld]

# Each API or SQL node that connects to the cluster requires a [mysqld] or [api]
# section of its own. Each such section defines a connection slot; you should
# have at least as many of these sections in the config.ini file as the total
# number of API nodes and SQL nodes that you wish to have connected to the
# cluster at any given time. There is no performance or other penalty for having
# extra slots available in case you find later that you want or need more API or
# SQL nodes to connect to the cluster at the same time. If no HostName is
# specified for a given [mysqld] or [api] section, then *any* API or SQL node
```

```
# may use that slot to connect to the cluster. You may wish to use an explicit
# HostName for one connection slot to guarantee that an API or SQL node from
# that host can always connect to the cluster. If you wish to prevent API or SQL
# nodes from connecting from other than a desired host or hosts, then use a
# HostName for every [mysqld] or [api] section in the config.ini file. You can
# if you wish define a node ID (NodeId parameter) for any API or SQL node, but
# this is not necessary; if you do so, it must be in the range 1 to 255
# inclusive and must be unique among all IDs specified for cluster nodes.
```

Recommended my.cnf options for SQL nodes. MySQL Servers acting as NDB Cluster SQL nodes must always be started with the `--ndbcluster` and `--ndb-connectstring` options, either on the command line or in `my.cnf`. In addition, set the following options for all `mysqld` processes in the cluster, unless your setup requires otherwise:

- `--ndb-use-exact-count=0`
- `--ndb-index-stat-enable=0`
- `--ndb-force-send=1`
- `--optimizer-switch=engine_condition_pushdown=on`

18.3.3.3 NDB Cluster Connection Strings

With the exception of the NDB Cluster management server (`ndb_mgmd`), each node that is part of an NDB Cluster requires a *connection string* that points to the management server's location. This connection string is used in establishing a connection to the management server as well as in performing other tasks depending on the node's role in the cluster. The syntax for a connection string is as follows:

```
[nodeid=node_id, ]host-definition[, host-definition[, ...]]

host-definition:
    host_name[:port_number]
```

`node_id` is an integer greater than or equal to 1 which identifies a node in `config.ini`. *`host_name`* is a string representing a valid Internet host name or IP address. *`port_number`* is an integer referring to a TCP/IP port number.

```
example 1 (long):    "nodeid=2,myhost1:1100,myhost2:1100,198.51.100.3:1200"
example 2 (short):   "myhost1"
```

`localhost:1186` is used as the default connection string value if none is provided. If *`port_num`* is omitted from the connection string, the default port is 1186. This port should always be available on the network because it has been assigned by IANA for this purpose (see http://www.iana.org/assignments/port-numbers for details).

By listing multiple host definitions, it is possible to designate several redundant management servers. An NDB Cluster data or API node attempts to contact successive management servers on each host in the order specified, until a successful connection has been established.

It is also possible to specify in a connection string one or more bind addresses to be used by nodes having multiple network interfaces for connecting to management servers. A bind address consists of a hostname or network address and an optional port number. This enhanced syntax for connection strings is shown here:

```
[nodeid=node_id, ]
    [bind-address=host-definition, ]
    host-definition[; bind-address=host-definition]
    host-definition[; bind-address=host-definition]
    [, ...]]

host-definition:
    host_name[:port_number]
```

If a single bind address is used in the connection string *prior* to specifying any management hosts, then this address is used as the default for connecting to any of them (unless overridden for a given management server; see later in this section for an example). For example, the following connection string causes the node to use `198.51.100.242` regardless of the management server to which it connects:

```
bind-address=198.51.100.242, poseidon:1186, perch:1186
```

If a bind address is specified *following* a management host definition, then it is used only for connecting to that management node. Consider the following connection string:

```
poseidon:1186;bind-address=localhost, perch:1186;bind-address=198.51.100.242
```

In this case, the node uses `localhost` to connect to the management server running on the host named `poseidon` and `198.51.100.242` to connect to the management server running on the host named `perch`.

You can specify a default bind address and then override this default for one or more specific management hosts. In the following example, `localhost` is used for connecting to the management server running on host `poseidon`; since `198.51.100.242` is specified first (before any management server definitions), it is the default bind address and so is used for connecting to the management servers on hosts `perch` and `orca`:

```
bind-address=198.51.100.242,poseidon:1186;bind-address=localhost,perch:1186,orca:2200
```

There are a number of different ways to specify the connection string:

- Each executable has its own command-line option which enables specifying the management server at startup. (See the documentation for the respective executable.)
- It is also possible to set the connection string for all nodes in the cluster at once by placing it in a `[mysql_cluster]` section in the management server's `my.cnf` file.
- For backward compatibility, two other options are available, using the same syntax:

  1. Set the `NDB_CONNECTSTRING` environment variable to contain the connection string.
  2. Write the connection string for each executable into a text file named `Ndb.cfg` and place this file in the executable's startup directory.

  However, these are now deprecated and should not be used for new installations.

The recommended method for specifying the connection string is to set it on the command line or in the `my.cnf` file for each executable.

18.3.3.4 Defining Computers in an NDB Cluster

The `[computer]` section has no real significance other than serving as a way to avoid the need of defining host names for each node in the system. All parameters mentioned here are required.

- `Id`

Version (or later)	NDB 7.3.1
Type or units	string
Default	[none]
Range	...
Restart Type	IS

This is a unique identifier, used to refer to the host computer elsewhere in the configuration file.

Important

The computer ID is *not* the same as the node ID used for a management, API, or data node. Unlike the case with node IDs, you cannot use `NodeId` in place of `Id` in the `[computer]` section of the `config.ini` file.

- `HostName`

Version (or later)	NDB 7.3.1
Type or units	name or IP address
Default	[none]
Range	...
Restart Type	N

This is the computer's hostname or IP address.

Restart types. Information about the restart types used by the parameter descriptions in this section is shown in the following table:

Table 18.6 NDB Cluster restart types

Symbol	Restart Type	Description
N	Node	The parameter can be updated using a rolling restart (see Section 18.5.5, "Performing a Rolling Restart of an NDB Cluster")
S	System	All cluster nodes must be shut down completely, then restarted, to effect a change in this parameter
I	Initial	Data nodes must be restarted using the `--initial` option

18.3.3.5 Defining an NDB Cluster Management Server

The `[ndb_mgmd]` section is used to configure the behavior of the management server. If multiple management servers are employed, you can specify parameters common to all of them in an `[ndb_mgmd default]` section. `[mgm]` and `[mgm default]` are older aliases for these, supported for backward compatibility.

All parameters in the following list are optional and assume their default values if omitted.

Note

If neither the `ExecuteOnComputer` nor the `HostName` parameter is present, the default value `localhost` is assumed for both.

- `Id`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	[none]
Range	1 - 255
Restart Type	IS

Each node in the cluster has a unique identity. For a management node, this is represented by an integer value in the range 1 to 255, inclusive. This ID is used by all internal cluster messages for addressing the node, and so must be unique for each NDB Cluster node, regardless of the type of node.

Note

Data node IDs must be less than 49. If you plan to deploy a large number of data nodes, it is a good idea to limit the node IDs for management nodes (and API nodes) to values greater than 48.

The use of the `Id` parameter for identifying management nodes is deprecated in favor of `NodeId`. Although `Id` continues to be supported for backward compatibility, it now generates a warning and is subject to removal in a future version of NDB Cluster.

- `NodeId`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	[none]
Range	1 - 255
Restart Type	IS

Each node in the cluster has a unique identity. For a management node, this is represented by an integer value in the range 1 to 255 inclusive. This ID is used by all internal cluster messages for addressing the node, and so must be unique for each NDB Cluster node, regardless of the type of node.

Note

Data node IDs must be less than 49. If you plan to deploy a large number of data nodes, it is a good idea to limit the node IDs for management nodes (and API nodes) to values greater than 48.

`NodeId` is the preferred parameter name to use when identifying management nodes. Although the older `Id` continues to be supported for backward compatibility, it is now deprecated and generates a warning when used; it is also subject to removal in a future NDB Cluster release.

- `ExecuteOnComputer`

Version (or later)	NDB 7.3.1
Type or units	name
Default	[none]
Range	...
Restart Type	S
Deprecated	Yes (in NDB 7.5)

This refers to the `Id` set for one of the computers defined in a `[computer]` section of the `config.ini` file.

- `PortNumber`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	1186
Range	0 - 64K

Restart Type	S

This is the port number on which the management server listens for configuration requests and management commands.

- `HostName`

Version (or later)	NDB 7.3.1
Type or units	name or IP address
Default	[none]
Range	...
Restart Type	N

Specifying this parameter defines the hostname of the computer on which the management node is to reside. To specify a hostname other than `localhost`, either this parameter or `ExecuteOnComputer` is required.

- `LogDestination`

Version (or later)	NDB 7.3.1
Type or units	{CONSOLE|SYSLOG|FILE}
Default	FILE: filename=ndb_nodeid_cluster.log, maxsize=1000000, maxfiles=6
Range	...
Restart Type	N

This parameter specifies where to send cluster logging information. There are three options in this regard—`CONSOLE`, `SYSLOG`, and `FILE`—with `FILE` being the default:

- `CONSOLE` outputs the log to `stdout`:

```
CONSOLE
```

- `SYSLOG` sends the log to a `syslog` facility, possible values being one of `auth`, `authpriv`, `cron`, `daemon`, `ftp`, `kern`, `lpr`, `mail`, `news`, `syslog`, `user`, `uucp`, `local0`, `local1`, `local2`, `local3`, `local4`, `local5`, `local6`, or `local7`.

Note

Not every facility is necessarily supported by every operating system.

```
SYSLOG:facility=syslog
```

- `FILE` pipes the cluster log output to a regular file on the same machine. The following values can be specified:

  - `filename`: The name of the log file.

    In NDB Cluster 7.3 and later, the default log file name used in such cases is `ndb_nodeid_cluster.log` (in some older versions, the log file's default name, used if `FILE` was specified without also setting `filename`, was `logger.log`.).

  - `maxsize`: The maximum size (in bytes) to which the file can grow before logging rolls over to a new file. When this occurs, the old log file is renamed by appending `.N` to the file name, where `N` is the next number not yet used with this name.

  - `maxfiles`: The maximum number of log files.

```
FILE:filename=cluster.log,maxsize=1000000,maxfiles=6
```

The default value for the `FILE` parameter is `FILE:filename=ndb_node_id_cluster.log,maxsize=1000000,maxfiles=6`, where *`node_id`* is the ID of the node.

It is possible to specify multiple log destinations separated by semicolons as shown here:

```
CONSOLE;SYSLOG:facility=local0;FILE:filename=/var/log/mgmd
```

- `ArbitrationRank`

Version (or later)	NDB 7.3.1
Type or units	0-2
Default	1
Range	0 - 2
Restart Type	N

This parameter is used to define which nodes can act as arbitrators. Only management nodes and SQL nodes can be arbitrators. `ArbitrationRank` can take one of the following values:

- `0`: The node is never used as an arbitrator.
- `1`: The node has high priority; that is, it is preferred as an arbitrator over low-priority nodes.
- `2`: Indicates a low-priority node which is used as an arbitrator only if a node with a higher priority is not available for that purpose.

Normally, the management server should be configured as an arbitrator by setting its `ArbitrationRank` to 1 (the default for management nodes) and those for all SQL nodes to 0 (the default for SQL nodes).

You can disable arbitration completely either by setting `ArbitrationRank` to 0 on all management and SQL nodes, or by setting the `Arbitration` parameter in the `[ndbd default]` section of the `config.ini` global configuration file. Setting `Arbitration` causes any settings for `ArbitrationRank` to be disregarded.

- `ArbitrationDelay`

Version (or later)	NDB 7.3.1
Type or units	milliseconds
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

An integer value which causes the management server's responses to arbitration requests to be delayed by that number of milliseconds. By default, this value is 0; it is normally not necessary to change it.

- `DataDir`

Version (or later)	NDB 7.3.1
Type or units	path
Default	.
Range	...

Restart Type	N

This specifies the directory where output files from the management server are placed. These files include cluster log files, process output files, and the daemon's process ID (PID) file. (For log files, this location can be overridden by setting the `FILE` parameter for `LogDestination` as discussed previously in this section.)

The default value for this parameter is the directory in which `ndb_mgmd` is located.

- `PortNumberStats`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	[none]
Range	0 - 64K
Restart Type	N

This parameter specifies the port number used to obtain statistical information from an NDB Cluster management server. It has no default value.

- `Wan`

Version (or later)	NDB 7.3.1
Type or units	boolean
Default	false
Range	true, false
Restart Type	N

Use WAN TCP setting as default.

- `HeartbeatThreadPriority`

Version (or later)	NDB 7.3.1
Type or units	string
Default	[none]
Range	...
Restart Type	S

Set the scheduling policy and priority of heartbeat threads for management and API nodes.

The syntax for setting this parameter is shown here:

```
HeartbeatThreadPriority = policy[, priority]

policy:
  {FIFO | RR}
```

When setting this parameter, you must specify a policy. This is one of `FIFO` (first in, first out) or `RR` (round robin). The policy value is followed optionally by the priority (an integer).

- `ExtraSendBufferMemory`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	0

Range	0 - 32G
Restart Type	N

This parameter specifies the amount of transporter send buffer memory to allocate in addition to any that has been set using `TotalSendBufferMemory`, `SendBufferMemory`, or both.

- `TotalSendBufferMemory`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	0
Range	256K - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter is used to determine the total amount of memory to allocate on this node for shared send buffer memory among all configured transporters.

If this parameter is set, its minimum permitted value is 256KB; 0 indicates that the parameter has not been set. For more detailed information, see Section 18.3.3.12, “Configuring NDB Cluster Send Buffer Parameters”.

- `HeartbeatIntervalMgmdMgmd`

Version (or later)	NDB 7.3.3
Type or units	milliseconds
Default	1500
Range	100 - 4294967039 (0xFFFFFEFF)
Restart Type	N
Added	NDB 7.3.3

Specify the interval between heartbeat messages used to determine whether another management node is on contact with this one. The management node waits after 3 of these intervals to declare the connection dead; thus, the default setting of 1500 milliseconds causes the management node to wait for approximately 1600 ms before timing out.

This parameter was added in NDB 7.3.3. (Bug #16426805)

Note

After making changes in a management node's configuration, it is necessary to perform a rolling restart of the cluster for the new configuration to take effect.

To add new management servers to a running NDB Cluster, it is also necessary to perform a rolling restart of all cluster nodes after modifying any existing `config.ini` files. For more information about issues arising when using multiple management nodes, see Section 18.1.7.10, “Limitations Relating to Multiple NDB Cluster Nodes”.

Restart types. Information about the restart types used by the parameter descriptions in this section is shown in the following table:

Table 18.7 NDB Cluster restart types

Symbol	Restart Type	Description
N	Node	The parameter can be updated using a rolling restart (see Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”)
S	System	All cluster nodes must be shut down completely, then restarted, to effect a change in this parameter
I	Initial	Data nodes must be restarted using the `--initial` option

18.3.3.6 Defining NDB Cluster Data Nodes

The `[ndbd]` and `[ndbd default]` sections are used to configure the behavior of the cluster's data nodes.

`[ndbd]` and `[ndbd default]` are always used as the section names whether you are using `ndbd` or `ndbmtd` binaries for the data node processes.

There are many parameters which control buffer sizes, pool sizes, timeouts, and so forth. The only mandatory parameter is either one of `ExecuteOnComputer` or `HostName`; this must be defined in the local `[ndbd]` section.

The parameter `NoOfReplicas` should be defined in the `[ndbd default]` section, as it is common to all Cluster data nodes. It is not strictly necessary to set `NoOfReplicas`, but it is good practice to set it explicitly.

Most data node parameters are set in the `[ndbd default]` section. Only those parameters explicitly stated as being able to set local values are permitted to be changed in the `[ndbd]` section. Where present, `HostName`, `NodeId` and `ExecuteOnComputer` *must* be defined in the local `[ndbd]` section, and not in any other section of `config.ini`. In other words, settings for these parameters are specific to one data node.

For those parameters affecting memory usage or buffer sizes, it is possible to use `K`, `M`, or `G` as a suffix to indicate units of 1024, 1024×1024, or 1024×1024×1024. (For example, `100K` means 100 × 1024 = 102400.)

Parameter names and values are case-insensitive, unless used in a MySQL Server `my.cnf` or `my.ini` file, in which case they are case-sensitive.

Information about configuration parameters specific to NDB Cluster Disk Data tables can be found later in this section (see Disk Data Configuration Parameters).

All of these parameters also apply to `ndbmtd` (the multithreaded version of `ndbd`). Three additional data node configuration parameters—`MaxNoOfExecutionThreads`, `ThreadConfig`, and `NoOfFragmentLogParts`—apply to `ndbmtd` only; these have no effect when used with `ndbd`. For more information, see Multi-Threading Configuration Parameters (ndbmtd). See also Section 18.4.3, “`ndbmtd` — The NDB Cluster Data Node Daemon (Multi-Threaded)”.

Identifying data nodes. The `NodeId` or `Id` value (that is, the data node identifier) can be allocated on the command line when the node is started or in the configuration file.

- `Id`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	[none]
Range	1 - 48
Restart Type	IS

A unique node ID is used as the node's address for all cluster internal messages. For data nodes, this is an integer in the range 1 to 48 inclusive. Each node in the cluster must have a unique identifier.

`NodeId` is the preferred parameter name to use when identifying data nodes. Although the older `Id` is still supported for backward compatibility, it is now deprecated, and generates a warning when used. `Id` is also subject to removal in a future NDB Cluster release.

- `NodeId`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	[none]
Range	1 - 48
Restart Type	IS

A unique node ID is used as the node's address for all cluster internal messages. For data nodes, this is an integer in the range 1 to 48 inclusive. Each node in the cluster must have a unique identifier.

`NodeId` is the preferred parameter name to use when identifying data nodes. Although `Id` continues to be supported for backward compatibility, it is now deprecated, generates a warning when used, and is subject to removal in a future version of NDB Cluster.

- `ExecuteOnComputer`

Version (or later)	NDB 7.3.1
Type or units	name
Default	[none]
Range	...
Restart Type	S
Deprecated	Yes (in NDB 7.5)

This refers to the `Id` set for one of the computers defined in a `[computer]` section.

- `HostName`

Version (or later)	NDB 7.3.1
Type or units	name or IP address
Default	localhost
Range	...
Restart Type	N

Specifying this parameter defines the hostname of the computer on which the data node is to reside. To specify a hostname other than `localhost`, either this parameter or `ExecuteOnComputer` is required.

- `ServerPort`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	[none]

Range	1 - 64K
Restart Type	S

Each node in the cluster uses a port to connect to other nodes. By default, this port is allocated dynamically in such a way as to ensure that no two nodes on the same host computer receive the same port number, so it should normally not be necessary to specify a value for this parameter.

However, if you need to be able to open specific ports in a firewall to permit communication between data nodes and API nodes (including SQL nodes), you can set this parameter to the number of the desired port in an `[ndbd]` section or (if you need to do this for multiple data nodes) the `[ndbd default]` section of the `config.ini` file, and then open the port having that number for incoming connections from SQL nodes, API nodes, or both.

Note

Connections from data nodes to management nodes is done using the `ndb_mgmd` management port (the management server's `PortNumber`) so outgoing connections to that port from any data nodes should always be permitted.

- `TcpBind_INADDR_ANY`

  Setting this parameter to `TRUE` or `1` binds `IP_ADDR_ANY` so that connections can be made from anywhere (for autogenerated connections). The default is `FALSE` (`0`).

- `NodeGroup`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	[none]
Range	0 - 65536
Restart Type	IS

This parameter can be used to assign a data node to a specific node group. It is read only when the cluster is started for the first time, and cannot be used to reassign a data node to a different node group online. It is generally not desirable to use this parameter in the `[ndbd default]` section of the `config.ini` file, and care must be taken not to assign nodes to node groups in such a way that an invalid numbers of nodes are assigned to any node groups.

The `NodeGroup` parameter is chiefly intended for use in adding a new node group to a running NDB Cluster without having to perform a rolling restart. For this purpose, you should set it to 65536 (the maximum value). You are not required to set a `NodeGroup` value for all cluster data nodes, only for those nodes which are to be started and added to the cluster as a new node group at a later time. For more information, see Section 18.5.7.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”.

- `NoOfReplicas`

Version (or later)	NDB 7.3.1
Type or units	integer
Default	2
Range	1 - 2

Restart Type	IS

This global parameter can be set only in the `[ndbd default]` section, and defines the number of fragment replicas for each table stored in the cluster. This parameter also specifies the size of node groups. A node group is a set of nodes all storing the same information.

Node groups are formed implicitly. The first node group is formed by the set of data nodes with the lowest node IDs, the next node group by the set of the next lowest node identities, and so on. By way of example, assume that we have 4 data nodes and that `NoOfReplicas` is set to 2. The four data nodes have node IDs 2, 3, 4 and 5. Then the first node group is formed from nodes 2 and 3, and the second node group by nodes 4 and 5. It is important to configure the cluster in such a manner that nodes in the same node groups are not placed on the same computer because a single hardware failure would cause the entire cluster to fail.

If no node IDs are provided, the order of the data nodes is the determining factor for the node group. Whether or not explicit assignments are made, they can be viewed in the output of the management client `SHOW` command.

The default and recommended maximum value for `NoOfReplicas` is 2. *This is the recommended value for most production environments*.

Important

While it is theoretically possible for the value of this parameter to be 3 or 4, **NDB Cluster 7.3 and NDB Cluster 7.4 do not support setting `NoOfReplicas` to a value greater than 2 in production**.

Warning

Setting `NoOfReplicas` to 1 means that there is only a single copy of all Cluster data; in this case, the loss of a single data node causes the cluster to fail because there are no additional copies of the data stored by that node.

The number of data nodes in the cluster must be evenly divisible by the value of this parameter. For example, if there are two data nodes, then `NoOfReplicas` must be equal to either 1 or 2, since 2/3 and 2/4 both yield fractional values; if there are four data nodes, then `NoOfReplicas` must be equal to 1, 2, or 4.

- `DataDir`

Version (or later)	NDB 7.3.1
Type or units	path
Default	.
Range	...
Restart Type	IN

This parameter specifies the directory where trace files, log files, pid files and error logs are placed.

The default is the data node process working directory.

- `FileSystemPath`

Version (or later)	NDB 7.3.1
Type or units	path
Default	DataDir
Range	...

Restart Type	IN

This parameter specifies the directory where all files created for metadata, REDO logs, UNDO logs (for Disk Data tables), and data files are placed. The default is the directory specified by `DataDir`.

Note

This directory must exist before the `ndbd` process is initiated.

The recommended directory hierarchy for NDB Cluster includes `/var/lib/mysql-cluster`, under which a directory for the node's file system is created. The name of this subdirectory contains the node ID. For example, if the node ID is 2, this subdirectory is named `ndb_2_fs`.

- `BackupDataDir`

Version (or later)	NDB 7.3.1
Type or units	path
Default	FileSystemPath
Range	...
Restart Type	IN

This parameter specifies the directory in which backups are placed.

Important

The string '`/BACKUP`' is always appended to this value. For example, if you set the value of `BackupDataDir` to `/var/lib/cluster-data`, then all backups are stored under `/var/lib/cluster-data/BACKUP`. This also means that the *effective* default backup location is the directory named `BACKUP` under the location specified by the `FileSystemPath` parameter.

Data Memory, Index Memory, and String Memory

`DataMemory` and `IndexMemory` are `[ndbd]` parameters specifying the size of memory segments used to store the actual records and their indexes. In setting values for these, it is important to understand how `DataMemory` and `IndexMemory` are used, as they usually need to be updated to reflect actual usage by the cluster:

- `DataMemory`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	80M
Range	1M - 1T
Restart Type	N

This parameter defines the amount of space (in bytes) available for storing database records. The entire amount specified by this value is allocated in memory, so it is extremely important that the machine has sufficient physical memory to accommodate it.

The memory allocated by `DataMemory` is used to store both the actual records and indexes. There is a 16-byte overhead on each record; an additional amount for each record is incurred because it is stored in a 32KB page with 128 byte page overhead (see below). There is also a small amount wasted per page due to the fact that each record is stored in only one page.

For variable-size table attributes, the data is stored on separate data pages, allocated from `DataMemory`. Variable-length records use a fixed-size part with an extra overhead of 4 bytes to reference the variable-size part. The variable-size part has 2 bytes overhead plus 2 bytes per attribute.

The maximum record size is 14000 bytes.

The memory space defined by `DataMemory` is also used to store ordered indexes, which use about 10 bytes per record. Each table row is represented in the ordered index. A common error among users is to assume that all indexes are stored in the memory allocated by `IndexMemory`, but this is not the case: Only primary key and unique hash indexes use this memory; ordered indexes use the memory allocated by `DataMemory`. However, creating a primary key or unique hash index also creates an ordered index on the same keys, unless you specify `USING HASH` in the index creation statement. This can be verified by running `ndb_desc -d db_name table_name` in the management client.

NDB Cluster can use a maximum of 512 MB for hash indexes per partition, which means in some cases it is possible to get `Table is full` errors in MySQL client applications even when `ndb_mgm -e "ALL REPORT MEMORYUSAGE"` shows significant free `DataMemory`. This can also pose a problem with data node restarts on nodes that are heavily loaded with data.

You can force `NDB` to create extra partitions for NDB Cluster tables and thus have more memory available for hash indexes by using the `MAX_ROWS` option for `CREATE TABLE`. In general, setting `MAX_ROWS` to twice the number of rows that you expect to store in the table should be sufficient.

You can also use the `MinFreePct` configuration parameter to help avoid problems with node restarts.

The memory space allocated by `DataMemory` consists of 32KB pages, which are allocated to table fragments. Each table is normally partitioned into the same number of fragments as there are data nodes in the cluster. Thus, for each node, there are the same number of fragments as are set in `NoOfReplicas`.

Once a page has been allocated, it is currently not possible to return it to the pool of free pages, except by deleting the table. (This also means that `DataMemory` pages, once allocated to a given table, cannot be used by other tables.) Performing a data node recovery also compresses the partition because all records are inserted into empty partitions from other live nodes.

The `DataMemory` memory space also contains UNDO information: For each update, a copy of the unaltered record is allocated in the `DataMemory`. There is also a reference to each copy in the ordered table indexes. Unique hash indexes are updated only when the unique index columns are updated, in which case a new entry in the index table is inserted and the old entry is deleted upon commit. For this reason, it is also necessary to allocate enough memory to handle the largest transactions performed by applications using the cluster. In any case, performing a few large transactions holds no advantage over using many smaller ones, for the following reasons:

- Large transactions are not any faster than smaller ones
- Large transactions increase the number of operations that are lost and must be repeated in event of transaction failure
- Large transactions use more memory

The default value for `DataMemory` is 80MB; the minimum is 1MB. There is no maximum size, but in reality the maximum size has to be adapted so that the process does not start swapping when the limit is reached. This limit is determined by the amount of physical RAM available on the machine and by the amount of memory that the operating system may commit to any one process. 32-bit operating systems are generally limited to 2−4GB per process; 64-bit operating systems can use more. For large databases, it may be preferable to use a 64-bit operating system for this reason.

- IndexMemory

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	18M
Range	1M - 1T
Restart Type	N
Deprecated	Yes (in NDB 7.6)

This parameter controls the amount of storage used for hash indexes in NDB Cluster. Hash indexes are always used for primary key indexes, unique indexes, and unique constraints. When defining a primary key or a unique index, two indexes are created, one of which is a hash index used for all tuple accesses as well as lock handling. This index is also used to enforce unique constraints.

You can estimate the size of a hash index using this formula:

```
size  = ( (fragments * 32K) + (rows * 18) )
        * fragment_replicas
```

fragments is the number of fragments, *fragment_replicas* is the number of fragment replicas (normally 2), and *rows* is the number of rows. If a table has one million rows, eight fragments, and two fragment replicas, the expected index memory usage is calculated as shown here:

```
((8 * 32K) + (1000000 * 18)) * 2 = ((8 * 32768) + (1000000 * 18)) * 2
= (262144 + 18000000) * 2
= 18262144 * 2 = 36524288 bytes = ~35MB
```

Index statistics (when enabled) for ordered indexes are stored in the mysql.ndb_index_stat_sample table. Since this table has a hash index, this adds to index memory usage. An upper bound to the number of rows for a given ordered index can be calculated as follows:

```
sample_size= key_size + ((key_attributes + 1) * 4)

sample_rows = IndexStatSaveSize
              * ((0.01 * IndexStatSaveScale * log2(rows * sample_size)) + 1)
              / sample_size
```

In the preceding formula, *key_size* is the size of the ordered index key in bytes, *key_attributes* is the number ot attributes in the ordered index key, and *rows* is the number of rows in the base table.

Assume that table t1 has 1 million rows and an ordered index named ix1 on two four-byte integers. Assume in addition that IndexStatSaveSize and IndexStatSaveScale are set to their default values (32K and 100, respectively). Using the previous 2 formulas, we can calculate as follows:

```
sample_size = 8  + ((1 + 2) * 4) = 20 bytes

sample_rows = 32K
              * ((0.01 * 100 * log2(1000000*20)) + 1)
              / 20
              = 32768 * ( (1 * ~16.811) +1) / 20
              = 32768 * ~17.811 / 20
              = ~29182 rows
```

The expected index memory usage is thus 2 * 18 * 29182 = ~1050550 bytes.

The default value for IndexMemory is 18MB. The minimum is 1MB.

- `StringMemory`

Version (or later)	NDB 7.3.1
Type or units	% or bytes
Default	25
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	S

This parameter determines how much memory is allocated for strings such as table names, and is specified in an `[ndbd]` or `[ndbd default]` section of the `config.ini` file. A value between `0` and `100` inclusive is interpreted as a percent of the maximum default value, which is calculated based on a number of factors including the number of tables, maximum table name size, maximum size of `.FRM` files, `MaxNoOfTriggers`, maximum column name size, and maximum default column value.

A value greater than `100` is interpreted as a number of bytes.

The default value is 25—that is, 25 percent of the default maximum.

Under most circumstances, the default value should be sufficient, but when you have a great many Cluster tables (1000 or more), it is possible to get Error 773 `Out of string memory, please modify StringMemory config parameter: Permanent error: Schema error`, in which case you should increase this value. `25` (25 percent) is not excessive, and should prevent this error from recurring in all but the most extreme conditions.

The following example illustrates how memory is used for a table. Consider this table definition:

```
CREATE TABLE example (
  a INT NOT NULL,
  b INT NOT NULL,
  c INT NOT NULL,
  PRIMARY KEY(a),
  UNIQUE(b)
) ENGINE=NDBCLUSTER;
```

For each record, there are 12 bytes of data plus 12 bytes overhead. Having no nullable columns saves 4 bytes of overhead. In addition, we have two ordered indexes on columns `a` and `b` consuming roughly 10 bytes each per record. There is a primary key hash index on the base table using roughly 29 bytes per record. The unique constraint is implemented by a separate table with `b` as primary key and `a` as a column. This other table consumes an additional 29 bytes of index memory per record in the `example` table as well 8 bytes of record data plus 12 bytes of overhead.

Thus, for one million records, we need 58MB for index memory to handle the hash indexes for the primary key and the unique constraint. We also need 64MB for the records of the base table and the unique index table, plus the two ordered index tables.

You can see that hash indexes takes up a fair amount of memory space; however, they provide very fast access to the data in return. They are also used in NDB Cluster to handle uniqueness constraints.

The only partitioning algorithm is hashing and ordered indexes are local to each node. Thus, ordered indexes cannot be used to handle uniqueness constraints in the general case.

An important point for both `IndexMemory` and `DataMemory` is that the total database size is the sum of all data memory and all index memory for each node group. Each node group is used to store replicated information, so if there are four nodes with two fragment replicas, there are two node groups. Thus, the total data memory available is 2 × `DataMemory` for each data node.

It is highly recommended that `DataMemory` and `IndexMemory` be set to the same values for all nodes. Data distribution is even over all nodes in the cluster, so the maximum amount of space available for any node can be no greater than that of the smallest node in the cluster.

`DataMemory` and `IndexMemory` can be changed, but decreasing either of these can be risky; doing so can easily lead to a node or even an entire NDB Cluster that is unable to restart due to there being insufficient memory space. Increasing these values should be acceptable, but it is recommended that such upgrades are performed in the same manner as a software upgrade, beginning with an update of the configuration file, and then restarting the management server followed by restarting each data node in turn.

MinFreePct. A proportion (5% by default) of data node resources including `DataMemory` and `IndexMemory` is kept in reserve to insure that the data node does not exhaust its memory when performing a restart. This can be adjusted using the `MinFreePct` data node configuration parameter (default 5).

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	5
Range	0 - 100
Restart Type	N

Updates do not increase the amount of index memory used. Inserts take effect immediately; however, rows are not actually deleted until the transaction is committed.

Transaction parameters. The next few `[ndbd]` parameters that we discuss are important because they affect the number of parallel transactions and the sizes of transactions that can be handled by the system. `MaxNoOfConcurrentTransactions` sets the number of parallel transactions possible in a node. `MaxNoOfConcurrentOperations` sets the number of records that can be in update phase or locked simultaneously.

Both of these parameters (especially `MaxNoOfConcurrentOperations`) are likely targets for users setting specific values and not using the default value. The default value is set for systems using small transactions, to ensure that these do not use excessive memory.

`MaxDMLOperationsPerTransaction` sets the maximum number of DML operations that can be performed in a given transaction.

- `MaxNoOfConcurrentTransactions`

Version (or later)	NDB 7.3.1
Type or units	integer
Default	4096
Range	32 - 4294967039 (0xFFFFFEFF)
Restart Type	N
Deprecated	Yes (in NDB 8.0)

Each cluster data node requires a transaction record for each active transaction in the cluster. The task of coordinating transactions is distributed among all of the data nodes. The total number of transaction records in the cluster is the number of transactions in any given node times the number of nodes in the cluster.

Transaction records are allocated to individual MySQL servers. Each connection to a MySQL server requires at least one transaction record, plus an additional transaction object per table accessed by that connection. This means that a reasonable minimum for the total number of transactions in the cluster can be expressed as

```
TotalNoOfConcurrentTransactions =
    (maximum number of tables accessed in any single transaction + 1)
    * number of SQL nodes
```

Suppose that there are 10 SQL nodes using the cluster. A single join involving 10 tables requires 11 transaction records; if there are 10 such joins in a transaction, then 10 * 11 = 110 transaction records are required for this transaction, per MySQL server, or 110 * 10 = 1100 transaction records total. Each data node can be expected to handle TotalNoOfConcurrentTransactions / number of data nodes. For an NDB Cluster having 4 data nodes, this would mean setting `MaxNoOfConcurrentTransactions` on each data node to 1100 / 4 = 275. In addition, you should provide for failure recovery by ensuring that a single node group can accommodate all concurrent transactions; in other words, that each data node's MaxNoOfConcurrentTransactions is sufficient to cover a number of transactions equal to TotalNoOfConcurrentTransactions / number of node groups. If this cluster has a single node group, then `MaxNoOfConcurrentTransactions` should be set to 1100 (the same as the total number of concurrent transactions for the entire cluster).

In addition, each transaction involves at least one operation; for this reason, the value set for `MaxNoOfConcurrentTransactions` should always be no more than the value of `MaxNoOfConcurrentOperations`.

This parameter must be set to the same value for all cluster data nodes. This is due to the fact that, when a data node fails, the oldest surviving node re-creates the transaction state of all transactions that were ongoing in the failed node.

It is possible to change this value using a rolling restart, but the amount of traffic on the cluster must be such that no more transactions occur than the lower of the old and new levels while this is taking place.

The default value is 4096.

- `MaxNoOfConcurrentOperations`

Version (or later)	NDB 7.3.1
Type or units	integer
Default	32K
Range	32 - 4294967039 (0xFFFFFEFF)
Restart Type	N

It is a good idea to adjust the value of this parameter according to the size and number of transactions. When performing transactions which involve only a few operations and records, the default value for this parameter is usually sufficient. Performing large transactions involving many records usually requires that you increase its value.

Records are kept for each transaction updating cluster data, both in the transaction coordinator and in the nodes where the actual updates are performed. These records contain state information needed to find UNDO records for rollback, lock queues, and other purposes.

This parameter should be set at a minimum to the number of records to be updated simultaneously in transactions, divided by the number of cluster data nodes. For example, in a cluster which has four data nodes and which is expected to handle one million concurrent updates using transactions, you should set this value to 1000000 / 4 = 250000. To help provide resiliency against failures, it is suggested that you set this parameter to a value that is high enough to permit an individual data node to handle the load for its node group. In other words, you should set the value equal to `total number of concurrent operations / number of node groups`. (In the case where there

is a single node group, this is the same as the total number of concurrent operations for the entire cluster.)

Because each transaction always involves at least one operation, the value of `MaxNoOfConcurrentOperations` should always be greater than or equal to the value of `MaxNoOfConcurrentTransactions`.

Read queries which set locks also cause operation records to be created. Some extra space is allocated within individual nodes to accommodate cases where the distribution is not perfect over the nodes.

When queries make use of the unique hash index, there are actually two operation records used per record in the transaction. The first record represents the read in the index table and the second handles the operation on the base table.

The default value is 32768.

This parameter actually handles two values that can be configured separately. The first of these specifies how many operation records are to be placed with the transaction coordinator. The second part specifies how many operation records are to be local to the database.

A very large transaction performed on an eight-node cluster requires as many operation records in the transaction coordinator as there are reads, updates, and deletes involved in the transaction. However, the operation records of the are spread over all eight nodes. Thus, if it is necessary to configure the system for one very large transaction, it is a good idea to configure the two parts separately. `MaxNoOfConcurrentOperations` is always used to calculate the number of operation records in the transaction coordinator portion of the node.

It is also important to have an idea of the memory requirements for operation records. These consume about 1KB per record.

- `MaxNoOfLocalOperations`

Version (or later)	NDB 7.3.1
Type or units	integer
Default	UNDEFINED
Range	32 - 4294967039 (0xFFFFFEFF)
Restart Type	N
Deprecated	Yes (in NDB 8.0)

By default, this parameter is calculated as 1.1 × `MaxNoOfConcurrentOperations`. This fits systems with many simultaneous transactions, none of them being very large. If there is a need to handle one very large transaction at a time and there are many nodes, it is a good idea to override the default value by explicitly specifying this parameter.

- `MaxDMLOperationsPerTransaction`

Version (or later)	NDB 7.3.1
Type or units	operations (DML)
Default	4294967295
Range	32 - 4294967295
Restart Type	N

This parameter limits the size of a transaction. The transaction is aborted if it requires more than this many DML operations. The minimum number of operations per transaction is 32; however, you

can set MaxDMLOperationsPerTransaction to 0 to disable any limitation on the number of DML operations per transaction. The maximum (and default) is 4294967295.

Transaction temporary storage. The next set of [ndbd] parameters is used to determine temporary storage when executing a statement that is part of a Cluster transaction. All records are released when the statement is completed and the cluster is waiting for the commit or rollback.

The default values for these parameters are adequate for most situations. However, users with a need to support transactions involving large numbers of rows or operations may need to increase these values to enable better parallelism in the system, whereas users whose applications require relatively small transactions can decrease the values to save memory.

- MaxNoOfConcurrentIndexOperations

Version (or later)	NDB 7.3.1
Type or units	integer
Default	8K
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N
Deprecated	Yes (in NDB 8.0)

For queries using a unique hash index, another temporary set of operation records is used during a query's execution phase. This parameter sets the size of that pool of records. Thus, this record is allocated only while executing a part of a query. As soon as this part has been executed, the record is released. The state needed to handle aborts and commits is handled by the normal operation records, where the pool size is set by the parameter MaxNoOfConcurrentOperations.

The default value of this parameter is 8192. Only in rare cases of extremely high parallelism using unique hash indexes should it be necessary to increase this value. Using a smaller value is possible and can save memory if the DBA is certain that a high degree of parallelism is not required for the cluster.

- MaxNoOfFiredTriggers

Version (or later)	NDB 7.3.1
Type or units	integer
Default	4000
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N
Deprecated	Yes (in NDB 8.0)

The default value of MaxNoOfFiredTriggers is 4000, which is sufficient for most situations. In some cases it can even be decreased if the DBA feels certain the need for parallelism in the cluster is not high.

A record is created when an operation is performed that affects a unique hash index. Inserting or deleting a record in a table with unique hash indexes or updating a column that is part of a unique hash index fires an insert or a delete in the index table. The resulting record is used to represent this index table operation while waiting for the original operation that fired it to complete. This operation is short-lived but can still require a large number of records in its pool for situations with many parallel write operations on a base table containing a set of unique hash indexes.

- TransactionBufferMemory

Version (or later)	NDB 7.3.1

Type or units	bytes
Default	1M
Range	1K - 4294967039 (0xFFFFFEFF)
Restart Type	N

The memory affected by this parameter is used for tracking operations fired when updating index tables and reading unique indexes. This memory is used to store the key and column information for these operations. It is only very rarely that the value for this parameter needs to be altered from the default.

The default value for `TransactionBufferMemory` is 1MB.

Normal read and write operations use a similar buffer, whose usage is even more short-lived. The compile-time parameter `ZATTRBUF_FILESIZE` (found in `ndb/src/kernel/blocks/Dbtc/Dbtc.hpp`) set to 4000 × 128 bytes (500KB). A similar buffer for key information, `ZDATABUF_FILESIZE` (also in `Dbtc.hpp`) contains 4000 × 16 = 62.5KB of buffer space. `Dbtc` is the module that handles transaction coordination.

Scans and buffering. There are additional `[ndbd]` parameters in the `Dblqh` module (in `ndb/src/kernel/blocks/Dblqh/Dblqh.hpp`) that affect reads and updates. These include `ZATTRINBUF_FILESIZE`, set by default to 10000 × 128 bytes (1250KB) and `ZDATABUF_FILE_SIZE`, set by default to 10000*16 bytes (roughly 156KB) of buffer space. To date, there have been neither any reports from users nor any results from our own extensive tests suggesting that either of these compile-time limits should be increased.

- `MaxNoOfConcurrentScans`

Version (or later)	NDB 7.3.1
Type or units	integer
Default	256
Range	2 - 500
Restart Type	N

This parameter is used to control the number of parallel scans that can be performed in the cluster. Each transaction coordinator can handle the number of parallel scans defined for this parameter. Each scan query is performed by scanning all partitions in parallel. Each partition scan uses a scan record in the node where the partition is located, the number of records being the value of this parameter times the number of nodes. The cluster should be able to sustain `MaxNoOfConcurrentScans` scans concurrently from all nodes in the cluster.

Scans are actually performed in two cases. The first of these cases occurs when no hash or ordered indexes exists to handle the query, in which case the query is executed by performing a full table scan. The second case is encountered when there is no hash index to support the query but there is an ordered index. Using the ordered index means executing a parallel range scan. The order is kept on the local partitions only, so it is necessary to perform the index scan on all partitions.

The default value of `MaxNoOfConcurrentScans` is 256. The maximum value is 500.

- `MaxNoOfLocalScans`

Version (or later)	NDB 7.3.1
Type or units	integer
Default	4 * MaxNoOfConcurrentScans * [# of data nodes] + 2
Range	32 - 4294967039 (0xFFFFFEFF)

Restart Type	N
Deprecated	Yes (in NDB 8.0)

Specifies the number of local scan records if many scans are not fully parallelized. When the number of local scan records is not provided, it is calculated as shown here:

```
4 * MaxNoOfConcurrentScans * [# data nodes] + 2
```

The minimum value is 32.

- `BatchSizePerLocalScan`

Version (or later)	NDB 7.3.1
Type or units	integer
Default	256
Range	1 - 992
Restart Type	N
Deprecated	Yes (in NDB 8.0)

This parameter is used to calculate the number of lock records used to handle concurrent scan operations.

`BatchSizePerLocalScan` has a strong connection to the `BatchSize` defined in the SQL nodes.

- `LongMessageBuffer`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	4M
Range	512K - 4294967039 (0xFFFFFEFF)
Restart Type	N
Version (or later)	NDB 7.3.5
Type or units	bytes
Default	64M
Range	512K - 4294967039 (0xFFFFFEFF)
Restart Type	N

This is an internal buffer used for passing messages within individual nodes and between nodes. The default is 64MB. (Prior to NDB 7.3.5, this was 4MB.)

This parameter seldom needs to be changed from the default.

- `MaxParallelCopyInstances`

Version (or later)	NDB 7.4.3
Type or units	integer
Default	0
Range	0 - 64
Restart Type	S

Added	NDB 7.4.3

This parameter sets the parallelization used in the copy phase of a node restart or system restart, when a node that is currently just starting is synchronised with a node that already has current data by copying over any changed records from the node that is up to date. Because full parallelism in such cases can lead to overload situations, `MaxParallelCopyInstances` was introduced in NDB 7.4.3 to provide a means to decrease it. This parameter's default value 0. This value means that the effective parallelism is equal to the number of LDM instances in the node just starting as well as the node updating it.

- `MaxParallelScansPerFragment`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	256
Range	1 - 4294967039 (0xFFFFFEFF)
Restart Type	N

It is possible to configure the maximum number of parallel scans (`TUP` scans and `TUX` scans) allowed before they begin queuing for serial handling. You can increase this to take advantage of any unused CPU when performing large number of scans in parallel and improve their performance.

Memory Allocation

`MaxAllocate`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	32M
Range	1M - 1G
Restart Type	N

This is the maximum size of the memory unit to use when allocating memory for tables. In cases where `NDB` gives `Out of memory` errors, but it is evident by examining the cluster logs or the output of `DUMP 1000` that all available memory has not yet been used, you can increase the value of this parameter (or `MaxNoOfTables`, or both) to cause `NDB` to make sufficient memory available.

Hash Map Size

`DefaultHashMapSize`

Version (or later)	NDB 7.3.1
Type or units	LDM threads
Default	240
Range	0 - 3840
Restart Type	N

The size of the table hash maps used by `NDB` is configurable using this parameter. `DefaultHashMapSize` can take any of three possible values (0, 240, 3840).

This parameter was intended to facilitate upgrades from very old NDB Cluster versions to NDB 7.3 and later, but should no longer need to be set.

Decreasing this parameter online after any tables have been created or modified with `DefaultHashMapSize` equal to 3840 is not currently supported.

Logging and checkpointing. The following `[ndbd]` parameters control log and checkpoint behavior.

- `NoOfFragmentLogFiles`

Version (or later)	NDB 7.3.1
Type or units	integer
Default	16
Range	3 - 4294967039 (0xFFFFFEFF)
Restart Type	IN

This parameter sets the number of REDO log files for the node, and thus the amount of space allocated to REDO logging. Because the REDO log files are organized in a ring, it is extremely important that the first and last log files in the set (sometimes referred to as the “head” and “tail” log files, respectively) do not meet. When these approach one another too closely, the node begins aborting all transactions encompassing updates due to a lack of room for new log records.

A `REDO` log record is not removed until the required number of local checkpoints has been completed since that log record was inserted. (In NDB Cluster 7.3 and later, only 2 local checkpoints are necessary). Checkpointing frequency is determined by its own set of configuration parameters discussed elsewhere in this chapter.

The default parameter value is 16, which by default means 16 sets of 4 16MB files for a total of 1024MB. The size of the individual log files is configurable using the `FragmentLogFileSize` parameter. In scenarios requiring a great many updates, the value for `NoOfFragmentLogFiles` may need to be set as high as 300 or even higher to provide sufficient space for REDO logs.

If the checkpointing is slow and there are so many writes to the database that the log files are full and the log tail cannot be cut without jeopardizing recovery, all updating transactions are aborted with internal error code 410 (`Out of log file space temporarily`). This condition prevails until a checkpoint has completed and the log tail can be moved forward.

Important

This parameter cannot be changed “on the fly”; you must restart the node using `--initial`. If you wish to change this value for all data nodes in a running cluster, you can do so using a rolling node restart (using `--initial` when starting each data node).

- `FragmentLogFileSize`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	16M
Range	4M - 1G
Restart Type	IN

Setting this parameter enables you to control directly the size of redo log files. This can be useful in situations when NDB Cluster is operating under a high load and it is unable to close fragment log files quickly enough before attempting to open new ones (only 2 fragment log files can be open at one time); increasing the size of the fragment log files gives the cluster more time before having to open each new fragment log file. The default value for this parameter is 16M.

For more information about fragment log files, see the description for `NoOfFragmentLogFiles`.

- `InitFragmentLogFiles`

Version (or later)	NDB 7.3.1
Type or units	[see values]
Default	SPARSE
Range	SPARSE, FULL
Restart Type	IN

By default, fragment log files are created sparsely when performing an initial start of a data node—that is, depending on the operating system and file system in use, not all bytes are necessarily written to disk. However, it is possible to override this behavior and force all bytes to be written, regardless of the platform and file system type being used, by means of this parameter. `InitFragmentLogFiles` takes either of two values:

- `SPARSE`. Fragment log files are created sparsely. This is the default value.
- `FULL`. Force all bytes of the fragment log file to be written to disk.

Depending on your operating system and file system, setting `InitFragmentLogFiles=FULL` may help eliminate I/O errors on writes to the REDO log.

- `MaxNoOfOpenFiles`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	0
Range	20 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter sets a ceiling on how many internal threads to allocate for open files. *Any situation requiring a change in this parameter should be reported as a bug*.

The default value is 0. However, the minimum value to which this parameter can be set is 20.

- `InitialNoOfOpenFiles`

Version (or later)	NDB 7.3.1
Type or units	files
Default	27
Range	20 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter sets the initial number of internal threads to allocate for open files.

The default value is 27.

- `MaxNoOfSavedMessages`

Version (or later)	NDB 7.3.1
Type or units	integer
Default	25
Range	0 - 4294967039 (0xFFFFFEFF)

Restart Type	N

This parameter sets the maximum number of errors iwritten n the error log as well as the maximum number of trace files that are kept before overwriting the existing ones. Trace files are generated when, for whatever reason, the node crashes.

The default is 25, which sets these maximums to 25 error messages and 25 trace files.

- `MaxLCPStartDelay`

Version (or later)	NDB 7.3.1
Type or units	seconds
Default	0
Range	0 - 600
Restart Type	N

In parallel data node recovery, only table data is actually copied and synchronized in parallel; synchronization of metadata such as dictionary and checkpoint information is done in a serial fashion. In addition, recovery of dictionary and checkpoint information cannot be executed in parallel with performing of local checkpoints. This means that, when starting or restarting many data nodes concurrently, data nodes may be forced to wait while a local checkpoint is performed, which can result in longer node recovery times.

It is possible to force a delay in the local checkpoint to permit more (and possibly all) data nodes to complete metadata synchronization; once each data node's metadata synchronization is complete, all of the data nodes can recover table data in parallel, even while the local checkpoint is being executed. To force such a delay, set `MaxLCPStartDelay`, which determines the number of seconds the cluster can wait to begin a local checkpoint while data nodes continue to synchronize metadata. This parameter should be set in the `[ndbd default]` section of the `config.ini` file, so that it is the same for all data nodes. The maximum value is 600; the default is 0.

- `LcpScanProgressTimeout`

Version (or later)	NDB 7.3.3
Type or units	second
Default	60
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N
Added	NDB 7.3.3

A local checkpoint fragment scan watchdog checks periodically for no progress in each fragment scan performed as part of a local checkpoint, and shuts down the node if there is no progress after a given amount of time has elapsed. Prior to NDB 7.3.3, this interval is always 60 seconds (Bug #16630410). In NDB 7.3.3 and later, this interval can be set using the `LcpScanProgressTimeout` data node configuration parameter, which sets the maximum time for which the local checkpoint can be stalled before the LCP fragment scan watchdog shuts down the node.

The default value is 60 seconds (providing compatibility with previous releases). Setting this parameter to 0 disables the LCP fragment scan watchdog altogether.

Metadata objects. The next set of `[ndbd]` parameters defines pool sizes for metadata objects, used to define the maximum number of attributes, tables, indexes, and trigger objects used by indexes, events, and replication between clusters.

Note

These act merely as “suggestions” to the cluster, and any that are not specified revert to the default values shown.

- `MaxNoOfAttributes`

Version (or later)	NDB 7.3.1
Type or units	integer
Default	1000
Range	32 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter sets a suggested maximum number of attributes that can be defined in the cluster; like `MaxNoOfTables`, it is not intended to function as a hard upper limit.

(In older NDB Cluster releases, this parameter was sometimes treated as a hard limit for certain operations. This caused problems with NDB Cluster Replication, when it was possible to create more tables than could be replicated, and sometimes led to confusion when it was possible [or not possible, depending on the circumstances] to create more than `MaxNoOfAttributes` attributes.)

The default value is 1000, with the minimum possible value being 32. The maximum is 4294967039. Each attribute consumes around 200 bytes of storage per node due to the fact that all metadata is fully replicated on the servers.

When setting `MaxNoOfAttributes`, it is important to prepare in advance for any `ALTER TABLE` statements that you might want to perform in the future. This is due to the fact, during the execution of `ALTER TABLE` on a Cluster table, 3 times the number of attributes as in the original table are used, and a good practice is to permit double this amount. For example, if the NDB Cluster table having the greatest number of attributes (*`greatest_number_of_attributes`*) has 100 attributes, a good starting point for the value of `MaxNoOfAttributes` would be `6 * greatest_number_of_attributes = 600`.

You should also estimate the average number of attributes per table and multiply this by `MaxNoOfTables`. If this value is larger than the value obtained in the previous paragraph, you should use the larger value instead.

Assuming that you can create all desired tables without any problems, you should also verify that this number is sufficient by trying an actual `ALTER TABLE` after configuring the parameter. If this is not successful, increase `MaxNoOfAttributes` by another multiple of `MaxNoOfTables` and test it again.

- `MaxNoOfTables`

Version (or later)	NDB 7.3.1
Type or units	integer
Default	128
Range	8 - 20320
Restart Type	N

A table object is allocated for each table and for each unique hash index in the cluster. This parameter sets a suggested maximum number of table objects for the cluster as a whole; like `MaxNoOfAttributes`, it is not intended to function as a hard upper limit.

(In older NDB Cluster releases, this parameter was sometimes treated as a hard limit for certain operations. This caused problems with NDB Cluster Replication, when it was possible to create

more tables than could be replicated, and sometimes led to confusion when it was possible [or not possible, depending on the circumstances] to create more than `MaxNoOfTables` tables.)

For each attribute that has a `BLOB` data type an extra table is used to store most of the `BLOB` data. These tables also must be taken into account when defining the total number of tables.

The default value of this parameter is 128. The minimum is 8 and the maximum is 20320. Each table object consumes approximately 20KB per node.

Note

The sum of `MaxNoOfTables`, `MaxNoOfOrderedIndexes`, and `MaxNoOfUniqueHashIndexes` must not exceed $2^{32} - 2$ (4294967294).

- `MaxNoOfOrderedIndexes`

Version (or later)	NDB 7.3.1
Type or units	integer
Default	128
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

For each ordered index in the cluster, an object is allocated describing what is being indexed and its storage segments. By default, each index so defined also defines an ordered index. Each unique index and primary key has both an ordered index and a hash index. `MaxNoOfOrderedIndexes` sets the total number of ordered indexes that can be in use in the system at any one time.

The default value of this parameter is 128. Each index object consumes approximately 10KB of data per node.

Note

The sum of `MaxNoOfTables`, `MaxNoOfOrderedIndexes`, and `MaxNoOfUniqueHashIndexes` must not exceed $2^{32} - 2$ (4294967294).

- `MaxNoOfUniqueHashIndexes`

Version (or later)	NDB 7.3.1
Type or units	integer
Default	64
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

For each unique index that is not a primary key, a special table is allocated that maps the unique key to the primary key of the indexed table. By default, an ordered index is also defined for each unique index. To prevent this, you must specify the `USING HASH` option when defining the unique index.

The default value is 64. Each index consumes approximately 15KB per node.

Note

The sum of `MaxNoOfTables`, `MaxNoOfOrderedIndexes`, and `MaxNoOfUniqueHashIndexes` must not exceed $2^{32} - 2$ (4294967294).

- `MaxNoOfTriggers`

Version (or later)	NDB 7.3.1

Type or units	integer
Default	768
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Internal update, insert, and delete triggers are allocated for each unique hash index. (This means that three triggers are created for each unique hash index.) However, an *ordered* index requires only a single trigger object. Backups also use three trigger objects for each normal table in the cluster.

Replication between clusters also makes use of internal triggers.

This parameter sets the maximum number of trigger objects in the cluster.

The default value is 768.

- `MaxNoOfIndexes`

  This parameter is deprecated in NDB 7.4 and is no longer available in NDB Cluster 7.5. You should use `MaxNoOfOrderedIndexes` and `MaxNoOfUniqueHashIndexes` instead.

  This parameter is used only by unique hash indexes. There needs to be one record in this pool for each unique hash index defined in the cluster.

  The default value of this parameter is 128.

- `MaxNoOfSubscriptions`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

  Each `NDB` table in an NDB Cluster requires a subscription in the NDB kernel. For some NDB API applications, it may be necessary or desirable to change this parameter. However, for normal usage with MySQL servers acting as SQL nodes, there is not any need to do so.

  The default value for `MaxNoOfSubscriptions` is 0, which is treated as equal to `MaxNoOfTables`. Each subscription consumes 108 bytes.

- `MaxNoOfSubscribers`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

  This parameter is of interest only when using NDB Cluster Replication. The default value is 0, which is treated as `2 * MaxNoOfTables`; that is, there is one subscription per `NDB` table for each of two MySQL servers (one acting as the replication source and the other as the replica). Each subscriber uses 16 bytes of memory.

  When using circular replication, multi-source replication, and other replication setups involving more than 2 MySQL servers, you should increase this parameter to the number of `mysqld` processes included in replication (this is often, but not always, the same as the number of clusters). For

example, if you have a circular replication setup using three NDB Clusters, with one `mysqld` attached to each cluster, and each of these `mysqld` processes acts as a source and as a replica, you should set `MaxNoOfSubscribers` equal to `3 * MaxNoOfTables`.

For more information, see Section 18.6, “NDB Cluster Replication”.

- `MaxNoOfConcurrentSubOperations`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	256
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter sets a ceiling on the number of operations that can be performed by all API nodes in the cluster at one time. The default value (256) is sufficient for normal operations, and might need to be adjusted only in scenarios where there are a great many API nodes each performing a high volume of operations concurrently.

Boolean parameters. The behavior of data nodes is also affected by a set of `[ndbd]` parameters taking on boolean values. These parameters can each be specified as `TRUE` by setting them equal to `1` or `Y`, and as `FALSE` by setting them equal to `0` or `N`.

- `LateAlloc`

Version (or later)	NDB 7.3.1
Type or units	numeric
Default	1
Range	0 - 1
Restart Type	N

Allocate memory for this data node after a connection to the management server has been established. Enabled by default.

- `LockPagesInMainMemory`

Version (or later)	NDB 7.3.1
Type or units	numeric
Default	0
Range	0 - 2
Restart Type	N

For a number of operating systems, including Solaris and Linux, it is possible to lock a process into memory and so avoid any swapping to disk. This can be used to help guarantee the cluster's real-time characteristics.

This parameter takes one of the integer values `0`, `1`, or `2`, which act as shown in the following list:

- `0`: Disables locking. This is the default value.
- `1`: Performs the lock after allocating memory for the process.
- `2`: Performs the lock before memory for the process is allocated.

If the operating system is not configured to permit unprivileged users to lock pages, then the data node process making use of this parameter may have to be run as system root. (`LockPagesInMainMemory` uses the `mlockall` function. From Linux kernel 2.6.9, unprivileged users can lock memory as limited by `max locked memory`. For more information, see `ulimit -l` and http://linux.die.net/man/2/mlock).

Note

In older NDB Cluster releases, this parameter was a Boolean. `0` or `false` was the default setting, and disabled locking. `1` or `true` enabled locking of the process after its memory was allocated. NDB Cluster 7.3 and later treats using `true` or `false` for the value of this parameter as an error.

Important

Beginning with `glibc` 2.10, `glibc` uses per-thread arenas to reduce lock contention on a shared pool, which consumes real memory. In general, a data node process does not need per-thread arenas, since it does not perform any memory allocation after startup. (This difference in allocators does not appear to affect performance significantly.)

The `glibc` behavior is intended to be configurable via the `MALLOC_ARENA_MAX` environment variable, but a bug in this mechanism prior to `glibc` 2.16 meant that this variable could not be set to less than 8, so that the wasted memory could not be reclaimed. (Bug #15907219; see also http://sourceware.org/bugzilla/show_bug.cgi?id=13137 for more information concerning this issue.)

One possible workaround for this problem is to use the `LD_PRELOAD` environment variable to preload a `jemalloc` memory allocation library to take the place of that supplied with `glibc`.

- `StopOnError`

Version (or later)	NDB 7.3.1
Type or units	boolean
Default	1
Range	0, 1
Restart Type	N

This parameter specifies whether a data node process should exit or perform an automatic restart when an error condition is encountered.

This parameter's default value is 1; this means that, by default, an error causes the data node process to halt.

When an error is encountered and `StopOnError` is 0, the data node process is restarted.

Prior to NDB Cluster 7.4.14, if the data node process exits in an uncontrolled fashion (due, for example, to performing `kill -9` on the data node process while performing a query, or to a segmentation fault), and `StopOnError` is set to 0, the angel process attempts to restart it in exactly the same way as it was started previously—that is, using the same startup options that were employed the last time the node was started. Thus, if the data node process was originally started using the `--initial` option, it is also restarted with `--initial`. This means that, in such cases, if the failure occurs on a sufficient number of data nodes in a very short interval, the effect is the same

as if you had performed an initial restart of the entire cluster, leading to loss of all data. This issue is resolved in NDB Cluster 7.4.14 and later NDB 7.4 releases (Bug #83510, Bug #24945638).

Users of MySQL Cluster Manager should note that, when `StopOnError` equals 1, this prevents the MySQL Cluster Manager agent from restarting any data nodes after it has performed its own restart and recovery. See Starting and Stopping the Agent on Linux, for more information.

- `CrashOnCorruptedTuple`

Version (or later)	NDB 7.3.1
Type or units	boolean
Default	true
Range	true, false
Restart Type	S

When this parameter is enabled, it forces a data node to shut down whenever it encounters a corrupted tuple. In NDB Cluster 7.3 and later, it is enabled by default.

- `Diskless`

Version (or later)	NDB 7.3.1
Type or units	true|false (1|0)
Default	false
Range	true, false
Restart Type	IS

It is possible to specify NDB Cluster tables as *diskless*, meaning that tables are not checkpointed to disk and that no logging occurs. Such tables exist only in main memory. A consequence of using diskless tables is that neither the tables nor the records in those tables survive a crash. However, when operating in diskless mode, it is possible to run `ndbd` on a diskless computer.

Important

This feature causes the *entire* cluster to operate in diskless mode.

When this feature is enabled, Cluster online backup is disabled. In addition, a partial start of the cluster is not possible.

`Diskless` is disabled by default.

- `ODirect`

Version (or later)	NDB 7.3.1
Type or units	boolean
Default	false
Range	true, false
Restart Type	N

Enabling this parameter causes `NDB` to attempt using `O_DIRECT` writes for LCP, backups, and redo logs, often lowering `kswapd` and CPU usage. When using NDB Cluster on Linux, enable `ODirect` if you are using a 2.6 or later kernel.

`ODirect` is disabled by default.

- `RestartOnErrorInsert`

Version (or later)	NDB 7.3.1
Type or units	error code
Default	2
Range	0 - 4
Restart Type	N

This feature is accessible only when building the debug version where it is possible to insert errors in the execution of individual blocks of code as part of testing.

This feature is disabled by default.

- `CompressedLCP`

Version (or later)	NDB 7.3.1
Type or units	boolean
Default	false
Range	true, false
Restart Type	N

Setting this parameter to `1` causes local checkpoint files to be compressed. The compression used is equivalent to `gzip --fast`, and can save 50% or more of the space required on the data node to store uncompressed checkpoint files. Compressed LCPs can be enabled for individual data nodes, or for all data nodes (by setting this parameter in the `[ndbd default]` section of the `config.ini` file).

Important

You cannot restore a compressed local checkpoint to a cluster running a MySQL version that does not support this feature.

The default value is `0` (disabled).

Controlling Timeouts, Intervals, and Disk Paging

There are a number of `[ndbd]` parameters specifying timeouts and intervals between various actions in Cluster data nodes. Most of the timeout values are specified in milliseconds. Any exceptions to this are mentioned where applicable.

- `TimeBetweenWatchDogCheck`

Version (or later)	NDB 7.3.1
Type or units	milliseconds
Default	6000
Range	70 - 4294967039 (0xFFFFFEFF)
Restart Type	N

To prevent the main thread from getting stuck in an endless loop at some point, a “watchdog” thread checks the main thread. This parameter specifies the number of milliseconds between checks. If the process remains in the same state after three checks, the watchdog thread terminates it.

This parameter can easily be changed for purposes of experimentation or to adapt to local conditions. It can be specified on a per-node basis although there seems to be little reason for doing so.

The default timeout is 6000 milliseconds (6 seconds).

- `TimeBetweenWatchDogCheckInitial`

Version (or later)	NDB 7.3.1
Type or units	milliseconds
Default	6000
Range	70 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This is similar to the `TimeBetweenWatchDogCheck` parameter, except that `TimeBetweenWatchDogCheckInitial` controls the amount of time that passes between execution checks inside a storage node in the early start phases during which memory is allocated.

The default timeout is 6000 milliseconds (6 seconds).

- `StartPartialTimeout`

Version (or later)	NDB 7.3.1
Type or units	milliseconds
Default	30000
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter specifies how long the Cluster waits for all data nodes to come up before the cluster initialization routine is invoked. This timeout is used to avoid a partial Cluster startup whenever possible.

This parameter is overridden when performing an initial start or initial restart of the cluster.

The default value is 30000 milliseconds (30 seconds). 0 disables the timeout, in which case the cluster may start only if all nodes are available.

- `StartPartitionedTimeout`

Version (or later)	NDB 7.3.1
Type or units	milliseconds
Default	60000
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

If the cluster is ready to start after waiting for `StartPartialTimeout` milliseconds but is still possibly in a partitioned state, the cluster waits until this timeout has also passed. If `StartPartitionedTimeout` is set to 0, the cluster waits indefinitely.

This parameter is overridden when performing an initial start or initial restart of the cluster.

The default timeout is 60000 milliseconds (60 seconds).

- `StartFailureTimeout`

Version (or later)	NDB 7.3.1
Type or units	milliseconds

Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

If a data node has not completed its startup sequence within the time specified by this parameter, the node startup fails. Setting this parameter to 0 (the default value) means that no data node timeout is applied.

For nonzero values, this parameter is measured in milliseconds. For data nodes containing extremely large amounts of data, this parameter should be increased. For example, in the case of a data node containing several gigabytes of data, a period as long as 10−15 minutes (that is, 600000 to 1000000 milliseconds) might be required to perform a node restart.

- `StartNoNodeGroupTimeout`

Version (or later)	NDB 7.3.1
Type or units	milliseconds
Default	15000
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

When a data node is configured with `Nodegroup = 65536`, is regarded as not being assigned to any node group. When that is done, the cluster waits `StartNoNodegroupTimeout` milliseconds, then treats such nodes as though they had been added to the list passed to the `--nowait-nodes` option, and starts. The default value is `15000` (that is, the management server waits 15 seconds). Setting this parameter equal to `0` means that the cluster waits indefinitely.

`StartNoNodegroupTimeout` must be the same for all data nodes in the cluster; for this reason, you should always set it in the `[ndbd default]` section of the `config.ini` file, rather than for individual data nodes.

See Section 18.5.7, “Adding NDB Cluster Data Nodes Online”, for more information.

- `HeartbeatIntervalDbDb`

Version (or later)	NDB 7.3.1
Type or units	milliseconds
Default	5000
Range	10 - 4294967039 (0xFFFFFEFF)
Restart Type	N

One of the primary methods of discovering failed nodes is by the use of heartbeats. This parameter states how often heartbeat signals are sent and how often to expect to receive them. Heartbeats cannot be disabled.

After missing four heartbeat intervals in a row, the node is declared dead. Thus, the maximum time for discovering a failure through the heartbeat mechanism is five times the heartbeat interval.

In NDB Cluster 7.3 and later, the default heartbeat interval is 5000 milliseconds (5 seconds). This parameter must not be changed drastically and should not vary widely between nodes. If one node uses 5000 milliseconds and the node watching it uses 1000 milliseconds, obviously the node is

declared dead very quickly. This parameter can be changed during an online software upgrade, but only in small increments.

See also Network communication and latency, as well as the description of the `ConnectCheckIntervalDelay` configuration parameter.

- `HeartbeatIntervalDbApi`

Version (or later)	NDB 7.3.1
Type or units	milliseconds
Default	1500
Range	100 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Each data node sends heartbeat signals to each MySQL server (SQL node) to ensure that it remains in contact. If a MySQL server fails to send a heartbeat in time it is declared “dead,” in which case all ongoing transactions are completed and all resources released. The SQL node cannot reconnect until all activities initiated by the previous MySQL instance have been completed. The three-heartbeat criteria for this determination are the same as described for `HeartbeatIntervalDbDb`.

The default interval is 1500 milliseconds (1.5 seconds). This interval can vary between individual data nodes because each data node watches the MySQL servers connected to it, independently of all other data nodes.

For more information, see Network communication and latency.

- `HeartbeatOrder`

Version (or later)	NDB 7.3.1
Type or units	numeric
Default	0
Range	0 - 65535
Restart Type	S

Data nodes send heartbeats to one another in a circular fashion whereby each data node monitors the previous one. If a heartbeat is not detected by a given data node, this node declares the previous data node in the circle “dead” (that is, no longer accessible by the cluster). The determination that a data node is dead is done globally; in other words; once a data node is declared dead, it is regarded as such by all nodes in the cluster.

It is possible for heartbeats between data nodes residing on different hosts to be too slow compared to heartbeats between other pairs of nodes (for example, due to a very low heartbeat interval or temporary connection problem), such that a data node is declared dead, even though the node can still function as part of the cluster. .

In this type of situation, it may be that the order in which heartbeats are transmitted between data nodes makes a difference as to whether or not a particular data node is declared dead. If this

declaration occurs unnecessarily, this can in turn lead to the unnecessary loss of a node group and as thus to a failure of the cluster.

Consider a setup where there are 4 data nodes A, B, C, and D running on 2 host computers `host1` and `host2`, and that these data nodes make up 2 node groups, as shown in the following table:

Table 18.8 Four data nodes A, B, C, D running on two host computers host1, host2; each data node belongs to one of two node groups.

Node Group	Nodes Running on `host1`	Nodes Running on `host2`
Node Group 0:	Node A	Node B
Node Group 1:	Node C	Node D

Suppose the heartbeats are transmitted in the order A->B->C->D->A. In this case, the loss of the heartbeat between the hosts causes node B to declare node A dead and node C to declare node B dead. This results in loss of Node Group 0, and so the cluster fails. On the other hand, if the order of transmission is A->B->D->C->A (and all other conditions remain as previously stated), the loss of the heartbeat causes nodes A and D to be declared dead; in this case, each node group has one surviving node, and the cluster survives.

The `HeartbeatOrder` configuration parameter makes the order of heartbeat transmission user-configurable. The default value for `HeartbeatOrder` is zero; allowing the default value to be used on all data nodes causes the order of heartbeat transmission to be determined by `NDB`. If this parameter is used, it must be set to a nonzero value (maximum 65535) for every data node in the cluster, and this value must be unique for each data node; this causes the heartbeat transmission to proceed from data node to data node in the order of their `HeartbeatOrder` values from lowest to highest (and then directly from the data node having the highest `HeartbeatOrder` to the data node having the lowest value, to complete the circle). The values need not be consecutive. For example, to force the heartbeat transmission order A->B->D->C->A in the scenario outlined previously, you could set the `HeartbeatOrder` values as shown here:

Table 18.9 HeartbeatOrder values to force a heartbeat transition order of A->B->D->C->A.

Node	`HeartbeatOrder` Value
A	10
B	20
C	30
D	25

To use this parameter to change the heartbeat transmission order in a running NDB Cluster, you must first set `HeartbeatOrder` for each data node in the cluster in the global configuration (`config.ini`) file (or files). To cause the change to take effect, you must perform either of the following:

- A complete shutdown and restart of the entire cluster.
- 2 rolling restarts of the cluster in succession. *All nodes must be restarted in the same order in both rolling restarts*.

You can use `DUMP 908` to observe the effect of this parameter in the data node logs.

- `ConnectCheckIntervalDelay`

Version (or later)	NDB 7.3.1
Type or units	milliseconds
Default	0

Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter enables connection checking between data nodes after one of them has failed heartbeat checks for 5 intervals of up to `HeartbeatIntervalDbDb` milliseconds.

Such a data node that further fails to respond within an interval of `ConnectCheckIntervalDelay` milliseconds is considered suspect, and is considered dead after two such intervals. This can be useful in setups with known latency issues.

The default value for this parameter is 0 (disabled).

- `TimeBetweenLocalCheckpoints`

Version (or later)	NDB 7.3.1
Type or units	number of 4-byte words, as base-2 logarithm
Default	20
Range	0 - 31
Restart Type	N

This parameter is an exception in that it does not specify a time to wait before starting a new local checkpoint; rather, it is used to ensure that local checkpoints are not performed in a cluster where relatively few updates are taking place. In most clusters with high update rates, it is likely that a new local checkpoint is started immediately after the previous one has been completed.

The size of all write operations executed since the start of the previous local checkpoints is added. This parameter is also exceptional in that it is specified as the base-2 logarithm of the number of 4-byte words, so that the default value 20 means 4MB (4×2^{20}) of write operations, 21 would mean 8MB, and so on up to a maximum value of 31, which equates to 8GB of write operations.

All the write operations in the cluster are added together. Setting `TimeBetweenLocalCheckpoints` to 6 or less means that local checkpoints are executed continuously without pause, independent of the cluster's workload.

- `TimeBetweenGlobalCheckpoints`

Version (or later)	NDB 7.3.1
Type or units	milliseconds
Default	2000
Range	20 - 32000
Restart Type	N

When a transaction is committed, it is committed in main memory in all nodes on which the data is mirrored. However, transaction log records are not flushed to disk as part of the commit. The reasoning behind this behavior is that having the transaction safely committed on at least two autonomous host machines should meet reasonable standards for durability.

It is also important to ensure that even the worst of cases—a complete crash of the cluster—is handled properly. To guarantee that this happens, all transactions taking place within a given interval are put into a global checkpoint, which can be thought of as a set of committed transactions that has been flushed to disk. In other words, as part of the commit process, a transaction is placed in a global checkpoint group. Later, this group's log records are flushed to disk, and then the entire group of transactions is safely committed to disk on all computers in the cluster.

This parameter defines the interval between global checkpoints. The default is 2000 milliseconds.

- `TimeBetweenGlobalCheckpointsTimeout`

Version (or later)	NDB 7.4.5
Type or units	milliseconds
Default	120000
Range	10 - 4294967039 (0xFFFFFEFF)
Restart Type	N
Version (or later)	NDB 7.3.9
Type or units	milliseconds
Default	120000
Range	10 - 4294967039 (0xFFFFFEFF)
Restart Type	N
Added	NDB 7.3.9
Added	NDB 7.4.5

This parameter defines the minimum timeout between global checkpoints. The default is 120000 milliseconds.

This parameter was added in NDB 7.3.9 and NDB 7.4.5. (Bug #20069617)

- `TimeBetweenEpochs`

Version (or later)	NDB 7.3.1
Type or units	milliseconds
Default	100
Range	0 - 32000
Restart Type	N

This parameter defines the interval between synchronization epochs for NDB Cluster Replication. The default value is 100 milliseconds.

`TimeBetweenEpochs` is part of the implementation of “micro-GCPs”, which can be used to improve the performance of NDB Cluster Replication.

- `TimeBetweenEpochsTimeout`

Version (or later)	NDB 7.3.1
Type or units	milliseconds
Default	0
Range	0 - 256000
Restart Type	N

This parameter defines a timeout for synchronization epochs for NDB Cluster Replication. If a node fails to participate in a global checkpoint within the time determined by this parameter, the node is shut down. In NDB Cluster 7.3 and later, the default value is 0; in other words, the timeout is disabled.

`TimeBetweenEpochsTimeout` is part of the implementation of “micro-GCPs”, which can be used to improve the performance of NDB Cluster Replication.

The current value of this parameter and a warning are written to the cluster log whenever a GCP save takes longer than 1 minute or a GCP commit takes longer than 10 seconds.

Setting this parameter to zero has the effect of disabling GCP stops caused by save timeouts, commit timeouts, or both. The maximum possible value for this parameter is 256000 milliseconds.

- `MaxBufferedEpochs`

Version (or later)	NDB 7.3.1
Type or units	epochs
Default	100
Range	0 - 100000
Restart Type	N

The number of unprocessed epochs by which a subscribing node can lag behind. Exceeding this number causes a lagging subscriber to be disconnected.

The default value of 100 is sufficient for most normal operations. If a subscribing node does lag enough to cause disconnections, it is usually due to network or scheduling issues with regard to processes or threads. (In rare circumstances, the problem may be due to a bug in the `NDB` client.) It may be desirable to set the value lower than the default when epochs are longer.

Disconnection prevents client issues from affecting the data node service, running out of memory to buffer data, and eventually shutting down. Instead, only the client is affected as a result of the disconnect (by, for example gap events in the binary log), forcing the client to reconnect or restart the process.

- `MaxBufferedEpochBytes`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	26214400
Range	26214400 (0x01900000) - 4294967039 (0xFFFFFEFF)
Restart Type	N

The total number of bytes allocated for buffering epochs by this node.

- `TimeBetweenInactiveTransactionAbortCheck`

Version (or later)	NDB 7.3.1
Type or units	milliseconds
Default	1000
Range	1000 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Timeout handling is performed by checking a timer on each transaction once for every interval specified by this parameter. Thus, if this parameter is set to 1000 milliseconds, every transaction is checked for timing out once per second.

The default value is 1000 milliseconds (1 second).

- `TransactionInactiveTimeout`

Version (or later)	NDB 7.3.1
Type or units	milliseconds

Default	4294967039 (0xFFFFFEFF)
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter states the maximum time that is permitted to lapse between operations in the same transaction before the transaction is aborted.

The default for this parameter is `4G` (also the maximum). For a real-time database that needs to ensure that no transaction keeps locks for too long, this parameter should be set to a relatively small value. Setting it to 0 means that the application never times out. The unit is milliseconds.

- `TransactionDeadlockDetectionTimeout`

Version (or later)	NDB 7.3.1
Type or units	milliseconds
Default	1200
Range	50 - 4294967039 (0xFFFFFEFF)
Restart Type	N

When a node executes a query involving a transaction, the node waits for the other nodes in the cluster to respond before continuing. This parameter sets the amount of time that the transaction can spend executing within a data node, that is, the time that the transaction coordinator waits for each data node participating in the transaction to execute a request.

A failure to respond can occur for any of the following reasons:

- The node is “dead”
- The operation has entered a lock queue
- The node requested to perform the action could be heavily overloaded.

This timeout parameter states how long the transaction coordinator waits for query execution by another node before aborting the transaction, and is important for both node failure handling and deadlock detection.

The default timeout value is 1200 milliseconds (1.2 seconds).

The minimum for this parameter is 50 milliseconds.

- `DiskSyncSize`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	4M
Range	32K - 4294967039 (0xFFFFFEFF)

Restart Type	N

This is the maximum number of bytes to store before flushing data to a local checkpoint file. This is done to prevent write buffering, which can impede performance significantly. This parameter is *not* intended to take the place of `TimeBetweenLocalCheckpoints`.

Note

When `ODirect` is enabled, it is not necessary to set `DiskSyncSize`; in fact, in such cases its value is simply ignored.

The default value is 4M (4 megabytes).

- `DiskCheckpointSpeed`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	10M
Range	1M - 4294967039 (0xFFFFFEFF)
Restart Type	N
Deprecated	NDB 7.4.1

The amount of data, in bytes per second, that is sent to disk during a local checkpoint. This allocation is shared by DML operations and backups (but not backup logging), which means that backups started during times of intensive DML may be impaired by flooding of the redo log buffer and may fail altogether if the contention is sufficiently severe.

The default value is 10M (10 megabytes per second).

This parameter is deprecated in NDB 7.4.1 and later, where setting it has no effect, and removed in NDB 7.5. Instead, use the configuration parameters `MinDiskWriteSpeed`, `MaxDiskWriteSpeed`, `MaxDiskWriteSpeedOtherNodeRestart`, and `MaxDiskWriteSpeedOwnRestart` to control write speeds for LCPs and backups.

- `DiskCheckpointSpeedInRestart`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	100M
Range	1M - 4294967039 (0xFFFFFEFF)
Restart Type	N
Deprecated	NDB 7.4.1

The amount of data, in bytes per second, that is sent to disk during a local checkpoint as part of a restart operation.

The default value is 100M (100 megabytes per second).

This parameter is deprecated in NDB 7.4.1 and later, where setting it has no effect, and removed in NDB 7.5. Instead, use the configuration parameters `MinDiskWriteSpeed`, `MaxDiskWriteSpeed`, `MaxDiskWriteSpeedOtherNodeRestart`, and `MaxDiskWriteSpeedOwnRestart` to control write speeds for LCPs and backups.

- `NoOfDiskPagesToDiskAfterRestartTUP`

  This parameter is deprecated and subject to removal in a future version of NDB Cluster. Use `DiskCheckpointSpeedInRestart` and `DiskSyncSize` instead. Beginning with NDB 7.4.1, you should instead use the configuration parameters `MinDiskWriteSpeed`, `MaxDiskWriteSpeed`, `MaxDiskWriteSpeedOtherNodeRestart`, and `MaxDiskWriteSpeedOwnRestart` introduced in that release.

- `MaxDiskWriteSpeed`

Version (or later)	NDB 7.4.1
Type or units	numeric
Default	20M
Range	1M - 1024G
Restart Type	S
Added	NDB 7.4.1

  Set the maximum rate for writing to disk, in bytes per second, by local checkpoints and backup operations when no restarts (by this data node or any other data node) are taking place in this NDB Cluster.

  For setting the maximum rate of disk writes allowed while this data node is restarting, use `MaxDiskWriteSpeedOwnRestart`. For setting the maximum rate of disk writes allowed while other data nodes are restarting, use `MaxDiskWriteSpeedOtherNodeRestart`. The minimum speed for disk writes by all LCPs and backup operations can be adjusted by setting `MinDiskWriteSpeed`.

  `MaxDiskWriteSpeed` was added in NDB 7.4.1.

- `MaxDiskWriteSpeedOtherNodeRestart`

Version (or later)	NDB 7.4.1
Type or units	numeric
Default	50M
Range	1M - 1024G
Restart Type	S
Added	NDB 7.4.1

  Set the maximum rate for writing to disk, in bytes per second, by local checkpoints and backup operations when one or more data nodes in this NDB Cluster are restarting, other than this node.

  For setting the maximum rate of disk writes allowed while this data node is restarting, use `MaxDiskWriteSpeedOwnRestart`. For setting the maximum rate of disk writes allowed when no data nodes are restarting anywhere in the cluster, use `MaxDiskWriteSpeed`. The minimum speed for disk writes by all LCPs and backup operations can be adjusted by setting `MinDiskWriteSpeed`.

  `MaxDiskWriteSpeedOtherNodeRestart` was added in NDB 7.4.1.

- `MaxDiskWriteSpeedOwnRestart`

Version (or later)	NDB 7.4.1
Type or units	numeric
Default	200M
Range	1M - 1024G
Restart Type	S

Added	NDB 7.4.1

Set the maximum rate for writing to disk, in bytes per second, by local checkpoints and backup operations while this data node is restarting.

For setting the maximum rate of disk writes allowed while other data nodes are restarting, use `MaxDiskWriteSpeedOtherNodeRestart`. For setting the maximum rate of disk writes allowed when no data nodes are restarting anywhere in the cluster, use `MaxDiskWriteSpeed`. The minimum speed for disk writes by all LCPs and backup operations can be adjusted by setting `MinDiskWriteSpeed`.

`MaxDiskWriteSpeedOwnRestart` was added in NDB 7.4.1.

- `MinDiskWriteSpeed`

Version (or later)	NDB 7.4.1
Type or units	numeric
Default	10M
Range	1M - 1024G
Restart Type	S
Added	NDB 7.4.1

Set the minimum rate for writing to disk, in bytes per second, by local checkpoints and backup operations.

The maximum rates of disk writes allowed for LCPs and backups under various conditions are adjustable using the parameters `MaxDiskWriteSpeed`, `MaxDiskWriteSpeedOwnRestart`, and `MaxDiskWriteSpeedOtherNodeRestart`. See the descriptions of these parameters for more information.

`MinDiskWriteSpeed` was added in NDB 7.4.1.

- `NoOfDiskPagesToDiskAfterRestartACC`

  This parameter is deprecated and subject to removal in a future version of NDB Cluster. In NDB Cluster 7.3, use `DiskCheckpointSpeedInRestart` and `DiskSyncSize` instead. In NDB 7.4.1 and later, you should use the parameters `MinDiskWriteSpeed`, `MaxDiskWriteSpeed`, `MaxDiskWriteSpeedOtherNodeRestart`, and `MaxDiskWriteSpeedOwnRestart`.

- `NoOfDiskPagesToDiskDuringRestartTUP` (DEPRECATED)

  This parameter is deprecated and subject to removal in a future version of NDB Cluster. In NDB Cluster 7.3, use `DiskCheckpointSpeedInRestart` and `DiskSyncSize` instead. In NDB 7.4.1 and later, you should use the parameters `MinDiskWriteSpeed`, `MaxDiskWriteSpeed`, `MaxDiskWriteSpeedOtherNodeRestart`, and `MaxDiskWriteSpeedOwnRestart`.

- `NoOfDiskPagesToDiskDuringRestartACC` (DEPRECATED)

  This parameter is deprecated and subject to removal in a future version of NDB Cluster. In NDB Cluster 7.3, use `DiskCheckpointSpeedInRestart` and `DiskSyncSize` instead. In NDB 7.4.1 and later, you should use the parameters `MinDiskWriteSpeed`, `MaxDiskWriteSpeed`, `MaxDiskWriteSpeedOtherNodeRestart`, and `MaxDiskWriteSpeedOwnRestart`.

- `ArbitrationTimeout`

Version (or later)	NDB 7.3.1
Type or units	milliseconds

Default	7500
Range	10 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter specifies how long data nodes wait for a response from the arbitrator to an arbitration message. If this is exceeded, the network is assumed to have split.

In NDB Cluster 7.3 and later, the default value is 7500 milliseconds (7.5 seconds).

- `Arbitration`

Version (or later)	NDB 7.3.1
Type or units	enumeration
Default	Default
Range	Default, Disabled, WaitExternal
Restart Type	N

The `Arbitration` parameter enables a choice of arbitration schemes, corresponding to one of 3 possible values for this parameter:

- **Default.** This enables arbitration to proceed normally, as determined by the `ArbitrationRank` settings for the management and API nodes. This is the default value.
- **Disabled.** Setting `Arbitration = Disabled` in the `[ndbd default]` section of the `config.ini` file to accomplishes the same task as setting `ArbitrationRank` to 0 on all management and API nodes. When `Arbitration` is set in this way, any `ArbitrationRank` settings are ignored.
- **WaitExternal.** The `Arbitration` parameter also makes it possible to configure arbitration in such a way that the cluster waits until after the time determined by `ArbitrationTimeout` has passed for an external cluster manager application to perform arbitration instead of handling arbitration internally. This can be done by setting `Arbitration = WaitExternal` in the `[ndbd default]` section of the `config.ini` file. For best results with the `WaitExternal` setting, it is recommended that `ArbitrationTimeout` be 2 times as long as the interval required by the external cluster manager to perform arbitration.

Important

This parameter should be used only in the `[ndbd default]` section of the cluster configuration file. The behavior of the cluster is unspecified when `Arbitration` is set to different values for individual data nodes.

- `RestartSubscriberConnectTimeout`

Version (or later)	NDB 7.3.6
Type or units	ms
Default	12000
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	S

Added	NDB 7.3.6

This parameter determines the time that a data node waits for subscribing API nodes to connect. Once this timeout expires, any “missing” API nodes are disconnected from the cluster. To disable this timeout, set `RestartSubscriberConnectTimeout` to 0.

While this parameter is specified in milliseconds, the timeout itself is resolved to the next-greatest whole second.

`RestartSubscriberConnectTimeout` was added in NDB 7.3.6.

Buffering and logging. Several `[ndbd]` configuration parameters enable the advanced user to have more control over the resources used by node processes and to adjust various buffer sizes at need.

These buffers are used as front ends to the file system when writing log records to disk. If the node is running in diskless mode, these parameters can be set to their minimum values without penalty due to the fact that disk writes are “faked” by the `NDB` storage engine's file system abstraction layer.

- `UndoIndexBuffer`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	2M
Range	1M - 4294967039 (0xFFFFFEFF)
Restart Type	N

The UNDO index buffer, whose size is set by this parameter, is used during local checkpoints. The `NDB` storage engine uses a recovery scheme based on checkpoint consistency in conjunction with an operational REDO log. To produce a consistent checkpoint without blocking the entire system for writes, UNDO logging is done while performing the local checkpoint. UNDO logging is activated on a single table fragment at a time. This optimization is possible because tables are stored entirely in main memory.

The UNDO index buffer is used for the updates on the primary key hash index. Inserts and deletes rearrange the hash index; the NDB storage engine writes UNDO log records that map all physical changes to an index page so that they can be undone at system restart. It also logs all active insert operations for each fragment at the start of a local checkpoint.

Reads and updates set lock bits and update a header in the hash index entry. These changes are handled by the page-writing algorithm to ensure that these operations need no UNDO logging.

This buffer is 2MB by default. The minimum value is 1MB, which is sufficient for most applications. For applications doing extremely large or numerous inserts and deletes together with large transactions and large primary keys, it may be necessary to increase the size of this buffer. If this buffer is too small, the NDB storage engine issues internal error code 677 (`Index UNDO buffers overloaded`).

Important

It is not safe to decrease the value of this parameter during a rolling restart.

- `UndoDataBuffer`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	16M

Range	1M - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter sets the size of the UNDO data buffer, which performs a function similar to that of the UNDO index buffer, except the UNDO data buffer is used with regard to data memory rather than index memory. This buffer is used during the local checkpoint phase of a fragment for inserts, deletes, and updates.

Because UNDO log entries tend to grow larger as more operations are logged, this buffer is also larger than its index memory counterpart, with a default value of 16MB.

This amount of memory may be unnecessarily large for some applications. In such cases, it is possible to decrease this size to a minimum of 1MB.

It is rarely necessary to increase the size of this buffer. If there is such a need, it is a good idea to check whether the disks can actually handle the load caused by database update activity. A lack of sufficient disk space cannot be overcome by increasing the size of this buffer.

If this buffer is too small and gets congested, the NDB storage engine issues internal error code 891 (`Data UNDO buffers overloaded`).

Important

It is not safe to decrease the value of this parameter during a rolling restart.

- `RedoBuffer`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	32M
Range	1M - 4294967039 (0xFFFFFEFF)
Restart Type	N

All update activities also need to be logged. The REDO log makes it possible to replay these updates whenever the system is restarted. The NDB recovery algorithm uses a “fuzzy” checkpoint of the data together with the UNDO log, and then applies the REDO log to play back all changes up to the restoration point.

`RedoBuffer` sets the size of the buffer in which the REDO log is written. The default value is 32MB; the minimum value is 1MB.

If this buffer is too small, the `NDB` storage engine issues error code 1221 (`REDO log buffers overloaded`). For this reason, you should exercise care if you attempt to decrease the value of `RedoBuffer` as part of an online change in the cluster's configuration.

`ndbmtd` allocates a separate buffer for each LDM thread (see `ThreadConfig`). For example, with 4 LDM threads, an `ndbmtd` data node actually has 4 buffers and allocates `RedoBuffer` bytes to each one, for a total of `4 * RedoBuffer` bytes.

- `EventLogBufferSize`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	8192
Range	0 - 64K

Restart Type	S

Controls the size of the circular buffer used for NDB log events within data nodes.

Controlling log messages. In managing the cluster, it is very important to be able to control the number of log messages sent for various event types to `stdout`. For each event category, there are 16 possible event levels (numbered 0 through 15). Setting event reporting for a given event category to level 15 means all event reports in that category are sent to `stdout`; setting it to 0 means that no event reports are made in that category.

By default, only the startup message is sent to `stdout`, with the remaining event reporting level defaults being set to 0. The reason for this is that these messages are also sent to the management server's cluster log.

An analogous set of levels can be set for the management client to determine which event levels to record in the cluster log.

- `LogLevelStartup`

Version (or later)	NDB 7.3.1
Type or units	integer
Default	1
Range	0 - 15
Restart Type	N

The reporting level for events generated during startup of the process.

The default level is 1.

- `LogLevelShutdown`

Version (or later)	NDB 7.3.1
Type or units	integer
Default	0
Range	0 - 15
Restart Type	N

The reporting level for events generated as part of graceful shutdown of a node.

The default level is 0.

- `LogLevelStatistic`

Version (or later)	NDB 7.3.1
Type or units	integer
Default	0
Range	0 - 15
Restart Type	N

The reporting level for statistical events such as number of primary key reads, number of updates, number of inserts, information relating to buffer usage, and so on.

The default level is 0.

- `LogLevelCheckpoint`

Version (or later)	NDB 7.3.1
Type or units	log level
Default	0
Range	0 - 15
Restart Type	N

The reporting level for events generated by local and global checkpoints.

The default level is 0.

- `LogLevelNodeRestart`

Version (or later)	NDB 7.3.1
Type or units	integer
Default	0
Range	0 - 15
Restart Type	N

The reporting level for events generated during node restart.

The default level is 0.

- `LogLevelConnection`

Version (or later)	NDB 7.3.1
Type or units	integer
Default	0
Range	0 - 15
Restart Type	N

The reporting level for events generated by connections between cluster nodes.

The default level is 0.

- `LogLevelError`

Version (or later)	NDB 7.3.1
Type or units	integer
Default	0
Range	0 - 15
Restart Type	N

The reporting level for events generated by errors and warnings by the cluster as a whole. These errors do not cause any node failure but are still considered worth reporting.

The default level is 0.

- `LogLevelCongestion`

Version (or later)	NDB 7.3.1
Type or units	level
Default	0

Range	0 - 15
Restart Type	N

The reporting level for events generated by congestion. These errors do not cause node failure but are still considered worth reporting.

The default level is 0.

- `LogLevelInfo`

Version (or later)	NDB 7.3.1
Type or units	integer
Default	0
Range	0 - 15
Restart Type	N

The reporting level for events generated for information about the general state of the cluster.

The default level is 0.

- `MemReportFrequency`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter controls how often data node memory usage reports are recorded in the cluster log; it is an integer value representing the number of seconds between reports.

Each data node's data memory and index memory usage is logged as both a percentage and a number of 32 KB pages of the `DataMemory` and `IndexMemory`, respectively, set in the `config.ini` file. For example, if `DataMemory` is equal to 100 MB, and a given data node is using 50 MB for data memory storage, the corresponding line in the cluster log might look like this:

```
2006-12-24 01:18:16 [MgmSrvr] INFO -- Node 2: Data usage is 50%(1280 32K pages of total 2560)
```

`MemReportFrequency` is not a required parameter. If used, it can be set for all cluster data nodes in the `[ndbd default]` section of `config.ini`, and can also be set or overridden for individual data nodes in the corresponding `[ndbd]` sections of the configuration file. The minimum value—which is also the default value—is 0, in which case memory reports are logged only when memory usage reaches certain percentages (80%, 90%, and 100%), as mentioned in the discussion of statistics events in Section 18.5.3.2, “NDB Cluster Log Events”.

- `StartupStatusReportFrequency`

Version (or later)	NDB 7.3.1
Type or units	seconds
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)

Restart Type	N

When a data node is started with the `--initial`, it initializes the redo log file during Start Phase 4 (see Section 18.5.4, “Summary of NDB Cluster Start Phases”). When very large values are set for `NoOfFragmentLogFiles`, `FragmentLogFileSize`, or both, this initialization can take a long time.You can force reports on the progress of this process to be logged periodically, by means of the `StartupStatusReportFrequency` configuration parameter. In this case, progress is reported in the cluster log, in terms of both the number of files and the amount of space that have been initialized, as shown here:

```
2009-06-20 16:39:23 [MgmSrvr] INFO -- Node 1: Local redo log file initialization status:
#Total files: 80, Completed: 60
#Total MBytes: 20480, Completed: 15557
2009-06-20 16:39:23 [MgmSrvr] INFO -- Node 2: Local redo log file initialization status:
#Total files: 80, Completed: 60
#Total MBytes: 20480, Completed: 15570
```

These reports are logged each `StartupStatusReportFrequency` seconds during Start Phase 4. If `StartupStatusReportFrequency` is 0 (the default), then reports are written to the cluster log only when at the beginning and at the completion of the redo log file initialization process.

Data Node Debugging Parameters. In NDB Cluster 7.3 and later, it is possible to cause logging of traces for events generated by creating and dropping tables using `DictTrace`. This parameter is useful only in debugging NDB kernel code. `DictTrace` takes an integer value. 0 (default - no logging) and 1 (logging enabled) are the only supported values prior to NDB 7.4.12. In NDB 7.4.12 and later, setting this parameter to 2 enables logging of additional `DBDICT` debugging output (Bug #20368450).

Backup parameters. The `[ndbd]` parameters discussed in this section define memory buffers set aside for execution of online backups.

- `BackupDataBufferSize`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	16M
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N
Version (or later)	NDB 7.4.8
Type or units	bytes
Default	16M
Range	2M - 4294967039 (0xFFFFFEFF)
Restart Type	N
Version (or later)	NDB 7.4.11
Type or units	bytes
Default	16M
Range	512K - 4294967039 (0xFFFFFEFF)
Restart Type	N
Deprecated	Yes (in NDB 7.6)

In creating a backup, there are two buffers used for sending data to the disk. The backup data buffer is used to fill in data recorded by scanning a node's tables. Once this buffer has been filled to the level specified as `BackupWriteSize`, the pages are sent to disk. While flushing data to disk, the backup process can continue filling this buffer until it runs out of space. When this happens,

the backup process pauses the scan and waits until some disk writes have completed freeing up memory so that scanning may continue.

The default value for this parameter is 16MB. The minimum was raised to 2M in NDB 7.4.8, then lowered to 512K in NDB 7.4.11. (Bug #22749509)

- `BackupDiskWriteSpeedPct`

Version (or later)	NDB 7.4.8
Type or units	percent
Default	50
Range	0 - 90
Restart Type	N
Added	NDB 7.4.8

During normal operation, data nodes attempt to maximize the disk write speed used for local checkpoints and backups while remaining within the bounds set by `MinDiskWriteSpeed` and `MaxDiskWriteSpeed`. In NDB Cluster 7.4, the implementation of disk write throttling has been changed to give each LDM thread an equal share of the total budget. This allows parallel LCPs to take place without exceeding the disk I/O budget. Because a backup is executed by only one LDM thread, this effectively caused a budget cut, resulting in longer backup completion times, and—if the rate of change is sufficiently high—in failure to complete the backup when the backup log buffer fill rate is higher than the achievable write rate.

This problem is addressed in NDB 7.4.8 and later by the addition of the `BackupDiskWriteSpeedPct` configuration parameter (Bug #20204854). This parameter takes a value in the range 0-90 (inclusive) which is interpreted as the percentage of the node's maximum write rate budget that is reserved prior to sharing out the remainder of the budget among LDM threads for LCPs. The LDM thread running the backup receives the whole write rate budget for the backup, plus its (reduced) share of the write rate budget for local checkpoints. This makes the disk write rate budget in NDB 7.4.8 and later behave similarly to how it is handled in NDB Cluster 7.3 and previous NDB Cluster release series.

The default value for this parameter is 50 (interpreted as 50%).

- `BackupLogBufferSize`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	16M
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N
Version (or later)	NDB 7.4.8
Type or units	bytes
Default	16M
Range	2M - 4294967039 (0xFFFFFEFF)
Restart Type	N

The backup log buffer fulfills a role similar to that played by the backup data buffer, except that it is used for generating a log of all table writes made during execution of the backup. The same principles apply for writing these pages as with the backup data buffer, except that when there is no more space in the backup log buffer, the backup fails. For that reason, the size of the backup log

buffer must be large enough to handle the load caused by write activities while the backup is being made. See Section 18.5.8.3, “Configuration for NDB Cluster Backups”.

The default value for this parameter should be sufficient for most applications. In fact, it is more likely for a backup failure to be caused by insufficient disk write speed than it is for the backup log buffer to become full. If the disk subsystem is not configured for the write load caused by applications, the cluster is unlikely to be able to perform the desired operations.

It is preferable to configure cluster nodes in such a manner that the processor becomes the bottleneck rather than the disks or the network connections.

The default value for this parameter is 16MB.

- `BackupMemory`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	32M
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N
Deprecated	NDB 7.4.8

This parameter is deprecated, and is subject to removal in a future version of NDB Cluster. In NDB Cluster 7.5 and later, it is ignored.

- `BackupReportFrequency`

Version (or later)	NDB 7.3.1
Type or units	seconds
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter controls how often backup status reports are issued in the management client during a backup, as well as how often such reports are written to the cluster log (provided cluster event logging is configured to permit it—see Logging and checkpointing). `BackupReportFrequency` represents the time in seconds between backup status reports.

The default value is 0.

- `BackupWriteSize`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	256K
Range	2K - 4294967039 (0xFFFFFEFF)
Restart Type	N
Version (or later)	NDB 7.4.8
Type or units	bytes
Default	256K
Range	32K - 4294967039 (0xFFFFFEFF)
Restart Type	N

Deprecated	Yes (in NDB 7.6)

This parameter specifies the default size of messages written to disk by the backup log and backup data buffers.

The default value for this parameter is 256KB.

- `BackupMaxWriteSize`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	1M
Range	2K - 4294967039 (0xFFFFFEFF)
Restart Type	N
Version (or later)	NDB 7.4.8
Type or units	bytes
Default	1M
Range	256K - 4294967039 (0xFFFFFEFF)
Restart Type	N
Deprecated	Yes (in NDB 7.6)

This parameter specifies the maximum size of messages written to disk by the backup log and backup data buffers.

The default value for this parameter is 1MB.

- `CompressedBackup`

Version (or later)	NDB 7.3.1
Type or units	boolean
Default	false
Range	true, false
Restart Type	N

Enabling this parameter causes backup files to be compressed. The compression used is equivalent to `gzip --fast`, and can save 50% or more of the space required on the data node to store uncompressed backup files. Compressed backups can be enabled for individual data nodes, or for all data nodes (by setting this parameter in the `[ndbd default]` section of the `config.ini` file).

Important

You cannot restore a compressed backup to a cluster running a MySQL version that does not support this feature.

The default value is `0` (disabled).

Note

The location of the backup files is determined by the `BackupDataDir` data node configuration parameter.

Additional requirements. When specifying these parameters, the following relationships must hold true. Otherwise, the data node cannot start.

- `BackupDataBufferSize >= BackupWriteSize + 188KB`

- `BackupLogBufferSize >= BackupWriteSize + 16KB`
- `BackupMaxWriteSize >= BackupWriteSize`

NDB Cluster Realtime Performance Parameters

The `[ndbd]` parameters discussed in this section are used in scheduling and locking of threads to specific CPUs on multiprocessor data node hosts.

Note

To make use of these parameters, the data node process must be run as system root.

- `BuildIndexThreads`

Version (or later)	NDB 7.3.1
Type or units	numeric
Default	0
Range	0 - 128
Restart Type	S

This parameter determines the number of threads to create when rebuilding ordered indexes during a system or node start, as well as when running `ndb_restore --rebuild-indexes`. It is supported only when there is more than one fragment for the table per data node (for example, when the `MAX_ROWS` option has been used with `CREATE TABLE`).

Setting this parameter to 0 (the default) disables multithreaded building of ordered indexes.

This parameter is supported when using `ndbd` or `ndbmtd`.

You can enable multithreaded builds during data node initial restarts by setting the `TwoPassInitialNodeRestartCopy` data node configuration parameter to `TRUE`.

- `LockExecuteThreadToCPU`

Version (or later)	NDB 7.3.1
Type or units	set of CPU IDs
Default	0
Range	...
Restart Type	N

When used with `ndbd`, this parameter (now a string) specifies the ID of the CPU assigned to handle the `NDBCLUSTER` execution thread. When used with `ndbmtd`, the value of this parameter is a comma-separated list of CPU IDs assigned to handle execution threads. Each CPU ID in the list should be an integer in the range 0 to 65535 (inclusive).

The number of IDs specified should match the number of execution threads determined by `MaxNoOfExecutionThreads`. However, there is no guarantee that threads are assigned to CPUs in any given order when using this parameter. You can obtain more finely-grained control of this type using `ThreadConfig`.

`LockExecuteThreadToCPU` has no default value.

- `LockMaintThreadsToCPU`

Version (or later)	NDB 7.3.1

Type or units	CPU ID
Default	0
Range	0 - 64K
Restart Type	N

This parameter specifies the ID of the CPU assigned to handle `NDBCLUSTER` maintenance threads.

The value of this parameter is an integer in the range 0 to 65535 (inclusive). In NDB Cluster 7.3 and later, there is no default value.

- `Numa`

Version (or later)	NDB 7.3.1
Type or units	numeric
Default	1
Range	...
Restart Type	N

This parameter determines whether Non-Uniform Memory Access (NUMA) is controlled by the operating system or by the data node process, whether the data node uses `ndbd` or `ndbmtd`. By default, `NDB` attempts to use an interleaved NUMA memory allocation policy on any data node where the host operating system provides NUMA support.

Setting `Numa = 0` means that the datanode process does not itself attempt to set a policy for memory allocation, and permits this behavior to be determined by the operating system, which may be further guided by the separate `numactl` tool. That is, `Numa = 0` yields the system default behavior, which can be customised by `numactl`. For many Linux systems, the system default behavior is to allocate socket-local memory to any given process at allocation time. This can be problematic when using `ndbmtd`; this is because `nbdmtd` allocates all memory at startup, leading to an imbalance, giving different access speeds for different sockets, especially when locking pages in main memory.

Setting `Numa = 1` means that the data node process uses `libnuma` to request interleaved memory allocation. (This can also be accomplished manually, on the operating system level, using `numactl`.) Using interleaved allocation in effect tells the data node process to ignore non-uniform memory access but does not attempt to take any advantage of fast local memory; instead, the data node process tries to avoid imbalances due to slow remote memory. If interleaved allocation is not desired, set `Numa` to 0 so that the desired behavior can be determined on the operating system level.

The `Numa` configuration parameter is supported only on Linux systems where `libnuma.so` is available.

- `RealtimeScheduler`

Version (or later)	NDB 7.3.1
Type or units	boolean
Default	false
Range	true, false

Restart Type	N

Setting this parameter to 1 enables real-time scheduling of data node threads.

Prior to NDB 7.3.3, this parameter did not work correctly with data nodes running `ndbmtd`. (Bug #16961971)

The default is 0 (scheduling disabled).

- `SchedulerExecutionTimer`

Version (or later)	NDB 7.3.1
Type or units	µs
Default	50
Range	0 - 11000
Restart Type	N

This parameter specifies the time in microseconds for threads to be executed in the scheduler before being sent. Setting it to 0 minimizes the response time; to achieve higher throughput, you can increase the value at the expense of longer response times.

The default is 50 µsec, which our testing shows to increase throughput slightly in high-load cases without materially delaying requests.

- `SchedulerResponsiveness`

Version (or later)	NDB 7.4.9
Type or units	integer
Default	5
Range	0 - 10
Restart Type	S
Added	NDB 7.4.9

Set the balance in the NDB scheduler between speed and throughput. This parameter takes an integer whose value is in the range 0-10 inclusive, with 5 as the default. (This is the same as the previous hard-coded value for this parameter.) Higher values provide better response times relative to throughput. Lower values provide increased throughput at the expense of longer response times.

The `SchedulerResponsiveness` parameter was added in NDB 7.4.9, but did not become effective until NDB 7.4.11 (Bug #80341, Bug #22712481).

- `SchedulerSpinTimer`

Version (or later)	NDB 7.3.1
Type or units	µs
Default	0
Range	0 - 500
Restart Type	N

This parameter specifies the time in microseconds for threads to be executed in the scheduler before sleeping.

The default value is 0.

- `TwoPassInitialNodeRestartCopy`

Version (or later)	NDB 7.3.1
Type or units	boolean
Default	false
Range	true, false
Restart Type	N

Multithreaded building of ordered indexes can be enabled for initial restarts of data nodes by setting this configuration parameter to `TRUE`, which enables two-pass copying of data during initial node restarts.

You must also set `BuildIndexThreads` to a nonzero value.

Multi-Threading Configuration Parameters (ndbmtd). `ndbmtd` runs by default as a single-threaded process and must be configured to use multiple threads, using either of two methods, both of which require setting configuration parameters in the `config.ini` file. The first method is simply to set an appropriate value for the `MaxNoOfExecutionThreads` configuration parameter. NDB Cluster 7.3 and later also support a second method, whereby it is possible to set up more complex rules for `ndbmtd` multithreading using `ThreadConfig`. The next few paragraphs provide information about these parameters and their use with multithreaded data nodes.

- `MaxNoOfExecutionThreads`

Version (or later)	NDB 7.3.1
Type or units	integer
Default	2
Range	2 - 8
Restart Type	IS
Version (or later)	NDB 7.3.2
Type or units	integer
Default	2
Range	2 - 36
Restart Type	IS
Version (or later)	NDB 7.3.3
Type or units	integer
Default	2
Range	2 - 72
Restart Type	IS

This parameter directly controls the number of execution threads used by `ndbmtd`, up to a maximum of 72 (previous to NDB 7.3.3, this was 36). Although this parameter is set in `[ndbd]` or `[ndbd default]` sections of the `config.ini` file, it is exclusive to `ndbmtd` and does not apply to `ndbd`.

Setting `MaxNoOfExecutionThreads` sets the number of threads for each type as determined by a matrix in the file `storage/ndb/src/kernel/vm/mt_thr_config.cpp`. This table shows these numbers of threads for possible values of `MaxNoOfExecutionThreads` in NDB 7.4.3 and later (Bug #75220, Bug #20215689). (A table with information about the matrix applicable in previous versions of NDB Cluster follows this one.) Rows containing values which changed in NDB 7.4.3 are shown in emphasized text.

Table 18.10 MaxNoOfExecutionThreads values and the corresponding number of threads by thread type (LQH, TC, Send, Receive), NDB 7.4.3 and later

`MaxNoOfExecutionThreads` Value	LDM Threads	TC Threads	Send Threads	Receive Threads
0 .. 3	1	0	0	1
4 .. 6	2	0	0	1
7 .. 8	4	0	0	1
9	4	2	0	1
10	4	2	1	1
11	4	3	1	1
12	6	2	1	1
13	6	3	1	1
14	6	3	1	2
15	6	3	2	2
16	8	3	1	2
17	8	4	1	2
18	8	4	2	2
19	8	5	2	2
20	10	4	2	2
21	10	5	2	2
22	10	5	2	3
23	10	6	2	3
24	12	5	2	3
25	12	6	2	3
26	12	6	3	3
27	12	7	3	3
28	12	7	3	4
29	12	8	3	4
30	12	8	4	4
31	12	9	4	4
32	16	8	3	3
33	16	8	3	4
34	16	8	4	4
35	16	9	4	4
36	16	10	4	4
37	16	10	4	5
38	16	11	4	5
39	16	11	5	5
40	20	10	4	4
41	20	10	4	5
42	20	11	4	5
43	20	11	5	5

MaxNoOfExecutionThreads Value	LDM Threads	TC Threads	Send Threads	Receive Threads
44	20	12	5	5
45	20	12	5	6
46	20	13	5	6
47	20	13	6	6
48	24	12	5	5
49	24	12	5	6
50	24	13	5	6
51	24	13	6	6
52	24	14	6	6
53	24	14	6	7
54	24	15	6	7
55	24	15	7	7
56	24	16	7	7
57	24	16	7	8
58	24	17	7	8
59	24	17	8	8
60	24	18	8	8
61	24	18	8	9
62	24	19	8	9
63	24	19	9	9
64	32	16	7	7
65	32	16	7	8
66	32	17	7	8
67	32	17	8	8
68	32	18	8	8
69	32	18	8	9
70	32	19	8	9
71	32	20	8	9
72	32	20	8	10

The following table shows how the number of threads for each type is obtained for values of `MaxNoOfExecutionThreads` in NDB 7.4.2 and earlier. This table can be also used with NDB 7.3.2 and earlier, except that in these versions the maximum value for `MaxNoOfExecutionThreads` is 36, and thus rows from this table which correspond to values greater than 36 do not apply in versions prior to NDB 7.3.3.

Table 18.11 MaxNoOfExecutionThreads values and the corresponding number of threads by thread type (LQH, TC, Send, Receive), NDB 7.4.2 and earlier

MaxNoOfExecutionThreads Value	LDM Threads	TC Threads	Send Threads	Receive Threads
0 .. 3	1	1	0	1
4 .. 6	2	1	0	1
7 .. 8	4	1	0	1

MaxNoOfExecutionThreads Value	LDM Threads	TC Threads	Send Threads	Receive Threads
9	4	2	0	1
10	4	2	1	1
11	4	3	1	1
12	4	3	1	2
13	4	3	2	2
14	4	4	2	2
15	4	5	2	2
16	8	3	1	2
17	8	4	1	2
18	8	4	2	2
19	8	5	2	2
20	8	5	2	3
21	8	5	3	3
22	8	6	3	3
23	8	7	3	3
24	12	5	2	3
25	12	6	2	3
26	12	6	3	3
27	12	7	3	3
28	12	7	3	4
29	12	8	3	4
30	12	8	4	4
31	12	9	4	4
32	16	8	3	3
33	16	8	3	4
34	16	8	4	4
35	16	9	4	4
36	16	10	4	4
37	16	10	4	5
38	16	11	4	5
39	16	11	5	5
40	16	12	5	5
41	16	12	5	6
42	16	13	5	6
43	16	13	6	6
44	16	14	6	6
45	16	14	6	7
46	16	15	6	7
47	16	15	7	7
48	24	12	5	5

MaxNoOfExecutionThreads Value	LDM Threads	TC Threads	Send Threads	Receive Threads
49	24	12	5	6
50	24	13	5	6
51	24	13	6	6
52	24	14	6	6
53	24	14	6	7
54	24	15	6	7
55	24	15	7	7
56	24	16	7	7
57	24	16	7	8
58	24	17	7	8
59	24	17	8	8
60	24	18	8	8
61	24	18	8	9
62	24	19	8	9
63	24	19	9	9
64	32	16	7	7
65	32	16	7	8
66	32	17	7	8
67	32	17	8	8
68	32	18	8	8
69	32	18	8	9
70	32	19	8	9
71	32	20	8	9
72	32	20	8	10

In NDB Cluster 7.3 and later, there is always one SUMA (replication) thread.

`NoOfFragmentLogParts` should be set equal to the number of LDM threads used by `ndbmtd` as determined by the setting for `MaxNoOfExecutionThreads`. This ratio should not be any greater than 4:1; beginning with NDB 7.4.16, a configuration in which this is the case is specifically disallowed. (Bug #25333414)

The number of LDM threads also determines the number of partitions used by an `NDB` table that is not explicitly partitioned; this is the number of LDM threads times the number of data nodes in the cluster. (If `ndbd` is used on the data nodes rather than `ndbmtd`, then there is always a single LDM thread; in this case, the number of partitions created automatically is simply equal to the number

of data nodes. See Section 18.1.2, “NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions”, for more information.

Adding large tablespaces for Disk Data tables when using more than the default number of LDM threads may cause issues with resource and CPU usage if the disk page buffer is insufficiently large; see the description of the `DiskPageBufferMemory` configuration parameter, for more information.

The thread types are described later in this section (see `ThreadConfig`).

Setting this parameter outside the permitted range of values causes the management server to abort on startup with the error `Error line number: Illegal value value for parameter MaxNoOfExecutionThreads`.

For `MaxNoOfExecutionThreads`, a value of 0 or 1 is rounded up internally by `NDB` to 2, so that 2 is considered this parameter's default and minimum value.

`MaxNoOfExecutionThreads` is generally intended to be set equal to the number of CPU threads available, and to allocate a number of threads of each type suitable to typical workloads. It does not assign particular threads to specified CPUs. For cases where it is desirable to vary from the settings provided, or to bind threads to CPUs, you should use `ThreadConfig` instead, which allows you to allocate each thread directly to a desired type, CPU, or both.

The multithreaded data node process always spawns, at a minimum, the threads listed here:

- 1 local query handler (LDM) thread
- 1 receive thread
- 1 subscription manager (SUMA or replication) thread

For a `MaxNoOfExecutionThreads` value of 8 or less, no TC threads are created, and TC handling is instead performed by the main thread.

Changing the number of LDM threads always requires a system restart, whether it is changed using this parameter or `ThreadConfig`. If the cluster's `IndexMemory` usage is greater than 50%, changing this requires an initial restart of the cluster. (A maximum of 30-35% `IndexMemory` usage is recommended in such cases.) Otherwise, resource usage and LDM thread allocation cannot be balanced between nodes, which can result in underutilized and overutilized LDM threads, and ultimately data node failures.

- `NoOfFragmentLogParts`

Version (or later)	NDB 7.3.1
Type or units	numeric
Default	4
Range	4, 8, 12, 16
Restart Type	IN
Version (or later)	NDB 7.3.3
Type or units	numeric
Default	4
Range	4, 6, 8, 10, 12, 16, 20, 24, 32
Restart Type	IN

Set the number of log file groups for redo logs belonging to this `ndbmtd`. The value of this parameter should be set equal to the number of LDM threads used by `ndbmtd` as determined by the setting for

MaxNoOfExecutionThreads. Beginning with NDB 7.4.16, a configuration where this number is any greater than 4 is disallowed. (Bug #25333414)

See the description of MaxNoOfExecutionThreads for more information.

- ThreadConfig

Version (or later)	NDB 7.3.1
Type or units	string
Default	''
Range	...
Restart Type	IS

This parameter is used with ndbmtd to assign threads of different types to different CPUs. Its value is a string whose format has the following syntax:

```
ThreadConfig := entry[,entry[,...]]

entry := type={param[,param[,...]]}

type := ldm | main | recv | send | rep | io | tc | watchdog

param := count=number
  | cpubind=cpu_list
  | cpuset=cpu_list
  | spintime=number
  | realtime={0|1}
```

The curly braces ({...}) surrounding the list of parameters are required, even if there is only one parameter in the list.

A *param* (parameter) specifies any or all of the following information:

- The number of threads of the given type (count).
- The set of CPUs to which the threads of the given type are to be nonexclusively bound. This is determined by either one of cpubind or cpuset). cpubind causes each thread to be bound (nonexclusively) to a CPU in the set; cpuset means that each thread is bound (nonexclusively) to the set of CPUs specified.

  Only one of cpubind or cpuset can be provided in a single configuration.
- spintime determines the wait time in microseconds the thread spins before going to sleep.

  The default value for spintime is the value of the SchedulerSpinTimer data node configuration parameter.

  spintime does not apply to I/O threads or watchdog threads and so cannot be set for these thread types.
- realtime can be set to 0 or 1. If it is set to 1, the threads run with real-time priority. This also means that thread_prio cannot be set.

  The realtime parameter is set by default to the value of the RealtimeScheduler data node configuration parameter.

The *type* attribute represents an NDB thread type. The thread types supported in NDB Cluster 7.3 and later, and the range of permitted count values for each, are provided in the following list:

- ldm: Local query handler (DBLQH kernel block) that handles data. The more LDM threads that are used, the more highly partitioned the data becomes. Each LDM thread maintains its own sets of

data and index partitions, as well as its own redo log. The value set for `ldm` must be one of the values 1, 2, 4, 6, 8, 12, 16, 24, or 32. (Prior to NDB 7.3.3, the maximum value was 16.)

Changing the number of LDM threads requires a system restart to be effective and safe for cluster operations. (This is also true when this is done using `MaxNoOfExecutionThreads`.) If `IndexMemory` usage is in excess of 50%, an initial restart of the cluster is required; a maximum of 30-35% `IndexMemory` usage is recommended in such cases. Otherwise, `IndexMemory` and `DataMemory` usage as well as the allocation of LDM threads cannot be balanced between nodes, which can ultimately lead to data node failures.

Adding large tablespaces (hundreds of gigabytes or more) for Disk Data tables when using more than the default number of LDMs may cause issues with resource and CPU usage if `DiskPageBufferMemory` is not sufficiently large.

- `tc`: Transaction coordinator thread (`DBTC` kernel block) containing the state of an ongoing transaction. In NDB Cluster 7.3 and later, the number of TC threads is configurable; a total of 32 is possible in NDB 7.3.3 and later; previously this was 16.

  Optimally, every new transaction can be assigned to a new TC thread. In most cases 1 TC thread per 2 LDM threads is sufficient to guarantee that this can happen. In cases where the number of writes is relatively small when compared to the number of reads, it is possible that only 1 TC thread per 4 LQH threads is required to maintain transaction states. Conversely, in applications that perform a great many updates, it may be necessary for the ratio of TC threads to LDM threads to approach 1 (for example, 3 TC threads to 4 LDM threads).

  Setting `tc` to 0 causes TC handling to be done by the main thread. In most cases, this is effectively the same as setting it to 1.

  Range: (*NDB 7.3.3 and later*) 0 - 32; (*NDB 7.3.2 and earlier*) 0 - 16.

- `main`: Data dictionary and transaction coordinator (`DBDIH` and `DBTC` kernel blocks), providing schema management. This is always handled by a single dedicated thread.

  Range: 1 only.

- `recv`: Receive thread (`CMVMI` kernel block). Each receive thread handles one or more sockets for communicating with other nodes in an NDB Cluster, with one socket per node. NDB Cluster 7.3 and later support multiple receive threads. In NDB 7.3.2 and earlier, the maximum is 8 such threads; in NDB 7.3.3 and later, the maximum is 16.

  Range: (*NDB 7.3.3 and later*) 1 - 16; (*NDB 7.3.2 and earlier*) 1 - 8.

- `send`: Send thread (`CMVMI` kernel block). To increase throughput, it is possible to perform sends from one or more separate, dedicated threads (maximum 8).

  Previously, all threads handled their own sending directly; this can still be made to happen by setting the number of send threads to 0 (this also happens when `MaxNoOfExecutionThreads` is set less than 10). While doing so can have an adeverse impact on throughput, it can also in some cases provide decreased latency.

  Range: (*NDB 7.3.3 and later*) 0 - 16; (*NDB 7.3.2 and earlier*) 0 - 8.

- `rep`: Replication thread (`SUMA` kernel block). Asynchronous replication operations are always handled by a single, dedicated thread.

  Range: 1 only.

- `io`: File system and other miscellaneous operations. These are not demanding tasks, and are always handled as a group by a single, dedicated I/O thread.

  Range: 1 only.

- `watchdog`: Settings to this parameter are actually applied to several threads of this type having specific uses. These threads include the `SocketServer` thread which receives connection setups from other nodes, the `SocketClient` thread which attempts to set up connections to other nodes, and the thread watchdog thread that checks that threads are progressing.

  Range: 1 only.

Simple examples:

```
# Example 1.

ThreadConfig=ldm={count=2,cpubind=1,2},main={cpubind=12},rep={cpubind=11}

# Example 2.

Threadconfig=main={cpubind=0},ldm={count=4,cpubind=1,2,5,6},io={cpubind=3}
```

It is usually desirable when configuring thread usage for a data node host to reserve one or more number of CPUs for operating system and other tasks. Thus, for a host machine with 24 CPUs, you might want to use 20 CPU threads (leaving 4 for other uses), with 8 LDM threads, 4 TC threads (half the number of LDM threads), 3 send threads, 3 receive threads, and 1 thread each for schema management, asynchronous replication, and I/O operations. (This is almost the same distribution of threads used when `MaxNoOfExecutionThreads` is set equal to 20.) The following `ThreadConfig` setting performs these assignments, additionally binding all of these threads to specific CPUs:

```
ThreadConfig=ldm{count=8,cpubind=1,2,3,4,5,6,7,8},main={cpubind=9},io={cpubind=9}, \
rep={cpubind=10},tc{count=4,cpubind=11,12,13,14},recv={count=3,cpubind=15,16,17}, \
send{count=3,cpubind=18,19,20}
```

It should be possible in most cases to bind the main (schema management) thread and the I/O thread to the same CPU, as we have done in the example just shown.

In order to take advantage of the enhanced stability that the use of `ThreadConfig` offers, it is necessary to insure that CPUs are isolated, and that they not subject to interrupts, or to being scheduled for other tasks by the operating system. On many Linux systems, you can do this by setting `IRQBALANCE_BANNED_CPUS` in `/etc/sysconfig/irqbalance` to `0xFFFFF0`, and by using the `isolcpus` boot option in `grub.conf`. For specific information, see your operating system or platform documentation.

Disk Data Configuration Parameters. Configuration parameters affecting Disk Data behavior include the following:

- `DiskPageBufferEntries`

Version (or later)	NDB 7.3.8
Type or units	32K pages
Default	10
Range	1 - 1000
Restart Type	N
Version (or later)	NDB 7.4.3
Type or units	32K pages
Default	10
Range	1 - 1000

Restart Type	N
Added	NDB 7.3.8
Added	NDB 7.4.3

This is the number of page entries (page references) to allocate. It is specified as a number of 32K pages in `DiskPageBufferMemory`. The default is sufficient for most cases but you may need to increase the value of this parameter if you encounter problems with very large transactions on Disk Data tables. Each page entry requires approximately 100 bytes.

- `DiskPageBufferMemory`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	64M
Range	4M - 1T
Restart Type	N

This determines the amount of space used for caching pages on disk, and is set in the `[ndbd]` or `[ndbd default]` section of the `config.ini` file. It is measured in bytes. Each page takes up 32 KB. This means that NDB Cluster Disk Data storage always uses *`N`* * 32 KB memory where *`N`* is some nonnegative integer.

The default value for this parameter is `64M` (2000 pages of 32 KB each).

If the value for `DiskPageBufferMemory` is set too low in conjunction with using more than the default number of LDM threads in `ThreadConfig` (for example `{ldm=6...}`), problems can arise when trying to add a large (for example 500G) data file to a disk-based `NDB` table, wherein the process takes indefinitely long while occupying one of the CPU cores.

This is due to the fact that, as part of adding a data file to a tablespace, extent pages are locked into memory in an extra PGMAN worker thread, for quick metadata access. When adding a large file, this worker has insufficient memory for all of the data file metadata. In such cases, you should either increase `DiskPageBufferMemory`, or add smaller tablespace files. You may also need to adjust `DiskPageBufferEntries`.

You can query the `ndbinfo.diskpagebuffer` table to help determine whether the value for this parameter should be increased to minimize unnecessary disk seeks. See Section 18.5.14.12, “The ndbinfo diskpagebuffer Table”, for more information.

- `SharedGlobalMemory`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	128M
Range	0 - 64T
Restart Type	N

This parameter determines the amount of memory that is used for log buffers, disk operations (such as page requests and wait queues), and metadata for tablespaces, log file groups, `UNDO` files, and data files. The shared global memory pool also provides memory used for satisfying the memory requirements of the `UNDO_BUFFER_SIZE` option used with `CREATE LOGFILE GROUP` and `ALTER LOGFILE GROUP` statements, including any default value implied for this options by the setting of the `InitialLogFileGroup` data node configuration parameter. `SharedGlobalMemory` can be set in

the [ndbd] or [ndbd default] section of the config.ini configuration file, and is measured in bytes.

The default value is 128M.

- DiskIOThreadPool

Version (or later)	NDB 7.3.1
Type or units	threads
Default	2
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter determines the number of unbound threads used for Disk Data file access. Before DiskIOThreadPool was introduced, exactly one thread was spawned for each Disk Data file, which could lead to performance issues, particularly when using very large data files. With DiskIOThreadPool, you can—for example—access a single large data file using several threads working in parallel.

This parameter applies to Disk Data I/O threads only.

The optimum value for this parameter depends on your hardware and configuration, and includes these factors:

- **Physical distribution of Disk Data files.** You can obtain better performance by placing data files, undo log files, and the data node file system on separate physical disks. If you do this with some or all of these sets of files, then you can set DiskIOThreadPool higher to enable separate threads to handle the files on each disk.
- **Disk performance and types.** The number of threads that can be accommodated for Disk Data file handling is also dependent on the speed and throughput of the disks. Faster disks and higher throughput allow for more disk I/O threads. Our test results indicate that solid-state disk drives can handle many more disk I/O threads than conventional disks, and thus higher values for DiskIOThreadPool.

The default value for this parameter is 2.

- **Disk Data file system parameters.** The parameters in the following list make it possible to place NDB Cluster Disk Data files in specific directories without the need for using symbolic links.

  - FileSystemPathDD

Version (or later)	NDB 7.3.1
Type or units	filename
Default	FileSystemPath
Range	...
Restart Type	IN

If this parameter is specified, then NDB Cluster Disk Data data files and undo log files are placed in the indicated directory. This can be overridden for data files, undo log files, or both, by specifying values for FileSystemPathDataFiles, FileSystemPathUndoFiles, or both, as explained for these parameters. It can also be overridden for data files by specifying a path in the ADD DATAFILE clause of a CREATE TABLESPACE or ALTER TABLESPACE statement, and for undo log files by specifying a path in the ADD UNDOFILE clause of a CREATE LOGFILE

GROUP or ALTER LOGFILE GROUP statement. If FileSystemPathDD is not specified, then FileSystemPath is used.

If a FileSystemPathDD directory is specified for a given data node (including the case where the parameter is specified in the [ndbd default] section of the config.ini file), then starting that data node with --initial causes all files in the directory to be deleted.

- FileSystemPathDataFiles

Version (or later)	NDB 7.3.1
Type or units	filename
Default	FileSystemPathDD
Range	...
Restart Type	IN

If this parameter is specified, then NDB Cluster Disk Data data files are placed in the indicated directory. This overrides any value set for FileSystemPathDD. This parameter can be overridden for a given data file by specifying a path in the ADD DATAFILE clause of a CREATE TABLESPACE or ALTER TABLESPACE statement used to create that data file. If FileSystemPathDataFiles is not specified, then FileSystemPathDD is used (or FileSystemPath, if FileSystemPathDD has also not been set).

If a FileSystemPathDataFiles directory is specified for a given data node (including the case where the parameter is specified in the [ndbd default] section of the config.ini file), then starting that data node with --initial causes all files in the directory to be deleted.

- FileSystemPathUndoFiles

Version (or later)	NDB 7.3.1
Type or units	filename
Default	FileSystemPathDD
Range	...
Restart Type	IN

If this parameter is specified, then NDB Cluster Disk Data undo log files are placed in the indicated directory. This overrides any value set for FileSystemPathDD. This parameter can be overridden for a given data file by specifying a path in the ADD UNDO clause of a CREATE LOGFILE GROUP or ALTER LOGFILE GROUP statement used to create that data file. If FileSystemPathUndoFiles is not specified, then FileSystemPathDD is used (or FileSystemPath, if FileSystemPathDD has also not been set).

If a FileSystemPathUndoFiles directory is specified for a given data node (including the case where the parameter is specified in the [ndbd default] section of the config.ini file), then starting that data node with --initial causes all files in the directory to be deleted.

For more information, see Section 18.5.10.1, “NDB Cluster Disk Data Objects”.

- **Disk Data object creation parameters.** The next two parameters enable you—when starting the cluster for the first time—to cause a Disk Data log file group, tablespace, or both, to be created without the use of SQL statements.

  - InitialLogFileGroup

Version (or later)	NDB 7.3.1
Type or units	string

Default	[see documentation]
Range	...
Restart Type	S

This parameter can be used to specify a log file group that is created when performing an initial start of the cluster. `InitialLogFileGroup` is specified as shown here:

```
InitialLogFileGroup = [name=name;] [undo_buffer_size=size;] file-specification-list

file-specification-list:
    file-specification[; file-specification[; ...]]

file-specification:
    filename:size
```

The `name` of the log file group is optional and defaults to `DEFAULT-LG`. The `undo_buffer_size` is also optional; if omitted, it defaults to `64M`. Each *`file-specification`* corresponds to an undo log file, and at least one must be specified in the *`file-specification-list`*. Undo log files are placed according to any values that have been set for `FileSystemPath`, `FileSystemPathDD`, and `FileSystemPathUndoFiles`, just as if they had been created as the result of a `CREATE LOGFILE GROUP` or `ALTER LOGFILE GROUP` statement.

Consider the following:

```
InitialLogFileGroup = name=LG1; undo_buffer_size=128M; undo1.log:250M; undo2.log:150M
```

This is equivalent to the following SQL statements:

```
CREATE LOGFILE GROUP LG1
    ADD UNDOFILE 'undo1.log'
    INITIAL_SIZE 250M
    UNDO_BUFFER_SIZE 128M
    ENGINE NDBCLUSTER;

ALTER LOGFILE GROUP LG1
    ADD UNDOFILE 'undo2.log'
    INITIAL_SIZE 150M
    ENGINE NDBCLUSTER;
```

This logfile group is created when the data nodes are started with `--initial`.

Prior to NDB 7.3.6, resources for the initial log file group are taken from the global memory pool whose size is determined by the value of the `SharedGlobalMemory` data node configuration parameter; in these versions, if this parameter is set too low and the values set in `InitialLogFileGroup` for the logfile group's initial size or undo buffer size are too high, the cluster may fail to create the default log file group when starting, or fail to start altogether. In NDB 7.3.6 and later, resources for the initial log file group are added to the global memory pool along with those indicated by the value of `SharedGlobalMemory` (Bug #11762867).

This parameter, if used, should always be set in the `[ndbd default]` section of the `config.ini` file. The behavior of an NDB Cluster when different values are set on different data nodes is not defined.

- `InitialTablespace`

Version (or later)	NDB 7.3.1
Type or units	string
Default	[see documentation]
Range	...

Restart Type	S

This parameter can be used to specify an NDB Cluster Disk Data tablespace that is created when performing an initial start of the cluster. `InitialTablespace` is specified as shown here:

```
InitialTablespace = [name=name;] [extent_size=size;] file-specification-list
```

The `name` of the tablespace is optional and defaults to `DEFAULT-TS`. The `extent_size` is also optional; it defaults to `1M`. The *`file-specification-list`* uses the same syntax as shown with the `InitialLogfileGroup` parameter, the only difference being that each *`file-specification`* used with `InitialTablespace` corresponds to a data file. At least one must be specified in the *`file-specification-list`*. Data files are placed according to any values that have been set for `FileSystemPath`, `FileSystemPathDD`, and `FileSystemPathDataFiles`, just as if they had been created as the result of a `CREATE TABLESPACE` or `ALTER TABLESPACE` statement.

For example, consider the following line specifying `InitialTablespace` in the `[ndbd default]` section of the `config.ini` file (as with `InitialLogfileGroup`, this parameter should always be set in the `[ndbd default]` section, as the behavior of an NDB Cluster when different values are set on different data nodes is not defined):

```
InitialTablespace = name=TS1; extent_size=8M; data1.dat:2G; data2.dat:4G
```

This is equivalent to the following SQL statements:

```
CREATE TABLESPACE TS1
    ADD DATAFILE 'data1.dat'
    EXTENT_SIZE 8M
    INITIAL_SIZE 2G
    ENGINE NDBCLUSTER;

ALTER TABLESPACE TS1
    ADD DATAFILE 'data2.dat'
    INITIAL_SIZE 4G
    ENGINE NDBCLUSTER;
```

This tablespace is created when the data nodes are started with `--initial`, and can be used whenever creating NDB Cluster Disk Data tables thereafter.

Disk Data and GCP Stop errors. Errors encountered when using Disk Data tables such as `Node nodeid killed this node because GCP stop was detected` (error 2303) are often referred to as “GCP stop errors”. Such errors occur when the redo log is not flushed to disk quickly enough; this is usually due to slow disks and insufficient disk throughput.

You can help prevent these errors from occurring by using faster disks, and by placing Disk Data files on a separate disk from the data node file system. Reducing the value of `TimeBetweenGlobalCheckpoints` tends to decrease the amount of data to be written for each global checkpoint, and so may provide some protection against redo log buffer overflows when trying to write a global checkpoint; however, reducing this value also permits less time in which to write the GCP, so this must be done with caution.

In addition to the considerations given for `DiskPageBufferMemory` as explained previously, it is also very important that the `DiskIOThreadPool` configuration parameter be set correctly; having `DiskIOThreadPool` set too high is very likely to cause GCP stop errors (Bug #37227).

GCP stops can be caused by save or commit timeouts; the `TimeBetweenEpochsTimeout` data node configuration parameter determines the timeout for commits. However, it is possible to disable both types of timeouts by setting this parameter to 0.

Parameters for configuring send buffer memory allocation. Send buffer memory is allocated dynamically from a memory pool shared between all transporters, which means that the size of the send buffer can be adjusted as necessary. (Previously, the NDB kernel used a fixed-size send buffer for every node in the cluster, which was allocated when the node started and could not be changed

while the node was running.) The TotalSendBufferMemory and OverLoadLimit data node configuration parameters permit the setting of limits on this memory allocation. For more information about the use of these parameters (as well as SendBufferMemory), see Section 18.3.3.12, “Configuring NDB Cluster Send Buffer Parameters”.

- ExtraSendBufferMemory

  This parameter specifies the amount of transporter send buffer memory to allocate in addition to any set using TotalSendBufferMemory, SendBufferMemory, or both.

- TotalSendBufferMemory

  This parameter is used to determine the total amount of memory to allocate on this node for shared send buffer memory among all configured transporters.

  If this parameter is set, its minimum permitted value is 256KB; 0 indicates that the parameter has not been set. For more detailed information, see Section 18.3.3.12, “Configuring NDB Cluster Send Buffer Parameters”.

- ReservedSendBufferMemory

  This parameter is present in NDBCLUSTER source code, but is not currently enabled; it is deprecated, and subject to removal in a future release of NDB Cluster (Bug #11760629, Bug #53053).

For more detailed information about the behavior and use of TotalSendBufferMemory and ReservedSendBufferMemory, and about configuring send buffer memory parameters in NDB Cluster, see Section 18.3.3.12, “Configuring NDB Cluster Send Buffer Parameters”.

See also Section 18.5.7, “Adding NDB Cluster Data Nodes Online”.

Redo log over-commit handling. It is possible to control a data node's handling of operations when too much time is taken flushing redo logs to disk. This occurs when a given redo log flush takes longer than RedoOverCommitLimit seconds, more than RedoOverCommitCounter times, causing any pending transactions to be aborted. When this happens, the API node that sent the transaction can handle the operations that should have been committed either by queuing the operations and re-trying them, or by aborting them, as determined by DefaultOperationRedoProblemAction. The data node configuration parameters for setting the timeout and number of times it may be exceeded before the API node takes this action are described in the following list:

- RedoOverCommitCounter

Version (or later)	NDB 7.3.1
Type or units	numeric
Default	3
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

  When RedoOverCommitLimit is exceeded when trying to write a given redo log to disk this many times or more, any transactions that were not committed as a result are aborted, and an API node where any of these transactions originated handles the operations making up those transactions according to its value for DefaultOperationRedoProblemAction (by either queuing the operations to be re-tried, or aborting them).

  RedoOverCommitCounter defaults to 3. Set it to 0 to disable the limit.

- RedoOverCommitLimit

Version (or later)	NDB 7.3.1
Type or units	seconds

Default	20
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter sets an upper limit in seconds for trying to write a given redo log to disk before timing out. The number of times the data node tries to flush this redo log, but takes longer than `RedoOverCommitLimit`, is kept and compared with `RedoOverCommitCounter`, and when flushing takes too long more times than the value of that parameter, any transactions that were not committed as a result of the flush timeout are aborted. When this occurs, the API node where any of these transactions originated handles the operations making up those transactions according to its `DefaultOperationRedoProblemAction` setting (it either queues the operations to be re-tried, or aborts them).

By default, `RedoOverCommitLimit` is 20 seconds. Set to 0 to disable checking for redo log flush timeouts. This parameter was added in NDB 7.1.10.

Controlling restart attempts. It is possible to exercise finely-grained control over restart attempts by data nodes when they fail to start using the `MaxStartFailRetries` and `StartFailRetryDelay` data node configuration parameters.

`MaxStartFailRetries` limits the total number of retries made before giving up on starting the data node, `StartFailRetryDelay` sets the number of seconds between retry attempts. These parameters are listed here:

- `StartFailRetryDelay`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Use this parameter to set the number of seconds between restart attempts by the data node in the event on failure on startup. The default is 0 (no delay).

Both this parameter and `MaxStartFailRetries` are ignored unless `StopOnError` is equal to 0.

- `MaxStartFailRetries`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	3
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Use this parameter to limit the number restart attempts made by the data node in the event that it fails on startup. The default is 3 attempts.

Both this parameter and `StartFailRetryDelay` are ignored unless `StopOnError` is equal to 0.

NDB index statistics parameters. The parameters in the following list relate to NDB index statistics generation.

- `IndexStatAutoCreate`

Version (or later)	NDB 7.3.1

Type or units	integer
Default	0
Range	0, 1
Restart Type	S

Enable (set equal to 1) or disable (set equal to 0) automatic statistics collection when indexes are created. Disabled by default.

- `IndexStatAutoUpdate`

Version (or later)	NDB 7.3.1
Type or units	integer
Default	0
Range	0, 1
Restart Type	S

Enable (set equal to 1) or disable (set equal to 0) monitoring of indexes for changes and trigger automatic statistics updates these are detected. The amount and degree of change needed to trigger the updates are determined by the settings for the `IndexStatTriggerPct` and `IndexStatTriggerScale` options.

- `IndexStatSaveSize`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	32768
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	IN

Maximum space in bytes allowed for the saved statistics of any given index in the `NDB` system tables and in the `mysqld` memory cache. This consumes `IndexMemory`.

At least one sample is always produced, regardless of any size limit. This size is scaled by `IndexStatSaveScale`.

The size specified by `IndexStatSaveSize` is scaled by the value of `IndexStatTriggerPct` for a large index, times 0.01. This is further multiplied by the logarithm to the base 2 of the index size. Setting `IndexStatTriggerPct` equal to 0 disables the scaling effect.

- `IndexStatSaveScale`

Version (or later)	NDB 7.3.1
Type or units	percentage
Default	100
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	IN

The size specified by `IndexStatSaveSize` is scaled by the value of `IndexStatTriggerPct` for a large index, times 0.01. This is further multiplied by the logarithm to the base 2 of the index size. Setting `IndexStatTriggerPct` equal to 0 disables the scaling effect.

- `IndexStatTriggerPct`

Version (or later)	NDB 7.3.1
Type or units	percentage
Default	100
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	IN

Percentage change in updates that triggers an index statistics update. The value is scaled by `IndexStatTriggerScale`. You can disable this trigger altogether by setting `IndexStatTriggerPct` to 0.

- `IndexStatTriggerScale`

Version (or later)	NDB 7.3.1
Type or units	percentage
Default	100
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	IN

Scale `IndexStatTriggerPct` by this amount times 0.01 for a large index. A value of 0 disables scaling.

- `IndexStatUpdateDelay`

Version (or later)	NDB 7.3.1
Type or units	seconds
Default	60
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	IN

Minimum delay in seconds between automatic index statistics updates for a given index. Setting this variable to 0 disables any delay. The default is 60 seconds.

Restart types. Information about the restart types used by the parameter descriptions in this section is shown in the following table:

Table 18.12 NDB Cluster restart types

Symbol	Restart Type	Description
N	Node	The parameter can be updated using a rolling restart (see Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”)
S	System	All cluster nodes must be shut down completely, then restarted, to effect a change in this parameter
I	Initial	Data nodes must be restarted using the `--initial` option

18.3.3.7 Defining SQL and Other API Nodes in an NDB Cluster

The `[mysqld]` and `[api]` sections in the `config.ini` file define the behavior of the MySQL servers (SQL nodes) and other applications (API nodes) used to access cluster data. None of the parameters shown is required. If no computer or host name is provided, any host can use this SQL or API node.

Generally speaking, a `[mysqld]` section is used to indicate a MySQL server providing an SQL interface to the cluster, and an `[api]` section is used for applications other than `mysqld` processes

accessing cluster data, but the two designations are actually synonymous; you can, for instance, list parameters for a MySQL server acting as an SQL node in an `[api]` section.

Note

For a discussion of MySQL server options for NDB Cluster, see MySQL Server Options for NDB Cluster. For information about MySQL server system variables relating to NDB Cluster, see NDB Cluster System Variables.

- `Id`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	[none]
Range	1 - 255
Restart Type	IS

The `Id` is an integer value used to identify the node in all cluster internal messages. The permitted range of values is 1 to 255 inclusive. This value must be unique for each node in the cluster, regardless of the type of node.

Note

Data node IDs must be less than 49, regardless of the NDB Cluster version used. If you plan to deploy a large number of data nodes, it is a good idea to limit the node IDs for API nodes (and management nodes) to values greater than 48.

`NodeId` is the preferred parameter name to use when identifying API nodes. (`Id` continues to be supported for backward compatibility, but is now deprecated and generates a warning when used. It is also subject to future removal.)

- `ConnectionMap`

Version (or later)	NDB 7.3.1
Type or units	string
Default	[none]
Range	...
Restart Type	N

Specifies which data nodes to connect.

- `NodeId`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	[none]
Range	1 - 255
Restart Type	IS

The `NodeId` is an integer value used to identify the node in all cluster internal messages. The permitted range of values is 1 to 255 inclusive. This value must be unique for each node in the cluster, regardless of the type of node.

Note

Data node IDs must be less than 49, regardless of the NDB Cluster version used. If you plan to deploy a large number of data nodes, it is a good idea to limit the node IDs for API nodes (and management nodes) to values greater than 48.

`NodeId` is the preferred parameter name to use when identifying management nodes. An alias, `Id`, was used for this purpose in very old versions of NDB Cluster, and continues to be supported for backward compatibility; it is now deprecated and generates a warning when used, and is subject to removal in a future release of NDB Cluster.

- `ExecuteOnComputer`

Version (or later)	NDB 7.3.1
Type or units	name
Default	[none]
Range	...
Restart Type	S
Deprecated	Yes (in NDB 7.5)

This refers to the `Id` set for one of the computers (hosts) defined in a `[computer]` section of the configuration file.

- `HostName`

Version (or later)	NDB 7.3.1
Type or units	name or IP address
Default	[none]
Range	...
Restart Type	N

Specifying this parameter defines the hostname of the computer on which the SQL node (API node) is to reside. To specify a hostname, either this parameter or `ExecuteOnComputer` is required.

If no `HostName` or `ExecuteOnComputer` is specified in a given `[mysql]` or `[api]` section of the `config.ini` file, then an SQL or API node may connect using the corresponding “slot” from any host which can establish a network connection to the management server host machine. *This differs from the default behavior for data nodes, where `localhost` is assumed for `HostName` unless otherwise specified*.

- `ArbitrationRank`

Version (or later)	NDB 7.3.1
Type or units	0-2
Default	0
Range	0 - 2
Restart Type	N

This parameter defines which nodes can act as arbitrators. Both management nodes and SQL nodes can be arbitrators. A value of 0 means that the given node is never used as an arbitrator, a value of 1 gives the node high priority as an arbitrator, and a value of 2 gives it low priority. A normal

configuration uses the management server as arbitrator, setting its `ArbitrationRank` to 1 (the default for management nodes) and those for all SQL nodes to 0 (the default for SQL nodes).

By setting `ArbitrationRank` to 0 on all management and SQL nodes, you can disable arbitration completely. You can also control arbitration by overriding this parameter; to do so, set the `Arbitration` parameter in the `[ndbd default]` section of the `config.ini` global configuration file.

- `ArbitrationDelay`

Version (or later)	NDB 7.3.1
Type or units	milliseconds
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

Setting this parameter to any other value than 0 (the default) means that responses by the arbitrator to arbitration requests are delayed by the stated number of milliseconds. It is usually not necessary to change this value.

- `BatchByteSize`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	16K
Range	1K - 1M
Restart Type	N

For queries that are translated into full table scans or range scans on indexes, it is important for best performance to fetch records in properly sized batches. It is possible to set the proper size both in terms of number of records (`BatchSize`) and in terms of bytes (`BatchByteSize`). The actual batch size is limited by both parameters.

The speed at which queries are performed can vary by more than 40% depending upon how this parameter is set.

This parameter is measured in bytes. The default value in NDB Cluster 7.3 and later is 16K.

- `BatchSize`

Version (or later)	NDB 7.3.1
Type or units	records
Default	256
Range	1 - 992
Restart Type	N

This parameter is measured in number of records and is by default set to 256. The maximum size is 992.

- `ExtraSendBufferMemory`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	0

Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter specifies the amount of transporter send buffer memory to allocate in addition to any that has been set using `TotalSendBufferMemory`, `SendBufferMemory`, or both.

- `HeartbeatThreadPriority`

Version (or later)	NDB 7.3.1
Type or units	string
Default	[none]
Range	...
Restart Type	S

Use this parameter to set the scheduling policy and priority of heartbeat threads for management and API nodes. The syntax for setting this parameter is shown here:

```
HeartbeatThreadPriority = policy[, priority]

policy:
  {FIFO | RR}
```

When setting this parameter, you must specify a policy. This is one of `FIFO` (first in, first in) or `RR` (round robin). This followed optionally by the priority (an integer).

- `MaxScanBatchSize`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	256K
Range	32K - 16M
Restart Type	N

The batch size is the size of each batch sent from each data node. Most scans are performed in parallel to protect the MySQL Server from receiving too much data from many nodes in parallel; this parameter sets a limit to the total batch size over all nodes.

The default value of this parameter is set to 256KB. Its maximum size is 16MB.

- `TotalSendBufferMemory`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	0
Range	256K - 4294967039 (0xFFFFFEFF)
Restart Type	N

This parameter is used to determine the total amount of memory to allocate on this node for shared send buffer memory among all configured transporters.

If this parameter is set, its minimum permitted value is 256KB; 0 indicates that the parameter has not been set. For more detailed information, see Section 18.3.3.12, “Configuring NDB Cluster Send Buffer Parameters”.

- `AutoReconnect`

Version (or later)	NDB 7.3.1
Type or units	boolean
Default	false
Range	true, false
Restart Type	N

This parameter is `false` by default. This forces disconnected API nodes (including MySQL Servers acting as SQL nodes) to use a new connection to the cluster rather than attempting to re-use an existing one, as re-use of connections can cause problems when using dynamically-allocated node IDs. (Bug #45921)

Note

This parameter can be overridden using the NDB API. For more information, see Ndb_cluster_connection::set_auto_reconnect(), and Ndb_cluster_connection::get_auto_reconnect().

- `DefaultOperationRedoProblemAction`

Version (or later)	NDB 7.3.1
Type or units	enumeration
Default	QUEUE
Range	ABORT, QUEUE
Restart Type	S

This parameter (along with `RedoOverCommitLimit` and `RedoOverCommitCounter`) controls the data node's handling of operations when too much time is taken flushing redo logs to disk. This occurs when a given redo log flush takes longer than `RedoOverCommitLimit` seconds, more than `RedoOverCommitCounter` times, causing any pending transactions to be aborted.

When this happens, the node can respond in either of two ways, according to the value of `DefaultOperationRedoProblemAction`, listed here:

- `ABORT`: Any pending operations from aborted transactions are also aborted.
- `QUEUE`: Pending operations from transactions that were aborted are queued up to be re-tried. This the default. In NDB 7.3.10 and later as well as NDB 7.4.7 and later, pending operations are still aborted when the redo log runs out of space—that is, when **`P_TAIL_PROBLEM`** errors occur. (Bug #20782580)

- `DefaultHashMapSize`

Version (or later)	NDB 7.3.1
Type or units	buckets
Default	3840
Range	0 - 3840

Restart Type	N

The size of the table hash maps used by `NDB` is configurable using this parameter. `DefaultHashMapSize` can take any of three possible values (0, 240, 3840).

The original intended use for this parameter was to facilitate upgrades and especially downgrades to and from very old releases with differing default hash map sizes. This is not an issue when upgrading from NDB Cluster 7.3 (or later) to later versions.

Decreasing this parameter online after any tables have been created or modified with `DefaultHashMapSize` equal to 3840 is not currently supported.

- `Wan`

Version (or later)	NDB 7.3.1
Type or units	boolean
Default	false
Range	true, false
Restart Type	N

Use WAN TCP setting as default.

- `ConnectBackoffMaxTime`

Version (or later)	NDB 7.3.7
Type or units	integer
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N
Version (or later)	NDB 7.4.2
Type or units	integer
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N
Added	NDB 7.3.7
Added	NDB 7.4.2

Starting with NDB 7.3.7 and NDB 7.4.2, in an NDB Cluster with many unstarted data nodes, the value of this parameter can be raised to circumvent connection attempts to data nodes which have not yet begun to function in the cluster, as well as moderate high traffic to management nodes. As long as the API node is not connected to any new data nodes, the value of the `StartConnectBackoffMaxTime` parameter is applied; otherwise, `ConnectBackoffMaxTime` is used to determine the length of time in milliseconds to wait between connection attempts.

Time elapsed *during* node connection attempts is not taken into account when calculating elapsed time for this parameter. The timeout is applied with approximately 100 ms resolution, starting with a 100 ms delay; for each subsequent attempt, the length of this period is doubled until it reaches `ConnectBackoffMaxTime` milliseconds, up to a maximum of 100000 ms (100s).

Once the API node is connected to a data node and that node reports (in a heartbeat message) that it has connected to other data nodes, connection attempts to those data nodes are no longer affected by this parameter, and are made every 100 ms thereafter until connected. Once a data node

has started, it can take up `HeartbeatIntervalDbApi` for the API node to be notified that this has occurred.

- `StartConnectBackoffMaxTime`

Version (or later)	NDB 7.3.7
Type or units	integer
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N
Version (or later)	NDB 7.4.2
Type or units	integer
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N
Added	NDB 7.3.7
Added	NDB 7.4.2

Starting with NDB 7.3.7 and NDB 7.4.2, in an NDB Cluster with many unstarted data nodes, the value of this parameter can be raised to circumvent connection attempts to data nodes which have not yet begun to function in the cluster, as well as moderate high traffic to management nodes. As long as the API node is not connected to any new data nodes, the value of the `StartConnectBackoffMaxTime` parameter is applied; otherwise, `ConnectBackoffMaxTime` is used to determine the length of time in milliseconds to wait between connection attempts.

Time elapsed *during* node connection attempts is not taken into account when calculating elapsed time for this parameter. The timeout is applied with approximately 100 ms resolution, starting with a 100 ms delay; for each subsequent attempt, the length of this period is doubled until it reaches `StartConnectBackoffMaxTime` milliseconds, up to a maximum of 100000 ms (100s).

Once the API node is connected to a data node and that node reports (in a heartbeat message) that it has connected to other data nodes, connection attempts to those data nodes are no longer affected by this parameter, and are made every 100 ms thereafter until connected. Once a data node has started, it can take up `HeartbeatIntervalDbApi` for the API node to be notified that this has occurred.

API Node Debugging Parameters. Beginning with NDB 7.4.12, you can use the `ApiVerbose` configuration parameter to enable debugging output from a given API node. This parameter takes an integer value. 0 is the default, and disables such debugging; 1 enables debugging output to the cluster log; 2 adds `DBDICT` debugging output as well. (Bug #20638450) See also DUMP 1229.

You can also obtain information from a MySQL server running as an NDB Cluster SQL node using `SHOW STATUS` in the `mysql` client, as shown here:

```
mysql> SHOW STATUS LIKE 'ndb%';
+-----------------------------+----------------+
| Variable_name               | Value          |
+-----------------------------+----------------+
| Ndb_cluster_node_id         | 5              |
| Ndb_config_from_host        | 198.51.100.112 |
| Ndb_config_from_port        | 1186           |
| Ndb_number_of_storage_nodes | 4              |
+-----------------------------+----------------+
4 rows in set (0.02 sec)
```

For information about the status variables appearing in the output from this statement, see NDB Cluster Status Variables.

Note

To add new SQL or API nodes to the configuration of a running NDB Cluster, it is necessary to perform a rolling restart of all cluster nodes after adding new `[mysqld]` or `[api]` sections to the `config.ini` file (or files, if you are using more than one management server). This must be done before the new SQL or API nodes can connect to the cluster.

It is *not* necessary to perform any restart of the cluster if new SQL or API nodes can employ previously unused API slots in the cluster configuration to connect to the cluster.

Restart types. Information about the restart types used by the parameter descriptions in this section is shown in the following table:

Table 18.13 NDB Cluster restart types

Symbol	Restart Type	Description
N	Node	The parameter can be updated using a rolling restart (see Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”)
S	System	All cluster nodes must be shut down completely, then restarted, to effect a change in this parameter
I	Initial	Data nodes must be restarted using the `--initial` option

18.3.3.8 MySQL Server Options and Variables for NDB Cluster

This section provides information about MySQL server options, server and status variables that are specific to NDB Cluster. For general information on using these, and for other options and variables not specific to NDB Cluster, see Section 5.1, “The MySQL Server”.

For NDB Cluster configuration parameters used in the cluster configuration file (usually named `config.ini`), see Section 18.3, “Configuration of NDB Cluster”.

MySQL Server Options for NDB Cluster

This section provides descriptions of `mysqld` server options relating to NDB Cluster. For information about `mysqld` options not specific to NDB Cluster, and for general information about the use of options with `mysqld`, see Section 5.1.6, “Server Command Options”.

For information about command-line options used with other NDB Cluster processes (`ndbd`, `ndb_mgmd`, and `ndb_mgm`), see Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”. For information about command-line options used with `NDB` utility programs (such as `ndb_desc`, `ndb_size.pl`, and `ndb_show_tables`), see Section 18.4, “NDB Cluster Programs”.

- `--ndbcluster`

Command-Line Format	`--ndbcluster[=value]`
Disabled by	`skip-ndbcluster`
Type	Boolean
Default Value	`OFF`

The `NDBCLUSTER` storage engine is necessary for using NDB Cluster. If a `mysqld` binary includes support for the `NDBCLUSTER` storage engine, the engine is disabled by default. Use the `--ndbcluster` option to enable it. Use `--skip-ndbcluster` to explicitly disable the engine.

- `--ndb-batch-size=#`

Command-Line Format	`--ndb-batch-size`
System Variable	`ndb_batch_size`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`32768`
Minimum Value	`0`
Maximum Value	`31536000`

This sets the size in bytes that is used for NDB transaction batches.

- `--ndb-cluster-connection-pool=#`

Command-Line Format	`--ndb-cluster-connection-pool`
System Variable	`ndb_cluster_connection_pool`
System Variable	`ndb_cluster_connection_pool`
Scope	Global
Scope	Global
Dynamic	No
Dynamic	No
Type	Integer
Default Value	`1`
Minimum Value	`1`
Maximum Value	`63`

By setting this option to a value greater than 1 (the default), a `mysqld` process can use multiple connections to the cluster, effectively mimicking several SQL nodes. Each connection requires its own `[api]` or `[mysqld]` section in the cluster configuration (`config.ini`) file, and counts against the maximum number of API connections supported by the cluster.

Suppose that you have 2 cluster host computers, each running an SQL node whose `mysqld` process was started with `--ndb-cluster-connection-pool=4`; this means that the cluster must have 8 API slots available for these connections (instead of 2). All of these connections are set up when the SQL node connects to the cluster, and are allocated to threads in a round-robin fashion.

This option is useful only when running `mysqld` on host machines having multiple CPUs, multiple cores, or both. For best results, the value should be smaller than the total number of cores available on the host machine. Setting it to a value greater than this is likely to degrade performance severely.

Important

Because each SQL node using connection pooling occupies multiple API node slots—each slot having its own node ID in the cluster—you must *not* use a node ID as part of the cluster connection string when starting any `mysqld` process that employs connection pooling.

Setting a node ID in the connection string when using the `--ndb-cluster-connection-pool` option causes node ID allocation errors when the SQL node attempts to connect to the cluster.

- `--ndb-blob-read-batch-bytes=bytes`

Command-Line Format	`--ndb-blob-read-batch-bytes`
System Variable	`ndb_blob_read_batch_bytes`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`65536`
Minimum Value	`0`
Maximum Value	`4294967295`

This option can be used to set the size (in bytes) for batching of `BLOB` data reads in NDB Cluster applications. When this batch size is exceeded by the amount of `BLOB` data to be read within the current transaction, any pending `BLOB` read operations are immediately executed.

The maximum value for this option is 4294967295; the default is 65536. Setting it to 0 has the effect of disabling `BLOB` read batching.

Note

In NDB API applications, you can control `BLOB` write batching with the `setMaxPendingBlobReadBytes()` and `getMaxPendingBlobReadBytes()` methods.

- `--ndb-blob-write-batch-bytes=`*`bytes`*

Command-Line Format	`--ndb-blob-write-batch-bytes`
System Variable	`ndb_blob_write_batch_bytes`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`65536`
Minimum Value	`0`
Maximum Value	`4294967295`

This option can be used to set the size (in bytes) for batching of `BLOB` data writes in NDB Cluster applications. When this batch size is exceeded by the amount of `BLOB` data to be written within the current transaction, any pending `BLOB` write operations are immediately executed.

The maximum value for this option is 4294967295; the default is 65536. Setting it to 0 has the effect of disabling `BLOB` write batching.

Note

In NDB API applications, you can control `BLOB` write batching with the `setMaxPendingBlobWriteBytes()` and `getMaxPendingBlobWriteBytes()` methods.

- `--ndb-connectstring=`*`connection_string`*

Command-Line Format	`--ndb-connectstring`

Type	String

When using the `NDBCLUSTER` storage engine, this option specifies the management server that distributes cluster configuration data. See Section 18.3.3.3, “NDB Cluster Connection Strings”, for syntax.

- `--ndb-deferred-constraints=[0|1]`

Command-Line Format	`--ndb-deferred-constraints`
System Variable	`ndb_deferred_constraints`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1`

Controls whether or not constraint checks on unique indexes are deferred until commit time, where such checks are supported. `0` is the default.

This option is not normally needed for operation of NDB Cluster or NDB Cluster Replication, and is intended primarily for use in testing.

- `--ndb-distribution=[KEYHASH|LINHASH]`

Command-Line Format	`--ndb-distribution={KEYHASH|LINHASH}`
System Variable	`ndb_distribution`
Scope	Global
Dynamic	Yes
Type	Enumeration
Default Value	`KEYHASH`

Valid Values	`LINHASH`
	
	`KEYHASH`

Controls the default distribution method for `NDB` tables. Can be set to either of `KEYHASH` (key hashing) or `LINHASH` (linear hashing). `KEYHASH` is the default.

- `--ndb-log-apply-status`

Command-Line Format	`--ndb-log-apply-status[={OFF|ON}]`
System Variable	`ndb_log_apply_status`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`OFF`

Causes a replica `mysqld` to log any updates received from its immediate source to the `mysql.ndb_apply_status` table in its own binary log using its own server ID rather than the server ID of the source. In a circular or chain replication setting, this allows such updates to propagate to the `mysql.ndb_apply_status` tables of any MySQL servers configured as replicas of the current `mysqld`.

In a chain replication setup, using this option allows downstream (replica) clusters to be aware of their positions relative to all of their upstream contributors (sources).

In a circular replication setup, this option causes changes to `ndb_apply_status` tables to complete the entire circuit, eventually propagating back to the originating NDB Cluster. This also allows a cluster acting as a source to see when its changes (epochs) have been applied to the other clusters in the circle.

This option has no effect unless the MySQL server is started with the `--ndbcluster` option.

- `--ndb-log-empty-epochs=[ON|OFF]`

Command-Line Format	`--ndb-log-empty-epochs[={OFF|ON}]`
System Variable	`ndb_log_empty_epochs`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

Causes epochs during which there were no changes to be written to the `ndb_apply_status` and `ndb_binlog_index` tables, even when `log_slave_updates` is enabled.

By default this option is disabled. Disabling `--ndb-log-empty-epochs` causes epoch transactions with no changes not to be written to the binary log, although a row is still written even for an empty epoch in `ndb_binlog_index`.

Because `--ndb-log-empty-epochs=1` causes the size of the `ndb_binlog_index` table to increase independently of the size of the binary log, users should be prepared to manage the growth of this table, even if they expect the cluster to be idle a large part of the time.

- `--ndb-log-empty-update=[ON|OFF]`

Command-Line Format	`--ndb-log-empty-update[={OFF|ON}]`
System Variable	`ndb_log_empty_update`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

Causes updates that produced no changes to be written to the `ndb_apply_status` and `ndb_binlog_index` tables, when when `log_slave_updates` is enabled.

By default this option is disabled (`OFF`). Disabling `--ndb-log-empty-update` causes updates with no changes not to be written to the binary log.

- `--ndb-log-exclusive-reads=[0|1]`

Command-Line Format	`--ndb-log-exclusive-reads[={OFF|ON}]`
Introduced	5.6.20-ndb-7.4.1
System Variable	`ndb_log_exclusive_reads`
Scope	Global, Session
Dynamic	Yes
Type	Boolean

Default Value	`0`

In NDB 7.4.1 and later, starting the server with this option causes primary key reads to be logged with exclusive locks, which allows for NDB Cluster Replication conflict detection and resolution based on read conflicts. You can also enable and disable these locks at runtime by setting the value of the `ndb_log_exclusive_reads` system variable to 1 or 0, respectively. 0 (disable locking) is the default.

For more information, see Read conflict detection and resolution.

- `--ndb-log-fail-terminate`

Command-Line Format	`--ndb-log-fail-terminate`
Introduced	5.6.47-ndb-7.4.28
System Variable	`ndb_log_fail_terminate`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`FALSE`

When this option is specified, and complete logging of all found row events is not possible, the `mysqld` process is terminated.

- `--ndb-log-orig`

Command-Line Format	`--ndb-log-orig[={OFF|ON}]`
System Variable	`ndb_log_orig`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`OFF`

Log the originating server ID and epoch in the `ndb_binlog_index` table.

Note

This makes it possible for a given epoch to have multiple rows in `ndb_binlog_index`, one for each originating epoch.

For more information, see Section 18.6.4, “NDB Cluster Replication Schema and Tables”.

- `--ndb-log-transaction-id`

Command-Line Format	`--ndb-log-transaction-id[={OFF|ON}]`
System Variable	`ndb_log_transaction_id`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`OFF`

Causes a replica `mysqld` to write the NDB transaction ID in each row of the binary log. Such logging requires the use of the Version 2 event format for the binary log; thus, the `log_bin_use_v1_row_events` system variable must be disabled to use this option.

This option is not supported in mainline MySQL Server 5.6. It is required to enable NDB Cluster Replication conflict detection and resolution using the `NDB$EPOCH_TRANS()` function (see NDB $EPOCH_TRANS()).

The default value is `FALSE`.

For more information, see Section 18.6.11, “NDB Cluster Replication Conflict Resolution”.

- `--ndb-log-update-minimal`

Command-Line Format	`--ndb-log-update-minimal[={OFF|ON}]`
Introduced	5.6.36-ndb-7.4.16
System Variable	`ndb_log_update_minimal`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

Log updates in a minimal fashion, by writing only the primary key values in the before image, and only the changed columns in the after image. This may cause compatibility problems if replicating to storage engines other than `NDB`.

- `--ndb-mgmd-host=host[:port]`

Command-Line Format	`--ndb-mgmd-host=host_name[:port_num]`
Type	String
Default Value	`localhost:1186`

Can be used to set the host and port number of a single management server for the program to connect to. If the program requires node IDs or references to multiple management servers (or both) in its connection information, use the `--ndb-connectstring` option instead.

- `--ndb-nodeid=#`

Command-Line Format	`--ndb-nodeid=#`
Status Variable	`Ndb_cluster_node_id`
Scope	Global
Dynamic	No
Type	Integer
Minimum Value	`1`
Maximum Value	`255`
Maximum Value	`63`

Set this MySQL server's node ID in an NDB Cluster.

The `--ndb-nodeid` option overrides any node ID set with `--ndb-connectstring`, regardless of the order in which the two options are used.

In addition, if `--ndb-nodeid` is used, then either a matching node ID must be found in a `[mysqld]` or `[api]` section of `config.ini`, or there must be an “open” `[mysqld]` or `[api]` section in the

file (that is, a section without a `NodeId` or `Id` parameter specified). This is also true if the node ID is specified as part of the connection string.

Regardless of how the node ID is determined, its is shown as the value of the global status variable `Ndb_cluster_node_id` in the output of `SHOW STATUS`, and as `cluster_node_id` in the `connection` row of the output of `SHOW ENGINE NDBCLUSTER STATUS`.

For more information about node IDs for NDB Cluster SQL nodes, see Section 18.3.3.7, "Defining SQL and Other API Nodes in an NDB Cluster".

- `--ndb-optimization-delay=`*`milliseconds`*

Command-Line Format	`--ndb-optimization-delay=#`
System Variable	`ndb_optimization_delay`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`10`
Minimum Value	`0`
Maximum Value	`100000`

Set the number of milliseconds to wait between sets of rows by `OPTIMIZE TABLE` statements on `NDB` tables. The default is 10.

- `--ndb-transid-mysql-connection-map=`*`state`*

Command-Line Format	`--ndb-transid-mysql-connection-map[=state]`
Type	Enumeration
Default Value	`ON`

Valid Values	`ON`
	`OFF`
	`FORCE`

Enables or disables the plugin that handles the `ndb_transid_mysql_connection_map` table in the `INFORMATION_SCHEMA` database. Takes one of the values `ON`, `OFF`, or `FORCE`. `ON` (the default) enables the plugin. `OFF` disables the plugin, which makes `ndb_transid_mysql_connection_map` inaccessible. `FORCE` keeps the MySQL Server from starting if the plugin fails to load and start.

You can see whether the `ndb_transid_mysql_connection_map` table plugin is running by checking the output of `SHOW PLUGINS`.

- `--ndb-wait-connected=`*`seconds`*

Command-Line Format	`--ndb-wait-connected=#`
System Variable	`ndb_wait_connected`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`30`
Minimum Value	`0`

Maximum Value	31536000

This option sets the period of time that the MySQL server waits for connections to NDB Cluster management and data nodes to be established before accepting MySQL client connections. The time is specified in seconds. The default value is 30.

- --ndb-wait-setup=*seconds*

Command-Line Format	--ndb-wait-setup=#
System Variable	ndb_wait_setup
Scope	Global
Dynamic	No
Type	Integer
Default Value	30
Default Value	30
Default Value	15
Default Value	15
Minimum Value	0
Maximum Value	31536000

This variable shows the period of time that the MySQL server waits for the NDB storage engine to complete setup before timing out and treating NDB as unavailable. The time is specified in seconds. The default value is 30.

- --server-id-bits=#

Command-Line Format	--server-id-bits=#
System Variable	server_id_bits
Scope	Global
Dynamic	No
Type	Integer
Default Value	32
Minimum Value	7
Maximum Value	32

This option indicates the number of least significant bits within the 32-bit server_id which actually identify the server. Indicating that the server is actually identified by fewer than 32 bits makes it possible for some of the remaining bits to be used for other purposes, such as storing user data generated by applications using the NDB API's Event API within the AnyValue of an OperationOptions structure (NDB Cluster uses the AnyValue to store the server ID).

When extracting the effective server ID from server_id for purposes such as detection of replication loops, the server ignores the remaining bits. The --server-id-bits option is used to

mask out any irrelevant bits of `server_id` in the I/O and SQL threads when deciding whether an event should be ignored based on the server ID.

This data can be read from the binary log by `mysqlbinlog`, provided that it is run with its own `--server-id-bits` option set to 32 (the default).

The value of `server_id` must be less than 2 ^ `server_id_bits`; otherwise, `mysqld` refuses to start.

This system variable is supported only by NDB Cluster. It is not supported in the standard MySQL 5.6 Server.

- `--skip-ndbcluster`

Command-Line Format	`--skip-ndbcluster`

Disable the `NDBCLUSTER` storage engine. This is the default for binaries that were built with `NDBCLUSTER` storage engine support; the server allocates memory and other resources for this storage engine only if the `--ndbcluster` option is given explicitly. See Section 18.3.1, "Quick Test Setup of NDB Cluster", for an example.

NDB Cluster System Variables

This section provides detailed information about MySQL server system variables that are specific to NDB Cluster and the `NDB` storage engine. For system variables not specific to NDB Cluster, see Section 5.1.7, "Server System Variables". For general information on using system variables, see Section 5.1.8, "Using System Variables".

- `create_old_temporals`

Command-Line Format	`--create-old-temporals[={OFF|ON}]`
Introduced	5.6.25-ndb-7.4.7
Deprecated	5.6.25-ndb-7.4.7
System Variable	`create_old_temporals`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`OFF`

Causes `mysqld` to use the storage formats for temporal data types that were used in the MySQL server prior to MySQL 5.6.4; that is, `TIME`, `DATETIME`, and `TIMESTAMP` columns are created without support for fractional seconds. This affects all `CREATE TABLE` and `ALTER TABLE` statements.

The `create_old_temporals` system variable is read-only, with a default value of `false`; to enable it, use the `--create-old-temporals` option on the command line or in the server configuration file.

Important

`avoid_temporal_upgrade` must also be enabled for this feature to work properly. It is also strongly recommended that you enable `show_old_temporals` as well. See the descriptions of these variables for more information, as well as Date and Time Type Storage Requirements.

This variable was added in NDB 7.3.10 and NDB 7.4.7; it is specific to NDB Cluster and is not available in standard MySQL Server releases. It was intended to facilitate upgrades from older

versions of NDB Cluster to NDB 7.3 and 7.4; it is now deprecated and scheduled for removal in a future version of NDB Cluster.

- `ndb_autoincrement_prefetch_sz`

Command-Line Format	`--ndb-autoincrement-prefetch-sz=#`
System Variable	`ndb_autoincrement_prefetch_sz`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`1`
Minimum Value	`1`
Maximum Value	`65536`

Determines the probability of gaps in an autoincremented column. Set it to `1` to minimize this. Setting it to a high value for optimization makes inserts faster, but decreases the likelihood that consecutive autoincrement numbers are used in a batch of inserts.

This variable affects only the number of `AUTO_INCREMENT` IDs that are fetched between statements; within a given statement, at least 32 IDs are obtained at a time.

Important

This variable does not affect inserts performed using `INSERT ... SELECT`.

- `ndb_cache_check_time`

Command-Line Format	`--ndb-cache-check-time=#`
System Variable	`ndb_cache_check_time`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`0`

The number of milliseconds that elapse between checks of NDB Cluster SQL nodes by the MySQL query cache. Setting this to 0 (the default and minimum value) means that the query cache checks for validation on every query.

The recommended maximum value for this variable is 1000, which means that the check is performed once per second. A larger value means that the check is performed and possibly invalidated due to updates on different SQL nodes less often. It is generally not desirable to set this to a value greater than 2000.

- `ndb_clear_apply_status`

Command-Line Format	`--ndb-clear-apply-status[={OFF|ON}]`
Introduced	5.6.28-ndb-7.4.9
System Variable	`ndb_clear_apply_status`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`ON`

By the default, executing `RESET SLAVE` causes an NDB Cluster replica to purge all rows from its `ndb_apply_status` table. In NDB 7.4.9 and later you can disable this by setting `ndb_clear_apply_status=OFF`.

- `ndb_deferred_constraints`

Command-Line Format	`--ndb-deferred-constraints=#`
System Variable	`ndb_deferred_constraints`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`1`

Controls whether or not constraint checks are deferred, where these are supported. `0` is the default.

This variable is not normally needed for operation of NDB Cluster or NDB Cluster Replication, and is intended primarily for use in testing.

- `ndb_distribution`

Command-Line Format	`--ndb-distribution={KEYHASH|LINHASH}`
System Variable	`ndb_distribution`
Scope	Global
Dynamic	Yes
Type	Enumeration
Default Value	`KEYHASH`

Valid Values	`LINHASH`
	
	`KEYHASH`

Controls the default distribution method for `NDB` tables. Can be set to either of `KEYHASH` (key hashing) or `LINHASH` (linear hashing). `KEYHASH` is the default.

- `ndb_eventbuffer_free_percent`

Command-Line Format	`--ndb-eventbuffer-free-percent=#`
Introduced	5.6.22-ndb-7.4.3
System Variable	`ndb_eventbuffer_free_percent`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`20`
Minimum Value	`1`

Maximum Value	99

Sets the percentage of the maximum memory allocated to the event buffer (ndb_eventbuffer_max_alloc) that should be available in event buffer after reaching the maximum, before starting to buffer again.

ndb_eventbuffer_free_percent was added in NDB 7.4.3.

- ndb_eventbuffer_max_alloc

Command-Line Format	--ndb-eventbuffer-max-alloc=#
Introduced	5.6.14-ndb-7.3.3
System Variable	ndb_eventbuffer_max_alloc
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	0
Minimum Value	0
Maximum Value	4294967295

Sets the maximum amount memory (in bytes) that can be allocated for buffering events by the NDB API. 0 means that no limit is imposed, and is the default.

This variable was added in NDB 7.3.3.

- ndb_extra_logging

Command-Line Format	ndb_extra_logging=#
System Variable	ndb_extra_logging
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	1

This variable enables recording in the MySQL error log of information specific to the NDB storage engine.

When this variable is set to 0, the only information specific to NDB that is written to the MySQL error log relates to transaction handling. If it set to a value greater than 0 but less than 10, NDB table schema and connection events are also logged, as well as whether or not conflict resolution is in use, and other NDB errors and information. If the value is set to 10 or more, information about NDB internals, such as the progress of data distribution among cluster nodes, is also written to the MySQL error log. The default is 1.

- ndb_force_send

Command-Line Format	--ndb-force-send[={OFF|ON}]
System Variable	ndb_force_send
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	ON

Forces sending of buffers to NDB immediately, without waiting for other threads. Defaults to ON.

- ndb_index_stat_cache_entries

Command-Line Format	--ndb-index-stat-cache-entries=#
Deprecated	Yes (removed in 5.6.16-ndb-7.3.5)
System Variable	ndb_index_stat_cache_entries
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	32
Minimum Value	0
Maximum Value	4294967295

Sets the granularity of the statistics by determining the number of starting and ending keys to store in the statistics memory cache. Zero means no caching takes place; in this case, the data nodes are always queried directly. Default value: 32.

Note

If ndb_index_stat_enable is OFF, then setting this variable has no effect.

This variable was deprecated in MySQL 5.1, and is removed from NDB 7.3.5 and later.

- ndb_index_stat_enable

Command-Line Format	--ndb-index-stat-enable[={OFF|ON}]
System Variable	ndb_index_stat_enable
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	ON

Use NDB index statistics in query optimization. The default is ON.

- ndb_index_stat_option

Command-Line Format	--ndb-index-stat-option=value
System Variable	ndb_index_stat_option
Scope	Global, Session
Dynamic	Yes
Type	String
Default Value	loop_checkon=1000ms,loop_idle=1000ms,loop_busy=100ms,update_batch=1,read_batch=4,idle_batch=32,check_batch=32,check_delay=1m,delete_batch=8,clean_delay=0,error_batch=4,error_delay=1m,evict_batch=8,evict_delay=1m,cache_limit=32M,cache_lowpct=90

This variable is used for providing tuning options for NDB index statistics generation. The list consist of comma-separated name-value pairs of option names and values, and this list must not contain any space characters.

Options not used when setting `ndb_index_stat_option` are not changed from their default values. For example, you can set `ndb_index_stat_option = 'loop_idle=1000ms,cache_limit=32M'`.

Time values can be optionally suffixed with `h` (hours), `m` (minutes), or `s` (seconds). Millisecond values can optionally be specified using `ms`; millisecond values cannot be specified using `h`, `m`, or `s`.) Integer values can be suffixed with `K`, `M`, or `G`.

The names of the options that can be set using this variable are shown in the table that follows. The table also provides brief descriptions of the options, their default values, and (where applicable) their minimum and maximum values.

Table 18.14 ndb_index_stat_option options and values

Name	Description	Default/Units	Minimum/Maximum
`loop_enable`		1000 ms	0/4G
`loop_idle`	Time to sleep when idle	1000 ms	0/4G
`loop_busy`	Time to sleep when more work is waiting	100 ms	0/4G
`update_batch`		1	0/4G
`read_batch`		4	1/4G
`idle_batch`		32	1/4G
`check_batch`		8	1/4G
`check_delay`	How often to check for new statistics	10 m	1/4G
`delete_batch`		8	0/4G
`clean_delay`		1 m	0/4G
`error_batch`		4	1/4G
`error_delay`		1 m	1/4G
`evict_batch`		8	1/4G
`evict_delay`	Clean LRU cache, from read time	1 m	0/4G
`cache_limit`	Maximum amount of memory in bytes used for cached index statistics by this `mysqld`; clean up the cache when this is exceeded.	32 M	0/4G
`cache_lowpct`		90	0/100
`zero_total`	Setting this to 1 resets all accumulating counters in `ndb_index_stat_status` to 0. This option value is also reset to 0 when this is done.	0	0/1

- `ndb_index_stat_update_freq`

Command-Line Format	`--ndb-index-stat-update-freq=#`

Deprecated	Yes (removed in 5.6.16-ndb-7.3.5)
System Variable	`ndb_index_stat_update_freq`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`20`
Minimum Value	`0`
Maximum Value	`4294967295`

How often to query data nodes instead of the statistics cache. For example, a value of `20` (the default) means to direct every 20th query to the data nodes.

Note

If `ndb_index_stat_cache_entries` is `0`, then setting this variable has no effect; in this case, every query is sent directly to the data nodes.

This variable was deprecated in MySQL 5.1, and is removed from NDB 7.3.5 and later.

- `ndb_join_pushdown`

System Variable	`ndb_join_pushdown`
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	`ON`

This variable controls whether joins on `NDB` tables are pushed down to the NDB kernel (data nodes). Previously, a join was handled using multiple accesses of `NDB` by the SQL node; however, when `ndb_join_pushdown` is enabled, a pushable join is sent in its entirety to the data nodes, where it can be distributed among the data nodes and executed in parallel on multiple copies of the data, with a single, merged result being returned to `mysqld`. This can reduce greatly the number of round trips between an SQL node and the data nodes required to handle such a join.

By default, `ndb_join_pushdown` is enabled.

Conditions for NDB pushdown joins. In order for a join to be pushable, it must meet the following conditions:

1. Only columns can be compared, and all columns to be joined must use *exactly* the same data type.

   This means that expressions such as `t1.a = t2.a + constant` cannot be pushed down, and that (for example) a join on an `INT` column and a `BIGINT` column also cannot be pushed down.

2. Queries referencing `BLOB` or `TEXT` columns are not supported.

3. Explicit locking is not supported; however, the `NDB` storage engine's characteristic implicit row-based locking is enforced.

   This means that a join using `FOR UPDATE` cannot be pushed down.

4. In order for a join to be pushed down, child tables in the join must be accessed using one of the `ref`, `eq_ref`, or `const` access methods, or some combination of these methods.

   Outer joined child tables can only be pushed using `eq_ref`.

If the root of the pushed join is an `eq_ref` or `const`, only child tables joined by `eq_ref` can be appended. (A table joined by `ref` is likely to become the root of another pushed join.)

If the query optimizer decides on `Using join cache` for a candidate child table, that table cannot be pushed as a child. However, it may be the root of another set of pushed tables.

5. Joins referencing tables explicitly partitioned by `[LINEAR] HASH`, `LIST`, or `RANGE` currently cannot be pushed down.

You can see whether a given join can be pushed down by checking it with `EXPLAIN`; when the join can be pushed down, you can see references to the `pushed join` in the `Extra` column of the output, as shown in this example:

```
mysql> EXPLAIN
    ->     SELECT e.first_name, e.last_name, t.title, d.dept_name
    ->         FROM employees e
    ->         JOIN dept_emp de ON e.emp_no=de.emp_no
    ->         JOIN departments d ON d.dept_no=de.dept_no
    ->         JOIN titles t ON e.emp_no=t.emp_no\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: d
         type: ALL
possible_keys: PRIMARY
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 9
        Extra: Parent of 4 pushed join@1
*************************** 2. row ***************************
           id: 1
  select_type: SIMPLE
        table: de
         type: ref
possible_keys: PRIMARY,emp_no,dept_no
          key: dept_no
      key_len: 4
          ref: employees.d.dept_no
         rows: 5305
        Extra: Child of 'd' in pushed join@1
*************************** 3. row ***************************
           id: 1
  select_type: SIMPLE
        table: e
         type: eq_ref
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: employees.de.emp_no
         rows: 1
        Extra: Child of 'de' in pushed join@1
*************************** 4. row ***************************
           id: 1
  select_type: SIMPLE
        table: t
         type: ref
possible_keys: PRIMARY,emp_no
          key: emp_no
      key_len: 4
          ref: employees.de.emp_no
         rows: 19
        Extra: Child of 'e' in pushed join@1
```

```
4 rows in set (0.00 sec)
```

Note

If inner joined child tables are joined by `ref`, *and* the result is ordered or grouped by a sorted index, this index cannot provide sorted rows, which forces writing to a sorted tempfile.

Two additional sources of information about pushed join performance are available:

1. The status variables `Ndb_pushed_queries_defined`, `Ndb_pushed_queries_dropped`, `Ndb_pushed_queries_executed`, and `Ndb_pushed_reads`.
2. The counters in the `ndbinfo.counters` table that belong to the `DBSPJ` kernel block.

- `ndb_log_apply_status`

Command-Line Format	`--ndb-log-apply-status[={OFF|ON}]`
System Variable	`ndb_log_apply_status`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`OFF`

A read-only variable which shows whether the server was started with the `--ndb-log-apply-status` option.

- `ndb_log_bin`

Command-Line Format	`--ndb-log-bin[={OFF|ON}]`
System Variable	`ndb_log_bin`
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	`ON`

Causes updates to `NDB` tables to be written to the binary log. Setting this variable has no effect if binary logging is not already enabled for the server using `log_bin`. `ndb_log_bin` defaults to 1 (ON); normally, there is never any need to change this value in a production environment.

- `ndb_log_binlog_index`

Command-Line Format	`--ndb-log-binlog-index[={OFF|ON}]`
System Variable	`ndb_log_binlog_index`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`ON`

Causes a mapping of epochs to positions in the binary log to be inserted into the `ndb_binlog_index` table. Setting this variable has no effect if binary logging is not already enabled for the server using `log_bin`. (In addition, `ndb_log_bin` must not be disabled.) `ndb_log_binlog_index` defaults to `1` (`ON`); normally, there is never any need to change this value in a production environment.

- `ndb_log_empty_epochs`

Command-Line Format	`--ndb-log-empty-epochs[={OFF|ON}]`
System Variable	`ndb_log_empty_epochs`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

When this variable is set to 0, epoch transactions with no changes are not written to the binary log, although a row is still written even for an empty epoch in `ndb_binlog_index`.

- `ndb_log_empty_update`

Command-Line Format	`--ndb-log-empty-update[={OFF|ON}]`
System Variable	`ndb_log_empty_update`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

When this variable is set to `ON` (`1`), update transactions with no changes are written to the binary log, even when `log_slave_updates` is enabled.

- `ndb_log_exclusive_reads`

Command-Line Format	`--ndb-log-exclusive-reads[={OFF|ON}]`
Introduced	5.6.20-ndb-7.4.1
System Variable	`ndb_log_exclusive_reads`
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	`0`

In NDB 7.4.1 and later, this variable determines whether primary key reads are logged with exclusive locks, which allows for NDB Cluster Replication conflict detection and resolution based on read conflicts. To enable these locks, set the value of `ndb_log_exclusive_reads` to 1. 0, which disables such locking, is the default.

For more information, see Read conflict detection and resolution.

- `ndb_log_orig`

Command-Line Format	`--ndb-log-orig[={OFF|ON}]`
System Variable	`ndb_log_orig`
Scope	Global
Dynamic	No
Type	Boolean

Default Value	OFF

Shows whether the originating server ID and epoch are logged in the ndb_binlog_index table. Set using the --ndb-log-orig server option.

- ndb_log_transaction_id

System Variable	ndb_log_transaction_id
Scope	Global
Dynamic	No
Type	Boolean
Default Value	OFF

This read-only, Boolean system variable shows whether a replica mysqld writes NDB transaction IDs in the binary log (required to use “active-active” NDB Cluster Replication with NDB $EPOCH_TRANS() conflict detection). To change the setting, use the --ndb-log-transaction-id option.

ndb_log_transaction_id is not supported in mainline MySQL Server 5.6.

For more information, see Section 18.6.11, “NDB Cluster Replication Conflict Resolution”.

- ndb_optimized_node_selection

Command-Line Format	--ndb-optimized-node-selection=#
System Variable	ndb_optimized_node_selection
Scope	Global
Dynamic	No
Type	Integer
Default Value	3
Minimum Value	0
Maximum Value	3

There are two forms of optimized node selection, described here:

1. The SQL node uses *promixity* to determine the transaction coordinator; that is, the “closest” data node to the SQL node is chosen as the transaction coordinator. For this purpose, a data node having a shared memory connection with the SQL node is considered to be “closest” to the SQL node; the next closest (in order of decreasing proximity) are: TCP connection to localhost, followed by TCP connection from a host other than localhost.

2. The SQL thread uses *distribution awareness* to select the data node. That is, the data node housing the cluster partition accessed by the first statement of a given transaction is used as the transaction coordinator for the entire transaction. (This is effective only if the first statement of the transaction accesses no more than one cluster partition.)

This option takes one of the integer values 0, 1, 2, or 3. 3 is the default. These values affect node selection as follows:

- 0: Node selection is not optimized. Each data node is employed as the transaction coordinator 8 times before the SQL thread proceeds to the next data node.

- 1: Proximity to the SQL node is used to determine the transaction coordinator.

- 2: Distribution awareness is used to select the transaction coordinator. However, if the first statement of the transaction accesses more than one cluster partition, the SQL node reverts to the round-robin behavior seen when this option is set to 0.
- 3: If distribution awareness can be employed to determine the transaction coordinator, then it is used; otherwise proximity is used to select the transaction coordinator. (This is the default behavior.)

Proximity is determined as follows:

1. Start with the value set for the Group parameter (default 55).
2. For an API node sharing the same host with other API nodes, decrement the value by 1. Assuming the default value for Group, the effective value for data nodes on same host as the API node is 54, and for remote data nodes 55.

- ndb_recv_thread_activation_threshold

Command-Line Format	--ndb-recv-thread-activation-threshold=#
Introduced	5.6.10-ndb-7.3.1
System Variable	ndb_recv_thread_activation_threshold
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	8
Minimum Value	0 (MIN_ACTIVATION_THRESHOLD)
Maximum Value	16 (MAX_ACTIVATION_THRESHOLD)

When this number of concurrently active threads is reached, the receive thread takes over polling of the cluster connection.

This variable is global in scope. It can also be set at startup.

- ndb_recv_thread_cpu_mask

Command-Line Format	--ndb-recv-thread-cpu-mask=mask
Introduced	5.6.10-ndb-7.3.1
System Variable	ndb_recv_thread_cpu_mask
Scope	Global
Dynamic	Yes
Type	Bitmap
Default Value	[empty]

CPU mask for locking receiver threads to specific CPUs. This is specified as a hexadecimal bitmask. For example, 0x33 means that one CPU is used per receiver thread. An empty string is the default; setting ndb_recv_thread_cpu_mask to this value removes any receiver thread locks previously set.

This variable is global in scope. It can also be set at startup.

- ndb_report_thresh_binlog_epoch_slip

Command-Line Format	--ndb-report-thresh-binlog-epoch-slip=#

System Variable	ndb_report_thresh_binlog_epoch_slip
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	3
Minimum Value	0
Maximum Value	256

This is a threshold on the number of epochs to be behind before reporting binary log status. For example, a value of 3 (the default) means that if the difference between which epoch has been received from the storage nodes and which epoch has been applied to the binary log is 3 or more, a status message is sent to the cluster log.

- ndb_report_thresh_binlog_mem_usage

Command-Line Format	--ndb-report-thresh-binlog-mem-usage=#
System Variable	ndb_report_thresh_binlog_mem_usage
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	10
Minimum Value	0
Maximum Value	10

This is a threshold on the percentage of free memory remaining before reporting binary log status. For example, a value of 10 (the default) means that if the amount of available memory for receiving binary log data from the data nodes falls below 10%, a status message is sent to the cluster log.

- ndb_show_foreign_key_mock_tables

Command-Line Format	--ndb-show-foreign-key-mock-tables[={OFF|ON}]
Introduced	5.6.11-ndb-7.3.2
System Variable	ndb_show_foreign_key_mock_tables
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	OFF

Show the mock tables used by NDB to support foreign_key_checks=0. When this is enabled, extra warnings are shown when creating and dropping the tables. The real (internal) name of the table can be seen in the output of SHOW CREATE TABLE.

- ndb_slave_conflict_role

Command-Line Format	--ndb-slave-conflict-role=value
Introduced	5.6.20-ndb-7.4.1
System Variable	ndb_slave_conflict_role
Scope	Global
Dynamic	Yes

Type	Enumeration
Default Value	NONE
Valid Values	NONE PRIMARY SECONDARY PASS

Determine the role of this SQL node (and NDB Cluster) in a circular (“active-active”) replication setup. ndb_slave_conflict_role can take any one of the values PRIMARY, SECONDARY, PASS, or NULL (the default). The replica SQL thread must be stopped before you can change ndb_slave_conflict_role. In addition, it is not possible to change directly between PASS and either of PRIMARY or SECONDARY directly; in such cases, you must ensure that the SQL thread is stopped, then execute SET @@GLOBAL.ndb_slave_conflict_role = 'NONE' first.

This variable was added in NDB 7.4.1. For more information, see Section 18.6.11, “NDB Cluster Replication Conflict Resolution”.

- ndb_table_no_logging

System Variable	ndb_table_no_logging
Scope	Session
Dynamic	Yes
Type	Boolean
Default Value	OFF

When this variable is set to ON or 1, it causes NDB tables not to be checkpointed to disk. More specifically, this setting applies to tables which are created or altered using ENGINE NDB when ndb_table_no_logging is enabled, and continues to apply for the lifetime of the table, even if ndb_table_no_logging is later changed. Suppose that A, B, C, and D are tables that we create (and perhaps also alter), and that we also change the setting for ndb_table_no_logging as shown here:

```
SET @@ndb_table_no_logging = 1;

CREATE TABLE A ... ENGINE NDB;

CREATE TABLE B ... ENGINE MYISAM;
CREATE TABLE C ... ENGINE MYISAM;

ALTER TABLE B ENGINE NDB;

SET @@ndb_table_no_logging = 0;

CREATE TABLE D ... ENGINE NDB;
ALTER TABLE C ENGINE NDB;

SET @@ndb_table_no_logging = 1;
```

After the previous sequence of events, tables A and B are not checkpointed; A was created with ENGINE NDB and B was altered to use NDB, both while ndb_table_no_logging was enabled. However, tables C and D are logged; C was altered to use NDB and D was created using ENGINE

NDB, both while ndb_table_no_logging was disabled. Setting ndb_table_no_logging back to 1 or ON does *not* cause table C or D to be checkpointed.

Note

ndb_table_no_logging has no effect on the creation of NDB table schema files; to suppress these, use ndb_table_temporary instead.

- ndb_table_temporary

System Variable	ndb_table_temporary
Scope	Session
Dynamic	Yes
Type	Boolean
Default Value	OFF

When set to ON or 1, this variable causes NDB tables not to be written to disk: This means that no table schema files are created, and that the tables are not logged.

Note

Setting this variable currently has no effect in NDB Cluster 7.0 and later. This is a known issue; see Bug #34036.

- ndb_use_copying_alter_table

System Variable	ndb_use_copying_alter_table
Scope	Global, Session
Dynamic	No

Forces NDB to use copying of tables in the event of problems with online ALTER TABLE operations. The default value is OFF.

- ndb_use_exact_count

System Variable	ndb_use_exact_count
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	OFF

Forces NDB to use a count of records during SELECT COUNT(*) query planning to speed up this type of query. The default value is OFF, which allows for faster queries overall.

- ndb_use_transactions

Command-Line Format	--ndb-use-transactions[={OFF|ON}]
System Variable	ndb_use_transactions
Scope	Global, Session
Dynamic	Yes
Type	Boolean

Default Value	ON

You can disable NDB transaction support by setting this variable's values to OFF (not recommended). The default is ON.

- ndb_version

System Variable	ndb_version
Scope	Global
Dynamic	No
Type	String
Default Value	

NDB engine version, as a composite integer.

- ndb_version_string

System Variable	ndb_version_string
Scope	Global
Dynamic	No
Type	String
Default Value	

NDB engine version in ndb-*x.y.z* format.

- server_id_bits

Command-Line Format	--server-id-bits=#
System Variable	server_id_bits
Scope	Global
Dynamic	No
Type	Integer
Default Value	32
Minimum Value	7
Maximum Value	32

The effective value of server_id if the server was started with the --server-id-bits option set to a nondefault value.

If the value of server_id greater than or equal to 2 to the power of server_id_bits, mysqld refuses to start.

This system variable is supported only by NDB Cluster. server_id_bits is not supported by the standard MySQL Server.

- slave_allow_batching

Command-Line Format	--slave-allow-batching[={OFF|ON}]
System Variable	slave_allow_batching
Scope	Global
Dynamic	Yes
Type	Boolean

Default Value	OFF

Whether or not batched updates are enabled on NDB Cluster replicas.

Setting this variable has an effect only when using replication with the NDB storage engine; in MySQL Server 5.6, it is present but does nothing. For more information, see Section 18.6.6, “Starting NDB Cluster Replication (Single Replication Channel)”.

- transaction_allow_batching

System Variable	transaction_allow_batching
Scope	Session
Dynamic	Yes
Type	Boolean
Default Value	OFF

When set to 1 or ON, this variable enables batching of statements within the same transaction. To use this variable, autocommit must first be disabled by setting it to 0 or OFF; otherwise, setting transaction_allow_batching has no effect.

It is safe to use this variable with transactions that performs writes only, as having it enabled can lead to reads from the “before” image. You should ensure that any pending transactions are committed (using an explicit COMMIT if desired) before issuing a SELECT.

Important

transaction_allow_batching should not be used whenever there is the possibility that the effects of a given statement depend on the outcome of a previous statement within the same transaction.

This variable is currently supported for NDB Cluster only.

The system variables in the following list all relate to the ndbinfo information database.

- ndbinfo_database

System Variable	ndbinfo_database
Scope	Global
Dynamic	No
Type	String
Default Value	ndbinfo

Shows the name used for the NDB information database; the default is ndbinfo. This is a read-only variable whose value is determined at compile time.

- ndbinfo_max_bytes

Command-Line Format	--ndbinfo-max-bytes=#
System Variable	ndbinfo_max_bytes
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	0

Used in testing and debugging only.

- `ndbinfo_max_rows`

Command-Line Format	`--ndbinfo-max-rows=#`
System Variable	`ndbinfo_max_rows`
Scope	Global, Session
Dynamic	Yes
Type	Integer
Default Value	`10`

Used in testing and debugging only.

- `ndbinfo_offline`

System Variable	`ndbinfo_offline`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`OFF`

Place the `ndbinfo` database into offline mode, in which tables and views can be opened even when they do not actually exist, or when they exist but have different definitions in `NDB`. No rows are returned from such tables (or views).

- `ndbinfo_show_hidden`

Command-Line Format	`--ndbinfo-show-hidden[={OFF|ON}]`
System Variable	`ndbinfo_show_hidden`
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	`OFF`
Valid Values	`ON` `OFF`

Whether or not the `ndbinfo` database's underlying internal tables are shown in the `mysql` client. The default is `OFF`.

- `ndbinfo_table_prefix`

System Variable	`ndbinfo_table_prefix`
Scope	Global
Dynamic	No
Type	String
Default Value	`ndb$`

The prefix used in naming the ndbinfo database's base tables (normally hidden, unless exposed by setting `ndbinfo_show_hidden`). This is a read-only variable whose default value is `ndb$`; the prefix itself is determined at compile time.

- `ndbinfo_version`

System Variable	`ndbinfo_version`
Scope	Global
Dynamic	No
Type	String
Default Value	

Shows the version of the `ndbinfo` engine in use; read-only.

NDB Cluster Status Variables

This section provides detailed information about MySQL server status variables that relate to NDB Cluster and the `NDB` storage engine. For status variables not specific to NDB Cluster, and for general information on using status variables, see Section 5.1.9, “Server Status Variables”.

- `Handler_discover`

  The MySQL server can ask the `NDBCLUSTER` storage engine if it knows about a table with a given name. This is called discovery. `Handler_discover` indicates the number of times that tables have been discovered using this mechanism.

- `Ndb_api_adaptive_send_deferred_count`

  Number of adaptive send calls that were not actually sent.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_adaptive_send_deferred_count_session`

  Number of adaptive send calls that were not actually sent.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_adaptive_send_deferred_count_slave`

  Number of adaptive send calls that were not actually sent by this replica.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_adaptive_send_forced_count`

  Number of adaptive send calls using forced-send sent by this MySQL Server (SQL node).

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_adaptive_send_forced_count_session`

  Number of adaptive send calls using forced-send sent in this client session.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_adaptive_send_forced_count_slave`

  Number of adaptive send calls using forced-send sent by this replica.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_adaptive_send_unforced_count`

  Number of adaptive send calls without forced-send sent by this MySQL server (SQL node).

For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_adaptive_send_unforced_count_session`

  Number of adaptive send calls without forced-send sent in this client session.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_adaptive_send_unforced_count_slave`

  Number of adaptive send calls without forced-send sent by this replica.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_bytes_sent_count_session`

  Amount of data (in bytes) sent to the data nodes in this client session.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_bytes_sent_count_slave`

  Amount of data (in bytes) sent to the data nodes by this replica.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_bytes_sent_count`

  Amount of data (in bytes) sent to the data nodes by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_bytes_received_count_session`

  Amount of data (in bytes) received from the data nodes in this client session.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_bytes_received_count_slave`

  Amount of data (in bytes) received from the data nodes by this replica.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_bytes_received_count`

Amount of data (in bytes) received from the data nodes by this MySQL Server (SQL node).

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_event_data_count_injector`

  The number of row change events received by the NDB binlog injector thread.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_event_data_count`

  The number of row change events received by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_event_nondata_count_injector`

  The number of events received, other than row change events, by the NDB binary log injector thread.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_event_nondata_count`

  The number of events received, other than row change events, by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_event_bytes_count_injector`

  The number of bytes of events received by the NDB binlog injector thread.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_event_bytes_count`

  The number of bytes of events received by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_pk_op_count_session`

The number of operations in this client session based on or using primary keys. This includes operations on blob tables, implicit unlock operations, and auto-increment operations, as well as user-visible primary key operations.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_pk_op_count_slave`

  The number of operations by this replica based on or using primary keys. This includes operations on blob tables, implicit unlock operations, and auto-increment operations, as well as user-visible primary key operations.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_pk_op_count`

  The number of operations by this MySQL Server (SQL node) based on or using primary keys. This includes operations on blob tables, implicit unlock operations, and auto-increment operations, as well as user-visible primary key operations.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_pruned_scan_count_session`

  The number of scans in this client session that have been pruned to a single partition.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_pruned_scan_count_slave`

  The number of scans by this replica that have been pruned to a single partition.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_pruned_scan_count`

  The number of scans by this MySQL Server (SQL node) that have been pruned to a single partition.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_range_scan_count_session`

  The number of range scans that have been started in this client session.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_range_scan_count_slave`

  The number of range scans that have been started by this replica.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_range_scan_count`

  The number of range scans that have been started by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_read_row_count_session`

  The total number of rows that have been read in this client session. This includes all rows read by any primary key, unique key, or scan operation made in this client session.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_read_row_count_slave`

  The total number of rows that have been read by this replica. This includes all rows read by any primary key, unique key, or scan operation made by this replica.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_read_row_count`

  The total number of rows that have been read by this MySQL Server (SQL node). This includes all rows read by any primary key, unique key, or scan operation made by this MySQL Server (SQL node).

  You should be aware that this value may not be completely accurate with regard to rows read by `SELECT COUNT(*)` queries, due to the fact that, in this case, the MySQL server actually reads pseudo-rows in the form `[table fragment ID]:[number of rows in fragment]` and

sums the rows per fragment for all fragments in the table to derive an estimated count for all rows. `Ndb_api_read_row_count` uses this estimate and not the actual number of rows in the table.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_scan_batch_count_session`

  The number of batches of rows received in this client session. 1 batch is defined as 1 set of scan results from a single fragment.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_scan_batch_count_slave`

  The number of batches of rows received by this replica. 1 batch is defined as 1 set of scan results from a single fragment.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_scan_batch_count`

  The number of batches of rows received by this MySQL Server (SQL node). 1 batch is defined as 1 set of scan results from a single fragment.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_table_scan_count_session`

  The number of table scans that have been started in this client session, including scans of internal tables,.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_table_scan_count_slave`

  The number of table scans that have been started by this replica, including scans of internal tables,.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_table_scan_count`

  The number of table scans that have been started by this MySQL Server (SQL node), including scans of internal tables,.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_abort_count_session`

  The number of transactions aborted in this client session.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_abort_count_slave`

  The number of transactions aborted by this replica.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_abort_count`

  The number of transactions aborted by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_close_count_session`

  The number of transactions closed in this client session. This value may be greater than the sum of `Ndb_api_trans_commit_count_session` and `Ndb_api_trans_abort_count_session`, since some transactions may have been rolled back.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_close_count_slave`

  The number of transactions closed by this replica. This value may be greater than the sum of `Ndb_api_trans_commit_count_slave` and `Ndb_api_trans_abort_count_slave`, since some transactions may have been rolled back.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_close_count`

  The number of transactions closed by this MySQL Server (SQL node). This value may be greater than the sum of `Ndb_api_trans_commit_count` and `Ndb_api_trans_abort_count`, since some transactions may have been rolled back.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_commit_count_session`

  The number of transactions committed in this client session.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_commit_count_slave`

  The number of transactions committed by this replica.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_commit_count`

  The number of transactions committed by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_local_read_row_count_session`

  The total number of rows that have been read in this client session. This includes all rows read by any primary key, unique key, or scan operation made in this client session.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_local_read_row_count_slave`

  The total number of rows that have been read by this replica. This includes all rows read by any primary key, unique key, or scan operation made by this replica.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_local_read_row_count`

  The total number of rows that have been read by this MySQL Server (SQL node). This includes all rows read by any primary key, unique key, or scan operation made by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_start_count_session`

  The number of transactions started in this client session.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_start_count_slave`

  The number of transactions started by this replica.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_trans_start_count`

  The number of transactions started by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_uk_op_count_session`

  The number of operations in this client session based on or using unique keys.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_uk_op_count_slave`

  The number of operations by this replica based on or using unique keys.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_uk_op_count`

  The number of operations by this MySQL Server (SQL node) based on or using unique keys.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_exec_complete_count_session`

  The number of times a thread has been blocked in this client session while waiting for execution of an operation to complete. This includes all `execute()` calls as well as implicit executes for blob and auto-increment operations not visible to clients.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_exec_complete_count_slave`

  The number of times a thread has been blocked by this replica while waiting for execution of an operation to complete. This includes all `execute()` calls as well as implicit executes for blob and auto-increment operations not visible to clients.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_exec_complete_count`

  The number of times a thread has been blocked by this MySQL Server (SQL node) while waiting for execution of an operation to complete. This includes all `execute()` calls as well as implicit executes for blob and auto-increment operations not visible to clients.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_meta_request_count_session`

  The number of times a thread has been blocked in this client session waiting for a metadata-based signal, such as is expected for DDL requests, new epochs, and seizure of transaction records.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_meta_request_count_slave`

  The number of times a thread has been blocked by this replica waiting for a metadata-based signal, such as is expected for DDL requests, new epochs, and seizure of transaction records.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_meta_request_count`

  The number of times a thread has been blocked by this MySQL Server (SQL node) waiting for a metadata-based signal, such as is expected for DDL requests, new epochs, and seizure of transaction records.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_nanos_count_session`

  Total time (in nanoseconds) spent in this client session waiting for any type of signal from the data nodes.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_nanos_count_slave`

  Total time (in nanoseconds) spent by this replica waiting for any type of signal from the data nodes.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_nanos_count`

  Total time (in nanoseconds) spent by this MySQL Server (SQL node) waiting for any type of signal from the data nodes.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_scan_result_count_session`

  The number of times a thread has been blocked in this client session while waiting for a scan-based signal, such as when waiting for more results from a scan, or when waiting for a scan to close.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_scan_result_count_slave`

The number of times a thread has been blocked by this replica while waiting for a scan-based signal, such as when waiting for more results from a scan, or when waiting for a scan to close.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_scan_result_count`

  The number of times a thread has been blocked by this MySQL Server (SQL node) while waiting for a scan-based signal, such as when waiting for more results from a scan, or when waiting for a scan to close.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 18.5.13, “NDB API Statistics Counters and Variables”.

- `Ndb_cluster_node_id`

  If the server is acting as an NDB Cluster node, then the value of this variable its node ID in the cluster.

  If the server is not part of an NDB Cluster, then the value of this variable is 0.

- `Ndb_config_from_host`

  If the server is part of an NDB Cluster, the value of this variable is the host name or IP address of the Cluster management server from which it gets its configuration data.

  If the server is not part of an NDB Cluster, then the value of this variable is an empty string.

- `Ndb_config_from_port`

  If the server is part of an NDB Cluster, the value of this variable is the number of the port through which it is connected to the Cluster management server from which it gets its configuration data.

  If the server is not part of an NDB Cluster, then the value of this variable is 0.

- `Ndb_conflict_fn_max_del_win`

  Shows the number of times that a row was rejected on the current SQL node due to NDB Cluster Replication conflict resolution using `NDB$MAX_DELETE_WIN()`, since the last time that this `mysqld` was started.

  For more information, see Section 18.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_conflict_fn_max`

  Used in NDB Cluster Replication conflict resolution, this variable shows the number of times that a row was not applied on the current SQL node due to “greatest timestamp wins” conflict resolution since the last time that this `mysqld` was started.

  For more information, see Section 18.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_conflict_fn_old`

  Used in NDB Cluster Replication conflict resolution, this variable shows the number of times that a row was not applied as the result of “same timestamp wins” conflict resolution on a given `mysqld` since the last time it was restarted.

For more information, see Section 18.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_conflict_fn_epoch`

  Used in NDB Cluster Replication conflict resolution, this variable shows the number of rows found to be in conflict using `NDB$EPOCH()` conflict resolution on a given `mysqld` since the last time it was restarted.

  For more information, see Section 18.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_conflict_fn_epoch2`

  Shows the number of rows found to be in conflict in NDB Cluster Replication conflict resolution, when using `NDB$EPOCH2()`, on the source designated as the primary since the last time it was restarted.

  Added in NDB 7.4.2.

  For more information, see NDB$EPOCH2().

- `Ndb_conflict_fn_epoch_trans`

  Used in NDB Cluster Replication conflict resolution, this variable shows the number of rows found to be in conflict using `NDB$EPOCH_TRANS()` conflict resolution on a given `mysqld` since the last time it was restarted.

  For more information, see Section 18.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_conflict_fn_epoch2_trans`

  Used in NDB Cluster Replication conflict resolution, this variable shows the number of rows found to be in conflict using `NDB$EPOCH_TRANS2()` conflict resolution on a given `mysqld` since the last time it was restarted.

  Added in NDB 7.4.2.

  For more information, see NDB$EPOCH2_TRANS().

- `Ndb_conflict_last_conflict_epoch`

  The most recent epoch in which a conflict was detected on this replica. You can compare this value with `Ndb_slave_max_replicated_epoch`; if `Ndb_slave_max_replicated_epoch` is greater than `Ndb_conflict_last_conflict_epoch`, no conflicts have yet been detected.

  This variable was added in NDB 7.4.2.

  See Section 18.6.11, “NDB Cluster Replication Conflict Resolution”, for more information.

- `Ndb_conflict_reflected_op_discard_count`

  When using NDB Cluster Replication conflict resolution, this is the number of reflected operations that were not applied on the secondary, due to encountering an error during execution.

  This variable was added in NDB 7.4.2.

  See Section 18.6.11, “NDB Cluster Replication Conflict Resolution”, for more information.

- `Ndb_conflict_reflected_op_prepare_count`

  When using conflict resolution with NDB Cluster Replication, this status variable contains the number of reflected operations that have been defined (that is, prepared for execution on the secondary).

  This variable was added in NDB 7.4.2.

See Section 18.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_conflict_refresh_op_count`

  When using conflict resolution with NDB Cluster Replication, this gives the number of refresh operations that have been prepared for execution on the secondary.

  This variable was added in NDB 7.4.2.

  See Section 18.6.11, “NDB Cluster Replication Conflict Resolution”, for more information.

- `Ndb_conflict_last_stable_epoch`

  Number of rows found to be in conflict by a transactional conflict function

  This variable was added in NDB 7.4.2.

  See Section 18.6.11, “NDB Cluster Replication Conflict Resolution”, for more information.

- `Ndb_conflict_trans_row_conflict_count`

  Used in NDB Cluster Replication conflict resolution, this status variable shows the number of rows found to be directly in-conflict by a transactional conflict function on a given `mysqld` since the last time it was restarted.

  Currently, the only transactional conflict detection function supported by NDB Cluster is NDB$EPOCH_TRANS(), so this status variable is effectively the same as `Ndb_conflict_fn_epoch_trans`.

  For more information, see Section 18.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_conflict_trans_row_reject_count`

  Used in NDB Cluster Replication conflict resolution, this status variable shows the total number of rows realigned due to being determined as conflicting by a transactional conflict detection function. This includes not only `Ndb_conflict_trans_row_conflict_count`, but any rows in or dependent on conflicting transactions.

  For more information, see Section 18.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_conflict_trans_reject_count`

  Used in NDB Cluster Replication conflict resolution, this status variable shows the number of transactions found to be in conflict by a transactional conflict detection function.

  For more information, see Section 18.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_conflict_trans_detect_iter_count`

  Used in NDB Cluster Replication conflict resolution, this shows the number of internal iterations required to commit an epoch transaction. Should be (slightly) greater than or equal to `Ndb_conflict_trans_conflict_commit_count`.

  For more information, see Section 18.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_conflict_trans_conflict_commit_count`

  Used in NDB Cluster Replication conflict resolution, this shows the number of epoch transactions committed after they required transactional conflict handling.

  For more information, see Section 18.6.11, “NDB Cluster Replication Conflict Resolution”.

- `Ndb_epoch_delete_delete_count`

  When using delete-delete conflict detection, this is the number of delete-delete conflicts detected, where a delete operation is applied, but the indicated row does not exist.

  Added in NDB 7.4.2.

- `Ndb_execute_count`

  Provides the number of round trips to the `NDB` kernel made by operations.

- `Ndb_last_commit_epoch_server`

  The epoch most recently committed by `NDB`.

  This variable was added in NDB 7.3.8 and NDB 7.4.1.

- `Ndb_last_commit_epoch_session`

  The epoch most recently committed by this `NDB` client.

  This variable was added in NDB 7.3.8 and NDB 7.4.1.

- `Ndb_number_of_data_nodes`

  If the server is part of an NDB Cluster, the value of this variable is the number of data nodes in the cluster.

  If the server is not part of an NDB Cluster, then the value of this variable is 0.

- `Ndb_pushed_queries_defined`

  The total number of joins pushed down to the NDB kernel for distributed handling on the data nodes.

  Note

  Joins tested using `EXPLAIN` that can be pushed down contribute to this number.

- `Ndb_pushed_queries_dropped`

  The number of joins that were pushed down to the NDB kernel but that could not be handled there.

- `Ndb_pushed_queries_executed`

  The number of joins successfully pushed down to `NDB` and executed there.

- `Ndb_pushed_reads`

  The number of rows returned to `mysqld` from the NDB kernel by joins that were pushed down.

  Note

  Executing `EXPLAIN` on joins that can be pushed down to `NDB` does not add to this number.

- `Ndb_pruned_scan_count`

  This variable holds a count of the number of scans executed by `NDBCLUSTER` since the NDB Cluster was last started where `NDBCLUSTER` was able to use partition pruning.

  Using this variable together with `Ndb_scan_count` can be helpful in schema design to maximize the ability of the server to prune scans to a single table partition, thereby involving only a single data node.

- `Ndb_scan_count`

  This variable holds a count of the total number of scans executed by `NDBCLUSTER` since the NDB Cluster was last started.

- `Ndb_slave_max_replicated_epoch`

  The most recently committed epoch on this replica. In NDB 7.4.1 and later, you can compare this value with `Ndb_conflict_last_conflict_epoch`; if `Ndb_slave_max_replicated_epoch` is the greater of the two, no conflicts have yet been detected.

  This variable was added in NDB 7.3.8 and NDB 7.4.1.

  For more information, see Section 18.6.11, “NDB Cluster Replication Conflict Resolution”.

18.3.3.9 NDB Cluster TCP/IP Connections

TCP/IP is the default transport mechanism for all connections between nodes in an NDB Cluster. Normally it is not necessary to define TCP/IP connections; NDB Cluster automatically sets up such connections for all data nodes, management nodes, and SQL or API nodes.

Note

For an exception to this rule, see Section 18.3.3.10, “NDB Cluster TCP/IP Connections Using Direct Connections”.

To override the default connection parameters, it is necessary to define a connection using one or more `[tcp]` sections in the `config.ini` file. Each `[tcp]` section explicitly defines a TCP/IP connection between two NDB Cluster nodes, and must contain at a minimum the parameters `NodeId1` and `NodeId2`, as well as any connection parameters to override.

It is also possible to change the default values for these parameters by setting them in the `[tcp default]` section.

Important

Any `[tcp]` sections in the `config.ini` file should be listed *last*, following all other sections in the file. However, this is not required for a `[tcp default]` section. This requirement is a known issue with the way in which the `config.ini` file is read by the NDB Cluster management server.

Connection parameters which can be set in `[tcp]` and `[tcp default]` sections of the `config.ini` file are listed here:

- `Checksum`

Version (or later)	NDB 7.3.1
Type or units	boolean
Default	false
Range	true, false
Restart Type	N

  This parameter is a boolean parameter (enabled by setting it to `Y` or `1`, disabled by setting it to `N` or `0`). It is disabled by default. When it is enabled, checksums for all messages are calculated before they placed in the send buffer. This feature ensures that messages are not corrupted while waiting in the send buffer, or by the transport mechanism.

- `Group`

When `ndb_optimized_node_selection` is enabled, node proximity is used in some cases to select which node to connect to. This parameter can be used to influence proximity by setting it to a lower value, which is interpreted as “closer”. See the description of the system variable for more information.

- `HostName1`

Version (or later)	NDB 7.3.1
Type or units	name or IP address
Default	[none]
Range	...
Restart Type	N

The `HostName1` and `HostName2` parameters can be used to specify specific network interfaces to be used for a given TCP connection between two nodes. The values used for these parameters can be host names or IP addresses.

- `HostName2`

Version (or later)	NDB 7.3.1
Type or units	name or IP address
Default	[none]
Range	...
Restart Type	N

The `HostName1` and `HostName2` parameters can be used to specify specific network interfaces to be used for a given TCP connection between two nodes. The values used for these parameters can be host names or IP addresses.

- `NodeId1`

Version (or later)	NDB 7.3.1
Type or units	numeric
Default	[none]
Range	1 - 255
Restart Type	N

To identify a connection between two nodes it is necessary to provide their node IDs in the `[tcp]` section of the configuration file as the values of `NodeId1` and `NodeId2`. These are the same unique `Id` values for each of these nodes as described in Section 18.3.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”.

- `NodeId2`

Version (or later)	NDB 7.3.1
Type or units	numeric
Default	[none]
Range	1 - 255
Restart Type	N

To identify a connection between two nodes it is necessary to provide their node IDs in the `[tcp]` section of the configuration file as the values of `NodeId1` and `NodeId2`. These are the same unique

`Id` values for each of these nodes as described in Section 18.3.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”.

- `OverloadLimit`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

When more than this many unsent bytes are in the send buffer, the connection is considered overloaded.

This parameter can be used to determine the amount of unsent data that must be present in the send buffer before the connection is considered overloaded. See Section 18.3.3.12, “Configuring NDB Cluster Send Buffer Parameters”, and Section 18.5.14.27, “The ndbinfo transporters Table”, for more information.

- `PortNumber` (*OBSOLETE* / *REMOVED*)

  This parameter formerly specified the port number to be used for listening for connections from other nodes, and was removed in NDB 7.5.1; use the `ServerPort` data node configuration parameter for this purpose instead (Bug #77405, Bug #21280456).

- `ReceiveBufferMemory`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	2M
Range	16K - 4294967039 (0xFFFFFEFF)
Restart Type	N

Specifies the size of the buffer used when receiving data from the TCP/IP socket.

The default value of this parameter is 2MB. The minimum possible value is 16KB; the theoretical maximum is 4GB.

- `SendBufferMemory`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	2M
Range	256K - 4294967039 (0xFFFFFEFF)
Restart Type	N

TCP transporters use a buffer to store all messages before performing the send call to the operating system. When this buffer reaches 64KB its contents are sent; these are also sent when a round of messages have been executed. To handle temporary overload situations it is also possible to define a bigger send buffer.

If this parameter is set explicitly, then the memory is not dedicated to each transporter; instead, the value used denotes the hard limit for how much memory (out of the total available memory —that is, `TotalSendBufferMemory`) that may be used by a single transporter. For more

information about configuring dynamic transporter send buffer memory allocation in NDB Cluster, see Section 18.3.3.12, “Configuring NDB Cluster Send Buffer Parameters”.

The default size of the send buffer is 2MB, which is the size recommended in most situations. The minimum size is 64 KB; the theoretical maximum is 4 GB.

- `SendSignalId`

Version (or later)	NDB 7.3.1
Type or units	boolean
Default	false (debug builds: true)
Range	true, false
Restart Type	N

To be able to retrace a distributed message datagram, it is necessary to identify each message. When this parameter is set to `Y`, message IDs are transported over the network. This feature is disabled by default in production builds, and enabled in `-debug` builds.

- `TcpBind_INADDR_ANY`

Setting this parameter to `TRUE` or `1` binds `IP_ADDR_ANY` so that connections can be made from anywhere (for autogenerated connections). The default is `FALSE` (`0`).

- `TCP_MAXSEG_SIZE`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	0
Range	0 - 2G
Restart Type	N

Determines the size of the memory set during TCP transporter initialization. The default is recommended for most common usage cases.

- `TCP_RCV_BUF_SIZE`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	0
Range	0 - 2G
Restart Type	N

Determines the size of the receive buffer set during TCP transporter initialization. Prior to NDB 7.3.1, the default was 70080 and the minimum was 1. In NDB 7.3.1 and later, the default and minimum value is 0, which allows the operating system or platform to set this value. The default is recommended for most common usage cases.

- `TCP_SND_BUF_SIZE`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	0
Range	0 - 2G

Restart Type	N
Version (or later)	NDB 7.4.8
Type or units	unsigned
Default	0
Range	0 - 2G
Restart Type	N

Determines the size of the send buffer set during TCP transporter initialization. Prior to NDB 7.3.1, the default was 71540 and the minimum was 1. In NDB 7.3.1 and later, the default and minimum value is 0, which allows the operating system or platform to set this value. The default is recommended for most common usage cases.

Restart types. Information about the restart types used by the parameter descriptions in this section is shown in the following table:

Table 18.15 NDB Cluster restart types

Symbol	Restart Type	Description
N	Node	The parameter can be updated using a rolling restart (see Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”)
S	System	All cluster nodes must be shut down completely, then restarted, to effect a change in this parameter
I	Initial	Data nodes must be restarted using the `--initial` option

18.3.3.10 NDB Cluster TCP/IP Connections Using Direct Connections

Setting up a cluster using direct connections between data nodes requires specifying explicitly the crossover IP addresses of the data nodes so connected in the `[tcp]` section of the cluster `config.ini` file.

In the following example, we envision a cluster with at least four hosts, one each for a management server, an SQL node, and two data nodes. The cluster as a whole resides on the `172.23.72.*` subnet of a LAN. In addition to the usual network connections, the two data nodes are connected directly using a standard crossover cable, and communicate with one another directly using IP addresses in the `1.1.0.*` address range as shown:

```
# Management Server
[ndb_mgmd]
Id=1
HostName=172.23.72.20

# SQL Node
[mysqld]
Id=2
HostName=172.23.72.21

# Data Nodes
[ndbd]
Id=3
HostName=172.23.72.22

[ndbd]
Id=4
HostName=172.23.72.23

# TCP/IP Connections
[tcp]
NodeId1=3
NodeId2=4
HostName1=1.1.0.1
HostName2=1.1.0.2
```

The `HostName1` and `HostName2` parameters are used only when specifying direct connections.

The use of direct TCP connections between data nodes can improve the cluster's overall efficiency by enabling the data nodes to bypass an Ethernet device such as a switch, hub, or router, thus cutting down on the cluster's latency.

Note

To take the best advantage of direct connections in this fashion with more than two data nodes, you must have a direct connection between each data node and every other data node in the same node group.

18.3.3.11 NDB Cluster Shared-Memory Connections

NDB Cluster attempts to use the shared memory transporter and configure it automatically where possible. `[shm]` sections in the `config.ini` file explicitly define shared-memory connections between nodes in the cluster. When explicitly defining shared memory as the connection method, it is necessary to define at least `NodeId1`, `NodeId2`, and `ShmKey`. All other parameters have default values that should work well in most cases.

Important

SHM functionality is considered experimental only. It is not officially supported in any current NDB Cluster release, and testing results indicate that SHM performance is not appreciably greater than when using TCP/IP for the transporter.

For these reasons, you must determine for yourself or by using our free resources (forums, mailing lists) whether SHM can be made to work correctly in your specific case.

- `Checksum`

Version (or later)	NDB 7.3.1
Type or units	boolean
Default	true
Range	true, false
Restart Type	N

This parameter is a boolean (`Y`/`N`) parameter which is disabled by default. When it is enabled, checksums for all messages are calculated before being placed in the send buffer.

This feature prevents messages from being corrupted while waiting in the send buffer. It also serves as a check against data being corrupted during transport.

- `Group`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	35
Range	0 - 200
Restart Type	N

Determines the group proximity; a smaller value is interpreted as being closer. The default value is sufficient for most conditions.

- `HostName1`

Version (or later)	NDB 7.3.1
Type or units	name or IP address
Default	[none]
Range	...
Restart Type	N

The `HostName1` and `HostName2` parameters can be used to specify specific network interfaces to be used for a given SHM connection between two nodes. The values used for these parameters can be host names or IP addresses.

- `HostName2`

Version (or later)	NDB 7.3.1
Type or units	name or IP address
Default	[none]
Range	...
Restart Type	N

The `HostName1` and `HostName2` parameters can be used to specify specific network interfaces to be used for a given SHM connection between two nodes. The values used for these parameters can be host names or IP addresses.

- `NodeId1`

Version (or later)	NDB 7.3.1
Type or units	numeric
Default	[none]
Range	1 - 255
Restart Type	N

To identify a connection between two nodes it is necessary to provide node identifiers for each of them, as `NodeId1` and `NodeId2`.

- `NodeId2`

Version (or later)	NDB 7.3.1
Type or units	numeric
Default	[none]
Range	1 - 255
Restart Type	N

To identify a connection between two nodes it is necessary to provide node identifiers for each of them, as `NodeId1` and `NodeId2`.

- `NodeIdServer`

Version (or later)	NDB 7.3.1
Type or units	numeric
Default	[none]
Range	1 - 63

Restart Type	N

Identify the server end of a shared memory connection.

- `OverloadLimit`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

When more than this many unsent bytes are in the send buffer, the connection is considered overloaded.

This parameter can be used to determine the amount of unsent data that must be present in the send buffer before the connection is considered overloaded. See Section 18.3.3.12, “Configuring NDB Cluster Send Buffer Parameters”, for more information.

- `SendSignalId`

Version (or later)	NDB 7.3.1
Type or units	boolean
Default	false
Range	true, false
Restart Type	N

To retrace the path of a distributed message, it is necessary to provide each message with a unique identifier. Setting this parameter to `Y` causes these message IDs to be transported over the network as well. This feature is disabled by default in production builds, and enabled in `-debug` builds.

- `ShmKey`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	0
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N

When setting up shared memory segments, a node ID, expressed as an integer, is used to identify uniquely the shared memory segment to use for the communication. There is no default value.

- `ShmSize`

Version (or later)	NDB 7.3.1
Type or units	bytes
Default	1M
Range	64K - 4294967039 (0xFFFFFEFF)
Restart Type	N

Each SHM connection has a shared memory segment where messages between nodes are placed by the sender and read by the reader. The size of this segment is defined by `ShmSize`. The default value is 1MB.

- `SigNum`

Version (or later)	NDB 7.3.1
Type or units	unsigned
Default	[none]
Range	0 - 4294967039 (0xFFFFFEFF)
Restart Type	N
Deprecated	Yes (in NDB 7.6)

When using the shared memory transporter, a process sends an operating system signal to the other process when there is new data available in the shared memory. Should that signal conflict with an existing signal, this parameter can be used to change it. This is a possibility when using SHM due to the fact that different operating systems use different signal numbers.

The default value of `SigNum` is 0; therefore, it must be set to avoid errors in the cluster log when using the shared memory transporter. Typically, this parameter is set to 10 in the `[shm default]` section of the `config.ini` file.

Restart types. Information about the restart types used by the parameter descriptions in this section is shown in the following table:

Table 18.16 NDB Cluster restart types

Symbol	Restart Type	Description
N	Node	The parameter can be updated using a rolling restart (see Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”)
S	System	All cluster nodes must be shut down completely, then restarted, to effect a change in this parameter
I	Initial	Data nodes must be restarted using the `--initial` option

18.3.3.12 Configuring NDB Cluster Send Buffer Parameters

Formerly, the NDB kernel employed a send buffer whose size was fixed at 2MB for each node in the cluster, this buffer being allocated when the node started. Because the size of this buffer could not be changed after the cluster was started, it was necessary to make it large enough in advance to accommodate the maximum possible load on any transporter socket. However, this was an inefficient use of memory, since much of it often went unused, and could result in large amounts of resources being wasted when scaling up to many API nodes.

This problem was eventually solved (in NDB Cluster 7.0) by employing a unified send buffer whose memory is allocated dynamically from a pool shared by all transporters. This means that the size of the send buffer can be adjusted as necessary. Configuration of the unified send buffer can accomplished by setting the following parameters:

- **TotalSendBufferMemory.** This parameter can be set for all types of NDB Cluster nodes—that is, it can be set in the `[ndbd]`, `[mgm]`, and `[api]` (or `[mysql]`) sections of the `config.ini` file. It represents the total amount of memory (in bytes) to be allocated by each node for which it is set for use among all configured transporters. If set, its minimum is 256KB; the maximum is 4294967039.

  To be backward-compatible with existing configurations, this parameter takes as its default value the sum of the maximum send buffer sizes of all configured transporters, plus an additional 32KB (one page) per transporter. The maximum depends on the type of transporter, as shown in the following table:

Table 18.17 Transporter types with maximum send buffer sizes

Transporter	Maximum Send Buffer Size (bytes)
TCP	`SendBufferMemory` (default = 2M)
SHM	20K

This enables existing configurations to function in close to the same way as they did with NDB Cluster 6.3 and earlier, with the same amount of memory and send buffer space available to each transporter. However, memory that is unused by one transporter is not available to other transporters.

- **OverloadLimit.** This parameter is used in the `config.ini` file `[tcp]` section, and denotes the amount of unsent data (in bytes) that must be present in the send buffer before the connection is considered overloaded. When such an overload condition occurs, transactions that affect the overloaded connection fail with NDB API Error 1218 (`Send Buffers overloaded in NDB kernel`) until the overload status passes. The default value is 0, in which case the effective overload limit is calculated as `SendBufferMemory * 0.8` for a given connection. The maximum value for this parameter is 4G.

- **SendBufferMemory.** This value denotes a hard limit for the amount of memory that may be used by a single transporter out of the entire pool specified by `TotalSendBufferMemory`. However, the sum of `SendBufferMemory` for all configured transporters may be greater than the `TotalSendBufferMemory` that is set for a given node. This is a way to save memory when many nodes are in use, as long as the maximum amount of memory is never required by all transporters at the same time.

- **ReservedSendBufferMemory.** This optional data node parameter, if set, gives an amount of memory (in bytes) that is reserved for connections between data nodes; this memory is not allocated to send buffers used for communications with management servers or API nodes. This provides a way to protect the cluster against misbehaving API nodes that use excess send memory and thus cause failures in communications internally in the NDB kernel. If set, its the minimum permitted value for this parameters is 256KB; the maximum is 4294967039.

You can use the `ndbinfo.transporters` table to monitor send buffer memory usage, and to detect slowdown and overload conditions that can adversely affect performance.

18.3.4 Using High-Speed Interconnects with NDB Cluster

Even before design of `NDBCLUSTER` began in 1996, it was evident that one of the major problems to be encountered in building parallel databases would be communication between the nodes in the network. For this reason, `NDBCLUSTER` was designed from the very beginning to permit the use of a number of different data transport mechanisms. In this Manual, we use the term *transporter* for these.

The NDB Cluster codebase provides for four different transporters:

- *TCP/IP using 100 Mbps or gigabit Ethernet*, as discussed in Section 18.3.3.9, “NDB Cluster TCP/IP Connections”.

- *Direct (machine-to-machine) TCP/IP*; although this transporter uses the same TCP/IP protocol as mentioned in the previous item, it requires setting up the hardware differently and is configured differently as well. For this reason, it is considered a separate transport mechanism for NDB Cluster. See Section 18.3.3.10, “NDB Cluster TCP/IP Connections Using Direct Connections”, for details.

- *Shared memory (SHM)*. For more information about SHM, see Section 18.3.3.11, “NDB Cluster Shared-Memory Connections”.

Note

SHM is considered experimental only, and is not officially supported.

- *Scalable Coherent Interface (SCI)*.

Note

Using SCI transporters in NDB Cluster requires specialized hardware, software, and MySQL binaries not available using an NDB 7.3 or 7.4 distributions.

Most users today employ TCP/IP over Ethernet because it is ubiquitous. TCP/IP is also by far the best-tested transporter for use with NDB Cluster.

Regardless of the transporter used, `NDB` attempts to make sure that communication with data node processes is done using chunks that are as large as possible since this benefits all types of data transmission.

18.4 NDB Cluster Programs

Using and managing an NDB Cluster requires several specialized programs, which we describe in this chapter. We discuss the purposes of these programs in an NDB Cluster, how to use the programs, and what startup options are available for each of them.

These programs include the NDB Cluster data, management, and SQL node processes (`ndbd`, `ndbmtd`, `ndb_mgmd`, and `mysqld`) and the management client (`ndb_mgm`).

Information about the program `ndb_setup.py` (added in NDB 7.3.1), used to start the NDB Cluster Auto-Installer (now deprecated), is also included in this section. You should be aware that Section 18.4.25, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”, contains information about the command-line client only; for information about using the GUI installer spawned by this program to configure and deploy an NDB Cluster, see Section 18.2.8, “The NDB Cluster Auto-Installer (No longer supported)”.

For information about using `mysqld` as an NDB Cluster process, see Section 18.5.9, “MySQL Server Usage for NDB Cluster”.

Other `NDB` utility, diagnostic, and example programs are included with the NDB Cluster distribution. These include `ndb_restore`, `ndb_show_tables`, and `ndb_config`. These programs are also covered in this section.

The final portion of this section contains tables of options that are common to all the various NDB Cluster programs.

18.4.1 ndbd — The NDB Cluster Data Node Daemon

`ndbd` is the process that is used to handle all the data in tables using the NDB Cluster storage engine. This is the process that empowers a data node to accomplish distributed transaction handling, node recovery, checkpointing to disk, online backup, and related tasks.

In an NDB Cluster, a set of `ndbd` processes cooperate in handling data. These processes can execute on the same computer (host) or on different computers. The correspondences between data nodes and Cluster hosts is completely configurable.

The following table includes command options specific to the NDB Cluster data node program `ndbd`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndbd`), see Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 18.18 Command-line options for the ndbd program

Format	Description	Added, Deprecated, or Removed
`--bind-address=name`	Local bind address	(Supported in all MySQL 5.6 based releases)

Format	Description	Added, Deprecated, or Removed
--connect-delay=#	Time to wait between attempts to contact a management server, in seconds; 0 means do not wait between attempts	DEPRECATED: NDB 7.4.9
--connect-retries=#	Set the number of times to retry a connection before giving up; 0 means 1 attempt only (and no retries)	(Supported in all MySQL 5.6 based releases)
--connect-retry-delay=#	Time to wait between attempts to contact a management server, in seconds; 0 means do not wait between attempts	ADDED: NDB 7.4.9
--daemon, -d	Start ndbd as daemon (default); override with --nodaemon	(Supported in all MySQL 5.6 based releases)
--foreground	Run ndbd in foreground, provided for debugging purposes (implies --nodaemon)	(Supported in all MySQL 5.6 based releases)
--initial	Perform initial start of ndbd, including file system cleanup; consult documentation before using this option	(Supported in all MySQL 5.6 based releases)
--initial-start	Perform partial initial start (requires --nowait-nodes)	(Supported in all MySQL 5.6 based releases)
--install[=name]	Used to install data node process as Windows service; does not apply on other platforms	(Supported in all MySQL 5.6 based releases)
--nodaemon	Do not start ndbd as daemon; provided for testing purposes	(Supported in all MySQL 5.6 based releases)
--nostart, -n	Do not start ndbd immediately; ndbd waits for command to start from ndb_mgm	(Supported in all MySQL 5.6 based releases)
--nowait-nodes=list	Do not wait for these data nodes to start (takes comma-separated list of node IDs); requires --ndb-nodeid	(Supported in all MySQL 5.6 based releases)
--remove[=name]	Used to remove data node process that was previously installed as Windows service; does not apply on other platforms	(Supported in all MySQL 5.6 based releases)
--verbose, -v	Write extra debugging information to node log	(Supported in all MySQL 5.6 based releases)

Note

All of these options also apply to the multithreaded version of this program (ndbmtd) and you may substitute “ndbmtd” for “ndbd” wherever the latter occurs in this section.

- --bind-address

Command-Line Format	--bind-address=name
Type	String
Default Value	

Causes ndbd to bind to a specific network interface (host name or IP address). This option has no default value.

- --connect-delay=#

Command-Line Format	--connect-delay=#
Deprecated	5.6.28-ndb-7.4.9
Type	Numeric
Default Value	5
Minimum Value	0
Maximum Value	3600

Determines the time to wait between attempts to contact a management server when starting (the number of attempts is controlled by the --connect-retries option). The default is 5 seconds.

This option is deprecated in NDB 7.4.9, and is subject to removal in a future release of NDB Cluster. Use --connect-retry-delay instead.

- --connect-retries=#

Command-Line Format	--connect-retries=#
Type	Numeric
Default Value	12
Minimum Value	0
Maximum Value	65535

Set the number of times to retry a connection before giving up; 0 means 1 attempt only (and no retries). The default is 12 attempts. The time to wait between attempts is controlled by the --connect-retry-delay option in MySQL NDB 7.4.9 and later (previously, this was --connect-delay).

- --connect-retry-delay=#

Command-Line Format	--connect-retry-delay=#
Introduced	5.6.28-ndb-7.4.9
Type	Numeric
Default Value	5
Minimum Value	0
Maximum Value	4294967295

Determines the time to wait between attempts to contact a management server when starting (the time between attempts is controlled by the --connect-retries option). The default is 5 seconds.

This option was added in NDB 7.4.9, and is intended to take the place of the --connect-delay option, which is now deprecated and subject to removal in a future release of NDB Cluster.

- `--daemon`, `-d`

Command-Line Format	`--daemon`
Type	Boolean
Default Value	`TRUE`

Instructs `ndbd` or `ndbmtd` to execute as a daemon process. This is the default behavior. `--nodaemon` can be used to prevent the process from running as a daemon.

This option has no effect when running `ndbd` or `ndbmtd` on Windows platforms.

- `--foreground`

Command-Line Format	`--foreground`
Type	Boolean
Default Value	`FALSE`

Causes `ndbd` or `ndbmtd` to execute as a foreground process, primarily for debugging purposes. This option implies the `--nodaemon` option.

This option has no effect when running `ndbd` or `ndbmtd` on Windows platforms.

- `--initial`

Command-Line Format	`--initial`
Type	Boolean
Default Value	`FALSE`

Instructs `ndbd` to perform an initial start. An initial start erases any files created for recovery purposes by earlier instances of `ndbd`. It also re-creates recovery log files. On some operating systems, this process can take a substantial amount of time.

An `--initial` start is to be used *only* when starting the `ndbd` process under very special circumstances; this is because this option causes all files to be removed from the NDB Cluster file system and all redo log files to be re-created. These circumstances are listed here:

- When performing a software upgrade which has changed the contents of any files.
- When restarting the node with a new version of `ndbd`.
- As a measure of last resort when for some reason the node restart or system restart repeatedly fails. In this case, be aware that this node can no longer be used to restore data due to the destruction of the data files.

Warning

To avoid the possibility of eventual data loss, it is recommended that you *not* use the `--initial` option together with `StopOnError = 0`. Instead, set `StopOnError` to 0 in `config.ini` only after the cluster has been started, then restart the data nodes normally—that is, without the `--initial` option.

See the description of the StopOnError parameter for a detailed explanation of this issue. (Bug #24945638)

Use of this option prevents the StartPartialTimeout and StartPartitionedTimeout configuration parameters from having any effect.

Important

This option does *not* affect either of the following types of files:

- Backup files that have already been created by the affected node
- NDB Cluster Disk Data files (see Section 18.5.10, "NDB Cluster Disk Data Tables").

This option also has no effect on recovery of data by a data node that is just starting (or restarting) from data nodes that are already running. This recovery of data occurs automatically, and requires no user intervention in an NDB Cluster that is running normally.

It is permissible to use this option when starting the cluster for the very first time (that is, before any data node files have been created); however, it is *not* necessary to do so.

- --initial-start

Command-Line Format	--initial-start
Type	Boolean
Default Value	FALSE

This option is used when performing a partial initial start of the cluster. Each node should be started with this option, as well as --nowait-nodes.

Suppose that you have a 4-node cluster whose data nodes have the IDs 2, 3, 4, and 5, and you wish to perform a partial initial start using only nodes 2, 4, and 5—that is, omitting node 3:

```
shell> ndbd --ndb-nodeid=2 --nowait-nodes=3 --initial-start
shell> ndbd --ndb-nodeid=4 --nowait-nodes=3 --initial-start
shell> ndbd --ndb-nodeid=5 --nowait-nodes=3 --initial-start
```

When using this option, you must also specify the node ID for the data node being started with the --ndb-nodeid option.

Important

Do not confuse this option with the --nowait-nodes option for ndb_mgmd, which can be used to enable a cluster configured with multiple management servers to be started without all management servers being online.

- --install[=*name*]

Command-Line Format	--install[=name]
Platform Specific	Windows
Type	String
Default Value	ndbd

Causes ndbd to be installed as a Windows service. Optionally, you can specify a name for the service; if not set, the service name defaults to ndbd. Although it is preferable to specify other ndbd program options in a my.ini or my.cnf configuration file, it is possible to use together with --

install. However, in such cases, the --install option must be specified first, before any other options are given, for the Windows service installation to succeed.

It is generally not advisable to use this option together with the --initial option, since this causes the data node file system to be wiped and rebuilt every time the service is stopped and started. Extreme care should also be taken if you intend to use any of the other ndbd options that affect the starting of data nodes—including --initial-start, --nostart, and --nowait-nodes—together with --install, and you should make absolutely certain you fully understand and allow for any possible consequences of doing so.

The --install option has no effect on non-Windows platforms.

- --nodaemon

Command-Line Format	--nodaemon
Type	Boolean
Default Value	FALSE

Prevents ndbd or ndbmtd from executing as a daemon process. This option overrides the --daemon option. This is useful for redirecting output to the screen when debugging the binary.

The default behavior for ndbd and ndbmtd on Windows is to run in the foreground, making this option unnecessary on Windows platforms, where it has no effect.

- --nostart, -n

Command-Line Format	--nostart
Type	Boolean
Default Value	FALSE

Instructs ndbd not to start automatically. When this option is used, ndbd connects to the management server, obtains configuration data from it, and initializes communication objects. However, it does not actually start the execution engine until specifically requested to do so by the management server. This can be accomplished by issuing the proper START command in the management client (see Section 18.5.1, “Commands in the NDB Cluster Management Client”).

- --nowait-nodes=*node_id_1*[, *node_id_2*[, ...]]

Command-Line Format	--nowait-nodes=list
Type	String
Default Value	

This option takes a list of data nodes which for which the cluster does not wait for before starting.

This can be used to start the cluster in a partitioned state. For example, to start the cluster with only half of the data nodes (nodes 2, 3, 4, and 5) running in a 4-node cluster, you can start each ndbd process with --nowait-nodes=3,5. In this case, the cluster starts as soon as nodes 2 and 4 connect, and does *not* wait StartPartitionedTimeout milliseconds for nodes 3 and 5 to connect as it would otherwise.

If you wanted to start up the same cluster as in the previous example without one ndbd (say, for example, that the host machine for node 3 has suffered a hardware failure) then start nodes 2, 4, and 5 with --nowait-nodes=3. Then the cluster starts as soon as nodes 2, 4, and 5 connect and does not wait for node 3 to start.

- `--remove[=name]`

Command-Line Format	`--remove[=name]`
Platform Specific	Windows
Type	String
Default Value	`ndbd`

  Causes an `ndbd` process that was previously installed as a Windows service to be removed. Optionally, you can specify a name for the service to be uninstalled; if not set, the service name defaults to `ndbd`.

  The `--remove` option has no effect on non-Windows platforms.

- `--verbose`, `-v`

  Causes extra debug output to be written to the node log.

`ndbd` generates a set of log files which are placed in the directory specified by `DataDir` in the `config.ini` configuration file.

These log files are listed below. *`node_id`* is and represents the node's unique identifier. For example, `ndb_2_error.log` is the error log generated by the data node whose node ID is `2`.

- `ndb_`*`node_id`*`_error.log` is a file containing records of all crashes which the referenced `ndbd` process has encountered. Each record in this file contains a brief error string and a reference to a trace file for this crash. A typical entry in this file might appear as shown here:

```
Date/Time: Saturday 30 July 2004 - 00:20:01
Type of error: error
Message: Internal program error (failed ndbrequire)
Fault ID: 2341
Problem data: DbtupFixAlloc.cpp
Object of reference: DBTUP (Line: 173)
ProgramName: NDB Kernel
ProcessID: 14909
TraceFile: ndb_2_trace.log.2
***EOM***
```

  Listings of possible `ndbd` exit codes and messages generated when a data node process shuts down prematurely can be found in Data Node Error Messages.

Important

The last entry in the error log file is not necessarily the newest one (nor is it likely to be). Entries in the error log are *not* listed in chronological order; rather, they correspond to the order of the trace files as determined in the `ndb_`*`node_id`*`_trace.log.next` file (see below). Error log entries are thus overwritten in a cyclical and not sequential fashion.

- `ndb_`*`node_id`*`_trace.log.`*`trace_id`* is a trace file describing exactly what happened just before the error occurred. This information is useful for analysis by the NDB Cluster development team.

  It is possible to configure the number of these trace files that are created before old files are overwritten. *`trace_id`* is a number which is incremented for each successive trace file.

- `ndb_`*`node_id`*`_trace.log.next` is the file that keeps track of the next trace file number to be assigned.

- `ndb_`*`node_id`*`_out.log` is a file containing any data output by the `ndbd` process. This file is created only if `ndbd` is started as a daemon, which is the default behavior.

- ndb_*node_id*.pid is a file containing the process ID of the ndbd process when started as a daemon. It also functions as a lock file to avoid the starting of nodes with the same identifier.

- ndb_*node_id*_signal.log is a file used only in debug versions of ndbd, where it is possible to trace all incoming, outgoing, and internal messages with their data in the ndbd process.

It is recommended not to use a directory mounted through NFS because in some environments this can cause problems whereby the lock on the .pid file remains in effect even after the process has terminated.

To start ndbd, it may also be necessary to specify the host name of the management server and the port on which it is listening. Optionally, one may also specify the node ID that the process is to use.

```
shell> ndbd --connect-string="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

See Section 18.3.3.3, “NDB Cluster Connection Strings”, for additional information about this issue. Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”, describes other command-line options which can be used with ndbd. For information about data node configuration parameters, see Section 18.3.3.6, “Defining NDB Cluster Data Nodes”.

When ndbd starts, it actually initiates two processes. The first of these is called the “angel process”; its only job is to discover when the execution process has been completed, and then to restart the ndbd process if it is configured to do so. Thus, if you attempt to kill ndbd using the Unix kill command, it is necessary to kill both processes, beginning with the angel process. The preferred method of terminating an ndbd process is to use the management client and stop the process from there.

The execution process uses one thread for reading, writing, and scanning data, as well as all other activities. This thread is implemented asynchronously so that it can easily handle thousands of concurrent actions. In addition, a watch-dog thread supervises the execution thread to make sure that it does not hang in an endless loop. A pool of threads handles file I/O, with each thread able to handle one open file. Threads can also be used for transporter connections by the transporters in the ndbd process. In a multi-processor system performing a large number of operations (including updates), the ndbd process can consume up to 2 CPUs if permitted to do so.

For a machine with many CPUs it is possible to use several ndbd processes which belong to different node groups; however, such a configuration is still considered experimental and is not supported for MySQL 5.6 in a production setting. See Section 18.1.7, “Known Limitations of NDB Cluster”.

18.4.2 ndbinfo_select_all — Select From ndbinfo Tables

ndbinfo_select_all is a client program that selects all rows and columns from one or more tables in the ndbinfo database.

Not all ndbinfo tables available in the mysql client can be read by this program. In addition, ndbinfo_select_all can show information about some tables internal to ndbinfo which cannot be accessed using SQL, including the tables and columns metadata tables.

To select from one or more ndbinfo tables using ndbinfo_select_all, it is necessary to supply the names of the tables when invoking the program as shown here:

```
shell> ndbinfo_select_all table_name1  [table_name2] [...]
```

For example:

```
shell> ndbinfo_select_all logbuffers logspaces
== logbuffers ==
node_id log_type        log_id  log_part        total   used    high
5       0       0       0       33554432        262144  0
6       0       0       0       33554432        262144  0
7       0       0       0       33554432        262144  0
8       0       0       0       33554432        262144  0
== logspaces ==
```

```
node_id log_type        log_id  log_part        total   used    high
5       0       0       0       268435456       0       0
5       0       0       1       268435456       0       0
5       0       0       2       268435456       0       0
5       0       0       3       268435456       0       0
6       0       0       0       268435456       0       0
6       0       0       1       268435456       0       0
6       0       0       2       268435456       0       0
6       0       0       3       268435456       0       0
7       0       0       0       268435456       0       0
7       0       0       1       268435456       0       0
7       0       0       2       268435456       0       0
7       0       0       3       268435456       0       0
8       0       0       0       268435456       0       0
8       0       0       1       268435456       0       0
8       0       0       2       268435456       0       0
8       0       0       3       268435456       0       0
shell>
```

The following table includes options that are specific to ndbinfo_select_all. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndbinfo_select_all), see Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 18.19 Command-line options for the ndbinfo_select_all program

Format	Description	Added, Deprecated, or Removed
--delay=#	Set delay in seconds between loops	(Supported in all MySQL 5.6 based releases)
--loops=#, -l	Set number of times to perform select	(Supported in all MySQL 5.6 based releases)
--database=db_name, -d	Name of database where table is located	(Supported in all MySQL 5.6 based releases)
--parallelism=#, -p	Set degree of parallelism	(Supported in all MySQL 5.6 based releases)

- --delay=seconds

Command-Line Format	--delay=#
Type	Numeric
Default Value	5
Minimum Value	0
Maximum Value	MAX_INT

This option sets the number of seconds to wait between executing loops. Has no effect if --loops is set to 0 or 1.

- --loops=number, -l *number*

Command-Line Format	--loops=#
Type	Numeric
Default Value	1
Minimum Value	0
Maximum Value	MAX_INT

This option sets the number of times to execute the select. Use `--delay` to set the time between loops.

18.4.3 ndbmtd — The NDB Cluster Data Node Daemon (Multi-Threaded)

`ndbmtd` is a multithreaded version of `ndbd`, the process that is used to handle all the data in tables using the `NDBCLUSTER` storage engine. `ndbmtd` is intended for use on host computers having multiple CPU cores. Except where otherwise noted, `ndbmtd` functions in the same way as `ndbd`; therefore, in this section, we concentrate on the ways in which `ndbmtd` differs from `ndbd`, and you should consult Section 18.4.1, “ndbd — The NDB Cluster Data Node Daemon”, for additional information about running NDB Cluster data nodes that apply to both the single-threaded and multithreaded versions of the data node process.

Command-line options and configuration parameters used with `ndbd` also apply to `ndbmtd`. For more information about these options and parameters, see Section 18.4.1, “ndbd — The NDB Cluster Data Node Daemon”, and Section 18.3.3.6, “Defining NDB Cluster Data Nodes”, respectively.

`ndbmtd` is also file system-compatible with `ndbd`. In other words, a data node running `ndbd` can be stopped, the binary replaced with `ndbmtd`, and then restarted without any loss of data. (However, when doing this, you must make sure that `MaxNoOfExecutionThreads` is set to an apppriate value before restarting the node if you wish for `ndbmtd` to run in multithreaded fashion.) Similarly, an `ndbmtd` binary can be replaced with `ndbd` simply by stopping the node and then starting `ndbd` in place of the multithreaded binary. It is not necessary when switching between the two to start the data node binary using `--initial`.

Using `ndbmtd` differs from using `ndbd` in two key respects:

1. Because `ndbmtd` runs by default in single-threaded mode (that is, it behaves like `ndbd`), you must configure it to use multiple threads. This can be done by setting an appropriate value in the `config.ini` file for the `MaxNoOfExecutionThreads` configuration parameter or the `ThreadConfig` configuration parameter. Using `MaxNoOfExecutionThreads` is simpler, but `ThreadConfig` offers more flexibility. For more information about these configuration parameters and their use, see Multi-Threading Configuration Parameters (ndbmtd).

2. Trace files are generated by critical errors in `ndbmtd` processes in a somewhat different fashion from how these are generated by `ndbd` failures. These differences are discussed in more detail in the next few paragraphs.

Like `ndbd`, `ndbmtd` generates a set of log files which are placed in the directory specified by `DataDir` in the `config.ini` configuration file. Except for trace files, these are generated in the same way and have the same names as those generated by `ndbd`.

In the event of a critical error, `ndbmtd` generates trace files describing what happened just prior to the error' occurrence. These files, which can be found in the data node's `DataDir`, are useful for analysis of problems by the NDB Cluster Development and Support teams. One trace file is generated for each `ndbmtd` thread. The names of these files have the following pattern:

```
ndb_node_id_trace.log.trace_id_tthread_id,
```

In this pattern, *`node_id`* stands for the data node's unique node ID in the cluster, *`trace_id`* is a trace sequence number, and *`thread_id`* is the thread ID. For example, in the event of the failure of an `ndbmtd` process running as an NDB Cluster data node having the node ID 3 and with `MaxNoOfExecutionThreads` equal to 4, four trace files are generated in the data node's data directory. If the is the first time this node has failed, then these files are named `ndb_3_trace.log.1_t1`, `ndb_3_trace.log.1_t2`, `ndb_3_trace.log.1_t3`, and `ndb_3_trace.log.1_t4`. Internally, these trace files follow the same format as `ndbd` trace files.

The `ndbd` exit codes and messages that are generated when a data node process shuts down prematurely are also used by `ndbmtd`. See Data Node Error Messages, for a listing of these.

Note

It is possible to use ndbd and ndbmtd concurrently on different data nodes in the same NDB Cluster. However, such configurations have not been tested extensively; thus, we cannot recommend doing so in a production setting at this time.

18.4.4 ndb_mgmd — The NDB Cluster Management Server Daemon

The management server is the process that reads the cluster configuration file and distributes this information to all nodes in the cluster that request it. It also maintains a log of cluster activities. Management clients can connect to the management server and check the cluster's status.

The following table includes options that are specific to the NDB Cluster management server program ndb_mgmd. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_mgmd), see Section 18.4.29, "Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs".

Table 18.20 Command-line options for the ndb_mgmd program

Format	Description	Added, Deprecated, or Removed
--bind-address=host	Local bind address	(Supported in all MySQL 5.6 based releases)
--config-cache[=TRUE|FALSE]	Enable management server configuration cache; true by default	(Supported in all MySQL 5.6 based releases)
--config-file=file, -f	Specify cluster configuration file; also specify --reload or --initial to override configuration cache if present	(Supported in all MySQL 5.6 based releases)
--configdir=directory, --config-dir=directory	Specify cluster management server configuration cache directory	(Supported in all MySQL 5.6 based releases)
--daemon, -d	Run ndb_mgmd in daemon mode (default)	(Supported in all MySQL 5.6 based releases)
--initial	Causes management server to reload configuration data from configuration file, bypassing configuration cache	(Supported in all MySQL 5.6 based releases)
--install[=name]	Used to install management server process as Windows service; does not apply on other platforms	(Supported in all MySQL 5.6 based releases)
--interactive	Run ndb_mgmd in interactive mode (not officially supported in production; for testing purposes only)	(Supported in all MySQL 5.6 based releases)
--log-name=name	Name to use when writing cluster log messages applying to this node	(Supported in all MySQL 5.6 based releases)
--mycnf	Read cluster configuration data from my.cnf file	(Supported in all MySQL 5.6 based releases)
--no-nodeid-checks	Do not provide any node ID checks	(Supported in all MySQL 5.6 based releases)

Format	Description	Added, Deprecated, or Removed
--nodaemon	Do not run ndb_mgmd as a daemon	(Supported in all MySQL 5.6 based releases)
--nowait-nodes=list	Do not wait for management nodes specified when starting this management server; requires --ndb-nodeid option	(Supported in all MySQL 5.6 based releases)
--print-full-config, -P	Print full configuration and exit	(Supported in all MySQL 5.6 based releases)
--reload	Causes management server to compare configuration file with configuration cache	(Supported in all MySQL 5.6 based releases)
--remove[=name]	Used to remove management server process that was previously installed as Windows service, optionally specifying name of service to be removed; does not apply on other platforms	(Supported in all MySQL 5.6 based releases)
--verbose, -v	Write additional information to log	(Supported in all MySQL 5.6 based releases)

- `--bind-address=host`

Command-Line Format	`--bind-address=host`
Type	String
Default Value	`[none]`

Causes the management server to bind to a specific network interface (host name or IP address). This option has no default value.

- `--config-cache`

Command-Line Format	`--config-cache[=TRUE|FALSE]`
Type	Boolean
Default Value	`TRUE`

This option, whose default value is `1` (or `TRUE`, or `ON`), can be used to disable the management server's configuration cache, so that it reads its configuration from `config.ini` every time it starts (see Section 18.3.3, “NDB Cluster Configuration Files”). You can do this by starting the `ndb_mgmd` process with any one of the following options:

- `--config-cache=0`
- `--config-cache=FALSE`
- `--config-cache=OFF`
- `--skip-config-cache`

Using one of the options just listed is effective only if the management server has no stored configuration at the time it is started. If the management server finds any configuration cache files, then the `--config-cache` option or the `--skip-config-cache` option is ignored. Therefore, to disable configuration caching, the option should be used the *first* time that the management server

is started. Otherwise—that is, if you wish to disable configuration caching for a management server that has *already* created a configuration cache—you must stop the management server, delete any existing configuration cache files manually, then restart the management server with `--skip-config-cache` (or with `--config-cache` set equal to 0, `OFF`, or `FALSE`).

Configuration cache files are normally created in a directory named `mysql-cluster` under the installation directory (unless this location has been overridden using the `--configdir` option). Each time the management server updates its configuration data, it writes a new cache file. The files are named sequentially in order of creation using the following format:

```
ndb_node-id_config.bin.seq-number
```

`node-id` is the management server's node ID; *`seq-number`* is a sequence number, beginning with 1. For example, if the management server's node ID is 5, then the first three configuration cache files would, when they are created, be named `ndb_5_config.bin.1`, `ndb_5_config.bin.2`, and `ndb_5_config.bin.3`.

If your intent is to purge or reload the configuration cache without actually disabling caching, you should start `ndb_mgmd` with one of the options `--reload` or `--initial` instead of `--skip-config-cache`.

To re-enable the configuration cache, simply restart the management server, but without the `--config-cache` or `--skip-config-cache` option that was used previously to disable the configuration cache.

`ndb_mgmd` does not check for the configuration directory (`--configdir`) or attempts to create one when `--skip-config-cache` is used. (Bug #13428853)

- `--config-file=filename`, `-f filename`

Command-Line Format	`--config-file=file`
Type	File name
Default Value	`[none]`

Instructs the management server as to which file it should use for its configuration file. By default, the management server looks for a file named `config.ini` in the same directory as the `ndb_mgmd` executable; otherwise the file name and location must be specified explicitly.

This option has no default value, and is ignored unless the management server is forced to read the configuration file, either because `ndb_mgmd` was started with the `--reload` or `--initial` option, or because the management server could not find any configuration cache. This option is also read if `ndb_mgmd` was started with `--config-cache=OFF`. See Section 18.3.3, "NDB Cluster Configuration Files", for more information.

Formerly, using this option together with `--initial` caused removal of the configuration cache even if the file was not found. This issue was resolved in NDB 7.3.2. (Bug #1299289)

- `--configdir=dir_name`

Command-Line Format	`--configdir=directory`
	
	`--config-dir=directory`

Type	File name
Default Value	`$INSTALLDIR/mysql-cluster`

Specifies the cluster management server's configuration cache directory. `--config-dir` is an alias for this option.

- `--daemon`, `-d`

Command-Line Format	--daemon
Type	Boolean
Default Value	TRUE

Instructs ndb_mgmd to start as a daemon process. This is the default behavior.

This option has no effect when running ndb_mgmd on Windows platforms.

- --initial

Command-Line Format	--initial
Type	Boolean
Default Value	FALSE

Configuration data is cached internally, rather than being read from the cluster global configuration file each time the management server is started (see Section 18.3.3, “NDB Cluster Configuration Files”). Using the --initial option overrides this behavior, by forcing the management server to delete any existing cache files, and then to re-read the configuration data from the cluster configuration file and to build a new cache.

This differs in two ways from the --reload option. First, --reload forces the server to check the configuration file against the cache and reload its data only if the contents of the file are different from the cache. Second, --reload does not delete any existing cache files.

If ndb_mgmd is invoked with --initial but cannot find a global configuration file, the management server cannot start.

When a management server starts, it checks for another management server in the same NDB Cluster and tries to use the other management server's configuration data. This behavior has implications when performing a rolling restart of an NDB Cluster with multiple management nodes. See Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”, for more information.

Formerly, using this option together with the --config-file option caused removal of the configuration cache even if the file was not found. Starting with NDB 7.3.2, the cache is cleared in such cases only if the configuration file is actually found. (Bug #1299289)

- --install[=*name*]

Command-Line Format	--install[=name]
Platform Specific	Windows
Type	String
Default Value	ndb_mgmd

Causes ndb_mgmd to be installed as a Windows service. Optionally, you can specify a name for the service; if not set, the service name defaults to ndb_mgmd. Although it is preferable to specify other ndb_mgmd program options in a my.ini or my.cnf configuration file, it is possible to use them together with --install. However, in such cases, the --install option must be specified first, before any other options are given, for the Windows service installation to succeed.

It is generally not advisable to use this option together with the --initial option, since this causes the configuration cache to be wiped and rebuilt every time the service is stopped and started. Care should also be taken if you intend to use any other ndb_mgmd options that affect the starting of the management server, and you should make absolutely certain you fully understand and allow for any possible consequences of doing so.

The --install option has no effect on non-Windows platforms.

- `--interactive`

Command-Line Format	`--interactive`
Type	Boolean
Default Value	`FALSE`

Starts `ndb_mgmd` in interactive mode; that is, an `ndb_mgm` client session is started as soon as the management server is running. This option does not start any other NDB Cluster nodes.

- `--log-name=name`

Command-Line Format	`--log-name=name`
Type	String
Default Value	`MgmtSrvr`

Provides a name to be used for this node in the cluster log.

- `--mycnf`

Command-Line Format	`--mycnf`
Type	Boolean
Default Value	`FALSE`

Read configuration data from the `my.cnf` file.

- `--no-nodeid-checks`

Command-Line Format	`--no-nodeid-checks`
Type	Boolean
Default Value	`FALSE`

Do not perform any checks of node IDs.

- `--nodaemon`

Command-Line Format	`--nodaemon`
Type	Boolean
Default Value	`FALSE`

Instructs `ndb_mgmd` not to start as a daemon process.

The default behavior for `ndb_mgmd` on Windows is to run in the foreground, making this option unnecessary on Windows platforms.

- `--nowait-nodes`

Command-Line Format	`--nowait-nodes=list`
Type	Numeric
Default Value	`[none]`
Minimum Value	`1`
Maximum Value	`255`

When starting an NDB Cluster is configured with two management nodes, each management server normally checks to see whether the other `ndb_mgmd` is also operational and whether the other

management server's configuration is identical to its own. However, it is sometimes desirable to start the cluster with only one management node (and perhaps to allow the other ndb_mgmd to be started later). This option causes the management node to bypass any checks for any other management nodes whose node IDs are passed to this option, permitting the cluster to start as though configured to use only the management node that was started.

For purposes of illustration, consider the following portion of a config.ini file (where we have omitted most of the configuration parameters that are not relevant to this example):

```
[ndbd]
NodeId = 1
HostName = 198.51.100.101

[ndbd]
NodeId = 2
HostName = 198.51.100.102

[ndbd]
NodeId = 3
HostName = 198.51.100.103

[ndbd]
NodeId = 4
HostName = 198.51.100.104

[ndb_mgmd]
NodeId = 10
HostName = 198.51.100.150

[ndb_mgmd]
NodeId = 11
HostName = 198.51.100.151

[api]
NodeId = 20
HostName = 198.51.100.200

[api]
NodeId = 21
HostName = 198.51.100.201
```

Assume that you wish to start this cluster using only the management server having node ID 10 and running on the host having the IP address 198.51.100.150. (Suppose, for example, that the host computer on which you intend to the other management server is temporarily unavailable due to a hardware failure, and you are waiting for it to be repaired.) To start the cluster in this way, use a command line on the machine at 198.51.100.150 to enter the following command:

```
shell> ndb_mgmd --ndb-nodeid=10 --nowait-nodes=11
```

As shown in the preceding example, when using --nowait-nodes, you must also use the --ndb-nodeid option to specify the node ID of this ndb_mgmd process.

You can then start each of the cluster's data nodes in the usual way. If you wish to start and use the second management server in addition to the first management server at a later time without restarting the data nodes, you must start each data node with a connection string that references both management servers, like this:

```
shell> ndbd -c 198.51.100.150,198.51.100.151
```

The same is true with regard to the connection string used with any mysqld processes that you wish to start as NDB Cluster SQL nodes connected to this cluster. See Section 18.3.3.3, “NDB Cluster Connection Strings”, for more information.

When used with ndb_mgmd, this option affects the behavior of the management node with regard to other management nodes only. Do not confuse it with the --nowait-nodes option used with ndbd

or ndbmtd to permit a cluster to start with fewer than its full complement of data nodes; when used with data nodes, this option affects their behavior only with regard to other data nodes.

Multiple management node IDs may be passed to this option as a comma-separated list. Each node ID must be no less than 1 and no greater than 255. In practice, it is quite rare to use more than two management servers for the same NDB Cluster (or to have any need for doing so); in most cases you need to pass to this option only the single node ID for the one management server that you do not wish to use when starting the cluster.

Note

When you later start the “missing” management server, its configuration must match that of the management server that is already in use by the cluster. Otherwise, it fails the configuration check performed by the existing management server, and does not start.

- --print-full-config, -P

Command-Line Format	--print-full-config
Type	Boolean
Default Value	FALSE

Shows extended information regarding the configuration of the cluster. With this option on the command line the ndb_mgmd process prints information about the cluster setup including an extensive list of the cluster configuration sections as well as parameters and their values. Normally used together with the --config-file (-f) option.

- --reload

Command-Line Format	--reload
Type	Boolean
Default Value	FALSE

NDB Cluster configuration data is stored internally rather than being read from the cluster global configuration file each time the management server is started (see Section 18.3.3, “NDB Cluster Configuration Files”). Using this option forces the management server to check its internal data store against the cluster configuration file and to reload the configuration if it finds that the configuration file does not match the cache. Existing configuration cache files are preserved, but not used.

This differs in two ways from the --initial option. First, --initial causes all cache files to be deleted. Second, --initial forces the management server to re-read the global configuration file and construct a new cache.

If the management server cannot find a global configuration file, then the --reload option is ignored.

When --reload is used, the management server must be able to communicate with data nodes and any other management servers in the cluster before it attempts to read the global configuration file; otherwise, the management server fails to start. This can happen due to changes in the networking environment, such as new IP addresses for nodes or an altered firewall configuration. In such cases, you must use --initial instead to force the exsiting cached configuration to be discarded and reloaded from the file. See Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”, for additional information.

- --remove{=name]

Command-Line Format	--remove[=name]

Platform Specific	Windows
Type	String
Default Value	ndb_mgmd

Remove a management server process that has been installed as a Windows service, optionally specifying the name of the service to be removed. Applies only to Windows platforms.

- --verbose, -v

Command-Line Format	--verbose
Type	Boolean
Default Value	FALSE

Remove a management server process that has been installed as a Windows service, optionally specifying the name of the service to be removed. Applies only to Windows platforms.

It is not strictly necessary to specify a connection string when starting the management server. However, if you are using more than one management server, a connection string should be provided and each node in the cluster should specify its node ID explicitly.

See Section 18.3.3.3, “NDB Cluster Connection Strings”, for information about using connection strings. Section 18.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”, describes other options for ndb_mgmd.

The following files are created or used by ndb_mgmd in its starting directory, and are placed in the DataDir as specified in the config.ini configuration file. In the list that follows, *node_id* is the unique node identifier.

- config.ini is the configuration file for the cluster as a whole. This file is created by the user and read by the management server. Section 18.3, “Configuration of NDB Cluster”, discusses how to set up this file.
- ndb_*node_id*_cluster.log is the cluster events log file. Examples of such events include checkpoint startup and completion, node startup events, node failures, and levels of memory usage. A complete listing of cluster events with descriptions may be found in Section 18.5, “Management of NDB Cluster”.

  By default, when the size of the cluster log reaches one million bytes, the file is renamed to ndb_*node_id*_cluster.log.*seq_id*, where *seq_id* is the sequence number of the cluster log file. (For example: If files with the sequence numbers 1, 2, and 3 already exist, the next log file is named using the number 4.) You can change the size and number of files, and other characteristics of the cluster log, using the LogDestination configuration parameter.
- ndb_*node_id*_out.log is the file used for stdout and stderr when running the management server as a daemon.
- ndb_*node_id*.pid is the process ID file used when running the management server as a daemon.

18.4.5 ndb_mgm — The NDB Cluster Management Client

The ndb_mgm management client process is actually not needed to run the cluster. Its value lies in providing a set of commands for checking the cluster's status, starting backups, and performing other administrative functions. The management client accesses the management server using a C API. Advanced users can also employ this API for programming dedicated management processes to perform tasks similar to those performed by ndb_mgm.

To start the management client, it is necessary to supply the host name and port number of the management server:

```
shell> ndb_mgm [host_name [port_num]]
```

For example:

```
shell> ndb_mgm ndb_mgmd.mysql.com 1186
```

The default host name and port number are `localhost` and 1186, respectively.

The following table includes options that are specific to the NDB Cluster management client program `ndb_mgm`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_mgm`), see Section 18.4.29, "Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs".

Table 18.21 Command-line options for the ndb_mgm program

Format	Description	Added, Deprecated, or Removed
`--connect-retries=#`	Set number of times to retry connection before giving up; 0 means 1 attempt only (and no retries)	ADDED: NDB 7.4.9
`--try-reconnect=#`, `-t`	Set number of times to retry connection before giving up; synonym for --connect-retries	DEPRECATED: NDB 7.4.9
`--execute=name`, `-e`	Execute command and exit	(Supported in all MySQL 5.6 based releases)

- `--connect-retries=#`

Command-Line Format	`--connect-retries=#`
Introduced	5.6.28-ndb-7.4.9
Type	Numeric
Default Value	`3`
Minimum Value	`0`
Maximum Value	`4294967295`

This option specifies the number of times following the first attempt to retry a connection before giving up (the client always tries the connection at least once). The length of time to wait per attempt is set using `--connect-retry-delay`.

This option was added in NDB 7.4.9, and is synonymous with the `--try-reconnect` option, which is now deprecated.

The default for this option this option differs from its default when used with other `NDB` programs. See Section 18.4.29, "Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs", for more information.

- `--execute=command`, `-e command`

Command-Line Format	`--execute=name`

This option can be used to send a command to the NDB Cluster management client from the system shell. For example, either of the following is equivalent to executing `SHOW` in the management client:

```
shell> ndb_mgm -e "SHOW"

shell> ndb_mgm --execute="SHOW"
```

This is analogous to how the `--execute` or `-e` option works with the `mysql` command-line client. See Section 4.2.2.1, “Using Options on the Command Line”.

Note

If the management client command to be passed using this option contains any space characters, then the command *must* be enclosed in quotation marks. Either single or double quotation marks may be used. If the management client command contains no space characters, the quotation marks are optional.

- `--try-reconnect=number`

Command-Line Format	`--try-reconnect=#`
Deprecated	5.6.28-ndb-7.4.9
Type	Integer
Default Value	`3`
Minimum Value	`0`
Maximum Value	`4294967295`

If the connection to the management server is broken, the node tries to reconnect to it every 5 seconds until it succeeds. By using this option, it is possible to limit the number of attempts to *`number`* before giving up and reporting an error instead.

This option is deprecated in NDB 7.4.9 and later, and is superseded by `--connect-retries`.

Additional information about using `ndb_mgm` can be found in Section 18.5.1, “Commands in the NDB Cluster Management Client”.

18.4.6 `ndb_blob_tool` — Check and Repair BLOB and TEXT columns of NDB Cluster Tables

This tool can be used to check for and remove orphaned BLOB column parts from `NDB` tables, as well as to generate a file listing any orphaned parts. It is sometimes useful in diagnosing and repairing corrupted or damaged `NDB` tables containing `BLOB` or `TEXT` columns.

The basic syntax for `ndb_blob_tool` is shown here:

```
ndb_blob_tool [options] table [column, ...]
```

Unless you use the `--help` option, you must specify an action to be performed by including one or more of the options `--check-orphans`, `--delete-orphans`, or `--dump-file`. These options cause `ndb_blob_tool` to check for orphaned BLOB parts, remove any orphaned BLOB parts, and generate a dump file listing orphaned BLOB parts, respectively, and are described in more detail later in this section.

You must also specify the name of a table when invoking `ndb_blob_tool`. In addition, you can optionally follow the table name with the (comma-separated) names of one or more `BLOB` or `TEXT` columns from that table. If no columns are listed, the tool works on all of the table's `BLOB` and `TEXT` columns. If you need to specify a database, use the `--database` (`-d`) option.

The `--verbose` option provides additional information in the output about the tool's progress.

The following table includes options that are specific to `ndb_blob_tool`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_blob_tool`), see Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 18.22 Command-line options for the ndb_blob_tool program

Format	Description	Added, Deprecated, or Removed
--check-orphans	Check for blob parts having no corresponding inline parts	(Supported in all MySQL 5.6 based releases)
--database=db_name, -d	Database to find the table in	(Supported in all MySQL 5.6 based releases)
--delete-orphans	Delete blob parts having no corresponding inline parts	(Supported in all MySQL 5.6 based releases)
--dump-file=file	Write orphan keys to specified file	(Supported in all MySQL 5.6 based releases)
--verbose, -v	Verbose output	(Supported in all MySQL 5.6 based releases)

- `--check-orphans`

Command-Line Format	`--check-orphans`
Type	Boolean
Default Value	`FALSE`

Check for orphaned BLOB parts in NDB Cluster tables.

- `--database=db_name`, `-d`

Command-Line Format	`--database=db_name`
Type	String
Default Value	`[none]`

Specify the database to find the table in.

- `--delete-orphans`

Command-Line Format	`--delete-orphans`
Type	Boolean
Default Value	`FALSE`

Remove orphaned BLOB parts from NDB Cluster tables.

- `--dump-file=file`

Command-Line Format	`--dump-file=file`
Type	File name
Default Value	`[none]`

Writes a list of orphaned BLOB column parts to *`file`*. The information written to the file includes the table key and BLOB part number for each orphaned BLOB part.

- `--verbose`

Command-Line Format	`--verbose`
Type	Boolean

Default Value	FALSE

Provide extra information in the tool's output regarding its progress.

Example

First we create an NDB table in the test database, using the CREATE TABLE statement shown here:

```
USE test;

CREATE TABLE btest (
    c0 BIGINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    c1 TEXT,
    c2 BLOB
)   ENGINE=NDB;
```

Then we insert a few rows into this table, using a series of statements similar to this one:

```
INSERT INTO btest VALUES (NULL, 'x', REPEAT('x', 1000));
```

When run with --check-orphans against this table, ndb_blob_tool generates the following output:

```
shell> ndb_blob_tool --check-orphans --verbose -d test btest
connected
processing 2 blobs
processing blob #0 c1 NDB$BLOB_19_1
NDB$BLOB_19_1: nextResult: res=1
total parts: 0
orphan parts: 0
processing blob #1 c2 NDB$BLOB_19_2
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=1
total parts: 10
orphan parts: 0
disconnected

NDBT_ProgramExit: 0 - OK
```

The tool reports that there are no NDB BLOB column parts associated with column c1, even though c1 is a TEXT column. This is due to the fact that, in an NDB table, only the first 256 bytes of a BLOB or TEXT column value are stored inline, and only the excess, if any, is stored separately; thus, if there are no values using more than 256 bytes in a given column of one of these types, no BLOB column parts are created by NDB for this column. See Section 11.6, “Data Type Storage Requirements”, for more information.

18.4.7 ndb_config — Extract NDB Cluster Configuration Information

This tool extracts current configuration information for data nodes, SQL nodes, and API nodes from one of a number of sources: an NDB Cluster management node, or its config.ini or my.cnf file. By default, the management node is the source for the configuration data; to override the default, execute ndb_config with the --config-file or --mycnf option. It is also possible to use a data node as the source by specifying its node ID with --config_from_node=*node_id*.

ndb_config can also provide an offline dump of all configuration parameters which can be used, along with their default, maximum, and minimum values and other information. The dump can be produced in either text or XML format; for more information, see the discussion of the --configinfo and --xml options later in this section).

You can filter the results by section (DB, SYSTEM, or CONNECTIONS) using one of the options --nodes, --system, or --connections.

The following table includes options that are specific to ndb_config. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_config), see Section 18.4.29, "Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs".

Table 18.23 Command-line options for the ndb_config program

Format	Description	Added, Deprecated, or Removed
--config-file=file_name	Set the path to config.ini file	(Supported in all MySQL 5.6 based releases)
--config-from-node=#	Obtain configuration data from the node having this ID (must be a data node)	(Supported in all MySQL 5.6 based releases)
--configinfo	Dumps information about all NDB configuration parameters in text format with default, maximum, and minimum values. Use with --xml to obtain XML output	(Supported in all MySQL 5.6 based releases)
--connections	Print connections information ([tcp], [tcp default], [sci], [sci default], [shm], or [shm default] sections of cluster configuration file) only. Cannot be used with --system or --nodes	(Supported in all MySQL 5.6 based releases)
--diff-default	Print only configuration parameters that have non-default values	ADDED: NDB 7.4.16
--fields=string, -f	Field separator	(Supported in all MySQL 5.6 based releases)
--host=name	Specify host	(Supported in all MySQL 5.6 based releases)
--mycnf	Read configuration data from my.cnf file	(Supported in all MySQL 5.6 based releases)
--nodeid, --id (<=7.5.0)	Get configuration of node with this ID	(Supported in all MySQL 5.6 based releases)
--nodes	Print node information ([ndbd] or [ndbd default] section of cluster configuration file) only. Cannot be used with --system or --connections	(Supported in all MySQL 5.6 based releases)
-c	Short form for --ndb-connectstring	(Supported in all MySQL 5.6 based releases)
--query=string, -q	One or more query options (attributes)	(Supported in all MySQL 5.6 based releases)
--query-all, -a	Dumps all parameters and values to a single comma-delimited string	(Supported in all MySQL 5.6 based releases)

Format	Description	Added, Deprecated, or Removed
--rows=string, -r	Row separator	(Supported in all MySQL 5.6 based releases)
--system	Print SYSTEM section information only (see ndb_config --configinfo output). Cannot be used with --nodes or --connections	(Supported in all MySQL 5.6 based releases)
--type=name	Specify node type	(Supported in all MySQL 5.6 based releases)
--configinfo --xml	Use --xml with --configinfo to obtain a dump of all NDB configuration parameters in XML format with default, maximum, and minimum values	(Supported in all MySQL 5.6 based releases)

- --configinfo

  The --configinfo option causes ndb_config to dump a list of each NDB Cluster configuration parameter supported by the NDB Cluster distribution of which ndb_config is a part, including the following information:

  - A brief description of each parameter's purpose, effects, and usage
  - The section of the config.ini file where the parameter may be used
  - The parameter's data type or unit of measurement
  - Where applicable, the parameter's default, minimum, and maximum values
  - NDB Cluster release version and build information

  By default, this output is in text format. Part of this output is shown here:

```
shell> ndb_config --configinfo

****** SYSTEM ******

Name (String)
Name of system (NDB Cluster)
MANDATORY

PrimaryMGMNode (Non-negative Integer)
Node id of Primary ndb_mgmd(MGM) node
Default: 0 (Min: 0, Max: 4294967039)

ConfigGenerationNumber (Non-negative Integer)
Configuration generation number
Default: 0 (Min: 0, Max: 4294967039)

****** DB ******

MaxNoOfSubscriptions (Non-negative Integer)
Max no of subscriptions (default 0 == MaxNoOfTables)
Default: 0 (Min: 0, Max: 4294967039)

MaxNoOfSubscribers (Non-negative Integer)
Max no of subscribers (default 0 == 2 * MaxNoOfTables)
Default: 0 (Min: 0, Max: 4294967039)
```

...

Use this option together with the `--xml` option to obtain output in XML format.

- `--config-file=path-to-file`

Command-Line Format	`--config-file=file_name`
Type	File name
Default Value	

Gives the path to the management server's configuration file (`config.ini`). This may be a relative or absolute path. If the management node resides on a different host from the one on which `ndb_config` is invoked, then an absolute path must be used.

- `--config_from_node=#`

Command-Line Format	`--config-from-node=#`
Type	Numeric
Default Value	`none`
Minimum Value	`1`
Maximum Value	`48`

Obtain the cluster's configuration data from the data node that has this ID.

If the node having this ID is not a data node, `ndb_config` fails with an error. (To obtain configuration data from the management node instead, simply omit this option.)

- `--connections`

Command-Line Format	`--connections`
Type	Boolean
Default Value	`FALSE`

Tells `ndb_config` to print `CONNECTIONS` information only—that is, information about parameters found in the `[tcp]`, `[tcp default]`, `[shm]`, or `[shm default]` sections of the cluster configuration file (see Section 18.3.3.9, “NDB Cluster TCP/IP Connections”, and Section 18.3.3.11, “NDB Cluster Shared-Memory Connections”, for more information).

This option is mutually exclusive with `--nodes` and `--system`; only one of these 3 options can be used.

- `--diff-default`

Command-Line Format	`--diff-default`
Introduced	5.6.36-ndb-7.4.16
Type	Boolean
Default Value	`FALSE`

Print only configuration parameters that have non-default values.

- `--fields=delimiter`, `-f delimiter`

Command-Line Format	`--fields=string`
Type	String
Default Value	

Specifies a *delimiter* string used to separate the fields in the result. The default is , (the comma character).

Note

If the *delimiter* contains spaces or escapes (such as \n for the linefeed character), then it must be quoted.

- --host=*hostname*

Command-Line Format	--host=name
Type	String
Default Value	

Specifies the host name of the node for which configuration information is to be obtained.

Note

While the hostname localhost usually resolves to the IP address 127.0.0.1, this may not necessarily be true for all operating platforms and configurations. This means that it is possible, when localhost is used in config.ini, for ndb_config --host=localhost to fail if ndb_config is run on a different host where localhost resolves to a different address (for example, on some versions of SUSE Linux, this is 127.0.0.2). In general, for best results, you should use numeric IP addresses for all NDB Cluster configuration values relating to hosts, or verify that all NDB Cluster hosts handle localhost in the same fashion.

- --mycnf

Command-Line Format	--mycnf
Type	Boolean
Default Value	FALSE

Read configuration data from the my.cnf file.

- --nodeid=*node_id*, --id=*node_id*

Command-Line Format	--ndb-nodeid=#
Type	Numeric
Default Value	0

Specify the node ID of the node for which configuration information is to be obtained. --nodeid is the preferred form; --id is deprecated and subject to removal in a later version of NDB Cluster.

- --nodes

Command-Line Format	--nodes
Type	Boolean

Default Value	FALSE

Tells ndb_config to print information relating only to parameters defined in an [ndbd] or [ndbd default] section of the cluster configuration file (see Section 18.3.3.6, "Defining NDB Cluster Data Nodes").

This option is mutually exclusive with --connections and --system; only one of these 3 options can be used.

- --ndb-connectstring=*connection_string*, -c *connection_string*

Command-Line Format	--ndb-connectstring=connectstring --connect-string=connectstring
Type	String
Default Value	localhost:1186

Specifies the connection string to use in connecting to the management server. The format for the connection string is the same as described in Section 18.3.3.3, "NDB Cluster Connection Strings", and defaults to localhost:1186.

- --query=*query-options*, -q *query-options*

Command-Line Format	--query=string
Type	String
Default Value	

This is a comma-delimited list of *query options*—that is, a list of one or more node attributes to be returned. These include nodeid (node ID), type (node type—that is, ndbd, mysqld, or ndb_mgmd), and any configuration parameters whose values are to be obtained.

For example, --query=nodeid,type,datamemory,datadir returns the node ID, node type, DataMemory, and DataDir for each node.

id is accepted as a synonym for nodeid, but is deprecated and subject to removal in a later version of NDB Cluster.

Note

If a given parameter is not applicable to a certain type of node, than an empty string is returned for the corresponding value. See the examples later in this section for more information.

- --query-all, -a

Command-Line Format	--query-all

Returns a comma-delimited list of all query options (node attributes; note that this list is a single string.

This option was introduced in NDB 7.4.16 (Bug #60095, Bug #11766869).

- --rows=*separator*, -r *separator*

Command-Line Format	--rows=string
Type	String

Default Value	

Specifies a *`separator`* string used to separate the rows in the result. The default is a space character.

Note

If the *`separator`* contains spaces or escapes (such as `\n` for the linefeed character), then it must be quoted.

- `--system`

Command-Line Format	`--system`
Type	Boolean
Default Value	`FALSE`

Tells `ndb_config` to print `SYSTEM` information only. This consists of system variables that cannot be changed at run time; thus, there is no corresponding section of the cluster configuration file for them. They can be seen (prefixed with `****** SYSTEM ******`) in the output of `ndb_config --configinfo`.

This option is mutually exclusive with `--nodes` and `--connections`; only one of these 3 options can be used.

- `--type=`*`node_type`*

Command-Line Format	`--type=name`
Type	Enumeration
Default Value	`[none]`

Valid Values	`ndbd`
	`mysqld`
	`ndb_mgmd`

Filters results so that only configuration values applying to nodes of the specified *`node_type`* (`ndbd`, `mysqld`, or `ndb_mgmd`) are returned.

- `--usage`, `--help`, or `-?`

Command-Line Format	`--help` `--usage`

Causes `ndb_config` to print a list of available options, and then exit.

- `--version`, `-V`

Command-Line Format	`--version`

Causes `ndb_config` to print a version information string, and then exit.

- `--configinfo --xml`

Command-Line Format	`--configinfo --xml`
Type	Boolean
Default Value	`false`

Cause ndb_config --configinfo to provide output as XML by adding this option. A portion of such output is shown in this example:

```
shell> ndb_config --configinfo --xml

<configvariables protocolversion="1" ndbversionstring="5.6.51-ndb-7.3.32"
                   ndbversion="458758" ndbversionmajor="7" ndbversionminor="0"
                   ndbversionbuild="6">
  <section name="SYSTEM">
    <param name="Name" comment="Name of system (NDB Cluster)" type="string"
              mandatory="true"/>
    <param name="PrimaryMGMNode" comment="Node id of Primary ndb_mgmd(MGM) node"
              type="unsigned" default="0" min="0" max="4294967039"/>
    <param name="ConfigGenerationNumber" comment="Configuration generation number"
              type="unsigned" default="0" min="0" max="4294967039"/>
  </section>
  <section name="NDBD">
    <param name="MaxNoOfSubscriptions"
              comment="Max no of subscriptions (default 0 == MaxNoOfTables)"
              type="unsigned" default="0" min="0" max="4294967039"/>
    <param name="MaxNoOfSubscribers"
              comment="Max no of subscribers (default 0 == 2 * MaxNoOfTables)"
              type="unsigned" default="0" min="0" max="4294967039"/>

    …

  </section>

  …

</configvariables>
```

Note

Normally, the XML output produced by ndb_config --configinfo --xml is formatted using one line per element; we have added extra whitespace in the previous example, as well as the next one, for reasons of legibility. This should not make any difference to applications using this output, since most XML processors either ignore nonessential whitespace as a matter of course, or can be instructed to do so.

The XML output also indicates when changing a given parameter requires that data nodes be restarted using the --initial option. This is shown by the presence of an initial="true" attribute in the corresponding <param> element. In addition, the restart type (system or node) is also shown; if a given parameter requires a system restart, this is indicated by the presence of a restart="system" attribute in the corresponding <param> element. For example, changing the value set for the Diskless parameter requires a system initial restart, as shown here (with the restart and initial attributes highlighted for visibility):

```
<param name="Diskless" comment="Run wo/ disk" type="bool" default="false"
          restart="system" initial="true"/>
```

Currently, no initial attribute is included in the XML output for <param> elements corresponding to parameters which do not require initial restarts; in other words, initial="false" is the default, and the value false should be assumed if the attribute is not present. Similarly, the default restart type is node (that is, an online or “rolling” restart of the cluster), but the restart attribute is included only if the restart type is system (meaning that all cluster nodes must be shut down at the same time, then restarted).

Beginning with NDB 7.4.7, deprecated parameters are indicated in the XML output with the addition of a deprecated attribute, as shown here:

```
<param name="NoOfDiskPagesToDiskAfterRestartACC" comment="DiskCheckpointSpeed"
```

```
        type="unsigned" default="20" min="1" max="4294967039" deprecated="true"/>
```

In such cases, the `comment` refers to one or more parameters that supersede the deprecated parameter. Similarly to `initial`, the `deprecated` attribute is indicated only when the parameter is deprecated, with `deprecated="true"`, and does not appear at all for parameters which are not deprecated. (Bug #21127135)

Important

The `--xml` option can be used only with the `--configinfo` option. Using `--xml` without `--configinfo` fails with an error.

Unlike the options used with this program to obtain current configuration data, `--configinfo` and `--xml` use information obtained from the NDB Cluster sources when `ndb_config` was compiled. For this reason, no connection to a running NDB Cluster or access to a `config.ini` or `my.cnf` file is required for these two options.

Combining other `ndb_config` options (such as `--query` or `--type`) with `--configinfo` (with or without the `--xml` option) is not supported. Currently, if you attempt to do so, the usual result is that all other options besides `--configinfo` or `--xml` are simply ignored. *However, this behavior is not guaranteed and is subject to change at any time*. In addition, since `ndb_config`, when used with the `--configinfo` option, does not access the NDB Cluster or read any files, trying to specify additional options such as `--ndb-connectstring` or `--config-file` with `--configinfo` serves no purpose.

Examples

1. To obtain the node ID and type of each node in the cluster:

```
shell> ./ndb_config -q nodeid,type --fields=':' --rows='\n'
1:ndbd
2:ndbd
3:ndbd
4:ndbd
5:ndb_mgmd
6:mysqld
7:mysqld
8:mysqld
9:mysqld
```

   In this example, we used the `--fields` options to separate the node ID and type of each node with a colon character (`:`), and the `--rows` options to place the values for each node on a new line in the output.

2. To produce a connection string that can be used by data, SQL, and API nodes to connect to the management server:

```
shell> ./ndb_config --config-file=usr/local/mysql/cluster-data/config.ini \
--query=hostname,portnumber --fields=: --rows=, --type=ndb_mgmd
198.51.100.179:1186
```

3. This invocation of `ndb_config` checks only data nodes (using the `--type` option), and shows the values for each node's ID and host name, as well as the values set for its `DataMemory`, `IndexMemory`, and `DataDir` parameters:

```
shell> ./ndb_config --type=ndbd --query=nodeid,host,datamemory,indexmemory,datadir -f ' : ' -r '\n'
1 : 198.51.100.193 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
2 : 198.51.100.112 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
3 : 198.51.100.176 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
4 : 198.51.100.119 : 83886080 : 18874368 : /usr/local/mysql/cluster-data
```

   In this example, we used the short options `-f` and `-r` for setting the field delimiter and row separator, respectively, as well as the short option `-q` to pass a list of parameters to be obtained.

4. To exclude results from any host except one in particular, use the `--host` option:

```
shell> ./ndb_config --host=198.51.100.176 -f : -r '\n' -q id,type
3:ndbd
5:ndb_mgmd
```

In this example, we also used the short form `-q` to determine the attributes to be queried.

Similarly, you can limit results to a node with a specific ID using the `--nodeid` option.

18.4.8 ndb_cpcd — Automate Testing for NDB Development

A utility having this name was formerly part of an internal automated test framework used in testing and debugging NDB Cluster. It is no longer included in NDB Cluster distributions provided by Oracle.

18.4.9 ndb_delete_all — Delete All Rows from an NDB Table

`ndb_delete_all` deletes all rows from the given `NDB` table. In some cases, this can be much faster than `DELETE` or even `TRUNCATE TABLE`.

Usage

```
ndb_delete_all -c connection_string tbl_name -d db_name
```

This deletes all rows from the table named *`tbl_name`* in the database named *`db_name`*. It is exactly equivalent to executing `TRUNCATE db_name.tbl_name` in MySQL.

The following table includes options that are specific to `ndb_delete_all`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_delete_all`), see Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 18.24 Command-line options for the ndb_delete_all program

Format	Description	Added, Deprecated, or Removed
`--database=dbname`, `-d`	Name of the database in which the table is found	(Supported in all MySQL 5.6 based releases)
`--transactional`, `-t`	Perform the delete in a single transaction (may run out of operations)	(Supported in all MySQL 5.6 based releases)
`--tupscan`	Run tup scan	(Supported in all MySQL 5.6 based releases)
`--diskscan`	Run disk scan	(Supported in all MySQL 5.6 based releases)

- `--transactional`, `-t`

  Use of this option causes the delete operation to be performed as a single transaction.

Warning

With very large tables, using this option may cause the number of operations available to the cluster to be exceeded.

18.4.10 ndb_desc — Describe NDB Tables

`ndb_desc` provides a detailed description of one or more `NDB` tables.

Usage

```
ndb_desc -c connection_string tbl_name -d db_name [options]
```

```
ndb_desc -c connection_string index_name -d db_name -t tbl_name
```

Additional options that can be used with ndb_desc are listed later in this section.

Sample Output

MySQL table creation and population statements:

```
USE test;

CREATE TABLE fish (
    id INT(11) NOT NULL AUTO_INCREMENT,
    name VARCHAR(20) NOT NULL,
    length_mm INT(11) NOT NULL,
    weight_gm INT(11) NOT NULL,

    PRIMARY KEY pk (id),
    UNIQUE KEY uk (name)
) ENGINE=NDB;

INSERT INTO fish VALUES
    ('','guppy', 35, 2), ('','tuna', 2500, 150000),
    ('','shark', 3000, 110000), ('','manta ray', 1500, 50000),
    ('','grouper', 900, 125000), ('','puffer', 250, 2500);
```

Output from ndb_desc:

```
shell> ./ndb_desc -c localhost fish -d test -p
-- fish --
Version: 2
Fragment type: 9
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 4
Number of primary keys: 1
Length of frm data: 311
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
FragmentCount: 2
TableStatus: Retrieved
-- Attributes --
id Int PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
name Varchar(20;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
length_mm Int NOT NULL AT=FIXED ST=MEMORY
weight_gm Int NOT NULL AT=FIXED ST=MEMORY

-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex
uk$unique(name) - UniqueHashIndex
uk(name) - OrderedIndex

-- Per partition info --
Partition  Row count  Commit count  Frag fixed memory ...
0          2          2             32768             ...
1          4          4             32768             ...

... Frag varsized memory  Extent_space  Free extent_space
... 32768                 0             0
... 32768                 0             0

NDBT_ProgramExit: 0 - OK
```

Information about multiple tables can be obtained in a single invocation of ndb_desc by using their names, separated by spaces. All of the tables must be in the same database.

You can obtain additional information about a specific index using the `--table` (short form: `-t`) option and supplying the name of the index as the first argument to `ndb_desc`, as shown here:

```
shell> ./ndb_desc uk -d test -t fish
-- uk --
Version: 3
Base table: fish
Number of attributes: 1
Logging: 0
Index type: OrderedIndex
Index status: Retrieved
-- Attributes --
name Varchar(20;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
-- IndexTable 10/uk --
Version: 3
Fragment type: FragUndefined
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: yes
Number of attributes: 2
Number of primary keys: 1
Length of frm data: 0
Row Checksum: 1
Row GCI: 1
SingleUserMode: 2
ForceVarPart: 0
FragmentCount: 4
ExtraRowGciBits: 0
ExtraRowAuthorBits: 0
TableStatus: Retrieved
-- Attributes --
name Varchar(20;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
NDB$TNODE Unsigned [64] PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY
-- Indexes --
PRIMARY KEY(NDB$TNODE) - UniqueHashIndex

NDBT_ProgramExit: 0 - OK
```

When an index is specified in this way, the `--extra-partition-info` and `--extra-node-info` options have no effect.

The `Version` column in the output contains the table's schema object version. For information about interpreting this value, see NDB Schema Object Versions.

The `Extent_space` and `Free extent_space` columns are applicable only to `NDB` tables having columns on disk; for tables having only in-memory columns, these columns always contain the value `0`.

To illustrate their use, we modify the previous example. First, we must create the necessary Disk Data objects, as shown here:

```
CREATE LOGFILE GROUP lg_1
    ADD UNDOFILE 'undo_1.log'
    INITIAL_SIZE 16M
    UNDO_BUFFER_SIZE 2M
    ENGINE NDB;

ALTER LOGFILE GROUP lg_1
    ADD UNDOFILE 'undo_2.log'
    INITIAL_SIZE 12M
    ENGINE NDB;

CREATE TABLESPACE ts_1
    ADD DATAFILE 'data_1.dat'
    USE LOGFILE GROUP lg_1
    INITIAL_SIZE 32M
    ENGINE NDB;

ALTER TABLESPACE ts_1
    ADD DATAFILE 'data_2.dat'
```

```
    INITIAL_SIZE 48M
    ENGINE NDB;
```

(For more information on the statements just shown and the objects created by them, see Section 18.5.10.1, “NDB Cluster Disk Data Objects”, as well as Section 13.1.14, “CREATE LOGFILE GROUP Statement”, and Section 13.1.18, “CREATE TABLESPACE Statement”.)

Now we can create and populate a version of the `fish` table that stores 2 of its columns on disk (deleting the previous version of the table first, if it already exists):

```
CREATE TABLE fish (
    id INT(11) NOT NULL AUTO_INCREMENT,
    name VARCHAR(20) NOT NULL,
    length_mm INT(11) NOT NULL,
    weight_gm INT(11) NOT NULL,

    PRIMARY KEY pk (id),
    UNIQUE KEY uk (name)
) TABLESPACE ts_1 STORAGE DISK
ENGINE=NDB;

INSERT INTO fish VALUES
    ('','guppy', 35, 2), ('','tuna', 2500, 150000),
    ('','shark', 3000, 110000), ('','manta ray', 1500, 50000),
    ('','grouper', 900, 125000), ('','puffer', 250, 2500);
```

When run against this version of the table, `ndb_desc` displays the following output:

```
shell> ./ndb_desc -c localhost fish -d test -p
-- fish --
Version: 3
Fragment type: 9
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 4
Number of primary keys: 1
Length of frm data: 321
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
FragmentCount: 2
TableStatus: Retrieved
-- Attributes --
id Int PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
name Varchar(20;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
length_mm Int NOT NULL AT=FIXED ST=DISK
weight_gm Int NOT NULL AT=FIXED ST=DISK

-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex
uk$unique(name) - UniqueHashIndex
uk(name) - OrderedIndex
-- Per partition info --
Partition  Row count  Commit count  Frag fixed memory ...
0          2          2             32768             ...
1          4          4             32768             ...

... Frag varsized memory  Extent_space  Free extent_space
... 32768                 0             0
... 32768                 0             0

NDBT_ProgramExit: 0 - OK
```

This means that 1048576 bytes are allocated from the tablespace for this table on each partition, of which 1044440 bytes remain free for additional storage. In other words, 1048576 - 1044440 = 4136 bytes per partition is currently being used to store the data from this table's disk-based

columns. The number of bytes shown as `Free extent_space` is available for storing on-disk column data from the `fish` table only; for this reason, it is not visible when selecting from the `INFORMATION_SCHEMA.FILES` table.

The following table includes options that are specific to `ndb_desc`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_desc`), see Section 18.4.29, "Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs".

Table 18.25 Command-line options for the ndb_desc program

Format	Description	Added, Deprecated, or Removed
`--blob-info`, `-b`	Include partition information for BLOB tables in output. Requires that the -p option also be used	(Supported in all MySQL 5.6 based releases)
`--database=dbname`, `-d`	Name of database containing table	(Supported in all MySQL 5.6 based releases)
`--extra-node-info`, `-n`	Include partition-to-data-node mappings in output. Requires that the -p option also be used	(Supported in all MySQL 5.6 based releases)
`--extra-partition-info`, `-p`	Display information about partitions	(Supported in all MySQL 5.6 based releases)
`--retries=#`, `-r`	Number of times to retry the connection (once per second)	(Supported in all MySQL 5.6 based releases)
`--table=tbl_name`, `-t`	Specify the table in which to find an index. When this option is used, -p and -n have no effect and are ignored	(Supported in all MySQL 5.6 based releases)
`--unqualified`, `-u`	Use unqualified table names	(Supported in all MySQL 5.6 based releases)

- `--blob-info`, `-b`

  Include information about subordinate `BLOB` and `TEXT` columns.

  Use of this option also requires the use of the `--extra-partition-info` (`-p`) option.

- `--database=db_name`, `-d`

  Specify the database in which the table should be found.

- `--extra-node-info`, `-n`

  Include information about the mappings between table partitions and the data nodes upon which they reside. This information can be useful for verifying distribution awareness mechanisms and supporting more efficient application access to the data stored in NDB Cluster.

  Use of this option also requires the use of the `--extra-partition-info` (`-p`) option.

- `--extra-partition-info`, `-p`

  Print additional information about the table's partitions.

- `--retries=#`, `-r`

  Try to connect this many times before giving up. One connect attempt is made per second.

- `--table=tbl_name`, `-t`

  Specify the table in which to look for an index.

- `--unqualified`, `-u`

  Use unqualified table names.

18.4.11 ndb_drop_index — Drop Index from an NDB Table

`ndb_drop_index` drops the specified index from an `NDB` table. *It is recommended that you use this utility only as an example for writing NDB API applications*—see the Warning later in this section for details.

Usage

```
ndb_drop_index -c connection_string table_name index -d db_name
```

The statement shown above drops the index named *`index`* from the *`table`* in the *`database`*.

The following table includes options that are specific to `ndb_drop_index`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_drop_index`), see Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 18.26 Command-line options for the ndb_drop_index program

Format	Description	Added, Deprecated, or Removed
`--database=dbname`, `-d`	Name of database in which table is found	(Supported in all MySQL 5.6 based releases)

Warning

Operations performed on Cluster table indexes using the NDB API are not visible to MySQL and make the table unusable by a MySQL server. If you use this program to drop an index, then try to access the table from an SQL node, an error results, as shown here:

```
shell> ./ndb_drop_index -c localhost dogs ix -d ctest1
Dropping index dogs/idx...OK

NDBT_ProgramExit: 0 - OK

shell> ./mysql -u jon -p ctest1
Enter password: *******
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 5.6.51-ndb-7.3.32

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SHOW TABLES;
+------------------+
| Tables_in_ctest1 |
+------------------+
| a                |
| bt1              |
| bt2              |
| dogs             |
| employees        |
| fish             |
+------------------+
```

```
6 rows in set (0.00 sec)

mysql> SELECT * FROM dogs;
ERROR 1296 (HY000): Got error 4243 'Index not found' from NDBCLUSTER
```

In such a case, your *only* option for making the table available to MySQL again is to drop the table and re-create it. You can use either the SQL statementDROP TABLE or the ndb_drop_table utility (see Section 18.4.12, “ndb_drop_table — Drop an NDB Table”) to drop the table.

18.4.12 ndb_drop_table — Drop an NDB Table

ndb_drop_table drops the specified NDB table. (If you try to use this on a table created with a storage engine other than NDB, the attempt fails with the error 723: No such table exists.) This operation is extremely fast; in some cases, it can be an order of magnitude faster than using a MySQL DROP TABLE statement on an NDB table.

Usage

```
ndb_drop_table -c connection_string tbl_name -d db_name
```

The following table includes options that are specific to ndb_drop_table. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_drop_table), see Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 18.27 Command-line options for the ndb_drop_table program

Format	Description	Added, Deprecated, or Removed
--database=dbname, -d	Name of database in which table is found	(Supported in all MySQL 5.6 based releases)

18.4.13 ndb_error_reporter — NDB Error-Reporting Utility

ndb_error_reporter creates an archive from data node and management node log files that can be used to help diagnose bugs or other problems with a cluster. *It is highly recommended that you make use of this utility when filing reports of bugs in NDB Cluster*.

The following table includes command options specific to the NDB Cluster program ndb_error_reporter. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_error_reporter), see Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

ndb_error_reporter did not support the --help option prior to NDB 7.3.3 (Bug #11756666, Bug #48606). The --connection-timeout --dry-scp, and --skip-nodegroup options were also added in this release (Bug #16602002).

Table 18.28 Command-line options for the ndb_error_reporter program

Format	Description	Added, Deprecated, or Removed
--connection-timeout=timeout	Number of seconds to wait when connecting to nodes before timing out	ADDED: NDB 7.3.3
--dry-scp	Disable scp with remote hosts; used only for testing	ADDED: NDB 7.3.3
--fs	Include file system data in error report; can use a large amount of disk space	(Supported in all MySQL 5.6 based releases)

Format	Description	Added, Deprecated, or Removed
--skip-nodegroup=nodegroup_id	Skip all nodes in the node group having this ID	ADDED: NDB 7.3.3

Usage

```
ndb_error_reporter path/to/config-file [username] [options]
```

This utility is intended for use on a management node host, and requires the path to the management host configuration file (usually named config.ini). Optionally, you can supply the name of a user that is able to access the cluster's data nodes using SSH, to copy the data node log files. ndb_error_reporter then includes all of these files in archive that is created in the same directory in which it is run. The archive is named ndb_error_report_*YYYYMMDDhhmmss*.tar.bz2, where *YYYYMMDDhhmmss* is a datetime string.

ndb_error_reporter also accepts the options listed here:

- --connection-timeout=*timeout*

Command-Line Format	--connection-timeout=timeout
Introduced	5.6.14-ndb-7.3.3
Type	Integer
Default Value	0

Wait this many seconds when trying to connect to nodes before timing out.

- --dry-scp

Command-Line Format	--dry-scp
Introduced	5.6.14-ndb-7.3.3
Type	Boolean
Default Value	TRUE

Run ndb_error_reporter without using scp from remote hosts. Used for testing only.

- --fs

Command-Line Format	--fs
Type	Boolean
Default Value	FALSE

Copy the data node file systems to the management host and include them in the archive.

Because data node file systems can be extremely large, even after being compressed, we ask that you please do *not* send archives created using this option to Oracle unless you are specifically requested to do so.

- --skip-nodegroup=*nodegroup_id*

Command-Line Format	--connection-timeout=timeout
Introduced	5.6.14-ndb-7.3.3
Type	Integer
Default Value	0

Skip all nodes belong to the node group having the supplied node group ID.

18.4.14 ndb_index_stat — NDB Index Statistics Utility

ndb_index_stat provides per-fragment statistical information about indexes on NDB tables. This includes cache version and age, number of index entries per partition, and memory consumption by indexes.

Usage

To obtain basic index statistics about a given NDB table, invoke ndb_index_stat as shown here, with the name of the table as the first argument and the name of the database containing this table specified immediately following it, using the --database (-d) option:

```
ndb_index_stat table -d database
```

In this example, we use ndb_index_stat to obtain such information about an NDB table named mytable in the test database:

```
shell> ndb_index_stat -d test mytable
table:City index:PRIMARY fragCount:2
sampleVersion:3 loadTime:1399585986 sampleCount:1994 keyBytes:7976
query cache: valid:1 sampleCount:1994 totalBytes:27916
times in ms: save: 7.133 sort: 1.974 sort per sample: 0.000

NDBT_ProgramExit: 0 - OK
```

sampleVersion is the version number of the cache from which the statistics data is taken. Running ndb_index_stat with the --update option causes sampleVersion to be incremented.

loadTime shows when the cache was last updated. This is expressed as seconds since the Unix Epoch.

sampleCount is the number of index entries found per partition. You can estimate the total number of entries by multiplying this by the number of fragments (shown as fragCount).

sampleCount can be compared with the cardinality of SHOW INDEX or INFORMATION_SCHEMA.STATISTICS, although the latter two provide a view of the table as a whole, while ndb_index_stat provides a per-fragment average.

keyBytes is the number of bytes used by the index. In this example, the primary key is an integer, which requires four bytes for each index, so keyBytes can be calculated in this case as shown here:

```
    keyBytes = sampleCount * (4 bytes per index) = 1994 * 4 = 7976
```

This information can also be obtained using the corresponding column definitions from INFORMATION_SCHEMA.COLUMNS (this requires a MySQL Server and a MySQL client application).

totalBytes is the total memory consumed by all indexes on the table, in bytes.

Timings shown in the preceding examples are specific to each invocation of ndb_index_stat.

The --verbose option provides some additional output, as shown here:

```
shell> ndb_index_stat -d test mytable --verbose
random seed 1337010518
connected
loop 1 of 1
table:mytable index:PRIMARY fragCount:4
sampleVersion:2 loadTime:1336751773 sampleCount:0 keyBytes:0
read stats
query cache created
query cache: valid:1 sampleCount:0 totalBytes:0
times in ms: save: 20.766 sort: 0.001
disconnected
```

```
NDBT_ProgramExit: 0 - OK

shell>
```

If the only output from the program is `NDBT_ProgramExit: 0 - OK`, this may indicate that no statistics yet exist. To force them to be created (or updated if they already exist), invoke `ndb_index_stat` with the `--update` option, or execute `ANALYZE TABLE` on the table in the `mysql` client.

Options

The following table includes options that are specific to the NDB Cluster `ndb_index_stat` utility. Additional descriptions are listed following the table. For options common to most NDB Cluster programs (including `ndb_index_stat`), see Section 18.4.29, "Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs".

Table 18.29 Command-line options for the ndb_index_stat program

Format	Description	Added, Deprecated, or Removed
`--database=name`, `-d`	Name of database containing table	(Supported in all MySQL 5.6 based releases)
`--delete`	Delete index statistics for table, stopping any auto-update previously configured	(Supported in all MySQL 5.6 based releases)
`--update`	Update index statistics for table, restarting any auto-update previously configured	(Supported in all MySQL 5.6 based releases)
`--dump`	Print query cache	(Supported in all MySQL 5.6 based releases)
`--query=#`	Perform random range queries on first key attr (must be int unsigned)	(Supported in all MySQL 5.6 based releases)
`--sys-drop`	Drop any statistics tables and events in NDB kernel (all statistics are lost)	(Supported in all MySQL 5.6 based releases)
`--sys-create`	Create all statistics tables and events in NDB kernel, if none of them already exist	(Supported in all MySQL 5.6 based releases)
`--sys-create-if-not-exist`	Create any statistics tables and events in NDB kernel that do not already exist	(Supported in all MySQL 5.6 based releases)
`--sys-create-if-not-valid`	Create any statistics tables or events that do not already exist in the NDB kernel, after dropping any that are invalid	(Supported in all MySQL 5.6 based releases)
`--sys-check`	Verify that NDB system index statistics and event tables exist	(Supported in all MySQL 5.6 based releases)
`--sys-skip-tables`	Do not apply sys-* options to tables	(Supported in all MySQL 5.6 based releases)
`--sys-skip-events`	Do not apply sys-* options to events	(Supported in all MySQL 5.6 based releases)
`--verbose`, `-v`	Turn on verbose output	(Supported in all MySQL 5.6 based releases)

Format	Description	Added, Deprecated, or Removed
`--loops=#`	Set the number of times to perform given command; default is 0	(Supported in all MySQL 5.6 based releases)

ndb_index_stat statistics options. The following options are used to generate index statistics. They work with a given table and database. They cannot be mixed with system options (see ndb_index_stat system options).

- `--database=name`, `-d name`

Command-Line Format	`--database=name`
Type	String
Default Value	`[none]`
Minimum Value	
Maximum Value	

The name of the database that contains the table being queried.

- `--delete`

Command-Line Format	`--delete`
Type	Boolean
Default Value	`false`
Minimum Value	
Maximum Value	

Delete the index statistics for the given table, stopping any auto-update that was previously configured.

- `--update`

Command-Line Format	`--update`
Type	Boolean
Default Value	`false`
Minimum Value	
Maximum Value	

Update the index statistics for the given table, and restart any auto-update that was previously configured.

- `--dump`

Command-Line Format	`--dump`
Type	Boolean
Default Value	`false`
Minimum Value	
Maximum Value	

Dump the contents of the query cache.

- `--query=#`

Command-Line Format	--query=#
Type	Numeric
Default Value	0
Minimum Value	0
Maximum Value	MAX_INT

Perform random range queries on first key attribute (must be int unsigned).

ndb_index_stat system options. The following options are used to generate and update the statistics tables in the NDB kernel. None of these options can be mixed with statistics options (see ndb_index_stat statistics options).

- --sys-drop

Command-Line Format	--sys-drop
Type	Boolean
Default Value	false
Minimum Value	
Maximum Value	

Drop all statistics tables and events in the NDB kernel. *This causes all statistics to be lost*.

- --sys-create

Command-Line Format	--sys-create
Type	Boolean
Default Value	false
Minimum Value	
Maximum Value	

Create all statistics tables and events in the NDB kernel. This works only if none of them exist previously.

- sys-create-if-not-exist

Command-Line Format	--sys-create-if-not-exist
Type	Boolean
Default Value	false
Minimum Value	
Maximum Value	

Create any NDB system statistics tables or events (or both) that do not already exist when the program is invoked.

- --sys-create-if-not-valid

Command-Line Format	--sys-create-if-not-valid
Type	Boolean
Default Value	false
Minimum Value	

Maximum Value	

Create any NDB system statistics tables or events that do not already exist, after dropping any that are invalid.

- `--sys-check`

Command-Line Format	`--sys-check`
Type	Boolean
Default Value	`false`
Minimum Value	
Maximum Value	

Verify that all required system statistics tables and events exist in the NDB kernel.

- `--sys-skip-tables`

Command-Line Format	`--sys-skip-tables`
Type	Boolean
Default Value	`false`
Minimum Value	
Maximum Value	

Do not apply any `--sys-*` options to any statistics tables.

- `--sys-skip-events`

Command-Line Format	`--sys-skip-events`
Type	Boolean
Default Value	`false`
Minimum Value	
Maximum Value	

Do not apply any `--sys-*` options to any events.

- `--verbose`

Command-Line Format	`--verbose`
Type	Boolean
Default Value	`false`
Minimum Value	
Maximum Value	

Turn on verbose output.

- `--loops=#`

Command-Line Format	`--loops=#`
Type	Numeric
Default Value	`0`
Minimum Value	`0`

Maximum Value	MAX_INT

Repeat commands this number of times (for use in testing).

18.4.15 ndb_move_data — NDB Data Copy Utility

ndb_move_data copies data from one NDB table to another.

Usage

The program is invoked with the names of the source and target tables; either or both of these may be qualified optionally with the database name. Both tables must use the NDB storage engine.

```
ndb_move_data options source target
```

The following table includes options that are specific to ndb_move_data. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_move_data), see Section 18.4.29, "Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs".

Table 18.30 Command-line options for the ndb_move_data program

Format	Description	Added, Deprecated, or Removed
--abort-on-error	Dump core on permanent error (debug option)	(Supported in all MySQL 5.6 based releases)
--character-sets-dir=name	Directory where character sets are	(Supported in all MySQL 5.6 based releases)
--database=dbname, -d	Name of database in which table is found	(Supported in all MySQL 5.6 based releases)
--drop-source	Drop source table after all rows have been moved	(Supported in all MySQL 5.6 based releases)
--error-insert	Insert random temporary errors (testing option)	(Supported in all MySQL 5.6 based releases)
--exclude-missing-columns	Ignore extra columns in source or target table	(Supported in all MySQL 5.6 based releases)
--lossy-conversions, -l	Allow attribute data to be truncated when converted to smaller type	(Supported in all MySQL 5.6 based releases)
--promote-attributes, -A	Allow attribute data to be converted to larger type	(Supported in all MySQL 5.6 based releases)
--staging-tries=x[,y[,z]]	Specify tries on temporary errors; format is x[,y[,z]] where x=max tries (0=no limit), y=min delay (ms), z=max delay (ms)	(Supported in all MySQL 5.6 based releases)
--verbose	Enable verbose messages	(Supported in all MySQL 5.6 based releases)

- --abort-on-error

Command-Line Format	--abort-on-error
Type	Boolean
Default Value	FALSE

Dump core on permanent error (debug option).

- `--character-sets-dir=name`

Command-Line Format	`--character-sets-dir=name`
Type	String
Default Value	`[none]`

Directory where character sets are.

- `--database=dbname`, `-d`

Command-Line Format	`--database=dbname`
Type	String
Default Value	`TEST_DB`

Name of the database in which the table is found.

- `--drop-source`

Command-Line Format	`--drop-source`
Type	Boolean
Default Value	`FALSE`

Drop source table after all rows have been moved.

- `--error-insert`

Command-Line Format	`--error-insert`
Type	Boolean
Default Value	`FALSE`

Insert random temporary errors (testing option).

- `--exclude-missing-columns`

Command-Line Format	`--exclude-missing-columns`
Type	Boolean
Default Value	`FALSE`

Ignore extra columns in source or target table.

- `--lossy-conversions`, `-l`

Command-Line Format	`--lossy-conversions`
Type	Boolean
Default Value	`FALSE`

Allow attribute data to be truncated when converted to a smaller type.

- `--promote-attributes`, `-A`

Command-Line Format	`--promote-attributes`
Type	Boolean

Default Value	FALSE

Allow attribute data to be converted to a larger type.

- `--staging-tries=x[,y[,z]]`

Command-Line Format	--staging-tries=x[,y[,z]]
Type	String
Default Value	0,1000,60000

Specify tries on temporary errors. Format is x[,y[,z]] where x=max tries (0=no limit), y=min delay (ms), z=max delay (ms).

- `--verbose`

Command-Line Format	--verbose
Type	Boolean
Default Value	FALSE

Enable verbose messages.

18.4.16 ndb_print_backup_file — Print NDB Backup File Contents

ndb_print_backup_file obtains diagnostic information from a cluster backup file.

Usage

```
ndb_print_backup_file file_name
```

file_name is the name of a cluster backup file. This can be any of the files (`.Data`, `.ctl`, or `.log` file) found in a cluster backup directory. These files are found in the data node's backup directory under the subdirectory `BACKUP-#`, where `#` is the sequence number for the backup. For more information about cluster backup files and their contents, see Section 18.5.8.1, “NDB Cluster Backup Concepts”.

Like ndb_print_schema_file and ndb_print_sys_file (and unlike most of the other NDB utilities that are intended to be run on a management server host or to connect to a management server) ndb_print_backup_file must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

Additional Options

None.

18.4.17 ndb_print_file — Print NDB Disk Data File Contents

ndb_print_file obtains information from an NDB Cluster Disk Data file.

Usage

```
ndb_print_file [-v] [-q] file_name+
```

file_name is the name of an NDB Cluster Disk Data file. Multiple filenames are accepted, separated by spaces.

Like ndb_print_schema_file and ndb_print_sys_file (and unlike most of the other NDB utilities that are intended to be run on a management server host or to connect to a management server) ndb_print_file must be run on an NDB Cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used

when the management server is not running, and even when the cluster has been completely shut down.

Additional Options

ndb_print_file supports the following options:

- -v: Make output verbose.
- -q: Suppress output (quiet mode).
- --help, -h, -?: Print help message.

  This option did not work correctly prior to NDB 7.3.7. (Bug #17069285)

For more information, see Section 18.5.10, “NDB Cluster Disk Data Tables”.

18.4.18 ndb_print_frag_file — Print NDB Fragment List File Contents

ndb_print_frag_file obtains information from a cluster fragment list file. It is intended for use in helping to diagnose issues with data node restarts.

Usage

```
ndb_print_frag_file file_name
```

file_name is the name of a cluster fragment list file, which matches the pattern S*X*.FragList, where *X* is a digit in the range 2-9 inclusive, and are found in the data node file system of the data node having the node ID *nodeid*, in directories named ndb_*nodeid*_fs/D*N*/DBDIH/, where *N* is 1 or 2. Each fragment file contains records of the fragments belonging to each NDB table. For more information about cluster fragment files, see NDB Cluster Data Node File System Directory.

Like ndb_print_backup_file, ndb_print_sys_file, and ndb_print_schema_file (and unlike most of the other NDB utilities that are intended to be run on a management server host or to connect to a management server), ndb_print_frag_file must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

Additional Options

None.

Sample Output

```
shell> ndb_print_frag_file /usr/local/mysqld/data/ndb_3_fs/D1/DBDIH/S2.FragList
Filename: /usr/local/mysqld/data/ndb_3_fs/D1/DBDIH/S2.FragList with size 8192
noOfPages = 1 noOfWords = 182
Table Data
----------
Num Frags: 2 NoOfReplicas: 2 hashpointer: 4294967040
kvalue: 6 mask: 0x00000000 method: HashMap
Storage is on Logged and checkpointed, survives SR
------ Fragment with FragId: 0 --------
Preferred Primary: 2 numStoredReplicas: 2 numOldStoredReplicas: 0 distKey: 0 LogPartId: 0
-------Stored Replica----------
Replica node is: 2 initialGci: 2 numCrashedReplicas = 0 nextLcpNo = 1
LcpNo[0]: maxGciCompleted: 1 maxGciStarted: 2 lcpId: 1 lcpStatus: valid
LcpNo[1]: maxGciCompleted: 0 maxGciStarted: 0 lcpId: 0 lcpStatus: invalid
-------Stored Replica----------
Replica node is: 3 initialGci: 2 numCrashedReplicas = 0 nextLcpNo = 1
LcpNo[0]: maxGciCompleted: 1 maxGciStarted: 2 lcpId: 1 lcpStatus: valid
LcpNo[1]: maxGciCompleted: 0 maxGciStarted: 0 lcpId: 0 lcpStatus: invalid
------ Fragment with FragId: 1 --------
Preferred Primary: 3 numStoredReplicas: 2 numOldStoredReplicas: 0 distKey: 0 LogPartId: 1
-------Stored Replica----------
```

```
Replica node is: 3 initialGci: 2 numCrashedReplicas = 0 nextLcpNo = 1
LcpNo[0]: maxGciCompleted: 1 maxGciStarted: 2 lcpId: 1 lcpStatus: valid
LcpNo[1]: maxGciCompleted: 0 maxGciStarted: 0 lcpId: 0 lcpStatus: invalid
-------Stored Replica----------
Replica node is: 2 initialGci: 2 numCrashedReplicas = 0 nextLcpNo = 1
LcpNo[0]: maxGciCompleted: 1 maxGciStarted: 2 lcpId: 1 lcpStatus: valid
LcpNo[1]: maxGciCompleted: 0 maxGciStarted: 0 lcpId: 0 lcpStatus: invalid
```

This program was added in MySQL NDB Cluster 7.4.3. (Bug #74594, Bug #19898269)

18.4.19 ndb_print_schema_file — Print NDB Schema File Contents

`ndb_print_schema_file` obtains diagnostic information from a cluster schema file.

Usage

```
ndb_print_schema_file file_name
```

`file_name` is the name of a cluster schema file. For more information about cluster schema files, see NDB Cluster Data Node File System Directory.

Like `ndb_print_backup_file` and `ndb_print_sys_file` (and unlike most of the other `NDB` utilities that are intended to be run on a management server host or to connect to a management server) `ndb_print_schema_file` must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

Additional Options

None.

18.4.20 ndb_print_sys_file — Print NDB System File Contents

`ndb_print_sys_file` obtains diagnostic information from an NDB Cluster system file.

Usage

```
ndb_print_sys_file file_name
```

`file_name` is the name of a cluster system file (sysfile). Cluster system files are located in a data node's data directory (`DataDir`); the path under this directory to system files matches the pattern `ndb_#_fs/D#/DBDIH/P#.sysfile`. In each case, the `#` represents a number (not necessarily the same number). For more information, see NDB Cluster Data Node File System Directory.

Like `ndb_print_backup_file` and `ndb_print_schema_file` (and unlike most of the other `NDB` utilities that are intended to be run on a management server host or to connect to a management server) `ndb_print_backup_file` must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

Additional Options

None.

18.4.21 ndb_redo_log_reader — Check and Print Content of Cluster Redo Log

Reads a redo log file, checking it for errors, printing its contents in a human-readable format, or both. `ndb_redo_log_reader` is intended for use primarily by NDB Cluster developers and Support personnel in debugging and diagnosing problems.

This utility remains under development, and its syntax and behavior are subject to change in future NDB Cluster releases.

The C++ source files for ndb_redo_log_reader can be found in the directory /storage/ndb/src/kernel/blocks/dblqh/redoLogReader.

The following table includes options that are specific to the NDB Cluster program ndb_redo_log_reader. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_redo_log_reader), see Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 18.31 Command-line options for the ndb_redo_log_reader program

Format	Description	Added, Deprecated, or Removed
-dump	Print dump info	(Supported in all MySQL 5.6 based releases)
-filedescriptors	Print file descriptors only	(Supported in all MySQL 5.6 based releases)
--help	Print usage information	ADDED: NDB 7.3.4
-lap	Provide lap info, with max GCI started and completed	(Supported in all MySQL 5.6 based releases)
-mbyte #	Starting megabyte	(Supported in all MySQL 5.6 based releases)
-mbyteheaders	Show only first page header of each megabyte in file	(Supported in all MySQL 5.6 based releases)
-nocheck	Do not check records for errors	(Supported in all MySQL 5.6 based releases)
-noprint	Do not print records	(Supported in all MySQL 5.6 based releases)
-page #	Start with this page	(Supported in all MySQL 5.6 based releases)
-pageheaders	Show page headers only	(Supported in all MySQL 5.6 based releases)
-pageindex #	Start with this page index	(Supported in all MySQL 5.6 based releases)
-twiddle	Bit-shifted dump	(Supported in all MySQL 5.6 based releases)

Usage

```
ndb_redo_log_reader file_name [options]
```

file_name is the name of a cluster redo log file. redo log files are located in the numbered directories under the data node's data directory (DataDir); the path under this directory to the redo log files matches the pattern ndb_*nodeid*_fs/D#/DBLQH/S#.FragLog. *nodeid* is the data node's node ID. The two instances of # each represent a number (not necessarily the same number); the number following D is in the range 8-39 inclusive; the range of the number following S varies according to the value of the NoOfFragmentLogFiles configuration parameter, whose default value is 16; thus, the default range of the number in the file name is 0-15 inclusive. For more information, see NDB Cluster Data Node File System Directory.

The name of the file to be read may be followed by one or more of the options listed here:

- -dump

Command-Line Format	-dump

Type	Boolean
Default Value	FALSE

Print dump info.

Command-Line Format	-filedescriptors
Type	Boolean
Default Value	FALSE

  -filedescriptors: Print file descriptors only.

Command-Line Format	--help
Introduced	5.6.15-ndb-7.3.4

  --help: Print usage information.

  Added in NDB 7.3.4. (Bug #11749591, Bug #36805)

- -lap

Command-Line Format	-lap
Type	Boolean
Default Value	FALSE

  Provide lap info, with max GCI started and completed.

Command-Line Format	-mbyte #
Type	Numeric
Default Value	0
Minimum Value	0
Maximum Value	15

  -mbyte #: Starting megabyte.

  # is an integer in the range 0 to 15, inclusive.

Command-Line Format	-mbyteheaders
Type	Boolean
Default Value	FALSE

  -mbyteheaders: Show only the first page header of every megabyte in the file.

Command-Line Format	-noprint
Type	Boolean
Default Value	FALSE

  -noprint: Do not print the contents of the log file.

Command-Line Format	-nocheck
Type	Boolean
Default Value	FALSE

  -nocheck: Do not check the log file for errors.

Command-Line Format	-page #
Type	Integer
Default Value	0
Minimum Value	0
Maximum Value	31

  -page #: Start at this page.

  # is an integer in the range 0 to 31, inclusive.

Command-Line Format	-pageheaders
Type	Boolean
Default Value	FALSE

  -pageheaders: Show page headers only.

Command-Line Format	-pageindex #
Type	Integer
Default Value	12
Minimum Value	12
Maximum Value	8191

  -pageindex #: Start at this page index.

  # is an integer between 12 and 8191, inclusive.

- -twiddle

Command-Line Format	-twiddle
Type	Boolean
Default Value	FALSE

  Bit-shifted dump.

Like ndb_print_backup_file and ndb_print_schema_file (and unlike most of the NDB utilities that are intended to be run on a management server host or to connect to a management server) ndb_redo_log_reader must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

18.4.22 ndb_restore — Restore an NDB Cluster Backup

The NDB Cluster restoration program is implemented as a separate command-line utility ndb_restore, which can normally be found in the MySQL bin directory. This program reads the files created as a result of the backup and inserts the stored information into the database.

Note

Beginning with NDB 7.3.25 and 7.4.24, this program no longer prints NDBT_ProgramExit: ... when it finishes its run. Applications depending on this behavior should be modified accordingly when upgrading from earlier releases.

ndb_restore must be executed once for each of the backup files that were created by the START BACKUP command used to create the backup (see Section 18.5.8.2, “Using The NDB Cluster

Management Client to Create a Backup"). This is equal to the number of data nodes in the cluster at the time that the backup was created.

Note

Before using ndb_restore, it is recommended that the cluster be running in single user mode, unless you are restoring multiple data nodes in parallel. See Section 18.5.6, "NDB Cluster Single User Mode", for more information.

The following table includes options that are specific to the NDB Cluster native backup restoration program ndb_restore. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_restore), see Section 18.4.29, "Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs".

Table 18.32 Command-line options for the ndb_restore program

Format	Description	Added, Deprecated, or Removed
--append	Append data to tab-delimited file	(Supported in all MySQL 5.6 based releases)
--backup-path=dir_name	Path to backup files directory	(Supported in all MySQL 5.6 based releases)
--backupid=#, -b	Restore from backup having this ID	(Supported in all MySQL 5.6 based releases)
--connect, -c	Alias for --connectstring	(Supported in all MySQL 5.6 based releases)
--disable-indexes	Causes indexes from backup to be ignored; may decrease time needed to restore data	(Supported in all MySQL 5.6 based releases)
--dont-ignore-systab-0, -f	Do not ignore system table during restore; experimental only; not for production use	(Supported in all MySQL 5.6 based releases)
--exclude-databases=db-list	List of one or more databases to exclude (includes those not named)	(Supported in all MySQL 5.6 based releases)
--exclude-intermediate-sql-tables[=TRUE|FALSE]	If TRUE (default), do not restore any intermediate tables (having names prefixed with '#sql-') that were left over from copying ALTER TABLE operations	ADDED: NDB 7.3.6
--exclude-missing-columns	Causes columns from backup version of table that are missing from version of table in database to be ignored	(Supported in all MySQL 5.6 based releases)
--exclude-missing-tables	Causes tables from backup that are missing from database to be ignored	ADDED: NDB 7.3.7
--exclude-tables=table-list	List of one or more tables to exclude (includes those in same database that are not named); each table reference must include database name	(Supported in all MySQL 5.6 based releases)
--fields-enclosed-by=char	Fields are enclosed by this character	(Supported in all MySQL 5.6 based releases)

Format	Description	Added, Deprecated, or Removed
--fields-optionally-enclosed-by	Fields are optionally enclosed by this character	(Supported in all MySQL 5.6 based releases)
--fields-terminated-by=char	Fields are terminated by this character	(Supported in all MySQL 5.6 based releases)
--hex	Print binary types in hexadecimal format	(Supported in all MySQL 5.6 based releases)
--include-databases=db-list	List of one or more databases to restore (excludes those not named)	(Supported in all MySQL 5.6 based releases)
--include-tables=table-list	List of one or more tables to restore (excludes those in same database that are not named); each table reference must include database name	(Supported in all MySQL 5.6 based releases)
--lines-terminated-by=char	Lines are terminated by this character	(Supported in all MySQL 5.6 based releases)
--lossy-conversions, -L	Allow lossy conversions of column values (type demotions or changes in sign) when restoring data from backup	(Supported in all MySQL 5.6 based releases)
--no-binlog	If mysqld is connected and using binary logging, do not log restored data	(Supported in all MySQL 5.6 based releases)
--no-restore-disk-objects, -d	Do not restore objects relating to Disk Data	(Supported in all MySQL 5.6 based releases)
--no-upgrade, -u	Do not upgrade array type for varsize attributes which do not already resize VAR data, and do not change column attributes	(Supported in all MySQL 5.6 based releases)
--ndb-nodegroup-map=map, -z	Nodegroup map for NDBCLUSTER storage engine; syntax: list of (source_nodegroup, destination_nodegroup)	(Supported in all MySQL 5.6 based releases)
--nodeid=#, -n	ID of node where backup was taken	(Supported in all MySQL 5.6 based releases)
--parallelism=#, -p	Number of parallel transactions to use while restoring data	(Supported in all MySQL 5.6 based releases)
--preserve-trailing-spaces, -P	Allow preservation of trailing spaces (including padding) when promoting fixed-width string types to variable-width types	(Supported in all MySQL 5.6 based releases)
--print	Print metadata, data, and log to stdout (equivalent to --print-meta --print-data --print-log)	(Supported in all MySQL 5.6 based releases)
--print-data	Print data to stdout	(Supported in all MySQL 5.6 based releases)

Format	Description	Added, Deprecated, or Removed
--print-log	Print log to stdout	(Supported in all MySQL 5.6 based releases)
--print-meta	Print metadata to stdout	(Supported in all MySQL 5.6 based releases)
--progress-frequency=#	Print status of restore each given number of seconds	(Supported in all MySQL 5.6 based releases)
--promote-attributes, -A	Allow attributes to be promoted when restoring data from backup	(Supported in all MySQL 5.6 based releases)
--rebuild-indexes	Causes multithreaded rebuilding of ordered indexes found in backup; number of threads used is determined by setting BuildIndexThreads	(Supported in all MySQL 5.6 based releases)
--restore-data, -r	Restore table data and logs into NDB Cluster using NDB API	(Supported in all MySQL 5.6 based releases)
--restore-epoch, -e	Restore epoch info into status table; useful on replica cluster for starting replication; updates or inserts row in mysql.ndb_apply_status with ID 0	(Supported in all MySQL 5.6 based releases)
--restore-meta, -m	Restore metadata to NDB Cluster using NDB API	(Supported in all MySQL 5.6 based releases)
--restore-privilege-tables	Restore MySQL privilege tables that were previously moved to NDB	(Supported in all MySQL 5.6 based releases)
--rewrite-database=olddb,newdb	Restore to differently named database	(Supported in all MySQL 5.6 based releases)
--skip-broken-objects	Ignore missing blob tables in backup file	(Supported in all MySQL 5.6 based releases)
--skip-table-check, -s	Skip table structure check during restore	(Supported in all MySQL 5.6 based releases)
--skip-unknown-objects	Causes schema objects not recognized by ndb_restore to be ignored when restoring backup made from newer NDB version to older version	(Supported in all MySQL 5.6 based releases)
--tab=dir_name, -T dir_name	Creates a tab-separated .txt file for each table in path provided	(Supported in all MySQL 5.6 based releases)
--verbose=#	Level of verbosity in output	(Supported in all MySQL 5.6 based releases)

Typical options for this utility are shown here:

```
ndb_restore [-c connection_string] -n node_id -b backup_id \
      [-m] -r --backup-path=/path/to/backup/files
```

Normally, when restoring from an NDB Cluster backup, ndb_restore requires at a minimum the --nodeid (short form: -n), --backupid (short form: -b), and --backup-path options. In addition, when ndb_restore is used to restore any tables containing unique indexes, you must include --disable-indexes or --rebuild-indexes. (Bug #57782, Bug #11764893)

The -c option is used to specify a connection string which tells ndb_restore where to locate the cluster management server (see Section 18.3.3.3, “NDB Cluster Connection Strings”). If this option is not used, then ndb_restore attempts to connect to a management server on localhost:1186. This utility acts as a cluster API node, and so requires a free connection “slot” to connect to the cluster management server. This means that there must be at least one [api] or [mysqld] section that can be used by it in the cluster config.ini file. It is a good idea to keep at least one empty [api] or [mysqld] section in config.ini that is not being used for a MySQL server or other application for this reason (see Section 18.3.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”).

You can verify that ndb_restore is connected to the cluster by using the SHOW command in the ndb_mgm management client. You can also accomplish this from a system shell, as shown here:

```
shell> ndb_mgm -e "SHOW"
```

Note

In NDB 7.3.11 and NDB 7.4.8 *only*, when ndb_restore is used to restore any tables containing unique indexes, you must include --disable-indexes or --rebuild-indexes. (Bug #57782, Bug #11764893) This is not a requirement in later versions. (Bug #22345748)

More detailed information about all options used by ndb_restore can be found in the following list:

- --append

Command-Line Format	--append

When used with the --tab and --print-data options, this causes the data to be appended to any existing files having the same names.

- --backup-path=*dir_name*

Command-Line Format	--backup-path=dir_name
Type	Directory name
Default Value	./

The path to the backup directory is required; this is supplied to ndb_restore using the --backup-path option, and must include the subdirectory corresponding to the ID backup of the backup to be restored. For example, if the data node's DataDir is /var/lib/mysql-cluster, then the backup directory is /var/lib/mysql-cluster/BACKUP, and the backup files for the backup with the ID 3 can be found in /var/lib/mysql-cluster/BACKUP/BACKUP-3. The path may be absolute or relative to the directory in which the ndb_restore executable is located, and may be optionally prefixed with backup-path=.

It is possible to restore a backup to a database with a different configuration than it was created from. For example, suppose that a backup with backup ID 12, created in a cluster with two storage nodes having the node IDs 2 and 3, is to be restored to a cluster with four nodes. Then ndb_restore must be run twice—once for each storage node in the cluster where the backup was taken. However, ndb_restore cannot always restore backups made from a cluster running one version of MySQL to a cluster running a different MySQL version. See Section 18.2.7, “Upgrading and Downgrading NDB Cluster”, for more information.

Important

It is not possible to restore a backup made from a newer version of NDB Cluster using an older version of ndb_restore. You can restore a backup made from a newer version of MySQL to an older cluster, but you must use a copy of ndb_restore from the newer NDB Cluster version to do so.

For example, to restore a cluster backup taken from a cluster running NDB Cluster 7.4.31 to a cluster running NDB Cluster 7.3.32, you must use the ndb_restore that comes with the NDB Cluster 7.4.31 distribution.

For more rapid restoration, the data may be restored in parallel, provided that there is a sufficient number of cluster connections available. That is, when restoring to multiple nodes in parallel, you must have an [api] or [mysqld] section in the cluster config.ini file available for each concurrent ndb_restore process. However, the data files must always be applied before the logs.

- --backupid=#, -b

Command-Line Format	--backupid=#
Type	Numeric
Default Value	none

This option is used to specify the ID or sequence number of the backup, and is the same number shown by the management client in the Backup *backup_id* completed message displayed upon completion of a backup. (See Section 18.5.8.2, "Using The NDB Cluster Management Client to Create a Backup".)

Important

When restoring cluster backups, you must be sure to restore all data nodes from backups having the same backup ID. Using files from different backups at best results in restoring the cluster to an inconsistent state, and may fail altogether.

- --connect, -c

Command-Line Format	--connect
Type	String
Default Value	localhost:1186

Alias for --ndb-connectstring.

- --disable-indexes

Command-Line Format	--disable-indexes

Disable restoration of indexes during restoration of the data from a native NDB backup. Afterwards, you can restore indexes for all tables at once with multithreaded building of indexes using --rebuild-indexes, which should be faster than rebuilding indexes concurrently for very large tables.

- --dont-ignore-systab-0, -f

Command-Line Format	--dont-ignore-systab-0

Normally, when restoring table data and metadata, ndb_restore ignores the copy of the NDB system table that is present in the backup. --dont-ignore-systab-0 causes the system table

to be restored. *This option is intended for experimental and development use only, and is not recommended in a production environment*.

- `--exclude-databases=db-list`

Command-Line Format	`--exclude-databases=db-list`
Type	String
Default Value	

Comma-delimited list of one or more databases which should not be restored.

This option is often used in combination with `--exclude-tables`; see that option's description for further information and examples.

- `--exclude-intermediate-sql-tables[=TRUE|FALSE]`

Command-Line Format	`--exclude-intermediate-sql-tables[=TRUE|FALSE]`
Introduced	5.6.17-ndb-7.3.6
Type	Boolean
Default Value	`TRUE`

When performing copying `ALTER TABLE` operations, `mysqld` creates intermediate tables (whose names are prefixed with `#sql-`). When `TRUE`, the `--exclude-intermediate-sql-tables` option keeps `ndb_restore` from restoring such tables that may have been left over from these operations. This option is `TRUE` by default.

This option was introduced in NDB 7.3.6. (Bug #17882305)

- `--exclude-missing-columns`

Command-Line Format	`--exclude-missing-columns`

It is possible to restore only selected table columns using this option, which causes `ndb_restore` to ignore any columns missing from tables being restored as compared to the versions of those tables found in the backup. This option applies to all tables being restored. If you wish to apply this option only to selected tables or databases, you can use it in combination with one or more of the `--include-*` or `--exclude-*` options described elsewhere in this section to do so, then restore data to the remaining tables using a complementary set of these options.

- `--exclude-missing-tables`

Command-Line Format	`--exclude-missing-tables`
Introduced	5.6.21-ndb-7.3.7

It is possible to restore only selected tables using this option, which causes `ndb_restore` to ignore any tables from the backup that are not found in the target database.

This option was introduced in NDB 7.3.7.

- `--exclude-tables=table-list`

Command-Line Format	`--exclude-tables=table-list`
Type	String
Default Value	

List of one or more tables to exclude; each table reference must include the database name. Often used together with `--exclude-databases`.

When --exclude-databases or --exclude-tables is used, only those databases or tables named by the option are excluded; all other databases and tables are restored by ndb_restore.

This table shows several invocations of ndb_restore usng --exclude-* options (other options possibly required have been omitted for clarity), and the effects these options have on restoring from an NDB Cluster backup:

Table 18.33 Several invocations of ndb_restore using --exclude-* options, and the effects these options have on restoring from an NDB Cluster backup.

Option	Result
--exclude-databases=db1	All tables in all databases except db1 are restored; no tables in db1 are restored
--exclude-databases=db1,db2 (or --exclude-databases=db1 --exclude-databases=db2)	All tables in all databases except db1 and db2 are restored; no tables in db1 or db2 are restored
--exclude-tables=db1.t1	All tables except t1 in database db1 are restored; all other tables in db1 are restored; all tables in all other databases are restored
--exclude-tables=db1.t2,db2.t1 (or --exclude-tables=db1.t2 --exclude-tables=db2.t1)	All tables in database db1 except for t2 and all tables in database db2 except for table t1 are restored; no other tables in db1 or db2 are restored; all tables in all other databases are restored

You can use these two options together. For example, the following causes all tables in all databases *except for* databases db1 and db2, and tables t1 and t2 in database db3, to be restored:

```
shell> ndb_restore [...] --exclude-databases=db1,db2 --exclude-tables=db3.t1,db3.t2
```

(Again, we have omitted other possibly necessary options in the interest of clarity and brevity from the example just shown.)

You can use --include-* and --exclude-* options together, subject to the following rules:

- The actions of all --include-* and --exclude-* options are cumulative.
- All --include-* and --exclude-* options are evaluated in the order passed to ndb_restore, from right to left.
- In the event of conflicting options, the first (rightmost) option takes precedence. In other words, the first option (going from right to left) that matches against a given database or table “wins”.

For example, the following set of options causes ndb_restore to restore all tables from database db1 except db1.t1, while restoring no other tables from any other databases:

```
--include-databases=db1 --exclude-tables=db1.t1
```

However, reversing the order of the options just given simply causes all tables from database db1 to be restored (including db1.t1, but no tables from any other database), because the --include-databases option, being farthest to the right, is the first match against database db1 and thus takes precedence over any other option that matches db1 or any tables in db1:

```
--exclude-tables=db1.t1 --include-databases=db1
```

- --fields-enclosed-by=*char*

Command-Line Format	--fields-enclosed-by=char

Type	String
Default Value	

Each column value is enclosed by the string passed to this option (regardless of data type; see the description of --fields-optionally-enclosed-by).

- --fields-optionally-enclosed-by

Command-Line Format	--fields-optionally-enclosed-by
Type	String
Default Value	

The string passed to this option is used to enclose column values containing character data (such as CHAR, VARCHAR, BINARY, TEXT, or ENUM).

- --fields-terminated-by=*char*

Command-Line Format	--fields-terminated-by=char
Type	String
Default Value	\t (tab)

The string passed to this option is used to separate column values. The default value is a tab character (\t).

- --hex

Command-Line Format	--hex

If this option is used, all binary values are output in hexadecimal format.

- --include-databases=*db-list*

Command-Line Format	--include-databases=db-list
Type	String
Default Value	

Comma-delimited list of one or more databases to restore. Often used together with --include-tables; see the description of that option for further information and examples.

- --include-tables=*table-list*

Command-Line Format	--include-tables=table-list
Type	String

Default Value	

Comma-delimited list of tables to restore; each table reference must include the database name.

When `--include-databases` or `--include-tables` is used, only those databases or tables named by the option are restored; all other databases and tables are excluded by `ndb_restore`, and are not restored.

The following table shows several invocations of `ndb_restore` using `--include-*` options (other options possibly required have been omitted for clarity), and the effects these have on restoring from an NDB Cluster backup:

Table 18.34 Several invocations of ndb_restore using --include-* options, and their effects on restoring from an NDB Cluster backup.

Option	Result
`--include-databases=db1`	Only tables in database `db1` are restored; all tables in all other databases are ignored
`--include-databases=db1,db2` (or `--include-databases=db1 --include-databases=db2`)	Only tables in databases `db1` and `db2` are restored; all tables in all other databases are ignored
`--include-tables=db1.t1`	Only table `t1` in database `db1` is restored; no other tables in `db1` or in any other database are restored
`--include-tables=db1.t2,db2.t1` (or `--include-tables=db1.t2 --include-tables=db2.t1`)	Only the table `t2` in database `db1` and the table `t1` in database `db2` are restored; no other tables in `db1`, `db2`, or any other database are restored

You can also use these two options together. For example, the following causes all tables in databases `db1` and `db2`, together with the tables `t1` and `t2` in database `db3`, to be restored (and no other databases or tables):

```
shell> ndb_restore [...] --include-databases=db1,db2 --include-tables=db3.t1,db3.t2
```

(Again we have omitted other, possibly required, options in the example just shown.)

It also possible to restore only selected databases, or selected tables from a single database, without any `--include-*` (or `--exclude-*`) options, using the syntax shown here:

```
ndb_restore other_options db_name,[db_name[,...] | tbl_name[,tbl_name][,...]]
```

In other words, you can specify either of the following to be restored:

- All tables from one or more databases
- One or more tables from a single database

- `--lines-terminated-by=char`

Command-Line Format	`--lines-terminated-by=char`
Type	String
Default Value	`\n (linebreak)`

Specifies the string used to end each line of output. The default is a linefeed character (`\n`).

- `--lossy-conversions`, `-L`

Command-Line Format	`--lossy-conversions`

Type	Boolean
Default Value	FALSE (If option is not used)

This option is intended to complement the --promote-attributes option. Using --lossy-conversions allows lossy conversions of column values (type demotions or changes in sign) when restoring data from backup. With some exceptions, the rules governing demotion are the same as for MySQL replication; see Replication of Columns Having Different Data Types, for information about specific type conversions currently supported by attribute demotion.

ndb_restore reports any truncation of data that it performs during lossy conversions once per attribute and column.

- --no-binlog

Command-Line Format	--no-binlog

This option prevents any connected SQL nodes from writing data restored by ndb_restore to their binary logs.

- --no-restore-disk-objects, -d

Command-Line Format	--no-restore-disk-objects
Type	Boolean
Default Value	FALSE

This option stops ndb_restore from restoring any NDB Cluster Disk Data objects, such as tablespaces and log file groups; see Section 18.5.10, “NDB Cluster Disk Data Tables”, for more information about these.

- --no-upgrade, -u

Command-Line Format	--no-upgrade

When using ndb_restore to restore a backup, VARCHAR columns created using the old fixed format are resized and recreated using the variable-width format now employed. This behavior can be overridden by specifying --no-upgrade.

- --ndb-nodegroup-map=*map*, -z

Command-Line Format	--ndb-nodegroup-map=map

This option can be used to restore a backup taken from one node group to a different node group. Its argument is a list of the form *source_node_group, target_node_group*.

- --nodeid=#, -n

Command-Line Format	--nodeid=#
Type	Numeric
Default Value	none

Specify the node ID of the data node on which the backup was taken.

When restoring to a cluster with different number of data nodes from that where the backup was taken, this information helps identify the correct set or sets of files to be restored to a given node. (In such cases, multiple files usually need to be restored to a single data node.) See Section 18.4.22.2, “Restoring to a different number of data nodes”, for additional information and examples.

- --parallelism=#, -p

Command-Line Format	--parallelism=#
Type	Numeric
Default Value	128
Minimum Value	1
Maximum Value	1024

ndb_restore uses single-row transactions to apply many rows concurrently. This parameter determines the number of parallel transactions (concurrent rows) that an instance of ndb_restore tries to use. By default, this is 128; the minimum is 1, and the maximum is 1024.

The work of performing the inserts is parallelized across the threads in the data nodes involved. This mechanism is employed for restoring bulk data from the .Data file—that is, the fuzzy snapshot of the data; it is not used for building or rebuilding indexes. The change log is applied serially; index drops and builds are DDL operations and handled separately. There is no thread-level parallelism on the client side of the restore.

- --preserve-trailing-spaces, -P

Command-Line Format	--preserve-trailing-spaces

Cause trailing spaces to be preserved when promoting a fixed-width character data type to its variable-width equivalent—that is, when promoting a CHAR column value to VARCHAR, or a BINARY column value to VARBINARY. Otherwise, any trailing spaces are dropped from such column values when they are inserted into the new columns.

Note

Although you can promote CHAR columns to VARCHAR and BINARY columns to VARBINARY, you cannot promote VARCHAR columns to CHAR or VARBINARY columns to BINARY.

- --print

Command-Line Format	--print
Type	Boolean
Default Value	FALSE

Causes ndb_restore to print all data, metadata, and logs to stdout. Equivalent to using the --print-data, --print-meta, and --print-log options together.

Note

Use of --print or any of the --print_* options is in effect performing a dry run. Including one or more of these options causes any output to be redirected to stdout; in such cases, ndb_restore makes no attempt to restore data or metadata to an NDB Cluster.

- --print-data

Command-Line Format	--print-data
Type	Boolean

Default Value	FALSE

Cause ndb_restore to direct its output to stdout. Often used together with one or more of --tab, --fields-enclosed-by, --fields-optionally-enclosed-by, --fields-terminated-by, --hex, and --append.

TEXT and BLOB column values are always truncated. In NDB 7.3.7 and earlier, such values are truncated to the first 240 bytes in the output; in NDB 7.3.8 and later, they are truncated to 256 bytes. (Bug #14571512, Bug #65467) This cannot currently be overridden when using --print-data.

- --print-log

Command-Line Format	--print-log
Type	Boolean
Default Value	FALSE

Cause ndb_restore to output its log to stdout.

- --print-meta

Command-Line Format	--print-meta
Type	Boolean
Default Value	FALSE

Print all metadata to stdout.

- --progress-frequency=*N*

Command-Line Format	--progress-frequency=#
Type	Numeric
Default Value	0
Minimum Value	0
Maximum Value	65535

Print a status report each *N* seconds while the restore is in progress. 0 (the default) causes no status reports to be printed. The maximum is 65535.

- --promote-attributes, -A

Command-Line Format	--promote-attributes

ndb_restore supports limited *attribute promotion* in much the same way that it is supported by MySQL replication; that is, data backed up from a column of a given type can generally be restored to a column using a “larger, similar” type. For example, data from a CHAR(20) column can be restored to a column declared as VARCHAR(20), VARCHAR(30), or CHAR(30); data from a MEDIUMINT column can be restored to a column of type INT or BIGINT. See Replication of Columns Having Different Data Types, for a table of type conversions currently supported by attribute promotion.

Attribute promotion by ndb_restore must be enabled explicitly, as follows:

1. Prepare the table to which the backup is to be restored. ndb_restore cannot be used to re-create the table with a different definition from the original; this means that you must either create

the table manually, or alter the columns which you wish to promote using ALTER TABLE after restoring the table metadata but before restoring the data.

2. Invoke ndb_restore with the --promote-attributes option (short form -A) when restoring the table data. Attribute promotion does not occur if this option is not used; instead, the restore operation fails with an error.

Prior to NDB 7.3.3, conversions between character data types and TEXT or BLOB were not handled correctly (Bug #17325051).

Prior to NDB 7.3.7, demotion of TEXT to TINYTEXT was not handled correctly (Bug #18875137).

When converting between character data types and TEXT or BLOB, only conversions between character types (CHAR and VARCHAR) and binary types (BINARY and VARBINARY) can be performed at the same time. For example, you cannot promote an INT column to BIGINT while promoting a VARCHAR column to TEXT in the same invocation of ndb_restore.

Converting between TEXT columns using different character sets is not supported. Beginning with NDB 7.3.7, it is expressly disallowed (Bug #18875137).

When performing conversions of character or binary types to TEXT or BLOB with ndb_restore, you may notice that it creates and uses one or more staging tables named *table_name*$ST*node_id*. These tables are not needed afterwards, and are normally deleted by ndb_restore following a successful restoration.

- --rebuild-indexes

Command-Line Format	--rebuild-indexes

Enable multithreaded rebuilding of the ordered indexes while restoring a native NDB backup. The number of threads used for building ordered indexes by ndb_restore with this option is controlled by the BuildIndexThreads data node configuration parameter and the number of LDMs.

It is necessary to use this option only for the first run of ndb_restore; this causes all ordered indexes to be rebuilt without using --rebuild-indexes again when restoring subsequent nodes. You should use this option prior to inserting new rows into the database; otherwise, it is possible for a row to be inserted that later causes a unique constraint violation when trying to rebuild the indexes.

Building of ordered indices is parallelized with the number of LDMs by default. Offline index builds performed during node and system restarts can be made faster using the BuildIndexThreads data node configuration parameter; this parameter has no effect on dropping and rebuilding of indexes by ndb_restore, which is performed online.

Rebuilding of unique indexes uses disk write bandwidth for redo logging and local checkpointing. An insufficient amount of this bandwith can lead to redo buffer overload or log overload errors. In such cases you can run ndb_restore --rebuild-indexes again; the process resumes at the point where the error occurred. You can also do this when you have encountered temporary errors. You can repeat execution of ndb_restore --rebuild-indexes indefinitely; you may be able to stop such errors by reducing the value of --parallelism. If the problem is insufficient space, you can increase the size of the redo log (FragmentLogFileSize node configuration parameter), or you can increase the speed at which LCPs are performed (MaxDiskWriteSpeed and related parameters), in order to free space more quickly.

- --restore-data, -r

Command-Line Format	--restore-data
Type	Boolean

Default Value	FALSE

Output NDB table data and logs.

- --restore-epoch, -e

Command-Line Format	--restore-epoch

Add (or restore) epoch information to the cluster replication status table. This is useful for starting replication on an NDB Cluster replica. When this option is used, the row in the mysql.ndb_apply_status having 0 in the id column is updated if it already exists; such a row is inserted if it does not already exist. (See Section 18.6.9, “NDB Cluster Backups With NDB Cluster Replication”.)

- --restore-meta, -m

Command-Line Format	--restore-meta
Type	Boolean
Default Value	FALSE

This option causes ndb_restore to print NDB table metadata.

The first time you run the ndb_restore restoration program, you also need to restore the metadata. In other words, you must re-create the database tables—this can be done by running it with the --restore-meta (-m) option. Restoring the metadata need be done only on a single data node; this is sufficient to restore it to the entire cluster.

Note

The cluster should have an empty database when starting to restore a backup. (In other words, you should start the data nodes with --initial prior to performing the restore.)

- --restore-privilege-tables

Command-Line Format	--restore-privilege-tables
Type	Boolean
Default Value	FALSE (If option is not used)

ndb_restore does not by default restore distributed MySQL privilege tables. This option causes ndb_restore to restore the privilege tables.

This works only if the privilege tables were converted to NDB before the backup was taken. For more information, see Section 18.5.12, “Distributed Privileges Using Shared Grant Tables”.

- --rewrite-database=*olddb,newdb*

Command-Line Format	--rewrite-database=olddb,newdb
Type	String
Default Value	none

This option makes it possible to restore to a database having a different name from that used in the backup. For example, if a backup is made of a database named products, you can restore the

data it contains to a database named inventory, use this option as shown here (omitting any other options that might be required):

```
shell> ndb_restore --rewrite-database=product,inventory
```

The option can be employed multiple times in a single invocation of ndb_restore. Thus it is possible to restore simultaneously from a database named db1 to a database named db2 and from a database named db3 to one named db4 using --rewrite-database=db1,db2 --rewrite-database=db3,db4. Other ndb_restore options may be used between multiple occurrences of --rewrite-database.

In the event of conflicts between multiple --rewrite-database options, the last --rewrite-database option used, reading from left to right, is the one that takes effect. For example, if --rewrite-database=db1,db2 --rewrite-database=db1,db3 is used, only --rewrite-database=db1,db3 is honored, and --rewrite-database=db1,db2 is ignored. It is also possible to restore from multiple databases to a single database, so that --rewrite-database=db1,db3 --rewrite-database=db2,db3 restores all tables and data from databases db1 and db2 into database db3.

Important

When restoring from multiple backup databases into a single target database using --rewrite-database, no check is made for collisions between table or other object names, and the order in which rows are restored is not guaranteed. This means that it is possible in such cases for rows to be overwritten and updates to be lost.

- --skip-broken-objects

Command-Line Format	--skip-broken-objects

This option causes ndb_restore to ignore corrupt tables while reading a native NDB backup, and to continue restoring any remaining tables (that are not also corrupted). Currently, the --skip-broken-objects option works only in the case of missing blob parts tables.

- --skip-table-check, -s

Command-Line Format	--skip-table-check

It is possible to restore data without restoring table metadata. By default when doing this, ndb_restore fails with an error if a mismatch is found between the table data and the table schema; this option overrides that behavior.

Some of the restrictions on mismatches in column definitions when restoring data using ndb_restore are relaxed; when one of these types of mismatches is encountered, ndb_restore does not stop with an error as it did previously, but rather accepts the data and inserts it into the target table while issuing a warning to the user that this is being done. This behavior occurs whether or not either of the options --skip-table-check or --promote-attributes is in use. These differences in column definitions are of the following types:

- Different COLUMN_FORMAT settings (FIXED, DYNAMIC, DEFAULT)
- Different STORAGE settings (MEMORY, DISK)
- Different default values
- Different distribution key settings

- `--skip-unknown-objects`

Command-Line Format	`--skip-unknown-objects`

This option causes `ndb_restore` to ignore any schema objects it does not recognize while reading a native `NDB` backup. This can be used for restoring a backup made from a cluster running (for example) NDB 7.4 to a cluster running NDB Cluster 7.3.

- `--tab=`*`dir_name`*, `-T` *`dir_name`*

Command-Line Format	`--tab=dir_name`
Type	Directory name

Causes `--print-data` to create dump files, one per table, each named *`tbl_name`*`.txt`. It requires as its argument the path to the directory where the files should be saved; use `.` for the current directory.

- `--verbose=#`

Command-Line Format	`--verbose=#`
Type	Numeric
Default Value	`1`
Minimum Value	`0`
Maximum Value	`255`

Sets the level for the verbosity of the output. The minimum is 0; the maximum is 255. The default value is 1.

Error reporting.
`ndb_restore` reports both temporary and permanent errors. In the case of temporary errors, it may able to recover from them, and reports `Restore successful, but encountered temporary error, please look at configuration` in such cases.

Important

After using `ndb_restore` to initialize an NDB Cluster for use in circular replication, binary logs on the SQL node acting as the replica are not automatically created, and you must cause them to be created manually. To cause the binary logs to be created, issue a `SHOW TABLES` statement on that SQL node before running `START SLAVE`. This is a known issue in NDB Cluster.

18.4.22.1 Restoring an NDB Backup to a Different Version of NDB Cluster

The following two sections provide information about restoring a native NDB backup to a different version of NDB Cluster from the version in which the backup was taken.

In addition, you should consult Section 18.2.7, “Upgrading and Downgrading NDB Cluster”, for other issues you may encounter when attempting to restore an NDB backup to a cluster running a different version of the NDB software.

It is also advisable to review What is New in NDB Cluster 8.0, as well as Section 2.11.3, “Changes in MySQL 5.6”, for other changes between NDB 8.0 and previous versions of NDB Cluster that may be relevant to your particular circumstances.

Restoring an NDB backup to a previous version of NDB Cluster

You may encounter issues when restoring a backup taken from a later version of NDB Cluster to a previous one, due to the use of features which do not exist in the earlier version. Some of these issues are listed here:

- Tables created in NDB 8.0 by default use the utf8mb4_ai_ci character set, which is not available in NDB 7.6 and earlier, and so cannot be read by an ndb_restore binary from one of these earlier versions. In such cases, it is necessary to alter any tables using utf8mb4_ai_ci so that they use a character set supported in the older version prior to performing the backup.
- Due to changes in how the MySQL Server and NDB handle table metadata, tables created or altered using the included MySQL server binary from NDB 8.0.14 or later cannot be restored using ndb_restore to an earlier version of NDB Cluster. Such tables use .sdi files which are not understood by older versions of mysqld.

  A backup taken in NDB 8.0.14 or later of tables which were created in NDB 8.0.13 or earlier, and which have not been altered since upgrading to NDB 8.0.14 or later, should be restorable to older versions of NDB Cluster.

  Since it is possible to restore metadata and table data separately, you can in such cases restore the table schemas from a dump made using mysqldump, or by executing the necessary CREATE TABLE statements manually, then import only the table data using ndb_restore with the --restore-data option.
- Encrypted backups created in NDB 8.0.22 and later cannot be restored using ndb_restore from NDB 8.0.21 or earlier.
- The NDB_STORED_USER privilege is not supported prior to NDB 8.0.18.
- NDB Cluster 8.0.18 and later supports up to 144 data nodes, while earlier versions support a maximum of only 48 data nodes. See Restoring to Fewer Nodes Than the Original, for information with situations in which this incompatibility causes an issue.

Restoring an NDB backup to a later version of NDB Cluster

In general, it should be possible to restore a backup created using the ndb_mgm client START BACKUP command in an older version of NDB to a newer version, provided that you use the ndb_restore binary that comes with the newer version. (It may be possible to use the older version of ndb_restore, but this is not recommended.) Additional potential issues are listed here:

- When restoring the metadata from a backup (--restore-meta option), ndb_restore normally attempts to reproduce the captured table schema exactly as it was when the backup was taken.

  Tables created in versions of NDB prior to 8.0.14 use .frm files for their metadata. These files can be read by the mysqld in NDB 8.0.14 and later, which can use the information contained therein to create the .sdi files used by the MySQL data dictionary in later versions.
- When restoring an older backup to a newer version of NDB, it may not be possible to take advantage of newer features such as hashmap partitioning, greater number of hashmap buckets, read backup, and different partitioning layouts. For this reason, it may be preferable to restore older schemas using mysqldump and the mysql client, which allows NDB to make use of the new schema features.
- Tables using the old temporal types which did not support fractional seconds (used prior to MySQL 5.6.4 and NDB 7.3.31) cannot be restored to NDB 8.0 using ndb_restore. You can check such tables using CHECK TABLE, and then upgrade them to the newer temporal column format, if necessary, using REPAIR TABLE in the mysql client; this must be done prior to taking the backup. See Preparing Your Installation for Upgrade, for more information.

  You also also restore such tables using a dump created with mysqldump.
- Distributed grant tables created in NDB 7.6 and earlier are not supported in NDB 8.0. Such tables can be restored to an NDB 8.0 cluster, but they have no effect on access control.

18.4.22.2 Restoring to a different number of data nodes

It is possible to restore from an NDB backup to a cluster having a different number of data nodes than the original from which the backup was taken. The following two sections discuss, respectively, the

cases where the target cluster has a lesser or greater number of data nodes than the source of the backup.

Restoring to Fewer Nodes Than the Original

You can restore to a cluster having fewer data nodes than the original provided that the larger number of nodes is an even multiple of the smaller number. In the following example, we use a backup taken on a cluster having four data nodes to a cluster having two data nodes.

1. The management server for the original cluster is on host `host10`. The original cluster has four data nodes, with the node IDs and host names shown in the following extract from the management server's `config.ini` file:

   ```
   [ndbd]
   NodeId=2
   HostName=host2

   [ndbd]
   NodeId=4
   HostName=host4

   [ndbd]
   NodeId=6
   HostName=host6

   [ndbd]
   NodeId=8
   HostName=host8
   ```

   We assume that each data node was originally started with `ndbmtd --ndb-connectstring=host10` or the equivalent.

2. Perform a backup in the normal manner. See Section 18.5.8.2, “Using The NDB Cluster Management Client to Create a Backup”, for information about how to do this.

3. The files created by the backup on each data node are listed here, where *`N`* is the node ID and *`B`* is the backup ID.

   - `BACKUP-B-0.N.Data`
   - `BACKUP-B.N.ctl`
   - `BACKUP-B.N.log`

   These files are found under `BackupDataDir/BACKUP/BACKUP-B`, on each data node. For the rest of this example, we assume that the backup ID is 1.

   Have all of these files available for later copying to the new data nodes (where they can be accessed on the data node's local file system by `ndb_restore`). It is simplest to copy them all to a single location; we assume that this is what you have done.

4. The management server for the target cluster is on host `host20`, and the target has two data nodes, with the node IDs and host names shown, from the management server `config.ini` file on `host20`:

   ```
   [ndbd]
   NodeId=3
   hostname=host3

   [ndbd]
   NodeId=5
   hostname=host5
   ```

Each of the data node processes on host3 and host5 should be started with ndbmtd -c host20 --initial or the equivalent, so that the new (target) cluster starts with clean data node file systems.

5. Copy two different sets of two backup files to each of the target data nodes. For this example, copy the backup files from nodes 2 and 4 from the original cluster to node 3 in the target cluster. These files are listed here:

   - BACKUP-1-0.2.Data
   - BACKUP-1.2.ctl
   - BACKUP-1.2.log
   - BACKUP-1-0.6.Data
   - BACKUP-1.6.ctl
   - BACKUP-1.6.log

   Then copy the backup files from nodes 6 and 8 to node 5; these files are shown in the following list:

   - BACKUP-1-0.4.Data
   - BACKUP-1.4.ctl
   - BACKUP-1.4.log
   - BACKUP-1-0.8.Data
   - BACKUP-1.8.ctl
   - BACKUP-1.8.log

   For the remainder of this example, we assume that the respective backup files have been saved to the directory /BACKUP-1 on each of nodes 3 and 5.

6. On each of the two target data nodes, you must restore from both sets of backups. First, restore the backups from nodes 2 and 4 to node 3 by invoking ndb_restore on host3 as shown here:

```
shell> ndb_restore -c host20 --nodeid=2 --backupid=1 --restore-data --backup-path=/BACKUP-1

shell> ndb_restore -c host20 --nodeid=4 --backupid=1 --restore-data --backup-path=/BACKUP-1
```

   Then restore the backups from nodes 6 and 8 to node 5 by invoking ndb_restore on host5, like this:

```
shell> ndb_restore -c host20 --nodeid=6 --backupid=1 --restore-data --backup-path=/BACKUP-1

shell> ndb_restore -c host20 --nodeid=8 --backupid=1 --restore-data --backup-path=/BACKUP-1
```

Restoring to More Nodes Than the Original

The node ID specified for a given ndb_restore command is that of the node in the original backup and not that of the data node to restore it to. When performing a backup using the method described in this section, ndb_restore connects to the management server and obtains a list of data nodes in the cluster the backup is being restored to. The restored data is distributed accordingly, so that the number of nodes in the target cluster does not need to be to be known or calculated when performing the backup.

Note

When changing the total number of LCP threads or LQH threads per node group, you should recreate the schema from backup created using `mysqldump`.

1. *Create the backup of the data*. You can do this by invoking the `ndb_mgm` client `START BACKUP` command from the system shell, like this:

   ```
   shell> ndb_mgm -e "START BACKUP 1"
   ```

   This assumes that the desired backup ID is 1.

2. Create a backup of the schema (see also below):

   ```
   shell> mysqldump --no-data --routines --events --triggers --databases > myschema.sql
   ```

   Important

   You must not make any schema changes between the first and second steps.

3. Copy the backup directories from above to the new cluster. For example if the backup you want to restore is has ID 1 and `BackupDataDir` = `/backups/node_nodeid`, then the path to the backup on this node is `/backups/node_1/BACKUP/BACKUP-1`. Inside this directory there are three files, listed here:

   - `BACKUP-1-0.1.Data`
   - `BACKUP-1.1.ctl`
   - `BACKUP-1.1.log`

   You should copy the entire directory to the new node.

There is no requirement for the backup to be restored from a specific node or nodes.

To restore from the backup just created, perform the following steps:

1. *Restore the schema*. Import the schema file using the `mysql` client, as shown here:

   ```
   shell> mysql < myschema.sql
   ```

2. *Restore the data*. The following commands can be run in parallel:

   ```
   ndb_restore --nodeid=1 --backupid=1 --restore-data --backup-path=/backups/node_1/BACKUP/BACKUP-1 --c
   ndb_restore --nodeid=2 --backupid=1 --restore-data --backup-path=/backups/node_2/BACKUP/BACKUP-1 --c
   ndb_restore --nodeid=3 --backupid=1 --restore-data --backup-path=/backups/node_3/BACKUP/BACKUP-1 --c
   ndb_restore --nodeid=4 --backupid=1 --restore-data --backup-path=/backups/node_4/BACKUP/BACKUP-1 --c
   ndb_restore --nodeid=5 --backupid=1 --restore-data --backup-path=/backups/node_5/BACKUP/BACKUP-1 --c
   ndb_restore --nodeid=6 --backupid=1 --restore-data --backup-path=/backups/node_6/BACKUP/BACKUP-1 --c
   ndb_restore --nodeid=7 --backupid=1 --restore-data --backup-path=/backups/node_7/BACKUP/BACKUP-1 --c
   ndb_restore --nodeid=8 --backupid=1 --restore-data --backup-path=/backups/node_8/BACKUP/BACKUP-1 --c
   ```

   Add the `--ndb-connectstring` option as needed.

   If you in 3. for example copied the backups from the “old” nodes having node IDs 1 and 2 to a “new” node whose node ID is 1, you should perform the two invocations of `ndb_restore` with `--nodeid=1` and `--nodeid=2` on the new node that has 1 as its node ID.

3. *Rebuild the indexes*. These were disabled by the `--disable-indexes` option used in the commands just shown. Recreating the indexes avoids errors due to the restore not being consistent at all points. Rebuilding the indexes can also improve performance in some cases. To rebuild the indexes, execute the following command once, on a single node:

   ```
   shell> ndb_restore --nodeid=1 --backupid=1 --backup-path=/backups/node_1/BACKUP/BACKUP-1 --rebuild-i
   ```

Important

You should be aware that the supported number of partitions in each table depends on the number of data nodes, node groups, and LDM threads in the cluster. Other conditions (such as the values of `MaxNoOfExecutionThreads`, `ThreadConfig`, `NoOfReplicas`, and so on) being the same, a cluster with (for example) two data nodes supports fewer partitions than a cluster with eight data nodes supports. This means that using `ndb_restore --restore-meta` to restore the schema does not always work since this restores a given table with the same number of partitions as in the original; it is safer to restore the schema from a backup written by `mysqldump`—as in the example shown previously—when restoring to a cluster having fewer data nodes, LDM threads, or both, than were used in the original cluster.

The support for fewer partitions when restoring to a smaller cluster also means the maximum number of rows per table is lower.

18.4.23 ndb_select_all — Print Rows from an NDB Table

`ndb_select_all` prints all rows from an `NDB` table to `stdout`.

Usage

```
ndb_select_all -c connection_string tbl_name -d db_name [> file_name]
```

The following table includes options that are specific to the NDB Cluster native backup restoration program `ndb_select_all`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_select_all`), see Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 18.35 Command-line options for the ndb_select_all program

Format	Description	Added, Deprecated, or Removed
`--database=dbname`, `-d`	Name of database in which table is found	(Supported in all MySQL 5.6 based releases)
`--parallelism=#`, `-p`	Degree of parallelism	(Supported in all MySQL 5.6 based releases)
`--lock=#`, `-l`	Lock type	(Supported in all MySQL 5.6 based releases)
`--order=index`, `-o`	Sort resultset according to index having this name	(Supported in all MySQL 5.6 based releases)
`--descending`, `-z`	Sort resultset in descending order (requires --order)	(Supported in all MySQL 5.6 based releases)
`--header`, `-h`	Print header (set to 0|FALSE to disable headers in output)	(Supported in all MySQL 5.6 based releases)
`--useHexFormat`, `-x`	Output numbers in hexadecimal format	(Supported in all MySQL 5.6 based releases)
`--delimiter=char`,	Set column delimiter	(Supported in all MySQL 5.6 based releases)

Format	Description	Added, Deprecated, or Removed
-D		
--disk	Print disk references (useful only for Disk Data tables having nonindexed columns)	(Supported in all MySQL 5.6 based releases)
--rowid	Print row ID	(Supported in all MySQL 5.6 based releases)
--gci	Include GCI in output	(Supported in all MySQL 5.6 based releases)
--gci64	Include GCI and row epoch in output	(Supported in all MySQL 5.6 based releases)
--tupscan,	Scan in tup order	(Supported in all MySQL 5.6 based releases)
-t		
--nodata	Do not print table column data	(Supported in all MySQL 5.6 based releases)

- `--database=dbname`, `-d dbname`

  Name of the database in which the table is found. The default value is `TEST_DB`.

- `parallelism=#`, `-p #`

  Specifies the degree of parallelism.

- `--lock=lock_type`, `-l lock_type`

  Employs a lock when reading the table. Possible values for *`lock_type`* are:

  - `0`: Read lock
  - `1`: Read lock with hold
  - `2`: Exclusive read lock

  There is no default value for this option.

- `--order=index_name`, `-o index_name`

  Orders the output according to the index named *`index_name`*.

  Note

  This is the name of an index, not of a column, and that the index must have been explicitly named when created.

- `--descending`, `-z`

  Sorts the output in descending order. This option can be used only in conjunction with the `-o` (`--order`) option.

- `--header=FALSE`

  Excludes column headers from the output.

- `--useHexFormat -x`

  Causes all numeric values to be displayed in hexadecimal format. This does not affect the output of numerals contained in strings or datetime values.

- `--delimiter=character`, `-D character`

  Causes the *`character`* to be used as a column delimiter. Only table data columns are separated by this delimiter.

  The default delimiter is the tab character.

- `--disk`

  Adds a disk reference column to the output. The column is nonempty only for Disk Data tables having nonindexed columns.

- `--rowid`

  Adds a `ROWID` column providing information about the fragments in which rows are stored.

- `--gci`

  Adds a `GCI` column to the output showing the global checkpoint at which each row was last updated. See Section 18.1, “NDB Cluster Overview”, and Section 18.5.3.2, “NDB Cluster Log Events”, for more information about checkpoints.

- `--gci64`

  Adds a `ROW$GCI64` column to the output showing the global checkpoint at which each row was last updated, as well as the number of the epoch in which this update occurred.

- `--tupscan`, `-t`

  Scan the table in the order of the tuples.

- `--nodata`

  Causes any table data to be omitted.

Sample Output

Output from a MySQL `SELECT` statement:

```
mysql> SELECT * FROM ctest1.fish;
+----+-----------+
| id | name      |
+----+-----------+
|  3 | shark     |
|  6 | puffer    |
|  2 | tuna      |
|  4 | manta ray |
|  5 | grouper   |
|  1 | guppy     |
+----+-----------+
6 rows in set (0.04 sec)
```

Output from the equivalent invocation of `ndb_select_all`:

```
shell> ./ndb_select_all -c localhost fish -d ctest1
id      name
3       [shark]
6       [puffer]
2       [tuna]
4       [manta ray]
5       [grouper]
1       [guppy]
6 rows returned

NDBT_ProgramExit: 0 - OK
```

All string values are enclosed by square brackets ([...]) in the output of ndb_select_all. Now consider the table created and populated as shown here:

```
CREATE TABLE dogs (
    id INT(11) NOT NULL AUTO_INCREMENT,
    name VARCHAR(25) NOT NULL,
    breed VARCHAR(50) NOT NULL,
    PRIMARY KEY pk (id),
    KEY ix (name)
)
TABLESPACE ts STORAGE DISK
ENGINE=NDBCLUSTER;

INSERT INTO dogs VALUES
    ('', 'Lassie', 'collie'),
    ('', 'Scooby-Doo', 'Great Dane'),
    ('', 'Rin-Tin-Tin', 'Alsatian'),
    ('', 'Rosscoe', 'Mutt');
```

This demonstrates the use of several additional ndb_select_all options:

```
shell> ./ndb_select_all -d ctest1 dogs -o ix -z --gci --disk
GCI     id name          breed        DISK_REF
834461  2  [Scooby-Doo]  [Great Dane] [ m_file_no: 0 m_page: 98 m_page_idx: 0 ]
834878  4  [Rosscoe]     [Mutt]       [ m_file_no: 0 m_page: 98 m_page_idx: 16 ]
834463  3  [Rin-Tin-Tin] [Alsatian]   [ m_file_no: 0 m_page: 34 m_page_idx: 0 ]
835657  1  [Lassie]      [Collie]     [ m_file_no: 0 m_page: 66 m_page_idx: 0 ]
4 rows returned

NDBT_ProgramExit: 0 - OK
```

18.4.24 ndb_select_count — Print Row Counts for NDB Tables

ndb_select_count prints the number of rows in one or more NDB tables. With a single table, the result is equivalent to that obtained by using the MySQL statement SELECT COUNT(*) FROM *tbl_name*.

Usage

```
ndb_select_count [-c connection_string] -ddb_name tbl_name[, tbl_name2[, ...]]
```

The following table includes options that are specific to the NDB Cluster native backup restoration program ndb_select_count. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_select_count), see Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 18.36 Command-line options for the ndb_select_count program

Format	Description	Added, Deprecated, or Removed
--database=dbname, -d	Name of database in which table is found	(Supported in all MySQL 5.6 based releases)
--parallelism=#, -p	Degree of parallelism	(Supported in all MySQL 5.6 based releases)
--lock=#, -l	Lock type	(Supported in all MySQL 5.6 based releases)

You can obtain row counts from multiple tables in the same database by listing the table names separated by spaces when invoking this command, as shown under **Sample Output**.

Sample Output

```
shell> ./ndb_select_count -c localhost -d ctest1 fish dogs
```

```
6 records in table fish
4 records in table dogs

NDBT_ProgramExit: 0 - OK
```

18.4.25 ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)

Note

This feature is deprecated and should be avoided. It is subject to removal in a future version of NDB Cluster.

ndb_setup.py starts the NDB Cluster Auto-Installer and opens the installer's Start page in the default Web browser.

Important

This program is intended to be invoked as a normal user, and not with the mysql, system root, or other administrative account.

This section describes usage of and program options for the command-line tool only. For information about using the Auto-Installer GUI that is spawned when ndb_setup.py is invoked, see Section 18.2.8, “The NDB Cluster Auto-Installer (No longer supported)”.

Usage

All platforms:

```
ndb_setup.py [options]
```

Additionally, on Windows platforms only:

```
setup.bat [options]
```

The following table includes all options that are supported by the NDB Cluster installation and configuration program ndb_setup.py. Additional descriptions follow the table.

Table 18.37 Command-line options for the ndb_setup.py program

Format	Description	Added, Deprecated, or Removed
--browser-start-page=filename, -s	Page that web browser opens when starting	(Supported in all MySQL 5.6 based releases)
--ca-certs-file=filename, -a	File containing list of client certificates allowed to connect to server	(Supported in all MySQL 5.6 based releases)
--cert-file=filename, -c	File containing X509 certificate identifying server	(Supported in all MySQL 5.6 based releases)
--debug-level=level, -d	Python logging module debug level; one of DEBUG, INFO, WARNING (default), ERROR, or CRITICAL	(Supported in all MySQL 5.6 based releases)
--help,	Print help message	(Supported in all MySQL 5.6 based releases)

Format	Description	Added, Deprecated, or Removed
-h --key-file=file, -k	Specify file containing private key (if not included in --cert-file)	(Supported in all MySQL 5.6 based releases)
--no-browser, -n	Do not open start page in browser, merely start tool	(Supported in all MySQL 5.6 based releases)
--port=#, -p	Specify port used by web server	(Supported in all MySQL 5.6 based releases)
--server-log-file=file, -o	Log requests to this file; use '-' to force logging to stderr instead	(Supported in all MySQL 5.6 based releases)
--server-name=name, -N	Name of server to connect to	(Supported in all MySQL 5.6 based releases)
--use-https, -S	Use encrypted (HTTPS) client/server connection	(Supported in all MySQL 5.6 based releases)

- --browser-start-page=*file*, -s

Command-Line Format	--browser-start-page=filename
Type	String
Default Value	index.html

Specify the file to open in the browser as the installation and configuration Start page. The default is index.html.

- --ca-certs-file=*file*, -a

Command-Line Format	--ca-certs-file=filename
Type	File name
Default Value	[none]

Specify a file containing a list of client certificates which are allowed to connect to the server. The default is an empty string, which means that no client authentication is used.

- --cert-file=*file*, -c

Command-Line Format	--cert-file=filename
Type	File name
Default Value	/usr/share/mysql/mcc/cfg.pem

Specify a file containing an X.509 certificate which identifies the server. It is possible for the certificate to be self-signed. The default is cfg.pem.

- --debug-level=*level*, -d

Command-Line Format	--debug-level=level
Type	Enumeration
Default Value	WARNING

Valid Values	WARNING DEBUG INFO ERROR CRITICAL

Set the Python logging module debug level. This is one of DEBUG, INFO, WARNING, ERROR, or CRITICAL. WARNING is the default.

- --help, -h

Command-Line Format	--help

Print a help message.

- --key-file=*file*, -d

Command-Line Format	--key-file=file
Type	File name
Default Value	[none]

Specify a file containing the private key if this is not included in the X. 509 certificate file (--cert-file). The default is an empty string, which means that no such file is used.

- --no-browser, -n

Command-Line Format	--no-browser

Start the installation and configuration tool, but do not open the Start page in a browser.

- --port=#, -p

Command-Line Format	--port=#
Type	Numeric
Default Value	8081
Minimum Value	1
Maximum Value	65535

Set the port used by the web server. The default is 8081.

- --server-log-file=*file*, -o

Command-Line Format	--server-log-file=file
Type	File name
Default Value	ndb_setup.log
Valid Values	ndb_setup.log - (Log to stderr)

Log requests to this file. The default is ndb_setup.log. To specify logging to stderr, rather than to a file, use a - (dash character) for the file name.

- --server-name=*host*, -N

Command-Line Format	--server-name=name
Type	String
Default Value	localhost

Specify the host name or IP address for the browser to use when connecting. The default is localhost.

- --use-https, -S

Command-Line Format	--use-https

Make the browser use a secure (HTTPS) connection with the server.

18.4.26 ndb_show_tables — Display List of NDB Tables

ndb_show_tables displays a list of all NDB database objects in the cluster. By default, this includes not only both user-created tables and NDB system tables, but NDB-specific indexes, internal triggers, and NDB Cluster Disk Data objects as well.

The following table includes options that are specific to the NDB Cluster native backup restoration program ndb_show_tables. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_show_tables), see Section 18.4.29, "Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs".

Table 18.38 Command-line options for the ndb_show_tables program

Format	Description	Added, Deprecated, or Removed
--database=string, -d	Specifies database in which table is found; database name must be followed by table name	(Supported in all MySQL 5.6 based releases)
--loops=#, -l	Number of times to repeat output	(Supported in all MySQL 5.6 based releases)
--parsable, -p	Return output suitable for MySQL LOAD DATA statement	(Supported in all MySQL 5.6 based releases)
--show-temp-status	Show table temporary flag	(Supported in all MySQL 5.6 based releases)
--type=#, -t	Limit output to objects of this type	(Supported in all MySQL 5.6 based releases)
--unqualified, -u	Do not qualify table names	(Supported in all MySQL 5.6 based releases)

Usage

```
ndb_show_tables [-c connection_string]
```

- --database, -d

Specifies the name of the database in which the desired table is found. If this option is given, the name of a table must follow the database name.

In NDB 7.4.8 and later, if this option has not been specified, and no tables are found in the TEST_DB database, ndb_show_tables issues a warning (Bug #50633, Bug #11758430).

- `--loops`, `-l`

  Specifies the number of times the utility should execute. This is 1 when this option is not specified, but if you do use the option, you must supply an integer argument for it.

- `--parsable`, `-p`

  Using this option causes the output to be in a format suitable for use with `LOAD DATA`.

- `--show-temp-status`

  If specified, this causes temporary tables to be displayed.

- `--type`, `-t`

  Can be used to restrict the output to one type of object, specified by an integer type code as shown here:

  - `1`: System table
  - `2`: User-created table
  - `3`: Unique hash index

  Any other value causes all `NDB` database objects to be listed (the default).

- `--unqualified`, `-u`

  If specified, this causes unqualified object names to be displayed.

Note

Only user-created NDB Cluster tables may be accessed from MySQL; system tables such as `SYSTAB_0` are not visible to `mysqld`. However, you can examine the contents of system tables using `NDB` API applications such as `ndb_select_all` (see Section 18.4.23, "`ndb_select_all` — Print Rows from an NDB Table").

Prior to NDB 7.4.28, this program printed `NDBT_ProgramExit - status` upon completion of its run, due to an unnecessary dependency on the `NDBT` testing library. This dependency is has now been removed, eliminating the extraneous output.

18.4.27 `ndb_size.pl` — NDBCLUSTER Size Requirement Estimator

This is a Perl script that can be used to estimate the amount of space that would be required by a MySQL database if it were converted to use the `NDBCLUSTER` storage engine. Unlike the other utilities discussed in this section, it does not require access to an NDB Cluster (in fact, there is no reason for it to do so). However, it does need to access the MySQL server on which the database to be tested resides.

Requirements

- A running MySQL server. The server instance does not have to provide support for NDB Cluster.
- A working installation of Perl.
- The `DBI` module, which can be obtained from CPAN if it is not already part of your Perl installation. (Many Linux and other operating system distributions provide their own packages for this library.)
- A MySQL user account having the necessary privileges. If you do not wish to use an existing account, then creating one using `GRANT USAGE ON db_name.*`—where *`db_name`* is the name of the database to be examined—is sufficient for this purpose.

ndb_size.pl can also be found in the MySQL sources in storage/ndb/tools.

The following table includes options that are specific to the NDB Cluster program ndb_size.pl. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_size.pl), see Section 18.4.29, "Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs".

Table 18.39 Command-line options for the ndb_size.pl program

Format	Description	Added, Deprecated, or Removed
--database=dbname	Database or databases to examine; a comma-delimited list; default is ALL (use all databases found on server)	(Supported in all MySQL 5.6 based releases)
--hostname[:port]	Specify host and optional port as host[:port]	(Supported in all MySQL 5.6 based releases)
--socket=file_name	Specify socket to connect to	(Supported in all MySQL 5.6 based releases)
--user=string	Specify MySQL user name	(Supported in all MySQL 5.6 based releases)
--password=string	Specify MySQL user password	(Supported in all MySQL 5.6 based releases)
--format=string	Set output format (text or HTML)	(Supported in all MySQL 5.6 based releases)
--excludetables=tbl_list	Skip any tables in comma-separated list	(Supported in all MySQL 5.6 based releases)
--excludedbs=db_list	Skip any databases in comma-separated list	(Supported in all MySQL 5.6 based releases)
--savequeries=file	Saves all queries on database into file specified	(Supported in all MySQL 5.6 based releases)
--loadqueries=file	Loads all queries from file specified; does not connect to database	(Supported in all MySQL 5.6 based releases)
--real_table_name=table	Designates table to handle unique index size calculations	(Supported in all MySQL 5.6 based releases)

Usage

```
perl ndb_size.pl [--database={db_name|ALL}] [--hostname=host[:port]] [--socket=socket] \
      [--user=user] [--password=password]  \
      [--help|-h] [--format={html|text}] \
      [--loadqueries=file_name] [--savequeries=file_name]
```

By default, this utility attempts to analyze all databases on the server. You can specify a single database using the --database option; the default behavior can be made explicit by using ALL for the name of the database. You can also exclude one or more databases by using the --excludedbs option with a comma-separated list of the names of the databases to be skipped. Similarly, you can cause specific tables to be skipped by listing their names, separated by commas, following the optional --excludetables option. A host name can be specified using --hostname; the default is localhost. You can specify a port in addition to the host using *host*:*port* format for the value of --hostname. The default port number is 3306. If necessary, you can also specify a socket; the default is /var/lib/mysql.sock. A MySQL user name and password can be specified the corresponding options shown. It also possible to control the format of the output using the --format option; this can take either of the values html or text, with text being the default. An example of the text output is shown here:

```
shell> ndb_size.pl --database=test --socket=/tmp/mysql.sock
ndb_size.pl report for database: 'test' (1 tables)
----------------------------------------------------
Connected to: DBI:mysql:host=localhost;mysql_socket=/tmp/mysql.sock

Including information for versions: 4.1, 5.0, 5.1

test.t1
-------

DataMemory for Columns (* means varsized DataMemory):
         Column Name            Type  Varsized   Key  4.1  5.0   5.1
     HIDDEN_NDB_PKEY          bigint             PRI    8    8     8
                  c2     varchar(50)         Y         52   52    4*
                  c1         int(11)                    4    4     4
                                                       --   --    --
Fixed Size Columns DM/Row                              64   64    12
   Varsize Columns DM/Row                               0    0     4

DataMemory for Indexes:
   Index Name                 Type        4.1        5.0        5.1
      PRIMARY                BTREE         16         16         16
                                           --         --         --
       Total Index DM/Row                  16         16         16

IndexMemory for Indexes:
               Index Name        4.1        5.0        5.1
                  PRIMARY         33         16         16
                                  --         --         --
           Indexes IM/Row         33         16         16

Summary (for THIS table):
                                 4.1        5.0        5.1
    Fixed Overhead DM/Row         12         12         16
           NULL Bytes/Row          4          4          4
           DataMemory/Row         96         96         48
                    (Includes overhead, bitmap and indexes)

  Varsize Overhead DM/Row          0          0          8
   Varsize NULL Bytes/Row          0          0          4
       Avg Varside DM/Row          0          0         16

                No. Rows           0          0          0

        Rows/32kb DM Page        340        340        680
Fixedsize DataMemory (KB)          0          0          0

Rows/32kb Varsize DM Page          0          0       2040
  Varsize DataMemory (KB)          0          0          0

         Rows/8kb IM Page        248        512        512
         IndexMemory (KB)          0          0          0

Parameter Minimum Requirements
------------------------------
* indicates greater than default

               Parameter     Default        4.1         5.0         5.1
          DataMemory (KB)       81920          0           0           0
       NoOfOrderedIndexes         128          1           1           1
                NoOfTables        128          1           1           1
          IndexMemory (KB)       18432          0           0           0
    NoOfUniqueHashIndexes           64          0           0           0
           NoOfAttributes         1000          3           3           3
             NoOfTriggers          768          5           5           5
```

For debugging purposes, the Perl arrays containing the queries run by this script can be read from the file specified using can be saved to a file using `--savequeries`; a file containing such arrays to be read during script execution can be specified using `--loadqueries`. Neither of these options has a default value.

To produce output in HTML format, use the `--format` option and redirect the output to a file, as shown here:

```
shell> ndb_size.pl --database=test --socket=/tmp/mysql.sock --format=html > ndb_size.html
```

(Without the redirection, the output is sent to `stdout`.)

The output from this script includes the following information:

- Minimum values for the `DataMemory`, `IndexMemory`, `MaxNoOfTables`, `MaxNoOfAttributes`, `MaxNoOfOrderedIndexes`, `MaxNoOfUniqueHashIndexes`, and `MaxNoOfTriggers` configuration parameters required to accommodate the tables analyzed.
- Memory requirements for all of the tables, attributes, ordered indexes, and unique hash indexes defined in the database.
- The `IndexMemory` and `DataMemory` required per table and table row.

18.4.28 ndb_waiter — Wait for NDB Cluster to Reach a Given Status

`ndb_waiter` repeatedly (each 100 milliseconds) prints out the status of all cluster data nodes until either the cluster reaches a given status or the `--timeout` limit is exceeded, then exits. By default, it waits for the cluster to achieve `STARTED` status, in which all nodes have started and connected to the cluster. This can be overridden using the `--no-contact` and `--not-started` options.

The node states reported by this utility are as follows:

- `NO_CONTACT`: The node cannot be contacted.
- `UNKNOWN`: The node can be contacted, but its status is not yet known. Usually, this means that the node has received a `START` or `RESTART` command from the management server, but has not yet acted on it.
- `NOT_STARTED`: The node has stopped, but remains in contact with the cluster. This is seen when restarting the node using the management client's `RESTART` command.
- `STARTING`: The node's `ndbd` process has started, but the node has not yet joined the cluster.
- `STARTED`: The node is operational, and has joined the cluster.
- `SHUTTING_DOWN`: The node is shutting down.
- `SINGLE USER MODE`: This is shown for all cluster data nodes when the cluster is in single user mode.

The following table includes options that are specific to the NDB Cluster native backup restoration program `ndb_waiter`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_waiter`), see Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 18.40 Command-line options for the ndb_waiter program

Format	Description	Added, Deprecated, or Removed
`--no-contact`, `-n`	Wait for cluster to reach NO CONTACT state	(Supported in all MySQL 5.6 based releases)
`--not-started`	Wait for cluster to reach NOT STARTED state	(Supported in all MySQL 5.6 based releases)
`--single-user`	Wait for cluster to enter single user mode	(Supported in all MySQL 5.6 based releases)

Format	Description	Added, Deprecated, or Removed
`--timeout=#`, `-t`	Wait this many seconds, then exit whether or not cluster has reached desired state	(Supported in all MySQL 5.6 based releases)
`--nowait-nodes=list`	List of nodes not to be waited for	(Supported in all MySQL 5.6 based releases)
`--wait-nodes=list`, `-w`	List of nodes to be waited for	(Supported in all MySQL 5.6 based releases)

Usage

```
ndb_waiter [-c connection_string]
```

Additional Options

- `--no-contact`, `-n`

  Instead of waiting for the `STARTED` state, `ndb_waiter` continues running until the cluster reaches `NO_CONTACT` status before exiting.

- `--not-started`

  Instead of waiting for the `STARTED` state, `ndb_waiter` continues running until the cluster reaches `NOT_STARTED` status before exiting.

- `--timeout=seconds`, `-t seconds`

  Time to wait. The program exits if the desired state is not achieved within this number of seconds. The default is 120 seconds (1200 reporting cycles).

- `--single-user`

  The program waits for the cluster to enter single user mode.

- `--nowait-nodes=list`

  When this option is used, `ndb_waiter` does not wait for the nodes whose IDs are listed. The list is comma-delimited; ranges can be indicated by dashes, as shown here:

  ```
  shell> ndb_waiter --nowait-nodes=1,3,7-9
  ```

  Important

  Do *not* use this option together with the `--wait-nodes` option.

- `--wait-nodes=list`, `-w list`

  When this option is used, `ndb_waiter` waits only for the nodes whose IDs are listed. The list is comma-delimited; ranges can be indicated by dashes, as shown here:

  ```
  shell> ndb_waiter --wait-nodes=2,4-6,10
  ```

  Important

  Do *not* use this option together with the `--nowait-nodes` option.

Sample Output. Shown here is the output from `ndb_waiter` when run against a 4-node cluster in which two nodes have been shut down and then started again manually. Duplicate reports (indicated by `...`) are omitted.

```
shell> ./ndb_waiter -c localhost

Connecting to mgmsrv at (localhost)
State node 1 STARTED
State node 2 NO_CONTACT
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 UNKNOWN
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 UNKNOWN
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 STARTING
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTED
State node 3 STARTED
State node 4 STARTING
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTED
State node 3 STARTED
State node 4 STARTED
Waiting for cluster enter state STARTED

NDBT_ProgramExit: 0 - OK
```

Note

If no connection string is specified, then `ndb_waiter` tries to connect to a management on `localhost`, and reports `Connecting to mgmsrv at (null)`.

Prior to NDB 7.4.28, this program printed `NDBT_ProgramExit - status` upon completion of its run, due to an unnecessary dependency on the `NDBT` testing library. This dependency is has now been removed, eliminating the extraneous output.

18.4.29 Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs

All NDB Cluster programs accept the options described in this section, with the following exceptions:

- `mysqld`
- `ndb_print_backup_file`
- `ndb_print_schema_file`
- `ndb_print_sys_file`

Note

Users of earlier NDB Cluster versions should note that some of these options have been changed to make them consistent with one another, and also with `mysqld`. You can use the `--help` option with any NDB Cluster program—with the exception of `ndb_print_backup_file`, `ndb_print_schema_file`, and `ndb_print_sys_file`—to view a list of the options which the program supports.

The options in the following table are common to all NDB Cluster executables (except those noted previously in this section).

Table 18.41 Command-line options common to all MySQL NDB Cluster programs

Format	Description	Added, Deprecated, or Removed
`--character-sets-dir=dir_name`	Directory where character sets are installed	(Supported in all MySQL 5.6 based releases)
`--connect-retries=#`	Set the number of times to retry a connection before giving up	ADDED: NDB 7.4.9
`--connect-retry-delay=#`	Time to wait between attempts to contact a management server, in seconds	ADDED: NDB 7.4.9
`--core-file`	Write core on errors (defaults to TRUE in debug builds)	(Supported in all MySQL 5.6 based releases)
`--debug=options`	Enable output from debug calls. Can be used only for versions compiled with debugging enabled	(Supported in all MySQL 5.6 based releases)
`--defaults-extra-file=filename`	Read this file after global option files are read	(Supported in all MySQL 5.6 based releases)
`--defaults-file=filename`	Read default options from this file	(Supported in all MySQL 5.6 based releases)
`--defaults-group-suffix`	Also read groups with names ending in this suffix	(Supported in all MySQL 5.6 based releases)
`--help`, `--usage`, `-?`	Display help message and exit	(Supported in all MySQL 5.6 based releases)
`--login-path=path`	Read this path from the login file	(Supported in all MySQL 5.6 based releases)
`--ndb-connectstring=connectstring`	Set connection string for connecting to ndb_mgmd. Syntax: [nodeid=<id>;]	(Supported in all MySQL 5.6 based releases)

Format	Description	Added, Deprecated, or Removed
`--connect-string=connectstring`, `-c`	[host=]<hostname>[:<port>]. Overrides entries specified in NDB_CONNECTSTRING or my.cnf	
`--ndb-mgmd-host=host[:port]`	Set the host (and port, if desired) for connecting to management server	(Supported in all MySQL 5.6 based releases)
`--ndb-nodeid=#`	Set node id for this node	(Supported in all MySQL 5.6 based releases)
`--ndb-optimized-node-selection`	Select nodes for transactions in a more optimal way	(Supported in all MySQL 5.6 based releases)
`--no-defaults`	Do not read default options from any option file other than login file	(Supported in all MySQL 5.6 based releases)
`--print-defaults`	Print the program argument list and exit	(Supported in all MySQL 5.6 based releases)
`--version`, `-V`	Output version information and exit	(Supported in all MySQL 5.6 based releases)

For options specific to individual NDB Cluster programs, see Section 18.4, “NDB Cluster Programs”.

See MySQL Server Options for NDB Cluster, for `mysqld` options relating to NDB Cluster.

- `--character-sets-dir=name`

Command-Line Format	`--character-sets-dir=dir_name`
Type	Directory name
Default Value	

Tells the program where to find character set information.

- `--connect-retries=#`

Command-Line Format	`--connect-retries=#`
Introduced	5.6.28-ndb-7.4.9
Type	Numeric
Default Value	`12`
Minimum Value	`0`
Maximum Value	`4294967295`

This option specifies the number of times following the first attempt to retry a connection before giving up (the client always tries the connection at least once). The length of time to wait per attempt is set using `--connect-retry-delay`.

Note

When used with `ndb_mgm`, this option has 3 as its default. See Section 18.4.5, “`ndb_mgm` — The NDB Cluster Management Client”, for more information.

This option was added in NDB 7.4.9.

- `--connect-retry-delay=#`

Command-Line Format	`--connect-retry-delay=#`
Introduced	5.6.28-ndb-7.4.9
Type	Numeric
Default Value	`5`
Minimum Value	`0`
Maximum Value	`4294967295`

This option specifies the length of time to wait per attempt a connection before giving up. The number of times to try connecting is set by `--connect-retries`.

This option was added in NDB 7.4.9.

- `--core-file`

Command-Line Format	`--core-file`
Type	Boolean
Default Value	`FALSE`

Write a core file if the program dies. The name and location of the core file are system-dependent. (For NDB Cluster programs nodes running on Linux, the default location is the program's working directory—for a data node, this is the node's `DataDir`.) For some systems, there may be restrictions or limitations. For example, it might be necessary to execute `ulimit -c unlimited` before starting the server. Consult your system documentation for detailed information.

If NDB Cluster was built using the `--debug` option for `configure`, then `--core-file` is enabled by default. For regular builds, `--core-file` is disabled by default.

- `--debug[=options]`

Command-Line Format	`--debug=options`
Type	String
Default Value	`d:t:O,/tmp/ndb_restore.trace`

This option can be used only for versions compiled with debugging enabled. It is used to enable output from debug calls in the same manner as for the `mysqld` process.

- `--defaults-extra-file=filename`

Command-Line Format	`--defaults-extra-file=filename`
Type	String
Default Value	`[none]`

Read this file after global option files are read.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-file=filename`

Command-Line Format	`--defaults-file=filename`
Type	String

Default Value	[none]

Read default options from this file.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--defaults-group-suffix`

Command-Line Format	--defaults-group-suffix
Type	String
Default Value	[none]

Also read groups with names ending in this suffix.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--help`, `--usage`, `-?`

Command-Line Format	--help --usage

Prints a short list with descriptions of the available command options.

- `--login-path=`*`path`*

Command-Line Format	--login-path=path
Type	String
Default Value	[none]

Read this path from the login file.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--ndb-connectstring=`*`connection_string`*, `--connect-string=`*`connection_string`*, `-c` *`connection_string`*

Command-Line Format	--ndb-connectstring=connectstring --connect-string=connectstring
Type	String
Default Value	localhost:1186

This option takes an NDB Cluster connection string that specifies the management server for the application to connect to, as shown here:

```
shell> ndbd --ndb-connectstring="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

For more information, see Section 18.3.3.3, “NDB Cluster Connection Strings”.

- `--ndb-mgmd-host=`*`host`*`[:`*`port`*`]`

Command-Line Format	--ndb-mgmd-host=host[:port]
Type	String

Default Value	`localhost:1186`

Can be used to set the host and port number of a single management server for the program to connect to. If the program requires node IDs or references to multiple management servers (or both) in its connection information, use the `--ndb-connectstring` option instead.

- `--ndb-nodeid=#`

Command-Line Format	`--ndb-nodeid=#`
Type	Numeric
Default Value	`0`

Sets this node's NDB Cluster node ID. *The range of permitted values depends on the node's type (data, management, or API) and the NDB Cluster software version*. See Section 18.1.7.2, “Limits and Differences of NDB Cluster from Standard MySQL Limits”, for more information.

- `--no-defaults`

Command-Line Format	`--no-defaults`
Type	Boolean
Default Value	`TRUE`

Do not read default options from any option file other than login file.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--ndb-optimized-node-selection`

Command-Line Format	`--ndb-optimized-node-selection`
Type	Boolean
Default Value	`TRUE`

Optimize selection of nodes for transactions. Enabled by default.

- `--print-defaults`

Command-Line Format	`--print-defaults`
Type	Boolean
Default Value	`TRUE`

Print the program argument list and exit.

For additional information about this and other option-file options, see Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”.

- `--version`, `-V`

Command-Line Format	`--version`

Prints the NDB Cluster version number of the executable. The version number is relevant because not all versions can be used together, and the NDB Cluster startup process verifies that the versions of the binaries being used can co-exist in the same cluster. This is also important when performing an online (rolling) software upgrade or downgrade of NDB Cluster.

See Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”), for more information.

18.5 Management of NDB Cluster

Managing an NDB Cluster involves a number of tasks, the first of which is to configure and start NDB Cluster. This is covered in Section 18.3, “Configuration of NDB Cluster”, and Section 18.4, “NDB Cluster Programs”.

The next few sections cover the management of a running NDB Cluster.

For information about security issues relating to management and deployment of an NDB Cluster, see Section 18.5.16, “NDB Cluster Security Issues”.

There are essentially two methods of actively managing a running NDB Cluster. The first of these is through the use of commands entered into the management client whereby cluster status can be checked, log levels changed, backups started and stopped, and nodes stopped and started. The second method involves studying the contents of the cluster log `ndb_node_id_cluster.log`; this is usually found in the management server's `DataDir` directory, but this location can be overridden using the `LogDestination` option. (Recall that *`node_id`* represents the unique identifier of the node whose activity is being logged.) The cluster log contains event reports generated by `ndbd`. It is also possible to send cluster log entries to a Unix system log.

Some aspects of the cluster's operation can be also be monitored from an SQL node using the `SHOW ENGINE NDB STATUS` statement.

More detailed information about NDB Cluster operations is available in real time through an SQL interface using the `ndbinfo` database. For more information, see Section 18.5.14, “ndbinfo: The NDB Cluster Information Database”.

NDB statistics counters provide improved monitoring using the `mysql` client. These counters, implemented in the NDB kernel, relate to operations performed by or affecting `Ndb` objects, such as starting, closing, and aborting transactions; primary key and unique key operations; table, range, and pruned scans; blocked threads waiting for various operations to complete; and data and events sent and received by NDB Cluster. The counters are incremented by the NDB kernel whenever NDB API calls are made or data is sent to or received by the data nodes.

`mysqld` exposes the NDB API statistics counters as system status variables, which can be identified from the prefix common to all of their names (`Ndb_api_`). The values of these variables can be read in the `mysql` client from the output of a `SHOW STATUS` statement, or by querying either the `SESSION_STATUS` table or the `GLOBAL_STATUS` table (in the `INFORMATION_SCHEMA` database). By comparing the values of the status variables before and after the execution of an SQL statement that acts on `NDB` tables, you can observe the actions taken on the NDB API level that correspond to this statement, which can be beneficial for monitoring and performance tuning of NDB Cluster.

MySQL Cluster Manager provides an advanced command-line interface that simplifies many otherwise complex NDB Cluster management tasks, such as starting, stopping, or restarting an NDB Cluster with a large number of nodes. The MySQL Cluster Manager client also supports commands for getting and setting the values of most node configuration parameters as well as `mysqld` server options and variables relating to NDB Cluster. See MySQL™ Cluster Manager 1.4.8 User Manual, for more information.

18.5.1 Commands in the NDB Cluster Management Client

In addition to the central configuration file, a cluster may also be controlled through a command-line interface available through the management client `ndb_mgm`. This is the primary administrative interface to a running cluster.

Commands for the event logs are given in Section 18.5.3, “Event Reports Generated in NDB Cluster”; commands for creating backups and restoring from them are provided in Section 18.5.8, “Online Backup of NDB Cluster”.

Using ndb_mgm with MySQL Cluster Manager. MySQL Cluster Manager handles starting and stopping processes and tracks their states internally, so it is not necessary to use `ndb_mgm` for

these tasks for an NDB Cluster that is under MySQL Cluster Manager control. it is recommended *not* to use the `ndb_mgm` command-line client that comes with the NDB Cluster distribution to perform operations that involve starting or stopping nodes. These include but are not limited to the `START`, `STOP`, `RESTART`, and `SHUTDOWN` commands. For more information, see MySQL Cluster Manager Process Commands.

The management client has the following basic commands. In the listing that follows, *`node_id`* denotes either a data node ID or the keyword `ALL`, which indicates that the command should be applied to all of the cluster's data nodes.

- `HELP`

  Displays information on all available commands.

- `CONNECT` *`connection-string`*

  Connects to the management server indicated by the connection string. If the client is already connected to this server, the client reconnects.

- `SHOW`

  Displays information on the cluster's status. Possible node status values include `UNKNOWN`, `NO_CONTACT`, `NOT_STARTED`, `STARTING`, `STARTED`, `SHUTTING_DOWN`, and `RESTARTING`. The output from this command also indicates when the cluster is in single user mode (status `SINGLE USER MODE`).

- *`node_id`* `START`

  Brings online the data node identified by *`node_id`* (or all data nodes).

  `ALL START` works on all data nodes only, and does not affect management nodes.

  Important

  To use this command to bring a data node online, the data node must have been started using `--nostart` or `-n`.

- *`node_id`* `STOP [-a] [-f]`

  Stops the data or management node identified by *`node_id`*.

  Note

  `ALL STOP` works to stop all data nodes only, and does not affect management nodes.

  A node affected by this command disconnects from the cluster, and its associated `ndbd` or `ndb_mgmd` process terminates.

  The `-a` option causes the node to be stopped immediately, without waiting for the completion of any pending transactions.

  Normally, `STOP` fails if the result would cause an incomplete cluster. The `-f` option forces the node to shut down without checking for this. If this option is used and the result is an incomplete cluster, the cluster immediately shuts down.

  Warning

  Use of the `-a` option also disables the safety check otherwise performed when `STOP` is invoked to insure that stopping the node does not cause an incomplete cluster. In other words, you should exercise extreme care when using the `-a` option with the `STOP` command, due to the fact that this option

makes it possible for the cluster to undergo a forced shutdown because it no longer has a complete copy of all data stored in NDB.

- *node_id* RESTART [-n] [-i] [-a] [-f]

  Restarts the data node identified by *node_id* (or all data nodes).

  Using the -i option with RESTART causes the data node to perform an initial restart; that is, the node's file system is deleted and recreated. The effect is the same as that obtained from stopping the data node process and then starting it again using ndbd --initial from the system shell.

  Note

  Backup files and Disk Data files are not removed when this option is used.

  Using the -n option causes the data node process to be restarted, but the data node is not actually brought online until the appropriate START command is issued. The effect of this option is the same as that obtained from stopping the data node and then starting it again using ndbd --nostart or ndbd -n from the system shell.

  Using the -a causes all current transactions relying on this node to be aborted. No GCP check is done when the node rejoins the cluster.

  Normally, RESTART fails if taking the node offline would result in an incomplete cluster. The -f option forces the node to restart without checking for this. If this option is used and the result is an incomplete cluster, the entire cluster is restarted.

- *node_id* STATUS

  Displays status information for the data node identified by *node_id* (or for all data nodes).

  The output from this command also indicates when the cluster is in single user mode.

- *node_id* REPORT *report-type*

  Displays a report of type *report-type* for the data node identified by *node_id*, or for all data nodes using ALL.

  Currently, there are three accepted values for *report-type*:

  - BackupStatus provides a status report on a cluster backup in progress
  - MemoryUsage displays how much data memory and index memory is being used by each data node as shown in this example:

    ```
    ndb_mgm> ALL REPORT MEMORY

    Node 1: Data usage is 5%(177 32K pages of total 3200)
    Node 1: Index usage is 0%(108 8K pages of total 12832)
    Node 2: Data usage is 5%(177 32K pages of total 3200)
    Node 2: Index usage is 0%(108 8K pages of total 12832)
    ```

    This information is also available from the ndbinfo.memoryusage table.

  - EventLog reports events from the event log buffers of one or more data nodes.

  report-type is case-insensitive and “fuzzy”; for MemoryUsage, you can use MEMORY (as shown in the prior example), memory, or even simply MEM (or mem). You can abbreviate BackupStatus in a similar fashion.

- `ENTER SINGLE USER MODE node_id`

  Enters single user mode, whereby only the MySQL server identified by the node ID *`node_id`* is permitted to access the database.

- `EXIT SINGLE USER MODE`

  Exits single user mode, enabling all SQL nodes (that is, all running `mysqld` processes) to access the database.

  Note

  It is possible to use `EXIT SINGLE USER MODE` even when not in single user mode, although the command has no effect in this case.

- `QUIT`, `EXIT`

  Terminates the management client.

  This command does not affect any nodes connected to the cluster.

- `SHUTDOWN`

  Shuts down all cluster data nodes and management nodes. To exit the management client after this has been done, use `EXIT` or `QUIT`.

  This command does *not* shut down any SQL nodes or API nodes that are connected to the cluster.

- `CREATE NODEGROUP nodeid[, nodeid, ...]`

  Creates a new NDB Cluster node group and causes data nodes to join it.

  This command is used after adding new data nodes online to an NDB Cluster, and causes them to join a new node group and thus to begin participating fully in the cluster. The command takes as its sole parameter a comma-separated list of node IDs—these are the IDs of the nodes just added and started that are to join the new node group. The number of nodes must be the same as the number of nodes in each node group that is already part of the cluster (each NDB Cluster node group must have the same number of nodes). In other words, if the NDB Cluster has 2 node groups of 2 data nodes each, then the new node group must also have 2 data nodes.

  The node group ID of the new node group created by this command is determined automatically, and always the next highest unused node group ID in the cluster; it is not possible to set it manually.

  For more information, see Section 18.5.7, “Adding NDB Cluster Data Nodes Online”.

- `DROP NODEGROUP nodegroup_id`

  Drops the NDB Cluster node group with the given *`nodegroup_id`*.

  This command can be used to drop a node group from an NDB Cluster. `DROP NODEGROUP` takes as its sole argument the node group ID of the node group to be dropped.

  `DROP NODEGROUP` acts only to remove the data nodes in the effected node group from that node group. It does not stop data nodes, assign them to a different node group, or remove them from the cluster's configuration. A data node that does not belong to a node group is indicated in the output of

the management client SHOW command with no nodegroup in place of the node group ID, like this (indicated using bold text):

```
id=3    @10.100.2.67  (5.6.51-ndb-7.4.31, no nodegroup)
```

Prior to NDB 7.0.4, the SHOW output was not updated correctly following DROP NODEGROUP. (Bug #43413)

DROP NODEGROUP works only when all data nodes in the node group to be dropped are completely empty of any table data and table definitions. Since there is currently no way using ndb_mgm or the mysql client to remove all data from a specific data node or node group, this means that the command succeeds only in the two following cases:

1. After issuing CREATE NODEGROUP in the ndb_mgm client, but before issuing any ALTER ONLINE TABLE ... REORGANIZE PARTITION statements in the mysql client.
2. After dropping all NDBCLUSTER tables using DROP TABLE.

   TRUNCATE TABLE does not work for this purpose because this removes only the table data; the data nodes continue to store an NDBCLUSTER table's definition until a DROP TABLE statement is issued that causes the table metadata to be dropped.

For more information about DROP NODEGROUP, see Section 18.5.7, “Adding NDB Cluster Data Nodes Online”.

Additional commands. A number of other commands available in the ndb_mgm client are described elsewhere, as shown in the following list:

- START BACKUP is used to perform an online backup in the ndb_mgm client; the ABORT BACKUP command is used to cancel a backup already in progress. For more information, see Section 18.5.8, “Online Backup of NDB Cluster”.
- The CLUSTERLOG command is used to perform various logging functions. See Section 18.5.3, “Event Reports Generated in NDB Cluster”, for more information and examples.
- For testing and diagnostics work, the client also supports a DUMP command which can be used to execute internal commands on the cluster. It should never be used in a production setting unless directed to do so by MySQL Support. For more information, see MySQL NDB Cluster Internals Manual.

18.5.2 NDB Cluster Log Messages

This section contains information about the messages written to the cluster log in response to different cluster log events. It provides additional, more specific information on NDB transporter errors.

18.5.2.1 NDB Cluster: Messages in the Cluster Log

The following table lists the most common NDB cluster log messages. For information about the cluster log, log events, and event types, see Section 18.5.3, “Event Reports Generated in NDB Cluster”. These log messages also correspond to log event types in the MGM API; see The Ndb_logevent_type Type, for related information of interest to Cluster API developers.

Table 18.42 Common NDB cluster log messages

Log Message	Description	Event Name	Event Type	Priority	Severity
Node *mgm_node_id*: Node *data_node_id* Connected	The data node having node ID *node_id* has connected to the management server (node *mgm_node_id*).	Connected	Connection	8	INFO

Log Message	Description	Event Name	Event Type	Priority	Severity
Node *mgm_node_id*: Node *data_node_id* Disconnected	The data node having node ID *data_node_id* has disconnected from the management server (node *mgm_node_id*).	Disconnected	Connection	8	ALERT
Node *data_node_id*: Communication to Node *api_node_id* closed	The API node or SQL node having node ID *api_node_id* is no longer communicating with data node *data_node_id*.	CommunicationClosed	Connection	8	INFO
Node *data_node_id*: Communication to Node *api_node_id* opened	The API node or SQL node having node ID *api_node_id* is now communicating with data node *data_node_id*.	CommunicationOpened	Connection	8	INFO
Node *mgm_node_id*: Node *api_node_id*: API *version*	The API node having node ID *api_node_id* has connected to management node *mgm_node_id* using NDB API version *version* (generally the same as the MySQL version number).	ConnectedApiVersion	Connection	8	INFO
Node *node_id*: Global checkpoint *gci* started	A global checkpoint with the ID *gci* has been started; node *node_id* is the master responsible for this global checkpoint.	GlobalCheckpointStarted	Checkpoint	9	INFO
Node *node_id*: Global checkpoint *gci* completed	The global checkpoint having the ID *gci* has been completed; node *node_id* was the master responsible for this global checkpoint.	GlobalCheckpointCompleted	Checkpoint	10	INFO
Node *node_id*: Local checkpoint	The local checkpoint having sequence ID *lcp*	LocalCheckpointStarted	Checkpoint	7	INFO

Log Message	Description	Event Name	Event Type	Priority	Severity
lcp started. Keep GCI = *current_gci* oldest restorable GCI = *old_gci*	has been started on node *node_id*. The most recent GCI that can be used has the index *current_gci*, and the oldest GCI from which the cluster can be restored has the index *old_gci*.				
Node *node_id*: Local checkpoint *lcp* completed	The local checkpoint having sequence ID *lcp* on node *node_id* has been completed.	LocalCheckpointCompleted	Checkpoint	8	INFO
Node *node_id*: Local Checkpoint stopped in CALCULATED_KEEP_GCI	The node was unable to determine the most recent usable GCI.	LCPStoppedInCalcKeepGci	Checkpoint	0	ALERT
Node *node_id*: Table ID = *table_id*, fragment *ID* = *fragment_id* has completed LCP on Node *node_id* maxGciStarted: *started_gci* maxGciCompleted *completed_gci*	A table fragment has been checkpointed to disk on node *node_id*. The GCI in progress has the index *started_gci*, and the most recent GCI to have been completed has the index *completed_gci*.	LCPFragmentCompleted	Checkpoint	11	INFO
Node *node_id*: ACC Blocked *num_1* and TUP Blocked *num_2* times last second	Undo logging is blocked because the log buffer is close to overflowing.	UndoLogBlocked	Checkpoint	7	INFO
Node *node_id*: Start initiated *version*	Data node *node_id*, running NDB version *version*, is beginning its startup process.	NDBStartStarted	StartUp	1	INFO
Node *node_id*: Started *version*	Data node *node_id*, running NDB version *version*, has started successfully.	NDBStartCompleted	StartUp	1	INFO

Log Message	Description	Event Name	Event Type	Priority	Severity
Node *node_id*: STTORRY received after restart finished	The node has received a signal indicating that a cluster restart has completed.	STTORRYRecieved	StartUp	15	INFO
Node *node_id*: Start phase *phase* completed (*type*)	The node has completed start phase *phase* of a *type* start. For a listing of start phases, see Section 18.5.4, "Summary of NDB Cluster Start Phases". (*type* is one of initial, system, node, initial node, or <Unknown>.)	StartPhaseCompleted	StartUp	4	INFO
Node *node_id*: CM_REGCONF president = *president_id*, own Node = *own_id*, our dynamic id = *dynamic_id*	Node *president_id* has been selected as "president". *own_id* and *dynamic_id* should always be the same as the ID (*node_id*) of the reporting node.	CM_REGCONF	StartUp	3	INFO
Node *node_id*: CM_REGREF from Node *president_id* to our Node *node_id*. Cause = *cause*	The reporting node (ID *node_id*) was unable to accept node *president_id* as president. The *cause* of the problem is given as one of Busy, Election with wait = false, Not president, Election without selecting new candidate, or No such cause.	CM_REGREF	StartUp	8	INFO
Node *node_id*: We are Node *own_id* with dynamic ID *dynamic_id*, our left neighbor is Node *id_1*, our	The node has discovered its neighboring nodes in the cluster (node *id_1* and node *id_2*). *node_id*, *own_id*, and *dynamic_id* should always	FIND_NEIGHBOURS	StartUp	8	INFO

Log Message	Description	Event Name	Event Type	Priority	Severity
`right is Node id_2`	be the same; if they are not, this indicates a serious misconfiguration of the cluster nodes.				
`Node node_id: type shutdown initiated`	The node has received a shutdown signal. The *type* of shutdown is either `Cluster` or `Node`.	`NDBStopStarted`	`StartUp`	1	`INFO`
`Node node_id: Node shutdown completed` [`, action`] [`Initiated by signal signal.`]	The node has been shut down. This report may include an *action*, which if present is one of `restarting`, `no start`, or `initial`. The report may also include a reference to an `NDB` Protocol *signal*; for possible signals, refer to Operations and Signals.	`NDBStopCompleted`	`StartUp`	1	`INFO`
`Node node_id: Forced node shutdown completed` [`, action`]. [`Occurred during startphase start_phase.`] [`Initiated by signal.`] [`Caused by error error_code: 'error_message(error_classification) error_status'.` [`(extra info extra_code)`]]	The node has been forcibly shut down. The *action* (one of `restarting`, `no start`, or `initial`) subsequently being taken, if any, is also reported. If the shutdown occurred while the node was starting, the report includes the *start_phase* during which the node failed. If this was a result of a *signal* sent to the node, this information is also provided (see Operations and Signals, for more information). If the error causing the failure is known, this is also included; for more information	`NDBStopForced`	`StartUp`	1	`ALERT`

Log Message	Description	Event Name	Event Type	Priority	Severity
	about NDB error messages and classifications, see NDB Cluster API Errors.				
Node *node_id*: Node shutdown aborted	The node shutdown process was aborted by the user.	NDBStopAborted	StartUp	1	INFO
Node *node_id*: StartLog: [GCI Keep: *keep_pos* LastCompleted: *last_pos* NewestRestorable: *restore_pos*]	This reports global checkpoints referenced during a node start. The redo log prior to *keep_pos* is dropped. *last_pos* is the last global checkpoint in which data node the participated; *restore_pos* is the global checkpoint which is actually used to restore all data nodes.	StartREDOLog	StartUp	4	INFO
startup_message [*Listed separately; see below.*]	There are a number of possible startup messages that can be logged under different circumstances. These are listed separately; see Section 18.5.2.2, "NDB Cluster Log Startup Messages".	StartReport	StartUp	4	INFO
Node *node_id*: Node restart completed copy of dictionary information	Copying of data dictionary information to the restarted node has been completed.	NR_CopyDict	NodeRestart	8	INFO
Node *node_id*: Node restart completed copy of distribution information	Copying of data distribution information to the restarted node has been completed.	NR_CopyDistr	NodeRestart	8	INFO
Node *node_id*: Node restart starting to copy the	Copy of fragments to starting data node *node_id* has begun	NR_CopyFragsStarted	NodeRestart	8	INFO

Log Message	Description	Event Name	Event Type	Priority	Severity
fragments to Node *node_id*					
Node *node_id*: Table ID = *table_id*, fragment ID = *fragment_id* have been copied to Node *node_id*	Fragment *fragment_id* from table *table_id* has been copied to data node *node_id*	NR_CopyFragDone	NodeRestart	10	INFO
Node *node_id*: Node restart completed copying the fragments to Node *node_id*	Copying of all table fragments to restarting data node *node_id* has been completed	NR_CopyFragsCompleted	NodeRestart	8	INFO
Node *node_id*: Node *node1_id* completed failure of Node *node2_id*	Data node *node1_id* has detected the failure of data node *node2_id*	NodeFailCompleted	NodeRestart	8	ALERT
All nodes completed failure of Node *node_id*	All (remaining) data nodes have detected the failure of data node *node_id*	NodeFailCompleted	NodeRestart	8	ALERT
Node failure of *node_id*block completed	The failure of data node *node_id* has been detected in the *block*NDB kernel block, where block is 1 of DBTC, DBDICT, DBDIH, or DBLQH; for more information, see NDB Kernel Blocks	NodeFailCompleted	NodeRestart	8	ALERT
Node *mgm_node_id*: Node *data_node_id* has failed. The Node state at failure was *state_code*	A data node has failed. Its state at the time of failure is described by an arbitration state code *state_code*: possible state code values can be found in the file include/kernel/signaldata/ArbitSignalData.hpp.	NODE_FAILREP	NodeRestart	8	ALERT
President restarts arbitration thread	This is a report on the current state and progress of arbitration in the	ArbitState	NodeRestart	6	INFO

Log Message	Description	Event Name	Event Type	Priority	Severity
[state=*state_code*] or Prepare arbitrator node *node_id* [ticket=*ticket_id*] or Receive arbitrator node *node_id* [ticket=*ticket_id*] or Started arbitrator node *node_id* [ticket=*ticket_id*] or Lost arbitrator node *node_id* - process failure [state=*state_code*] or Lost arbitrator node *node_id* - process exit [state=*state_code*] or Lost arbitrator node *node_id* - *error_message* [state=*state_code*]	cluster. *node_id* is the node ID of the management node or SQL node selected as the arbitrator. *state_code* is an arbitration state code, as found in include/kernel/signaldata/ArbitSignalData.hpp. When an error has occurred, an *error_message*, also defined in ArbitSignalData.hpp, is provided. *ticket_id* is a unique identifier handed out by the arbitrator when it is selected to all the nodes that participated in its selection; this is used to ensure that each node requesting arbitration was one of the nodes that took part in the selection process.				
Arbitration check lost - less than 1/2 nodes left or Arbitration check won - all node groups and more than 1/2 nodes left or Arbitration check won - node group majority or Arbitration check lost - missing node group or Network partitioning - arbitration required or Arbitration	This message reports on the result of arbitration. In the event of arbitration failure, an *error_message* and an arbitration *state_code* are provided; definitions for both of these are found in include/kernel/signaldata/ArbitSignalData.hpp.	ArbitResult	NodeRestart	2	ALERT

Log Message	Description	Event Name	Event Type	Priority	Severity
won - positive reply from node *node_id* or Arbitration lost - negative reply from node *node_id* or Network partitioning - no arbitrator available or Network partitioning - no arbitrator configured or Arbitration failure - *error_message* [state=*state_code*]					
Node *node_id*: GCP Take over started	This node is attempting to assume responsibility for the next global checkpoint (that is, it is becoming the master node)	GCP_TakeoverStarted	NodeRestart	7	INFO
Node *node_id*: GCP Take over completed	This node has become the master, and has assumed responsibility for the next global checkpoint	GCP_TakeoverCompleted	NodeRestart	7	INFO
Node *node_id*: LCP Take over started	This node is attempting to assume responsibility for the next set of local checkpoints (that is, it is becoming the master node)	LCP_TakeoverStarted	NodeRestart	7	INFO
Node *node_id*: LCP Take over completed	This node has become the master, and has assumed responsibility for the next set of local checkpoints	LCP_TakeoverCompleted	NodeRestart	7	INFO
Node *node_id*: Trans. Count = *transactions*, Commit Count	This report of transaction activity is given	TransReportCounters	Statistic	8	INFO

Log Message	Description	Event Name	Event Type	Priority	Severity
= *commits*, Read Count = *reads*, Simple Read Count = *simple_reads*, Write Count = *writes*, AttrInfo Count = *AttrInfo_objects*, Concurrent Operations = *concurrent_operations*, Abort Count = *aborts*, Scans = *scans*, Range scans = *range_scans*	approximately once every 10 seconds				
Node *node_id*: Operations=*operations*	Number of operations performed by this node, provided approximately once every 10 seconds	OperationReportCounters	Statistic	8	INFO
Node *node_id*: Table with ID = *table_id* created	A table having the table ID shown has been created	TableCreated	Statistic	7	INFO
Node *node_id*: Mean loop Counter in doJob last 8192 times = *count*		JobStatistic	Statistic	9	INFO
Mean send size to Node = *node_id* last 4096 sends = *bytes* bytes	This node is sending an average of *bytes* bytes per send to node *node_id*	SendBytesStatistic	Statistic	9	INFO
Mean receive size to Node = *node_id* last 4096 sends = *bytes* bytes	This node is receiving an average of *bytes* of data each time it receives data from node *node_id*	ReceiveBytesStatistic	Statistic	9	INFO
Node *node_id*: Data usage is *data_memory_percentage*% (*data_pages_used* 32K pages of total *data_pages_total*) / Node *node_id*: Index usage is	This report is generated when a DUMP 1000 command is issued in the cluster management client; for more information, see DUMP 1000,	MemoryUsage	Statistic	5	INFO

Log Message	Description	Event Name	Event Type	Priority	Severity
index_memory_per (*index_pages_used* 8K pages of total *index_pages_total*)	in MySQL NDB Cluster Internals Manual				
Node *node1_id*: Transporter to node *node2_id* reported error *error_code*: *error_message*	A transporter error occurred while communicating with node *node2_id*; for a listing of transporter error codes and messages, see NDB Transporter Errors, in MySQL NDB Cluster Internals Manual	TransporterError	Error	2	ERROR
Node *node1_id*: Transporter to node *node2_id* reported error *error_code*: *error_message*	A warning of a potential transporter problem while communicating with node *node2_id*; for a listing of transporter error codes and messages, see NDB Transporter Errors, for more information	TransporterWarning	Error	8	WARNING
Node *node1_id*: Node *node2_id* missed heartbeat *heartbeat_id*	This node missed a heartbeat from node *node2_id*	MissedHeartbeat	Error	8	WARNING
Node *node1_id*: Node *node2_id* declared dead due to missed heartbeat	This node has missed at least 3 heartbeats from node *node2_id*, and so has declared that node “dead”	DeadDueToHeartbeat	Error	8	ALERT
Node *node1_id*: Node Sent Heartbeat to node = *node2_id*	This node has sent a heartbeat to node *node2_id*	SentHeartbeat	Info	12	INFO
(*NDB 7.5.0 and earlier*:) Node *node_id*: Event buffer status: used=*bytes_used*	This report is seen during heavy event buffer usage, for example, when many updates are	EventBufferStatus	Info	7	INFO

Log Message	Description	Event Name	Event Type	Priority	Severity
`(percent_used%) alloc=bytes_allocated (percent_available%) max=bytes_available apply_epoch=latest_consumable_epoch latest_epoch=latest_epoch`	being applied in a relatively short period of time; the report shows the number of bytes and the percentage of event buffer memory used, the bytes allocated and percentage still available, and the latest and latest restorable epochs				
`Node node_id: Entering single user mode`, `Node node_id: Entered single user mode Node API_node_id has exclusive access`, `Node node_id: Entering single user mode`	These reports are written to the cluster log when entering and exiting single user mode; *API_node_id* is the node ID of the API or SQL having exclusive access to the cluster (for more information, see Section 18.5.6, “NDB Cluster Single User Mode”); the message `Unknown single user report API_node_id` indicates an error has taken place and should never be seen in normal operation	`SingleUser`	`Info`	7	`INFO`
`Node node_id: Backup backup_id started from node mgm_node_id`	A backup has been started using the management node having *mgm_node_id*; this message is also displayed in the cluster management client when the `START BACKUP` command is issued; for more information, see Section 18.5.8.2, “Using The NDB Cluster Management	`BackupStarted`	`Backup`	7	`INFO`

Log Message	Description	Event Name	Event Type	Priority	Severity
	Client to Create a Backup”				
Node *node_id*: Backup *backup_id* started from node *mgm_node_id* completed. StartGCP: *start_gcp* StopGCP: *stop_gcp* #Records: *records* #LogRecords: *log_records* Data: *data_bytes* bytes Log: *log_bytes* bytes	The backup having the ID *backup_id* has been completed; for more information, see Section 18.5.8.2, “Using The NDB Cluster Management Client to Create a Backup”	BackupCompleted	Backup	7	INFO
Node *node_id*: Backup request from *mgm_node_id* failed to start. Error: *error_code*	The backup failed to start; for error codes, see MGM API Errors	BackupFailedToStart	Backup	7	ALERT
Node *node_id*: Backup *backup_id* started from *mgm_node_id* has been aborted. Error: *error_code*	The backup was terminated after starting, possibly due to user intervention	BackupAborted	Backup	7	ALERT

18.5.2.2 NDB Cluster Log Startup Messages

Possible startup messages with descriptions are provided in the following list:

- `Initial start, waiting for %s to connect, nodes [ all: %s connected: %s no-wait: %s ]`
- `Waiting until nodes: %s connects, nodes [ all: %s connected: %s no-wait: %s ]`
- `Waiting %u sec for nodes %s to connect, nodes [ all: %s connected: %s no-wait: %s ]`
- `Waiting for non partitioned start, nodes [ all: %s connected: %s missing: %s no-wait: %s ]`
- `Waiting %u sec for non partitioned start, nodes [ all: %s connected: %s missing: %s no-wait: %s ]`

- `Initial start with nodes %s [ missing: %s no-wait: %s ]`
- `Start with all nodes %s`
- `Start with nodes %s [ missing: %s no-wait: %s ]`
- `Start potentially partitioned with nodes %s [ missing: %s no-wait: %s ]`
- `Unknown startreport: 0x%x [ %s %s %s %s ]`

18.5.2.3 NDB Cluster: NDB Transporter Errors

This section lists error codes, names, and messages that are written to the cluster log in the event of transporter errors.

`0x00`	**`TE_NO_ERROR`** `No error`
`0x01`	**`TE_ERROR_CLOSING_SOCKET`** `Error found during closing of socket`
`0x02`	**`TE_ERROR_IN_SELECT_BEFORE_ACCEPT`** `Error found before accept. The transporter will retry`
`0x03`	**`TE_INVALID_MESSAGE_LENGTH`** `Error found in message (invalid message length)`
`0x04`	**`TE_INVALID_CHECKSUM`** `Error found in message (checksum)`
`0x05`	**`TE_COULD_NOT_CREATE_SOCKET`** `Error found while creating socket(can't create socket)`
`0x06`	**`TE_COULD_NOT_BIND_SOCKET`** `Error found while binding server socket`
`0x07`	**`TE_LISTEN_FAILED`** `Error found while listening to server socket`
`0x08`	**`TE_ACCEPT_RETURN_ERROR`** `Error found during accept(accept return error)`
`0x0b`	**`TE_SHM_DISCONNECT`** `The remote node has disconnected`
`0x0c`	**`TE_SHM_IPC_STAT`** `Unable to check shm segment`
`0x0d`	**`TE_SHM_UNABLE_TO_CREATE_SEGMENT`** `Unable to create shm segment`

0x0e	**TE_SHM_UNABLE_TO_ATTACH_SEGMENT** Unable to attach shm segment
0x0f	**TE_SHM_UNABLE_TO_REMOVE_SEGMENT** Unable to remove shm segment
0x10	**TE_TOO_SMALL_SIGID** Sig ID too small
0x11	**TE_TOO_LARGE_SIGID** Sig ID too large
0x12	**TE_WAIT_STACK_FULL** Wait stack was full
0x13	**TE_RECEIVE_BUFFER_FULL** Receive buffer was full
0x14	**TE_SIGNAL_LOST_SEND_BUFFER_FULL** Send buffer was full,and trying to force send fails
0x15	**TE_SIGNAL_LOST** Send failed for unknown reason(signal lost)
0x16	**TE_SEND_BUFFER_FULL** The send buffer was full, but sleeping for a while solved
0x21	**TE_SHM_IPC_PERMANENT** Shm ipc Permanent error

Note

Transporter error codes `0x17` through `0x20` and `0x22` are reserved for SCI connections, which are not supported in this version of NDB Cluster, and so are not included here.

18.5.3 Event Reports Generated in NDB Cluster

In this section, we discuss the types of event logs provided by NDB Cluster, and the types of events that are logged.

NDB Cluster provides two types of event log:

- The *cluster log*, which includes events generated by all cluster nodes. The cluster log is the log recommended for most uses because it provides logging information for an entire cluster in a single location.

  By default, the cluster log is saved to a file named `ndb_node_id_cluster.log`, (where *`node_id`* is the node ID of the management server) in the management server's `DataDir`.

  Cluster logging information can also be sent to `stdout` or a `syslog` facility in addition to or instead of being saved to a file, as determined by the values set for the `DataDir` and `LogDestination`

configuration parameters. See Section 18.3.3.5, “Defining an NDB Cluster Management Server”, for more information about these parameters.

- *Node logs* are local to each node.

  Output generated by node event logging is written to the file `ndb_node_id_out.log` (where `node_id` is the node's node ID) in the node's `DataDir`. Node event logs are generated for both management nodes and data nodes.

  Node logs are intended to be used only during application development, or for debugging application code.

Both types of event logs can be set to log different subsets of events.

Each reportable event can be distinguished according to three different criteria:

- *Category*: This can be any one of the following values: `STARTUP`, `SHUTDOWN`, `STATISTICS`, `CHECKPOINT`, `NODERESTART`, `CONNECTION`, `ERROR`, or `INFO`.
- *Priority*: This is represented by one of the numbers from 0 to 15 inclusive, where 0 indicates “most important” and 15 “least important.”
- *Severity Level*: This can be any one of the following values: `ALERT`, `CRITICAL`, `ERROR`, `WARNING`, `INFO`, or `DEBUG`.

Both the cluster log and the node log can be filtered on these properties.

The format used in the cluster log is as shown here:

```
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 1: Data usage is 2%(60 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 1: Index usage is 1%(24 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 1: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 2: Data usage is 2%(76 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 2: Index usage is 1%(24 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 2: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 3: Data usage is 2%(58 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 3: Index usage is 1%(25 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 3: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 4: Data usage is 2%(74 32K pages of total 2560)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 4: Index usage is 1%(25 8K pages of total 2336)
2007-01-26 19:35:55 [MgmSrvr] INFO     -- Node 4: Resource 0 min: 0 max: 639 curr: 0
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 4: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 1: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 1: Node 9: API 5.6.51-ndb-7.4.31
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 2: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 2: Node 9: API 5.6.51-ndb-7.4.31
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 3: Node 9 Connected
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 3: Node 9: API 5.6.51-ndb-7.4.31
2007-01-26 19:39:42 [MgmSrvr] INFO     -- Node 4: Node 9: API 5.6.51-ndb-7.4.31
2007-01-26 19:59:22 [MgmSrvr] ALERT    -- Node 2: Node 7 Disconnected
2007-01-26 19:59:22 [MgmSrvr] ALERT    -- Node 2: Node 7 Disconnected
```

Each line in the cluster log contains the following information:

- A timestamp in `YYYY-MM-DD HH:MM:SS` format.
- The type of node which is performing the logging. In the cluster log, this is always `[MgmSrvr]`.
- The severity of the event.
- The ID of the node reporting the event.
- A description of the event. The most common types of events to appear in the log are connections and disconnections between different nodes in the cluster, and when checkpoints occur. In some cases, the description may contain status information.

18.5.3.1 NDB Cluster Logging Management Commands

`ndb_mgm` supports a number of management commands related to the cluster log. In the listing that follows, *`node_id`* denotes either a data node ID or the keyword `ALL`, which indicates that the command should be applied to all of the cluster's data nodes.

- `CLUSTERLOG ON`

  Turns the cluster log on.

- `CLUSTERLOG OFF`

  Turns the cluster log off.

- `CLUSTERLOG INFO`

  Provides information about cluster log settings.

- *`node_id`* `CLUSTERLOG` *`category=threshold`*

  Logs *`category`* events with priority less than or equal to *`threshold`* in the cluster log.

- `CLUSTERLOG FILTER` *`severity_level`*

  Toggles cluster logging of events of the specified *`severity_level`*.

The following table describes the default setting (for all data nodes) of the cluster log category threshold. If an event has a priority with a value lower than or equal to the priority threshold, it is reported in the cluster log.

Note

Events are reported per data node, and that the threshold can be set to different values on different nodes.

Table 18.43 Cluster log categories, with default threshold setting

Category	Default threshold (All data nodes)
`STARTUP`	**7**
`SHUTDOWN`	**7**
`STATISTICS`	**7**
`CHECKPOINT`	**7**
`NODERESTART`	**7**
`CONNECTION`	**7**
`ERROR`	**15**
`INFO`	**7**

The `STATISTICS` category can provide a great deal of useful data. See Section 18.5.3.3, “Using CLUSTERLOG STATISTICS in the NDB Cluster Management Client”, for more information.

Thresholds are used to filter events within each category. For example, a `STARTUP` event with a priority of 3 is not logged unless the threshold for `STARTUP` is set to 3 or higher. Only events with priority 3 or lower are sent if the threshold is 3.

The following table shows the event severity levels.

Note

These correspond to Unix `syslog` levels, except for `LOG_EMERG` and `LOG_NOTICE`, which are not used or mapped.

Table 18.44 Event severity levels

Severity Level Value	Severity	Description
1	ALERT	A condition that should be corrected immediately, such as a corrupted system database
2	CRITICAL	Critical conditions, such as device errors or insufficient resources
3	ERROR	Conditions that should be corrected, such as configuration errors
4	WARNING	Conditions that are not errors, but that might require special handling
5	INFO	Informational messages
6	DEBUG	Debugging messages used for NDBCLUSTER development

Event severity levels can be turned on or off (using CLUSTERLOG FILTER—see above). If a severity level is turned on, then all events with a priority less than or equal to the category thresholds are logged. If the severity level is turned off then no events belonging to that severity level are logged.

Important

Cluster log levels are set on a per ndb_mgmd, per subscriber basis. This means that, in an NDB Cluster with multiple management servers, using a CLUSTERLOG command in an instance of ndb_mgm connected to one management server affects only logs generated by that management server but not by any of the others. This also means that, should one of the management servers be restarted, only logs generated by that management server are affected by the resetting of log levels caused by the restart.

18.5.3.2 NDB Cluster Log Events

An event report reported in the event logs has the following format:

```
datetime [string] severity -- message
```

For example:

```
09:19:30 2005-07-24 [NDB] INFO -- Node 4 Start phase 4 completed
```

This section discusses all reportable events, ordered by category and severity level within each category.

In the event descriptions, GCP and LCP mean “Global Checkpoint” and “Local Checkpoint”, respectively.

CONNECTION Events

These events are associated with connections between Cluster nodes.

Table 18.45 Events associated with connections between cluster nodes

Event	Priority	Severity Level	Description
Connected	8	INFO	Data nodes connected
Disconnected	8	ALERT	Data nodes disconnected
CommunicationClosed	8	INFO	SQL node or data node connection closed
CommunicationOpened	8	INFO	SQL node or data node connection open

Event	Priority	Severity Level	Description
ConnectedApiVersion	8	INFO	Connection using API version

CHECKPOINT Events

The logging messages shown here are associated with checkpoints.

Table 18.46 Events associated with checkpoints

Event	Priority	Severity Level	Description
GlobalCheckpointStarted	9	INFO	Start of GCP: REDO log is written to disk
GlobalCheckpointCompleted	10	INFO	GCP finished
LocalCheckpointStarted	7	INFO	Start of LCP: data written to disk
LocalCheckpointCompleted	7	INFO	LCP completed normally
LCPStoppedInCalcKeepGci	0	ALERT	LCP stopped
LCPFragmentCompleted	11	INFO	LCP on a fragment has been completed
UndoLogBlocked	7	INFO	UNDO logging blocked; buffer near overflow
RedoStatus	7	INFO	Redo status

STARTUP Events

The following events are generated in response to the startup of a node or of the cluster and of its success or failure. They also provide information relating to the progress of the startup process, including information concerning logging activities.

Table 18.47 Events relating to the startup of a node or cluster

Event	Priority	Severity Level	Description
NDBStartStarted	1	INFO	Data node start phases initiated (all nodes starting)
NDBStartCompleted	1	INFO	Start phases completed, all data nodes
STTORRYRecieved	15	INFO	Blocks received after completion of restart
StartPhaseCompleted	4	INFO	Data node start phase *X* completed
CM_REGCONF	3	INFO	Node has been successfully included into the cluster; shows the node, managing node, and dynamic ID
CM_REGREF	8	INFO	Node has been refused for inclusion in the cluster; cannot be included in cluster due to misconfiguration, inability to establish communication, or other problem
FIND_NEIGHBOURS	8	INFO	Shows neighboring data nodes
NDBStopStarted	1	INFO	Data node shutdown initiated
NDBStopCompleted	1	INFO	Data node shutdown complete
NDBStopForced	1	ALERT	Forced shutdown of data node
NDBStopAborted	1	INFO	Unable to shut down data node normally
StartREDOLog	4	INFO	New redo log started; GCI keep *X*, newest restorable GCI *Y*

Event	Priority	Severity Level	Description
StartLog	10	INFO	New log started; log part *X*, start MB *Y*, stop MB *Z*
UNDORecordsExecuted	15	INFO	Undo records executed
StartReport	4	INFO	Report started
LogFileInitStatus	7	INFO	Log file initialization status
LogFileInitCompStatus	7	INFO	Log file completion status
StartReadLCP	10	INFO	Start read for local checkpoint
ReadLCPComplete	10	INFO	Read for local checkpoint completed
RunRedo	8	INFO	Running the redo log
RebuildIndex	10	INFO	Rebuilding indexes

NODERESTART Events

The following events are generated when restarting a node and relate to the success or failure of the node restart process.

Table 18.48 Events relating to restarting a node

Event	Priority	Severity Level	Description
NR_CopyDict	7	INFO	Completed copying of dictionary information
NR_CopyDistr	7	INFO	Completed copying distribution information
NR_CopyFragsStarted	7	INFO	Starting to copy fragments
NR_CopyFragDone	10	INFO	Completed copying a fragment
NR_CopyFragsCompleted	7	INFO	Completed copying all fragments
NodeFailCompleted	8	ALERT	Node failure phase completed
NODE_FAILREP	8	ALERT	Reports that a node has failed
ArbitState	6	INFO	Report whether an arbitrator is found or not; there are seven different possible outcomes when seeking an arbitrator, listed here: • Management server restarts arbitration thread [state=*X*] • Prepare arbitrator node *X* [ticket=*Y*] • Receive arbitrator node *X* [ticket=*Y*] • Started arbitrator node *X* [ticket=*Y*] • Lost arbitrator node *X* - process failure [state=*Y*] • Lost arbitrator node *X* - process exit [state=*Y*] • Lost arbitrator node *X* <error msg> [state=*Y*]
ArbitResult	2	ALERT	Report arbitrator results; there are eight different possible results for arbitration attempts, listed here:

Event	Priority	Severity Level	Description
			• Arbitration check failed: less than 1/2 nodes left • Arbitration check succeeded: node group majority • Arbitration check failed: missing node group • Network partitioning: arbitration required • Arbitration succeeded: affirmative response from node *X* • Arbitration failed: negative response from node *X* • Network partitioning: no arbitrator available • Network partitioning: no arbitrator configured
GCP_TakeoverStarted	7	INFO	GCP takeover started
GCP_TakeoverCompleted	7	INFO	GCP takeover complete
LCP_TakeoverStarted	7	INFO	LCP takeover started
LCP_TakeoverCompleted	7	INFO	LCP takeover complete (state = *X*)
ConnectCheckStarted	6	INFO	Connection check started
ConnectCheckCompleted	6	INFO	Connection check completed
NodeFailRejected	6	ALERT	Node failure phase failed

STATISTICS Events

The following events are of a statistical nature. They provide information such as numbers of transactions and other operations, amount of data sent or received by individual nodes, and memory usage.

Table 18.49 Events of a statistical nature

Event	Priority	Severity Level	Description
TransReportCounters	8	INFO	Report transaction statistics, including numbers of transactions, commits, reads, simple reads, writes, concurrent operations, attribute information, and aborts
OperationReportCounters	8	INFO	Number of operations
TableCreated	7	INFO	Report number of tables created
JobStatistic	9	INFO	Mean internal job scheduling statistics
ThreadConfigLoop	9	INFO	Number of thread configuration loops
SendBytesStatistic	9	INFO	Mean number of bytes sent to node *X*
ReceiveBytesStatistic	9	INFO	Mean number of bytes received from node *X*
MemoryUsage	5	INFO	Data and index memory usage (80%, 90%, and 100%)

Event	Priority	Severity Level	Description
MTSignalStatistics	9	INFO	Multithreaded signals

SCHEMA Events

These events relate to NDB Cluster schema operations.

Table 18.50 Events relating to NDB Cluster schema operations

Event	Priority	Severity Level	Description
CreateSchemaObject	8	INFO	Schema objected created
AlterSchemaObject	8	INFO	Schema object updated
DropSchemaObject	8	INFO	Schema object dropped

ERROR Events

These events relate to Cluster errors and warnings. The presence of one or more of these generally indicates that a major malfunction or failure has occurred.

Table 18.51 Events relating to cluster errors and warnings

Event	Priority	Severity Level	Description
TransporterError	2	ERROR	Transporter error
TransporterWarning	8	WARNING	Transporter warning
MissedHeartbeat	8	WARNING	Node *X* missed heartbeat number *Y*
DeadDueToHeartbeat	8	ALERT	Node *X* declared “dead” due to missed heartbeat
WarningEvent	2	WARNING	General warning event
SubscriptionStatus	4	WARNING	Change in subscription status

INFO Events

These events provide general information about the state of the cluster and activities associated with Cluster maintenance, such as logging and heartbeat transmission.

Table 18.52 Information events

Event	Priority	Severity Level	Description
SentHeartbeat	12	INFO	Sent heartbeat
CreateLogBytes	11	INFO	Create log: Log part, log file, size in MB
InfoEvent	2	INFO	General informational event
EventBufferStatus	7	INFO	Event buffer status

Note

SentHeartbeat events are available only if NDB Cluster was compiled with VM_TRACE enabled.

SINGLEUSER Events

These events are associated with entering and exiting single user mode.

Table 18.53 Events relating to single user mode

Event	Priority	Severity Level	Description
SingleUser	7	INFO	Entering or exiting single user mode

BACKUP Events

These events provide information about backups being created or restored.

Table 18.54 Backup events

Event	Priority	Severity Level	Description
BackupStarted	7	INFO	Backup started
BackupStatus	7	INFO	Backup status
BackupCompleted	7	INFO	Backup completed
BackupFailedToStart	7	ALERT	Backup failed to start
BackupAborted	7	ALERT	Backup aborted by user
RestoreStarted	7	INFO	Started restoring from backup
RestoreMetaData	7	INFO	Restoring metadata
RestoreData	7	INFO	Restoring data
RestoreLog	7	INFO	Restoring log files
RestoreCompleted	7	INFO	Completed restoring from backup
SavedEvent	7	INFO	Event saved

18.5.3.3 Using CLUSTERLOG STATISTICS in the NDB Cluster Management Client

The NDB management client's CLUSTERLOG STATISTICS command can provide a number of useful statistics in its output. Counters providing information about the state of the cluster are updated at 5-second reporting intervals by the transaction coordinator (TC) and the local query handler (LQH), and written to the cluster log.

Transaction coordinator statistics. Each transaction has one transaction coordinator, which is chosen by one of the following methods:

- In a round-robin fashion
- By communication proximity
- By supplying a data placement hint when the transaction is started

Note

You can determine which TC selection method is used for transactions started from a given SQL node using the ndb_optimized_node_selection system variable.

All operations within the same transaction use the same transaction coordinator, which reports the following statistics:

- **Trans count.** This is the number transactions started in the last interval using this TC as the transaction coordinator. Any of these transactions may have committed, have been aborted, or remain uncommitted at the end of the reporting interval.

Note

Transactions do not migrate between TCs.

- **Commit count.** This is the number of transactions using this TC as the transaction coordinator that were committed in the last reporting interval. Because some transactions committed in this reporting interval may have started in a previous reporting interval, it is possible for `Commit count` to be greater than `Trans count`.

- **Read count.** This is the number of primary key read operations using this TC as the transaction coordinator that were started in the last reporting interval, including simple reads. This count also includes reads performed as part of unique index operations. A unique index read operation generates 2 primary key read operations—1 for the hidden unique index table, and 1 for the table on which the read takes place.

- **Simple read count.** This is the number of simple read operations using this TC as the transaction coordinator that were started in the last reporting interval.

- **Write count.** This is the number of primary key write operations using this TC as the transaction coordinator that were started in the last reporting interval. This includes all inserts, updates, writes and deletes, as well as writes performed as part of unique index operations.

  Note

  A unique index update operation can generate multiple PK read and write operations on the index table and on the base table.

- **AttrInfoCount.** This is the number of 32-bit data words received in the last reporting interval for primary key operations using this TC as the transaction coordinator. For reads, this is proportional to the number of columns requested. For inserts and updates, this is proportional to the number of columns written, and the size of their data. For delete operations, this is usually zero.

  Unique index operations generate multiple PK operations and so increase this count. However, data words sent to describe the PK operation itself, and the key information sent, are *not* counted here. Attribute information sent to describe columns to read for scans, or to describe ScanFilters, is also not counted in `AttrInfoCount`.

- **Concurrent Operations.** This is the number of primary key or scan operations using this TC as the transaction coordinator that were started during the last reporting interval but that were not completed. Operations increment this counter when they are started and decrement it when they are completed; this occurs after the transaction commits. Dirty reads and writes—as well as failed operations—decrement this counter.

  The maximum value that `Concurrent Operations` can have is the maximum number of operations that a TC block can support; currently, this is `(2 * MaxNoOfConcurrentOperations) + 16 + MaxNoOfConcurrentTransactions`. (For more information about these configuration parameters, see the *Transaction Parameters* section of Section 18.3.3.6, “Defining NDB Cluster Data Nodes”.)

- **Abort count.** This is the number of transactions using this TC as the transaction coordinator that were aborted during the last reporting interval. Because some transactions that were aborted in the last reporting interval may have started in a previous reporting interval, `Abort count` can sometimes be greater than `Trans count`.

- **Scans.** This is the number of table scans using this TC as the transaction coordinator that were started during the last reporting interval. This does not include range scans (that is, ordered index scans).

- **Range scans.** This is the number of ordered index scans using this TC as the transaction coordinator that were started in the last reporting interval.

- **Local reads.** This is the number of primary-key read operations performed using a transaction coordinator on a node that also holds the primary replica of the record. This count can also be obtained from the `LOCAL_READS` counter in the `ndbinfo.counters` table.
- **Local writes.** This contains the number of primary-key read operations that were performed using a transaction coordinator on a node that also holds the primary fragment replica of the record. This count can also be obtained from the `LOCAL_WRITES` counter in the `ndbinfo.counters` table.

Local query handler statistics (Operations). There is 1 cluster event per local query handler block (that is, 1 per data node process). Operations are recorded in the LQH where the data they are operating on resides.

Note

A single transaction may operate on data stored in multiple LQH blocks.

The `Operations` statistic provides the number of local operations performed by this LQH block in the last reporting interval, and includes all types of read and write operations (insert, update, write, and delete operations). This also includes operations used to replicate writes. For example, in a cluster with two fragment replicas, the write to the primary fragment replica is recorded in the primary LQH, and the write to the backup is recorded in the backup LQH. Unique key operations may result in multiple local operations; however, this does *not* include local operations generated as a result of a table scan or ordered index scan, which are not counted.

Process scheduler statistics. In addition to the statistics reported by the transaction coordinator and local query handler, each `ndbd` process has a scheduler which also provides useful metrics relating to the performance of an NDB Cluster. This scheduler runs in an infinite loop; during each loop the scheduler performs the following tasks:

1. Read any incoming messages from sockets into a job buffer.
2. Check whether there are any timed messages to be executed; if so, put these into the job buffer as well.
3. Execute (in a loop) any messages in the job buffer.
4. Send any distributed messages that were generated by executing the messages in the job buffer.
5. Wait for any new incoming messages.

Process scheduler statistics include the following:

- **Mean Loop Counter.** This is the number of loops executed in the third step from the preceding list. This statistic increases in size as the utilization of the TCP/IP buffer improves. You can use this to monitor changes in performance as you add new data node processes.
- **Mean send size and Mean receive size.** These statistics enable you to gauge the efficiency of, respectively writes and reads between nodes. The values are given in bytes. Higher values mean a lower cost per byte sent or received; the maximum value is 64K.

To cause all cluster log statistics to be logged, you can use the following command in the `NDB` management client:

```
ndb_mgm> ALL CLUSTERLOG STATISTICS=15
```

Note

Setting the threshold for `STATISTICS` to 15 causes the cluster log to become very verbose, and to grow quite rapidly in size, in direct proportion to the number of cluster nodes and the amount of activity in the NDB Cluster.

For more information about NDB Cluster management client commands relating to logging and reporting, see Section 18.5.3.1, “NDB Cluster Logging Management Commands”.

18.5.4 Summary of NDB Cluster Start Phases

This section provides a simplified outline of the steps involved when NDB Cluster data nodes are started. More complete information can be found in NDB Cluster Start Phases, in the *NDB Internals Guide*.

These phases are the same as those reported in the output from the `node_id STATUS` command in the management client (see Section 18.5.1, “Commands in the NDB Cluster Management Client”). These start phases are also reported in the `start_phase` column of the `ndbinfo.nodes` table.

Start types. There are several different startup types and modes, as shown in the following list:

- **Initial start.** The cluster starts with a clean file system on all data nodes. This occurs either when the cluster started for the very first time, or when all data nodes are restarted using the `--initial` option.

  Note

  Disk Data files are not removed when restarting a node using `--initial`.

- **System restart.** The cluster starts and reads data stored in the data nodes. This occurs when the cluster has been shut down after having been in use, when it is desired for the cluster to resume operations from the point where it left off.

- **Node restart.** This is the online restart of a cluster node while the cluster itself is running.

- **Initial node restart.** This is the same as a node restart, except that the node is reinitialized and started with a clean file system.

Setup and initialization (phase -1). Prior to startup, each data node (`ndbd` process) must be initialized. Initialization consists of the following steps:

1. Obtain a node ID
2. Fetch configuration data
3. Allocate ports to be used for inter-node communications
4. Allocate memory according to settings obtained from the configuration file

When a data node or SQL node first connects to the management node, it reserves a cluster node ID. To make sure that no other node allocates the same node ID, this ID is retained until the node has managed to connect to the cluster and at least one `ndbd` reports that this node is connected. This retention of the node ID is guarded by the connection between the node in question and `ndb_mgmd`.

After each data node has been initialized, the cluster startup process can proceed. The stages which the cluster goes through during this process are listed here:

- **Phase 0.** The `NDBFS` and `NDBCNTR` blocks start. Data node file systems are cleared on those data nodes that were started with `--initial` option.

- **Phase 1.** In this stage, all remaining `NDB` kernel blocks are started. NDB Cluster connections are set up, inter-block communications are established, and heartbeats are started. In the case of a node restart, API node connections are also checked.

  Note

  When one or more nodes hang in Phase 1 while the remaining node or nodes hang in Phase 2, this often indicates network problems. One possible

> cause of such issues is one or more cluster hosts having multiple network interfaces. Another common source of problems causing this condition is the blocking of TCP/IP ports needed for communications between cluster nodes. In the latter case, this is often due to a misconfigured firewall.

- **Phase 2.** The `NDBCNTR` kernel block checks the states of all existing nodes. The master node is chosen, and the cluster schema file is initialized.
- **Phase 3.** The `DBLQH` and `DBTC` kernel blocks set up communications between them. The startup type is determined; if this is a restart, the `DBDIH` block obtains permission to perform the restart.
- **Phase 4.** For an initial start or initial node restart, the redo log files are created. The number of these files is equal to `NoOfFragmentLogFiles`.

  For a system restart:

  - Read schema or schemas.
  - Read data from the local checkpoint.
  - Apply all redo information until the latest restorable global checkpoint has been reached.

  For a node restart, find the tail of the redo log.
- **Phase 5.** Most of the database-related portion of a data node start is performed during this phase. For an initial start or system restart, a local checkpoint is executed, followed by a global checkpoint. Periodic checks of memory usage begin during this phase, and any required node takeovers are performed.
- **Phase 6.** In this phase, node groups are defined and set up.
- **Phase 7.** The arbitrator node is selected and begins to function. The next backup ID is set, as is the backup disk write speed. Nodes reaching this start phase are marked as `Started`. It is now possible for API nodes (including SQL nodes) to connect to the cluster.
- **Phase 8.** If this is a system restart, all indexes are rebuilt (by `DBDIH`).
- **Phase 9.** The node internal startup variables are reset.
- **Phase 100 (OBSOLETE).** Formerly, it was at this point during a node restart or initial node restart that API nodes could connect to the node and begin to receive events. Currently, this phase is empty.
- **Phase 101.** At this point in a node restart or initial node restart, event delivery is handed over to the node joining the cluster. The newly-joined node takes over responsibility for delivering its primary data to subscribers. This phase is also referred to as *SUMA handover phase*.

After this process is completed for an initial start or system restart, transaction handling is enabled. For a node restart or initial node restart, completion of the startup process means that the node may now act as a transaction coordinator.

18.5.5 Performing a Rolling Restart of an NDB Cluster

This section discusses how to perform a *rolling restart* of an NDB Cluster installation, so called because it involves stopping and starting (or restarting) each node in turn, so that the cluster itself remains operational. This is often done as part of a *rolling upgrade* or *rolling downgrade*, where high availability of the cluster is mandatory and no downtime of the cluster as a whole is permissible. Where we refer to upgrades, the information provided here also generally applies to downgrades as well.

There are a number of reasons why a rolling restart might be desirable. These are described in the next few paragraphs.

Configuration change.
To make a change in the cluster's configuration, such as adding an SQL node to the cluster, or setting a configuration parameter to a new value.

NDB Cluster software upgrade or downgrade. To upgrade the cluster to a newer version of the NDB Cluster software (or to downgrade it to an older version). This is usually referred to as a "rolling upgrade" (or "rolling downgrade", when reverting to an older version of NDB Cluster).

Change on node host. To make changes in the hardware or operating system on which one or more NDB Cluster node processes are running.

System reset (cluster reset).
To reset the cluster because it has reached an undesirable state. In such cases it is often desirable to reload the data and metadata of one or more data nodes. This can be done in any of three ways:

- Start each data node process (`ndbd` or possibly `ndbmtd`) with the `--initial` option, which forces the data node to clear its file system and to reload all NDB Cluster data and metadata from the other data nodes.
- Create a backup using the `ndb_mgm` client `START BACKUP` command prior to performing the restart. Following the upgrade, restore the node or nodes using `ndb_restore`.

  See Section 18.5.8, "Online Backup of NDB Cluster", and Section 18.4.22, "`ndb_restore` — Restore an NDB Cluster Backup", for more information.
- Use `mysqldump` to create a backup prior to the upgrade; afterward, restore the dump using `LOAD DATA`.

Resource Recovery.
To free memory previously allocated to a table by successive `INSERT` and `DELETE` operations, for re-use by other NDB Cluster tables.

The process for performing a rolling restart may be generalized as follows:

1. Stop all cluster management nodes (`ndb_mgmd` processes), reconfigure them, then restart them. (See Rolling restarts with multiple management servers.)
2. Stop, reconfigure, then restart each cluster data node (`ndbd` process) in turn.

   Some node configuration parameters can be updated by issuing `RESTART` for each of the data nodes in the `ndb_mgm` client following the previous step. Other parameters require that the data node be stopped completely using the management client `STOP` command, then started again from a system shell by invoking the `ndbd` or `ndbmtd` executable as appropriate. (A shell command such as `kill` can also be used on most Unix systems to stop a data node process, but the `STOP` command is preferred and usually simpler.)

   Note

   On Windows, you can also use `SC STOP` and `SC START` commands, `NET STOP` and `NET START` commands, or the Windows Service Manager to stop and start nodes which have been installed as Windows services (see Section 18.2.2.4, "Installing NDB Cluster Processes as Windows Services").

   The type of restart required is indicated in the documentation for each node configuration parameter. See Section 18.3.3, "NDB Cluster Configuration Files".
3. Stop, reconfigure, then restart each cluster SQL node (`mysqld` process) in turn.

NDB Cluster supports a somewhat flexible order for upgrading nodes. When upgrading an NDB Cluster, you may upgrade API nodes (including SQL nodes) before upgrading the management nodes, data nodes, or both. In other words, you are permitted to upgrade the API and SQL nodes in any order. This is subject to the following provisions:

- This functionality is intended for use as part of an online upgrade only. A mix of node binaries from different NDB Cluster releases is neither intended nor supported for continuous, long-term use in a production setting.
- All management nodes must be upgraded before any data nodes are upgraded. This remains true regardless of the order in which you upgrade the cluster's API and SQL nodes.
- Features specific to the “new” version must not be used until all management nodes and data nodes have been upgraded.

  This also applies to any MySQL Server version change that may apply, in addition to the NDB engine version change, so do not forget to take this into account when planning the upgrade. (This is true for online upgrades of NDB Cluster in general.)

It is not possible for any API node to perform schema operations (such as data definition statements) during a node restart. Due in part to this limitation, schema operations are also not supported during an online upgrade or downgrade.

Rolling restarts with multiple management servers. When performing a rolling restart of an NDB Cluster with multiple management nodes, you should keep in mind that `ndb_mgmd` checks to see if any other management node is running, and, if so, tries to use that node's configuration data. To keep this from occurring, and to force `ndb_mgmd` to reread its configuration file, perform the following steps:

1. Stop all NDB Cluster `ndb_mgmd` processes.
2. Update all `config.ini` files.
3. Start a single `ndb_mgmd` with `--reload`, `--initial`, or both options as desired.
4. If you started the first `ndb_mgmd` with the `--initial` option, you must also start any remaining `ndb_mgmd` processes using `--initial`.

   Regardless of any other options used when starting the first `ndb_mgmd`, you should not start any remaining `ndb_mgmd` processes after the first one using `--reload`.
5. Complete the rolling restarts of the data nodes and API nodes as normal.

When performing a rolling restart to update the cluster's configuration, you can use the `config_generation` column of the `ndbinfo.nodes` table to keep track of which data nodes have been successfully restarted with the new configuration. See Section 18.5.14.18, “The ndbinfo nodes Table”.

18.5.6 NDB Cluster Single User Mode

Single user mode enables the database administrator to restrict access to the database system to a single API node, such as a MySQL server (SQL node) or an instance of `ndb_restore`. When entering single user mode, connections to all other API nodes are closed gracefully and all running transactions are aborted. No new transactions are permitted to start.

Once the cluster has entered single user mode, only the designated API node is granted access to the database.

You can use the `ALL STATUS` command in the `ndb_mgm` client to see when the cluster has entered single user mode. You can also check the `status` column of the `ndbinfo.nodes` table (see Section 18.5.14.18, “The ndbinfo nodes Table”, for more information).

Example:

```
ndb_mgm> ENTER SINGLE USER MODE 5
```

After this command has executed and the cluster has entered single user mode, the API node whose node ID is 5 becomes the cluster's only permitted user.

The node specified in the preceding command must be an API node; attempting to specify any other type of node is rejected.

Note

When the preceding command is invoked, all transactions running on the designated node are aborted, the connection is closed, and the server must be restarted.

The command `EXIT SINGLE USER MODE` changes the state of the cluster's data nodes from single user mode to normal mode. API nodes—such as MySQL Servers—waiting for a connection (that is, waiting for the cluster to become ready and available), are again permitted to connect. The API node denoted as the single-user node continues to run (if still connected) during and after the state change.

Example:

```
ndb_mgm> EXIT SINGLE USER MODE
```

There are two recommended ways to handle a node failure when running in single user mode:

- Method 1:

  1. Finish all single user mode transactions
  2. Issue the `EXIT SINGLE USER MODE` command
  3. Restart the cluster's data nodes

- Method 2:

  Restart data nodes prior to entering single user mode.

18.5.7 Adding NDB Cluster Data Nodes Online

This section describes how to add NDB Cluster data nodes “online”—that is, without needing to shut down the cluster completely and restart it as part of the process.

Important

Currently, you must add new data nodes to an NDB Cluster as part of a new node group. In addition, it is not possible to change the number of fragment replicas (or the number of nodes per node group) online.

18.5.7.1 Adding NDB Cluster Data Nodes Online: General Issues

This section provides general information about the behavior of and current limitations in adding NDB Cluster nodes online.

Redistribution of Data. The ability to add new nodes online includes a means to reorganize `NDBCLUSTER` table data and indexes so that they are distributed across all data nodes, including the new ones, by means of the `ALTER ONLINE TABLE ... REORGANIZE PARTITION` statement. Table reorganization of both in-memory and Disk Data tables is supported. This redistribution does not currently include unique indexes (only ordered indexes are redistributed). Prior to NDB 7.3.3, `BLOB` table data is also not redistributed using this method (Bug #13714148).

The redistribution for `NDBCLUSTER` tables already existing before the new data nodes were added is not automatic, but can be accomplished using simple SQL statements in `mysql` or another MySQL

client application. However, all data and indexes added to tables created after a new node group has been added are distributed automatically among all cluster data nodes, including those added as part of the new node group.

Partial starts. It is possible to add a new node group without all of the new data nodes being started. It is also possible to add a new node group to a degraded cluster—that is, a cluster that is only partially started, or where one or more data nodes are not running. In the latter case, the cluster must have enough nodes running to be viable before the new node group can be added.

Effects on ongoing operations. Normal DML operations using NDB Cluster data are not prevented by the creation or addition of a new node group, or by table reorganization. However, it is not possible to perform DDL concurrently with table reorganization—that is, no other DDL statements can be issued while an `ALTER TABLE ... REORGANIZE PARTITION` statement is executing. In addition, during the execution of `ALTER TABLE ... REORGANIZE PARTITION` (or the execution of any other DDL statement), it is not possible to restart cluster data nodes.

Failure handling. Failures of data nodes during node group creation and table reorganization are handled as shown in the following table:

Table 18.55 Data node failure handling during node group creation and table reorganization

Failure during	Failure in “Old” data node	Failure in “New” data node	System Failure

Node group creation	• **If a node other than the master fails:** The creation of the node group is always rolled forward.
	• **If the master fails:**
	  • **If the internal commit point has been reached:** The creation of the node group is rolled forward.
	  • **If the internal commit point has not yet been reached.** The creation of the node group is rolled back
		• **If a node other than the master fails:** The creation of the node group is always rolled forward.
		• **If the master fails:**
		  • **If the internal commit point has been reached:** The creation of the node group is rolled forward.
		  • **If the internal commit point has not yet been reached.** The creation of the node group is rolled back
			• **If the execution of CREATE NODEGROUP has reached the internal commit point:** When restarted, the cluster includes the new node group. Otherwise it without.
			• **If the execution of CREATE NODEGROUP has not yet reached the internal commit point:** When restarted, the cluster does not include the new node group.

Table reorganization	• **If a node other than the master fails:** The table reorganization is always rolled forward.
	• **If the master fails:**
	  • **If the internal commit point has been reached:** The table reorganization is rolled forward.
		• **If a node other than the master fails:** The table reorganization is always rolled forward.
		• **If the master fails:**
		  • **If the internal commit point has been reached:** The table reorganization is rolled forward.
			• **If the execution of an ALTER TABLE ... REORGANIZE PARTITION statement has reached the internal commit point:** When the cluster is restarted, the data and indexes belonging to *`table`* are distributed using the “new” data nodes.

Failure during	Failure in “Old” data node	Failure in “New” data node	System Failure
	• **If the internal commit point has not yet been reached.** The table reorganization is rolled back.	• **If the internal commit point has not yet been reached.** The table reorganization is rolled back.	• **If the execution of an ALTER TABLE ... REORGANIZE PARTITION statement has not yet reached the internal commit point:** When the cluster is restarted, the data and indexes belonging to *`table`* are distributed using only the “old” data nodes.

Dropping node groups. The `ndb_mgm` client supports a `DROP NODEGROUP` command, but it is possible to drop a node group only when no data nodes in the node group contain any data. Since there is currently no way to “empty” a specific data node or node group, this command works only the following two cases:

1. After issuing `CREATE NODEGROUP` in the `ndb_mgm` client, but before issuing any `ALTER ONLINE TABLE ... REORGANIZE PARTITION` statements in the `mysql` client.
2. After dropping all `NDBCLUSTER` tables using `DROP TABLE`.

   `TRUNCATE TABLE` does not work for this purpose because the data nodes continue to store the table definitions.

18.5.7.2 Adding NDB Cluster Data Nodes Online: Basic procedure

In this section, we list the basic steps required to add new data nodes to an NDB Cluster. This procedure applies whether you are using `ndbd` or `ndbmtd` binaries for the data node processes. For a more detailed example, see Section 18.5.7.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”.

Assuming that you already have a running NDB Cluster, adding data nodes online requires the following steps:

1. Edit the cluster configuration `config.ini` file, adding new `[ndbd]` sections corresponding to the nodes to be added. In the case where the cluster uses multiple management servers, these changes need to be made to all `config.ini` files used by the management servers.

   You must be careful that node IDs for any new data nodes added in the `config.ini` file do not overlap node IDs used by existing nodes. In the event that you have API nodes using dynamically allocated node IDs and these IDs match node IDs that you want to use for new data nodes, it is possible to force any such API nodes to “migrate”, as described later in this procedure.

2. Perform a rolling restart of all NDB Cluster management servers.

   Important

   All management servers must be restarted with the `--reload` or `--initial` option to force the reading of the new configuration.

3. Perform a rolling restart of all existing NDB Cluster data nodes. It is not necessary (or usually even desirable) to use `--initial` when restarting the existing data nodes.

   If you are using API nodes with dynamically allocated IDs matching any node IDs that you wish to assign to new data nodes, you must restart all API nodes (including SQL nodes) before restarting

any of the data nodes processes in this step. This causes any API nodes with node IDs that were previously not explicitly assigned to relinquish those node IDs and acquire new ones.

4. Perform a rolling restart of any SQL or API nodes connected to the NDB Cluster.

5. Start the new data nodes.

   The new data nodes may be started in any order. They can also be started concurrently, as long as they are started after the rolling restarts of all existing data nodes have been completed, and before proceeding to the next step.

6. Execute one or more `CREATE NODEGROUP` commands in the NDB Cluster management client to create the new node group or node groups to which the new data nodes belong.

7. Redistribute the cluster's data among all data nodes, including the new ones. Normally this is done by issuing an `ALTER ONLINE TABLE ... REORGANIZE PARTITION` statement in the `mysql` client for each `NDBCLUSTER` table.

   Exception: For tables created using the `MAX_ROWS` option, this statement does not work; instead, use `ALTER ONLINE TABLE ... MAX_ROWS=...` to reorganize such tables.

   Note

   This needs to be done only for tables already existing at the time the new node group is added. Data in tables created after the new node group is added is distributed automatically; however, data added to any given table `tbl` that existed before the new nodes were added is not distributed using the new nodes until that table has been reorganized.

8. `ALTER ONLINE TABLE ... REORGANIZE PARTITION` reorganizes partitions but does not reclaim the space freed on the “old” nodes. You can do this by issuing, for each `NDBCLUSTER` table, an `OPTIMIZE TABLE` statement in the `mysql` client.

   This works for space used by variable-width columns of in-memory `NDB` tables. `OPTIMIZE TABLE` is not supported for fixed-width columns of in-memory tables; it is also not supported for Disk Data tables.

You can add all the nodes desired, then issue several `CREATE NODEGROUP` commands in succession to add the new node groups to the cluster.

18.5.7.3 Adding NDB Cluster Data Nodes Online: Detailed Example

In this section we provide a detailed example illustrating how to add new NDB Cluster data nodes online, starting with an NDB Cluster having 2 data nodes in a single node group and concluding with a cluster having 4 data nodes in 2 node groups.

Starting configuration. For purposes of illustration, we assume a minimal configuration, and that the cluster uses a `config.ini` file containing only the following information:

```
[ndbd default]
DataMemory = 100M
IndexMemory = 100M
NoOfReplicas = 2
DataDir = /usr/local/mysql/var/mysql-cluster

[ndbd]
Id = 1
HostName = 198.51.100.1

[ndbd]
Id = 2
HostName = 198.51.100.2
```

```
[mgm]
HostName = 198.51.100.10
Id = 10

[api]
Id=20
HostName = 198.51.100.20

[api]
Id=21
HostName = 198.51.100.21
```

Note

We have left a gap in the sequence between data node IDs and other nodes. This make it easier later to assign node IDs that are not already in use to data nodes which are newly added.

We also assume that you have already started the cluster using the appropriate command line or `my.cnf` options, and that running `SHOW` in the management client produces output similar to what is shown here:

```
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: 198.51.100.10:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=1    @198.51.100.1  (5.6.51-ndb-7.4.31, Nodegroup: 0, *)
id=2    @198.51.100.2  (5.6.51-ndb-7.4.31, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=10   @198.51.100.10  (5.6.51-ndb-7.4.31)

[mysqld(API)]   2 node(s)
id=20   @198.51.100.20  (5.6.51-ndb-7.4.31)
id=21   @198.51.100.21  (5.6.51-ndb-7.4.31)
```

Finally, we assume that the cluster contains a single `NDBCLUSTER` table created as shown here:

```
USE n;

CREATE TABLE ips (
    id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    country_code CHAR(2) NOT NULL,
    type CHAR(4) NOT NULL,
    ip_address VARCHAR(15) NOT NULL,
    addresses BIGINT UNSIGNED DEFAULT NULL,
    date BIGINT UNSIGNED DEFAULT NULL
)   ENGINE NDBCLUSTER;
```

The memory usage and related information shown later in this section was generated after inserting approximately 50000 rows into this table.

Note

In this example, we show the single-threaded `ndbd` being used for the data node processes. However—beginning with NDB 7.0.4—you can also apply this example, if you are using the multithreaded `ndbmtd` by substituting `ndbmtd` for `ndbd` wherever it appears in the steps that follow. (Bug #43108)

Step 1: Update configuration file. Open the cluster global configuration file in a text editor and add `[ndbd]` sections corresponding to the 2 new data nodes. (We give these data nodes IDs 3 and 4, and assume that they are to be run on host machines at addresses 198.51.100.3 and 198.51.100.4, respectively.) After you have added the new sections, the contents of the `config.ini` file should look like what is shown here, where the additions to the file are shown in bold type:

```
[ndbd default]
DataMemory = 100M
IndexMemory = 100M
NoOfReplicas = 2
DataDir = /usr/local/mysql/var/mysql-cluster

[ndbd]
Id = 1
HostName = 198.51.100.1

[ndbd]
Id = 2
HostName = 198.51.100.2

[ndbd]
Id = 3
HostName = 198.51.100.3

[ndbd]
Id = 4
HostName = 198.51.100.4

[mgm]
HostName = 198.51.100.10
Id = 10

[api]
Id=20
HostName = 198.51.100.20

[api]
Id=21
HostName = 198.51.100.21
```

Once you have made the necessary changes, save the file.

Step 2: Restart the management server. Restarting the cluster management server requires that you issue separate commands to stop the management server and then to start it again, as follows:

1. Stop the management server using the management client `STOP` command, as shown here:

   ```
   ndb_mgm> 10 STOP
   Node 10 has shut down.
   Disconnecting to allow Management Server to shutdown

   shell>
   ```

2. Because shutting down the management server causes the management client to terminate, you must start the management server from the system shell. For simplicity, we assume that `config.ini` is in the same directory as the management server binary, but in practice, you must supply the correct path to the configuration file. You must also supply the `--reload` or `--initial` option so that the management server reads the new configuration from the file rather than its configuration cache. If your shell's current directory is also the same as the directory where the management server binary is located, then you can invoke the management server as shown here:

   ```
   shell> ndb_mgmd -f config.ini --reload
   2008-12-08 17:29:23 [MgmSrvr] INFO     -- NDB Cluster Management Server. 5.6.51-ndb-7.4.31
   2008-12-08 17:29:23 [MgmSrvr] INFO     -- Reading cluster configuration from 'config.ini'
   ```

If you check the output of `SHOW` in the management client after restarting the `ndb_mgm` process, you should now see something like this:

```
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: 198.51.100.10:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=1    @198.51.100.1  (5.6.51-ndb-7.4.31, Nodegroup: 0, *)
```

```
id=2    @198.51.100.2  (5.6.51-ndb-7.4.31, Nodegroup: 0)
id=3 (not connected, accepting connect from 198.51.100.3)
id=4 (not connected, accepting connect from 198.51.100.4)

[ndb_mgmd(MGM)] 1 node(s)
id=10   @198.51.100.10  (5.6.51-ndb-7.4.31)

[mysqld(API)]   2 node(s)
id=20   @198.51.100.20  (5.6.51-ndb-7.4.31)
id=21   @198.51.100.21  (5.6.51-ndb-7.4.31)
```

Step 3: Perform a rolling restart of the existing data nodes. This step can be accomplished entirely within the cluster management client using the `RESTART` command, as shown here:

```
ndb_mgm> 1 RESTART
Node 1: Node shutdown initiated
Node 1: Node shutdown completed, restarting, no start.
Node 1 is being restarted

ndb_mgm> Node 1: Start initiated (version 7.4.31)
Node 1: Started (version 7.4.31)

ndb_mgm> 2 RESTART
Node 2: Node shutdown initiated
Node 2: Node shutdown completed, restarting, no start.
Node 2 is being restarted

ndb_mgm> Node 2: Start initiated (version 7.4.31)

ndb_mgm> Node 2: Started (version 7.4.31)
```

Important

After issuing each *`X`* `RESTART` command, wait until the management client reports `Node` *`X`*`: Started (version ...)` *before* proceeding any further.

You can verify that all existing data nodes were restarted using the updated configuration by checking the `ndbinfo.nodes` table in the `mysql` client.

Step 4: Perform a rolling restart of all cluster API nodes. Shut down and restart each MySQL server acting as an SQL node in the cluster using `mysqladmin shutdown` followed by `mysqld_safe` (or another startup script). This should be similar to what is shown here, where *`password`* is the MySQL `root` password for a given MySQL server instance:

```
shell> mysqladmin -uroot -ppassword shutdown
081208 20:19:56 mysqld_safe mysqld from pid file
/usr/local/mysql/var/tonfisk.pid ended
shell> mysqld_safe --ndbcluster --ndb-connectstring=198.51.100.10 &
081208 20:20:06 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
081208 20:20:06 mysqld_safe Starting mysqld daemon with databases
from /usr/local/mysql/var
```

Of course, the exact input and output depend on how and where MySQL is installed on the system, as well as which options you choose to start it (and whether or not some or all of these options are specified in a `my.cnf` file).

Step 5: Perform an initial start of the new data nodes. From a system shell on each of the hosts for the new data nodes, start the data nodes as shown here, using the `--initial` option:

```
shell> ndbd -c 198.51.100.10 --initial
```

Note

Unlike the case with restarting the existing data nodes, you can start the new data nodes concurrently; you do not need to wait for one to finish starting before starting the other.

Wait until both of the new data nodes have started before proceeding with the next step. Once the new data nodes have started, you can see in the output of the management client `SHOW` command that they do not yet belong to any node group (as indicated with bold type here):

```
ndb_mgm> SHOW
Connected to Management Server at: 198.51.100.10:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=1    @198.51.100.1  (5.6.51-ndb-7.4.31, Nodegroup: 0, *)
id=2    @198.51.100.2  (5.6.51-ndb-7.4.31, Nodegroup: 0)
id=3    @198.51.100.3  (5.6.51-ndb-7.4.31, no nodegroup)
id=4    @198.51.100.4  (5.6.51-ndb-7.4.31, no nodegroup)

[ndb_mgmd(MGM)] 1 node(s)
id=10   @198.51.100.10  (5.6.51-ndb-7.4.31)

[mysqld(API)]   2 node(s)
id=20   @198.51.100.20  (5.6.51-ndb-7.4.31)
id=21   @198.51.100.21  (5.6.51-ndb-7.4.31)
```

Step 6: Create a new node group. You can do this by issuing a `CREATE NODEGROUP` command in the cluster management client. This command takes as its argument a comma-separated list of the node IDs of the data nodes to be included in the new node group, as shown here:

```
ndb_mgm> CREATE NODEGROUP 3,4
Nodegroup 1 created
```

By issuing `SHOW` again, you can verify that data nodes 3 and 4 have joined the new node group (again indicated in bold type):

```
ndb_mgm> SHOW
Connected to Management Server at: 198.51.100.10:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=1    @198.51.100.1  (5.6.51-ndb-7.4.31, Nodegroup: 0, *)
id=2    @198.51.100.2  (5.6.51-ndb-7.4.31, Nodegroup: 0)
id=3    @198.51.100.3  (5.6.51-ndb-7.4.31, Nodegroup: 1)
id=4    @198.51.100.4  (5.6.51-ndb-7.4.31, Nodegroup: 1)

[ndb_mgmd(MGM)] 1 node(s)
id=10   @198.51.100.10  (5.6.51-ndb-7.4.31)

[mysqld(API)]   2 node(s)
id=20   @198.51.100.20  (5.6.51-ndb-7.4.31)
id=21   @198.51.100.21  (5.6.51-ndb-7.4.31)
```

Step 7: Redistribute cluster data. When a node group is created, existing data and indexes are not automatically distributed to the new node group's data nodes, as you can see by issuing the appropriate `REPORT` command in the management client:

```
ndb_mgm> ALL REPORT MEMORY

Node 1: Data usage is 5%(177 32K pages of total 3200)
Node 1: Index usage is 0%(108 8K pages of total 12832)
Node 2: Data usage is 5%(177 32K pages of total 3200)
Node 2: Index usage is 0%(108 8K pages of total 12832)
Node 3: Data usage is 0%(0 32K pages of total 3200)
Node 3: Index usage is 0%(0 8K pages of total 12832)
Node 4: Data usage is 0%(0 32K pages of total 3200)
Node 4: Index usage is 0%(0 8K pages of total 12832)
```

By using `ndb_desc` with the `-p` option, which causes the output to include partitioning information, you can see that the table still uses only 2 partitions (in the `Per partition info` section of the output, shown here in bold text):

```
shell> ndb_desc -c 198.51.100.10 -d n ips -p
-- ips --
```

```
Version: 1
Fragment type: 9
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 6
Number of primary keys: 1
Length of frm data: 340
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
FragmentCount: 2
TableStatus: Retrieved
-- Attributes --
id Bigint PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
country_code Char(2;latin1_swedish_ci) NOT NULL AT=FIXED ST=MEMORY
type Char(4;latin1_swedish_ci) NOT NULL AT=FIXED ST=MEMORY
ip_address Varchar(15;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
addresses Bigunsigned NULL AT=FIXED ST=MEMORY
date Bigunsigned NULL AT=FIXED ST=MEMORY

-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex

-- Per partition info --
Partition   Row count   Commit count  Frag fixed memory   Frag varsized memory
0           26086       26086         1572864             557056
1           26329       26329         1605632             557056

NDBT_ProgramExit: 0 - OK
```

You can cause the data to be redistributed among all of the data nodes by performing, for each `NDB` table, an `ALTER ONLINE TABLE ... REORGANIZE PARTITION` statement in the `mysql` client.

Important

`ALTER ONLINE TABLE ... REORGANIZE PARTITION` does not work on tables that were created with the `MAX_ROWS` option. Instead, use `ALTER ONLINE TABLE ... MAX_ROWS=...` to reorganize such tables.

After issuing the statement `ALTER ONLINE TABLE ips REORGANIZE PARTITION`, you can see using `ndb_desc` that the data for this table is now stored using 4 partitions, as shown here (with the relevant portions of the output in bold type):

```
shell> ndb_desc -c 198.51.100.10 -d n ips -p
-- ips --
Version: 16777217
Fragment type: 9
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 6
Number of primary keys: 1
Length of frm data: 341
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
FragmentCount: 4
TableStatus: Retrieved
-- Attributes --
id Bigint PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
country_code Char(2;latin1_swedish_ci) NOT NULL AT=FIXED ST=MEMORY
type Char(4;latin1_swedish_ci) NOT NULL AT=FIXED ST=MEMORY
ip_address Varchar(15;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
addresses Bigunsigned NULL AT=FIXED ST=MEMORY
```

```
date Bigunsigned NULL AT=FIXED ST=MEMORY

-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex

-- Per partition info --
Partition   Row count   Commit count  Frag fixed memory   Frag varsized memory
0           12981       52296         1572864             557056
1           13236       52515         1605632             557056
2           13105       13105         819200              294912
3           13093       13093         819200              294912

NDBT_ProgramExit: 0 - OK
```

Note

Normally, `ALTER [ONLINE] TABLE table_name REORGANIZE PARTITION` is used with a list of partition identifiers and a set of partition definitions to create a new partitioning scheme for a table that has already been explicitly partitioned. Its use here to redistribute data onto a new NDB Cluster node group is an exception in this regard; when used in this way, only the name of the table is used following the `TABLE` keyword, and no other keywords or identifiers follow `REORGANIZE PARTITION`.

For more information, see Section 13.1.7, “ALTER TABLE Statement”.

In addition, for each table, the `ALTER ONLINE TABLE` statement should be followed by an `OPTIMIZE TABLE` to reclaim wasted space. You can obtain a list of all `NDBCLUSTER` tables using the following query against the `INFORMATION_SCHEMA.TABLES` table:

```
SELECT TABLE_SCHEMA, TABLE_NAME
    FROM INFORMATION_SCHEMA.TABLES
    WHERE ENGINE = 'NDBCLUSTER';
```

Note

The `INFORMATION_SCHEMA.TABLES.ENGINE` value for an NDB Cluster table is always `NDBCLUSTER`, regardless of whether the `CREATE TABLE` statement used to create the table (or `ALTER TABLE` statement used to convert an existing table from a different storage engine) used `NDB` or `NDBCLUSTER` in its `ENGINE` option.

You can see after performing these statements in the output of `ALL REPORT MEMORY` that the data and indexes are now redistributed between all cluster data nodes, as shown here:

```
ndb_mgm> ALL REPORT MEMORY

Node 1: Data usage is 5%(176 32K pages of total 3200)
Node 1: Index usage is 0%(76 8K pages of total 12832)
Node 2: Data usage is 5%(176 32K pages of total 3200)
Node 2: Index usage is 0%(76 8K pages of total 12832)
Node 3: Data usage is 2%(80 32K pages of total 3200)
Node 3: Index usage is 0%(51 8K pages of total 12832)
Node 4: Data usage is 2%(80 32K pages of total 3200)
Node 4: Index usage is 0%(50 8K pages of total 12832)
```

Note

Since only one DDL operation on `NDBCLUSTER` tables can be executed at a time, you must wait for each `ALTER ONLINE TABLE ... REORGANIZE PARTITION` statement to finish before issuing the next one.

It is not necessary to issue `ALTER ONLINE TABLE ... REORGANIZE PARTITION` statements for `NDBCLUSTER` tables created *after* the new data nodes have been added; data added to such tables is

distributed among all data nodes automatically. However, in `NDBCLUSTER` tables that existed *prior to* the addition of the new nodes, neither existing nor new data is distributed using the new nodes until these tables have been reorganized using `ALTER ONLINE TABLE ... REORGANIZE PARTITION`.

Alternative procedure, without rolling restart. It is possible to avoid the need for a rolling restart by configuring the extra data nodes, but not starting them, when first starting the cluster. We assume, as before, that you wish to start with two data nodes—nodes 1 and 2—in one node group and later to expand the cluster to four data nodes, by adding a second node group consisting of nodes 3 and 4:

```
[ndbd default]
DataMemory = 100M
IndexMemory = 100M
NoOfReplicas = 2
DataDir = /usr/local/mysql/var/mysql-cluster

[ndbd]
Id = 1
HostName = 198.51.100.1

[ndbd]
Id = 2
HostName = 198.51.100.2

[ndbd]
Id = 3
HostName = 198.51.100.3
Nodegroup = 65536

[ndbd]
Id = 4
HostName = 198.51.100.4
Nodegroup = 65536

[mgm]
HostName = 198.51.100.10
Id = 10

[api]
Id=20
HostName = 198.51.100.20

[api]
Id=21
HostName = 198.51.100.21
```

The data nodes to be brought online at a later time (nodes 3 and 4) can be configured with `NodeGroup = 65536`, in which case nodes 1 and 2 can each be started as shown here:

```
shell> ndbd -c 198.51.100.10 --initial
```

The data nodes configured with `NodeGroup = 65536` are treated by the management server as though you had started nodes 1 and 2 using `--nowait-nodes=3,4` after waiting for a period of time determined by the setting for the `StartNoNodeGroupTimeout` data node configuration parameter. By default, this is 15 seconds (15000 milliseconds).

Note

`StartNoNodegroupTimeout` must be the same for all data nodes in the cluster; for this reason, you should always set it in the `[ndbd default]` section of the `config.ini` file, rather than for individual data nodes.

When you are ready to add the second node group, you need only perform the following additional steps:

1. Start data nodes 3 and 4, invoking the data node process once for each new node:

   ```
   shell> ndbd -c 198.51.100.10 --initial
   ```

2. Issue the appropriate `CREATE NODEGROUP` command in the management client:

```
ndb_mgm> CREATE NODEGROUP 3,4
```

3. In the `mysql` client, issue `ALTER ONLINE TABLE ... REORGANIZE PARTITION` and `OPTIMIZE TABLE` statements for each existing `NDBCLUSTER` table. (As noted elsewhere in this section, existing NDB Cluster tables cannot use the new nodes for data distribution until this has been done.)

18.5.8 Online Backup of NDB Cluster

The next few sections describe how to prepare for and then to create an NDB Cluster backup using the functionality for this purpose found in the `ndb_mgm` management client. To distinguish this type of backup from a backup made using `mysqldump`, we sometimes refer to it as a "native" NDB Cluster backup. (For information about the creation of backups with `mysqldump`, see Section 4.5.4, "`mysqldump` — A Database Backup Program".) Restoration of NDB Cluster backups is done using the `ndb_restore` utility provided with the NDB Cluster distribution; for information about `ndb_restore` and its use in restoring NDB Cluster backups, see Section 18.4.22, "`ndb_restore` — Restore an NDB Cluster Backup".

18.5.8.1 NDB Cluster Backup Concepts

A backup is a snapshot of the database at a given time. The backup consists of three main parts:

- **Metadata.** The names and definitions of all database tables
- **Table records.** The data actually stored in the database tables at the time that the backup was made
- **Transaction log.** A sequential record telling how and when data was stored in the database

Each of these parts is saved on all nodes participating in the backup. During backup, each node saves these three parts into three files on disk:

- `BACKUP-`*`backup_id`*`.`*`node_id`*`.ctl`

  A control file containing control information and metadata. Each node saves the same table definitions (for all tables in the cluster) to its own version of this file.

- `BACKUP-`*`backup_id`*`-0.`*`node_id`*`.data`

  A data file containing the table records, which are saved on a per-fragment basis. That is, different nodes save different fragments during the backup. The file saved by each node starts with a header that states the tables to which the records belong. Following the list of records there is a footer containing a checksum for all records.

- `BACKUP-`*`backup_id`*`.`*`node_id`*`.log`

  A log file containing records of committed transactions. Only transactions on tables stored in the backup are stored in the log. Nodes involved in the backup save different records because different nodes host different database fragments.

In the listing just shown, *`backup_id`* stands for the backup identifier and *`node_id`* is the unique identifier for the node creating the file.

The location of the backup files is determined by the `BackupDataDir` parameter.

18.5.8.2 Using The NDB Cluster Management Client to Create a Backup

Before starting a backup, make sure that the cluster is properly configured for performing one. (See Section 18.5.8.3, "Configuration for NDB Cluster Backups".)

The START BACKUP command is used to create a backup:

```
START BACKUP [backup_id] [wait_option] [snapshot_option]

wait_option:
WAIT {STARTED | COMPLETED} | NOWAIT

snapshot_option:
SNAPSHOTSTART | SNAPSHOTEND
```

Successive backups are automatically identified sequentially, so the *backup_id*, an integer greater than or equal to 1, is optional; if it is omitted, the next available value is used. If an existing *backup_id* value is used, the backup fails with the error Backup failed: file already exists. If used, the *backup_id* must follow START BACKUP immediately, before any other options are used.

The *wait_option* can be used to determine when control is returned to the management client after a START BACKUP command is issued, as shown in the following list:

- If NOWAIT is specified, the management client displays a prompt immediately, as seen here:

  ```
  ndb_mgm> START BACKUP NOWAIT
  ndb_mgm>
  ```

  In this case, the management client can be used even while it prints progress information from the backup process.

- With WAIT STARTED the management client waits until the backup has started before returning control to the user, as shown here:

  ```
  ndb_mgm> START BACKUP WAIT STARTED
  Waiting for started, this may take several minutes
  Node 2: Backup 3 started from node 1
  ndb_mgm>
  ```

- **WAIT COMPLETED** causes the management client to wait until the backup process is complete before returning control to the user.

WAIT COMPLETED is the default.

A *snapshot_option* can be used to determine whether the backup matches the state of the cluster when START BACKUP was issued, or when it was completed. SNAPSHOTSTART causes the backup to match the state of the cluster when the backup began; SNAPSHOTEND causes the backup to reflect the state of the cluster when the backup was finished. SNAPSHOTEND is the default, and matches the behavior found in previous NDB Cluster releases.

Note

If you use the SNAPSHOTSTART option with START BACKUP, and the CompressedBackup parameter is enabled, only the data and control files are compressed—the log file is not compressed.

If both a *wait_option* and a *snapshot_option* are used, they may be specified in either order. For example, all of the following commands are valid, assuming that there is no existing backup having 4 as its ID:

```
START BACKUP WAIT STARTED SNAPSHOTSTART
START BACKUP SNAPSHOTSTART WAIT STARTED
START BACKUP 4 WAIT COMPLETED SNAPSHOTSTART
START BACKUP SNAPSHOTEND WAIT COMPLETED
START BACKUP 4 NOWAIT SNAPSHOTSTART
```

The procedure for creating a backup consists of the following steps:

1. Start the management client (ndb_mgm), if it not running already.

2. Execute the **START BACKUP** command. This produces several lines of output indicating the progress of the backup, as shown here:

```
ndb_mgm> START BACKUP
Waiting for completed, this may take several minutes
Node 2: Backup 1 started from node 1
Node 2: Backup 1 started from node 1 completed
 StartGCP: 177 StopGCP: 180
 #Records: 7362 #LogRecords: 0
 Data: 453648 bytes Log: 0 bytes
ndb_mgm>
```

3. When the backup has started the management client displays this message:

```
Backup backup_id started from node node_id
```

backup_id is the unique identifier for this particular backup. This identifier is saved in the cluster log, if it has not been configured otherwise. *node_id* is the identifier of the management server that is coordinating the backup with the data nodes. At this point in the backup process the cluster has received and processed the backup request. It does not mean that the backup has finished. An example of this statement is shown here:

```
Node 2: Backup 1 started from node 1
```

4. The management client indicates with a message like this one that the backup has started:

```
Backup backup_id started from node node_id completed
```

As is the case for the notification that the backup has started, *backup_id* is the unique identifier for this particular backup, and *node_id* is the node ID of the management server that is coordinating the backup with the data nodes. This output is accompanied by additional information including relevant global checkpoints, the number of records backed up, and the size of the data, as shown here:

```
Node 2: Backup 1 started from node 1 completed
 StartGCP: 177 StopGCP: 180
 #Records: 7362 #LogRecords: 0
 Data: 453648 bytes Log: 0 bytes
```

It is also possible to perform a backup from the system shell by invoking `ndb_mgm` with the `-e` or `--execute` option, as shown in this example:

```
shell> ndb_mgm -e "START BACKUP 6 WAIT COMPLETED SNAPSHOTSTART"
```

When using `START BACKUP` in this way, you must specify the backup ID.

Cluster backups are created by default in the `BACKUP` subdirectory of the `DataDir` on each data node. This can be overridden for one or more data nodes individually, or for all cluster data nodes in the `config.ini` file using the `BackupDataDir` configuration parameter. The backup files created for a backup with a given *backup_id* are stored in a subdirectory named `BACKUP-`*backup_id* in the backup directory.

Cancelling backups. To cancel or abort a backup that is already in progress, perform the following steps:

1. Start the management client.

2. Execute this command:

```
ndb_mgm> ABORT BACKUP backup_id
```

The number *backup_id* is the identifier of the backup that was included in the response of the management client when the backup was started (in the message `Backup` *backup_id* `started from node` *management_node_id*).

3. The management client acknowledges the abort request with `Abort of backup backup_id ordered`.

Note

At this point, the management client has not yet received a response from the cluster data nodes to this request, and the backup has not yet actually been aborted.

4. After the backup has been aborted, the management client reports this fact in a manner similar to what is shown here:

```
Node 1: Backup 3 started from 5 has been aborted.
  Error: 1321 - Backup aborted by user request: Permanent error: User defined error
Node 3: Backup 3 started from 5 has been aborted.
  Error: 1323 - 1323: Permanent error: Internal error
Node 2: Backup 3 started from 5 has been aborted.
  Error: 1323 - 1323: Permanent error: Internal error
Node 4: Backup 3 started from 5 has been aborted.
  Error: 1323 - 1323: Permanent error: Internal error
```

In this example, we have shown sample output for a cluster with 4 data nodes, where the sequence number of the backup to be aborted is 3, and the management node to which the cluster management client is connected has the node ID 5. The first node to complete its part in aborting the backup reports that the reason for the abort was due to a request by the user. (The remaining nodes report that the backup was aborted due to an unspecified internal error.)

Note

There is no guarantee that the cluster nodes respond to an `ABORT BACKUP` command in any particular order.

The `Backup backup_id started from node management_node_id has been aborted` messages mean that the backup has been terminated and that all files relating to this backup have been removed from the cluster file system.

It is also possible to abort a backup in progress from a system shell using this command:

```
shell> ndb_mgm -e "ABORT BACKUP backup_id"
```

Note

If there is no backup having the ID *backup_id* running when an `ABORT BACKUP` is issued, the management client makes no response, nor is it indicated in the cluster log that an invalid abort command was sent.

18.5.8.3 Configuration for NDB Cluster Backups

Five configuration parameters are essential for backup:

- `BackupDataBufferSize`

  The amount of memory used to buffer data before it is written to disk.

- `BackupLogBufferSize`

  The amount of memory used to buffer log records before these are written to disk.

- `BackupMemory`

  The total memory allocated in a data node for backups. This should be the sum of the memory allocated for the backup data buffer and the backup log buffer.

- `BackupWriteSize`

The default size of blocks written to disk. This applies for both the backup data buffer and the backup log buffer.

- `BackupMaxWriteSize`

  The maximum size of blocks written to disk. This applies for both the backup data buffer and the backup log buffer.

In addition, `CompressedBackup` causes `NDB` to use compression when creating and writing to backup files.

More detailed information about these parameters can be found in Backup Parameters.

You can also set a location for the backup files using the `BackupDataDir` configuration parameter. The default is `FileSystemPath/BACKUP/BACKUP-backup_id`.

18.5.8.4 NDB Cluster Backup Troubleshooting

If an error code is returned when issuing a backup request, the most likely cause is insufficient memory or disk space. You should check that there is enough memory allocated for the backup.

Important

If you have set `BackupDataBufferSize` and `BackupLogBufferSize` and their sum is greater than 4MB, then you must also set `BackupMemory` as well.

You should also make sure that there is sufficient space on the hard drive partition of the backup target.

`NDB` does not support repeatable reads, which can cause problems with the restoration process. Although the backup process is “hot”, restoring an NDB Cluster from backup is not a 100% “hot” process. This is due to the fact that, for the duration of the restore process, running transactions get nonrepeatable reads from the restored data. This means that the state of the data is inconsistent while the restore is in progress.

18.5.9 MySQL Server Usage for NDB Cluster

`mysqld` is the traditional MySQL server process. To be used with NDB Cluster, `mysqld` needs to be built with support for the `NDB` storage engine, as it is in the precompiled binaries available from https://dev.mysql.com/downloads/. If you build MySQL from source, you must invoke `CMake` with the `-DWITH_NDBCLUSTER=1` option to include support for `NDB`.

For more information about compiling NDB Cluster from source, see Section 18.2.1.4, “Building NDB Cluster from Source on Linux”, and Section 18.2.2.2, “Compiling and Installing NDB Cluster from Source on Windows”.

(For information about `mysqld` options and variables, in addition to those discussed in this section, which are relevant to NDB Cluster, see Section 18.3.3.8, “MySQL Server Options and Variables for NDB Cluster”.)

If the `mysqld` binary has been built with Cluster support, the `NDBCLUSTER` storage engine is still disabled by default. You can use either of two possible options to enable this engine:

- Use `--ndbcluster` as a startup option on the command line when starting `mysqld`.
- Insert a line containing `ndbcluster` in the `[mysqld]` section of your `my.cnf` file.

An easy way to verify that your server is running with the `NDBCLUSTER` storage engine enabled is to issue the `SHOW ENGINES` statement in the MySQL Monitor (`mysql`). You should see the value `YES`

as the `Support` value in the row for `NDBCLUSTER`. If you see `NO` in this row or if there is no such row displayed in the output, you are not running an `NDB`-enabled version of MySQL. If you see `DISABLED` in this row, you need to enable it in either one of the two ways just described.

To read cluster configuration data, the MySQL server requires at a minimum three pieces of information:

- The MySQL server's own cluster node ID
- The host name or IP address for the management server
- The number of the TCP/IP port on which it can connect to the management server

Node IDs can be allocated dynamically, so it is not strictly necessary to specify them explicitly.

The `mysqld` parameter `ndb-connectstring` is used to specify the connection string either on the command line when starting `mysqld` or in `my.cnf`. The connection string contains the host name or IP address where the management server can be found, as well as the TCP/IP port it uses.

In the following example, `ndb_mgmd.mysql.com` is the host where the management server resides, and the management server listens for cluster messages on port 1186:

```
shell> mysqld --ndbcluster --ndb-connectstring=ndb_mgmd.mysql.com:1186
```

See Section 18.3.3.3, “NDB Cluster Connection Strings”, for more information on connection strings.

Given this information, the MySQL server can become a full participant in the cluster. (We often refer to a `mysqld` process running in this manner as an SQL node.) It is fully aware of all cluster data nodes as well as their status, and establishes connections to all data nodes. In this case, it is able to use any data node as a transaction coordinator and to read and update node data.

You can see in the `mysql` client whether a MySQL server is connected to the cluster using `SHOW PROCESSLIST`. If the MySQL server is connected to the cluster, and you have the `PROCESS` privilege, then the first row of the output is as shown here:

```
mysql> SHOW PROCESSLIST \G
*************************** 1. row ***************************
     Id: 1
   User: system user
   Host:
     db:
Command: Daemon
   Time: 1
  State: Waiting for event from ndbcluster
   Info: NULL
```

Important

To participate in an NDB Cluster, the `mysqld` process must be started with *both* the options `--ndbcluster` and `--ndb-connectstring` (or their equivalents in `my.cnf`). If `mysqld` is started with only the `--ndbcluster` option, or if it is unable to contact the cluster, it is not possible to work with `NDB` tables, *nor is it possible to create any new tables regardless of storage engine*. The latter restriction is a safety measure intended to prevent the creation of tables having the same names as `NDB` tables while the SQL node is not connected to the cluster. If you wish to create tables using a different storage engine while the `mysqld` process is not participating in an NDB Cluster, you must restart the server *without* the `--ndbcluster` option.

18.5.10 NDB Cluster Disk Data Tables

It is possible to store the nonindexed columns of `NDB` tables on disk, rather than in RAM.

As part of implementing NDB Cluster Disk Data work, a number of improvements were made in NDB Cluster for the efficient handling of very large amounts (terabytes) of data during node recovery and restart. These include a “no-steal” algorithm for synchronizing a starting node with very large data sets. For more information, see the paper *Recovery Principles of NDB Cluster 5.1*, by NDB Cluster developers Mikael Ronström and Jonas Oreland.

NDB Cluster Disk Data performance can be influenced by a number of configuration parameters. For information about these parameters and their effects, see *NDB Cluster Disk Data configuration parameters* and *NDB Cluster Disk Data storage and `GCP Stop` errors*

The performance of an NDB Cluster that uses Disk Data storage can also be greatly improved by separating data node file systems from undo log files and tablespace data files, which can be done using symbolic links. For more information, see Section 18.5.10.2, “Using Symbolic Links with Disk Data Objects”.

18.5.10.1 NDB Cluster Disk Data Objects

NDB Cluster Disk Data storage is implemented using a number of *Disk Data objects*. These include the following:

- *Tablespaces* act as containers for other Disk Data objects.
- *Undo log files* undo information required for rolling back transactions.
- One or more undo log files are assigned to a *log file group*, which is then assigned to a tablespace.
- *Data files* store Disk Data table data. A data file is assigned directly to a tablespace.

Undo log files and data files are actual files in the file system of each data node; by default they are placed in `ndb_node_id_fs` in the `DataDir` specified in the NDB Cluster `config.ini` file, and where `node_id` is the data node's node ID. It is possible to place these elsewhere by specifying either an absolute or relative path as part of the filename when creating the undo log or data file. Statements that create these files are shown later in this section.

NDB Cluster tablespaces and log file groups are not implemented as files.

Important

Although not all Disk Data objects are implemented as files, they all share the same namespace. This means that *each Disk Data object* must be uniquely named (and not merely each Disk Data object of a given type). For example, you cannot have a tablespace and a log file group both named `dd1`.

Assuming that you have already set up an NDB Cluster with all nodes (including management and SQL nodes), the basic steps for creating an NDB Cluster table on disk are as follows:

1. Create a log file group, and assign one or more undo log files to it (an undo log file is also sometimes referred to as an *undofile*).

   Note

   Undo log files are necessary only for Disk Data tables; they are not used for `NDBCLUSTER` tables that are stored only in memory.

2. Create a tablespace; assign the log file group, as well as one or more data files, to the tablespace.
3. Create a Disk Data table that uses this tablespace for data storage.

Each of these tasks can be accomplished using SQL statements in the `mysql` client or other MySQL client application, as shown in the example that follows.

1. We create a log file group named `lg_1` using `CREATE LOGFILE GROUP`. This log file group is to be made up of two undo log files, which we name `undo_1.log` and `undo_2.log`, whose initial sizes are 16 MB and 12 MB, respectively. (The default initial size for an undo log file is 128 MB.) Optionally, you can also specify a size for the log file group's undo buffer, or permit it to assume the default value of 8 MB. In this example, we set the UNDO buffer's size at 2 MB. A log file group must be created with an undo log file; so we add `undo_1.log` to `lg_1` in this `CREATE LOGFILE GROUP` statement:

```
CREATE LOGFILE GROUP lg_1
    ADD UNDOFILE 'undo_1.log'
    INITIAL_SIZE 16M
    UNDO_BUFFER_SIZE 2M
    ENGINE NDBCLUSTER;
```

To add `undo_2.log` to the log file group, use the following `ALTER LOGFILE GROUP` statement:

```
ALTER LOGFILE GROUP lg_1
    ADD UNDOFILE 'undo_2.log'
    INITIAL_SIZE 12M
    ENGINE NDBCLUSTER;
```

Some items of note:

- The `.log` file extension used here is not required. We use it merely to make the log files easily recognisable.
- Every `CREATE LOGFILE GROUP` and `ALTER LOGFILE GROUP` statement must include an `ENGINE` option. The only permitted values for this option are `NDBCLUSTER` and `NDB`.

Important

There can exist at most one log file group in the same NDB Cluster at any given time.

- When you add an undo log file to a log file group using `ADD UNDOFILE 'filename'`, a file with the name *`filename`* is created in the `ndb_node_id_fs` directory within the `DataDir` of each data node in the cluster, where *`node_id`* is the node ID of the data node. Each undo log file is of the size specified in the SQL statement. For example, if an NDB Cluster has 4 data nodes, then the `ALTER LOGFILE GROUP` statement just shown creates 4 undo log files, 1 each on in the data directory of each of the 4 data nodes; each of these files is named `undo_2.log` and each file is 12 MB in size.
- `UNDO_BUFFER_SIZE` is limited by the amount of system memory available.
- For more information about the `CREATE LOGFILE GROUP` statement, see Section 13.1.14, “CREATE LOGFILE GROUP Statement”. For more information about `ALTER LOGFILE GROUP`, see Section 13.1.4, “ALTER LOGFILE GROUP Statement”.

2. Now we can create a tablespace, which contains files to be used by NDB Cluster Disk Data tables for storing their data. A tablespace is also associated with a particular log file group. When creating a new tablespace, you must specify the log file group which it is to use for undo logging; you must also specify a data file. You can add more data files to the tablespace after the tablespace is created; it is also possible to drop data files from a tablespace (an example of dropping data files is provided later in this section).

   Assume that we wish to create a tablespace named `ts_1` which uses `lg_1` as its log file group. This tablespace is to contain two data files named `data_1.dat` and `data_2.dat`, whose initial sizes are 32 MB and 48 MB, respectively. (The default value for `INITIAL_SIZE` is 128 MB.) We can do this using two SQL statements, as shown here:

```
CREATE TABLESPACE ts_1
    ADD DATAFILE 'data_1.dat'
    USE LOGFILE GROUP lg_1
```

```
    INITIAL_SIZE 32M
    ENGINE NDBCLUSTER;

ALTER TABLESPACE ts_1
    ADD DATAFILE 'data_2.dat'
    INITIAL_SIZE 48M
    ENGINE NDBCLUSTER;
```

The `CREATE TABLESPACE` statement creates a tablespace `ts_1` with the data file `data_1.dat`, and associates `ts_1` with log file group `lg_1`. The `ALTER TABLESPACE` adds the second data file (`data_2.dat`).

Some items of note:

- As is the case with the `.log` file extension used in this example for undo log files, there is no special significance for the `.dat` file extension; it is used merely for easy recognition of data files.
- When you add a data file to a tablespace using `ADD DATAFILE 'filename'`, a file with the name *`filename`* is created in the `ndb_node_id_fs` directory within the `DataDir` of each data node in the cluster, where *`node_id`* is the node ID of the data node. Each data file is of the size specified in the SQL statement. For example, if an NDB Cluster has 4 data nodes, then the `ALTER TABLESPACE` statement just shown creates 4 data files, 1 each in the data directory of each of the 4 data nodes; each of these files is named `data_2.dat` and each file is 48 MB in size.
- All `CREATE TABLESPACE` and `ALTER TABLESPACE` statements must contain an `ENGINE` clause; only tables using the same storage engine as the tablespace can be created in the tablespace. For NDB Cluster tablespaces, the only permitted values for this option are `NDBCLUSTER` and `NDB`.
- For more information about the `CREATE TABLESPACE` and `ALTER TABLESPACE` statements, see Section 13.1.18, “CREATE TABLESPACE Statement”, and Section 13.1.8, “ALTER TABLESPACE Statement”.

3. Now it is possible to create a table whose nonindexed columns are stored on disk in the tablespace `ts_1`:

```
CREATE TABLE dt_1 (
    member_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    last_name VARCHAR(50) NOT NULL,
    first_name VARCHAR(50) NOT NULL,
    dob DATE NOT NULL,
    joined DATE NOT NULL,
    INDEX(last_name, first_name)
    )
    TABLESPACE ts_1 STORAGE DISK
    ENGINE NDBCLUSTER;
```

The `TABLESPACE ... STORAGE DISK` option tells the `NDBCLUSTER` storage engine to use tablespace `ts_1` for disk data storage.

Once table `ts_1` has been created as shown, you can perform `INSERT`, `SELECT`, `UPDATE`, and `DELETE` statements on it just as you would with any other MySQL table.

It is also possible to specify whether an individual column is stored on disk or in memory by using a `STORAGE` clause as part of the column's definition in a `CREATE TABLE` or `ALTER TABLE` statement. `STORAGE DISK` causes the column to be stored on disk, and `STORAGE MEMORY` causes in-memory storage to be used. See Section 13.1.17, “CREATE TABLE Statement”, for more information.

Indexing of columns implicitly stored on disk. For table `dt_1` as defined in the example just shown, only the `dob` and `joined` columns are stored on disk. This is because there are indexes on the `id`, `last_name`, and `first_name` columns, and so data belonging to these columns is stored in

RAM. Only nonindexed columns can be held on disk; indexes and indexed column data continue to be stored in memory. This tradeoff between the use of indexes and conservation of RAM is something you must keep in mind as you design Disk Data tables.

You cannot add an index to a column that has been explicitly declared `STORAGE DISK`, without first changing its storage type to `MEMORY`; any attempt to do so fails with an error. A column which *implicitly* uses disk storage can be indexed; when this is done, the column's storage type is changed to `MEMORY` automatically. By “implicitly”, we mean a column whose storage type is not declared, but which is which inherited from the parent table. In the following CREATE TABLE statement (using the tablespace `ts_1` defined previously), columns `c2` and `c3` use disk storage implicitly:

```
mysql> CREATE TABLE ti (
    ->     c1 INT PRIMARY KEY,
    ->     c2 INT,
    ->     c3 INT,
    ->     c4 INT
    -> )
    ->     STORAGE DISK
    ->     TABLESPACE ts_1
    ->     ENGINE NDBCLUSTER;
Query OK, 0 rows affected (1.31 sec)
```

Because `c2`, `c3`, and `c4` are themselves not declared with `STORAGE DISK`, it is possible to index them. Here, we add indexes to `c2` and `c3`, using, respectively, `CREATE INDEX` and `ALTER TABLE`:

```
mysql> CREATE INDEX i1 ON ti(c2);
Query OK, 0 rows affected (2.72 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE ti ADD INDEX i2(c3);
Query OK, 0 rows affected (0.92 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

`SHOW CREATE TABLE` confirms that the indexes were added.

```
mysql> SHOW CREATE TABLE ti\G
*************************** 1. row ***************************
       Table: ti
Create Table: CREATE TABLE `ti` (
  `c1` int(11) NOT NULL,
  `c2` int(11) DEFAULT NULL,
  `c3` int(11) DEFAULT NULL,
  `c4` int(11) DEFAULT NULL,
  PRIMARY KEY (`c1`),
  KEY `i1` (`c2`),
  KEY `i2` (`c3`)
) /*!50100 TABLESPACE `ts_1` STORAGE DISK */ ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

You can see using `ndb_desc` that the indexed columns (emphasized text) now use in-memory rather than on-disk storage:

```
shell> ./ndb_desc -d test t1
-- t1 --
Version: 33554433
Fragment type: HashMapPartition
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 4
Number of primary keys: 1
Length of frm data: 317
Max Rows: 0
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
PartitionCount: 4
FragmentCount: 4
```

```
PartitionBalance: FOR_RP_BY_LDM
ExtraRowGciBits: 0
ExtraRowAuthorBits: 0
TableStatus: Retrieved
Table options:
HashMap: DEFAULT-HASHMAP-3840-4
-- Attributes --
c1 Int PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY
c2 Int NULL AT=FIXED ST=MEMORY
c3 Int NULL AT=FIXED ST=MEMORY
c4 Int NULL AT=FIXED ST=DISK
-- Indexes --
PRIMARY KEY(c1) - UniqueHashIndex
i2(c3) - OrderedIndex
PRIMARY(c1) - OrderedIndex
i1(c2) - OrderedIndex

NDBT_ProgramExit: 0 - OK
```

Performance note. The performance of a cluster using Disk Data storage is greatly improved if Disk Data files are kept on a separate physical disk from the data node file system. This must be done for each data node in the cluster to derive any noticeable benefit.

You may use absolute and relative file system paths with `ADD UNDOFILE` and `ADD DATAFILE`. Relative paths are calculated relative to the data node's data directory. You may also use symbolic links; see Section 18.5.10.2, “Using Symbolic Links with Disk Data Objects”, for more information and examples.

A log file group, a tablespace, and any Disk Data tables using these must be created in a particular order. The same is true for dropping any of these objects:

- A log file group cannot be dropped as long as any tablespaces are using it.
- A tablespace cannot be dropped as long as it contains any data files.
- You cannot drop any data files from a tablespace as long as there remain any tables which are using the tablespace.
- It is not possible to drop files created in association with a different tablespace than the one with which the files were created. (Bug #20053)

For example, to drop all the objects created so far in this section, you would use the following statements:

```
mysql> DROP TABLE dt_1;

mysql> ALTER TABLESPACE ts_1
    -> DROP DATAFILE 'data_2.dat'
    -> ENGINE NDBCLUSTER;

mysql> ALTER TABLESPACE ts_1
    -> DROP DATAFILE 'data_1.dat'
    -> ENGINE NDBCLUSTER;

mysql> DROP TABLESPACE ts_1
    -> ENGINE NDBCLUSTER;

mysql> DROP LOGFILE GROUP lg_1
    -> ENGINE NDBCLUSTER;
```

These statements must be performed in the order shown, except that the two `ALTER TABLESPACE ... DROP DATAFILE` statements may be executed in either order.

You can obtain information about data files used by Disk Data tables by querying the `FILES` table in the `INFORMATION_SCHEMA` database. An extra “`NULL` row” provides additional information about undo log files. For more information and examples, see Section 21.31.1, “The INFORMATION_SCHEMA FILES Table”.

18.5.10.2 Using Symbolic Links with Disk Data Objects

The performance of an NDB Cluster that uses Disk Data storage can be greatly improved by separating data node file systems from undo log files and tablespace data files and placing these on different disks. In early versions of NDB Cluster, there was no direct support for this in NDB Cluster, and it was necessary to achieve this separation using symbolic links as described later in this section. NDB Cluster 7.3 and later supports the data node configuration parameters `FileSystemPathDD`, `FileSystemPathDataFiles`, and `FileSystemPathUndoFiles`, which make the use of symbolic links for this purpose unnecessary. For more information about these parameters, see Disk Data file system parameters.

The procedure described in the remainder of this section is of historical interest only.

Each data node in the cluster creates a file system in the directory named `ndb_node_id_fs` under the data node's `DataDir` as defined in the `config.ini` file. In this example, we assume that each data node host has 3 disks, aliased as `/data0`, `/data1`, and `/data2`, and that the cluster's `config.ini` includes the following:

```
[ndbd default]
DataDir= /data0
```

Our objective is to place all Disk Data log files in `/data1`, and all Disk Data data files in `/data2`, on each data node host.

Note

In this example, we assume that the cluster's data node hosts are all using Linux operating systems. For other platforms, you may need to substitute you operating system's commands for those shown here.

To accomplish this, perform the following steps:

- Under the data node file system create symbolic links pointing to the other drives:

```
shell> cd /data0/ndb_2_fs
shell> ls
D1  D10  D11  D2  D8  D9  LCP
shell> ln -s /data0 dnlogs
shell> ln -s /data1 dndata
```

  You should now have two symbolic links:

```
shell> ls -l --hide=D*
lrwxrwxrwx 1 user group   30 2007-03-19 13:58 dndata -> /data1
lrwxrwxrwx 1 user group   30 2007-03-19 13:59 dnlogs -> /data2
```

  We show this only for the data node with node ID 2; however, you must do this for *each* data node.

- Now, in the `mysql` client, create a log file group and tablespace using the symbolic links, as shown here:

```
mysql> CREATE LOGFILE GROUP lg1
    ->    ADD UNDOFILE 'dnlogs/undo1.log'
    ->    INITIAL_SIZE 150M
    ->    UNDO_BUFFER_SIZE = 1M
    ->    ENGINE=NDBCLUSTER;

mysql> CREATE TABLESPACE ts1
    ->    ADD DATAFILE 'dndata/data1.log'
    ->    USE LOGFILE GROUP lg1
    ->    INITIAL_SIZE 1G
    ->    ENGINE=NDBCLUSTER;
```

  Verify that the files were created and placed correctly as shown here:

```
shell> cd /data1
shell> ls -l
total 2099304
-rw-rw-r--  1 user group 157286400 2007-03-19 14:02 undo1.dat

shell> cd /data2
shell> ls -l
total 2099304
-rw-rw-r--  1 user group 1073741824 2007-03-19 14:02 data1.dat
```

- If you are running multiple data nodes on one host, you must take care to avoid having them try to use the same space for Disk Data files. You can make this easier by creating a symbolic link in each data node file system. Suppose you are using `/data0` for both data node file systems, but you wish to have the Disk Data files for both nodes on `/data1`. In this case, you can do something similar to what is shown here:

```
shell> cd /data0
shell> ln -s /data1/dn2 ndb_2_fs/dd
shell> ln -s /data1/dn3 ndb_3_fs/dd
shell> ls -l --hide=D* ndb_2_fs
lrwxrwxrwx 1 user group   30 2007-03-19 14:22 dd -> /data1/dn2
shell> ls -l --hide=D* ndb_3_fs
lrwxrwxrwx 1 user group   30 2007-03-19 14:22 dd -> /data1/dn3
```

- Now you can create a logfile group and tablespace using the symbolic link, like this:

```
mysql> CREATE LOGFILE GROUP lg1
    ->    ADD UNDOFILE 'dd/undo1.log'
    ->    INITIAL_SIZE 150M
    ->    UNDO_BUFFER_SIZE = 1M
    ->    ENGINE=NDBCLUSTER;

mysql> CREATE TABLESPACE ts1
    ->    ADD DATAFILE 'dd/data1.log'
    ->    USE LOGFILE GROUP lg1
    ->    INITIAL_SIZE 1G
    ->    ENGINE=NDBCLUSTER;
```

Verify that the files were created and placed correctly as shown here:

```
shell> cd /data1
shell> ls
dn2        dn3
shell> ls dn2
undo1.log        data1.log
shell> ls dn3
undo1.log        data1.log
```

18.5.10.3 NDB Cluster Disk Data Storage Requirements

The following items apply to Disk Data storage requirements:

- Variable-length columns of Disk Data tables take up a fixed amount of space. For each row, this is equal to the space required to store the largest possible value for that column.

  For general information about calculating these values, see Section 11.6, “Data Type Storage Requirements”.

  You can obtain an estimate the amount of space available in data files and undo log files by querying the `INFORMATION_SCHEMA.FILES` table. For more information and examples, see Section 21.31.1, “The INFORMATION_SCHEMA FILES Table”.

Note

The `OPTIMIZE TABLE` statement does not have any effect on Disk Data tables.

- In a Disk Data table, the first 256 bytes of a `TEXT` or `BLOB` column are stored in memory; only the remainder is stored on disk.
- Each row in a Disk Data table uses 8 bytes in memory to point to the data stored on disk. This means that, in some cases, converting an in-memory column to the disk-based format can actually result in greater memory usage. For example, converting a `CHAR(4)` column from memory-based to disk-based format increases the amount of `DataMemory` used per row from 4 to 8 bytes.

Important

Starting the cluster with the `--initial` option does *not* remove Disk Data files. You must remove these manually prior to performing an initial restart of the cluster.

Performance of Disk Data tables can be improved by minimizing the number of disk seeks by making sure that `DiskPageBufferMemory` is of sufficient size. You can query the `diskpagebuffer` table to help determine whether the value for this parameter needs to be increased.

18.5.11 Online Operations with ALTER TABLE in NDB Cluster

MySQL NDB Cluster 7.5 supports online table schema changes using the standard `ALTER TABLE` syntax employed by the MySQL Server (`ALGORITHM=DEFAULT|INPLACE|COPY`), and described elsewhere.

Important

Online table schema changes as implemented in NDB Cluster 5.1 used a syntax specific to `NDB` that is deprecated in NDB 7.3 and 7.4, and no longer supported in NDB 7.5 or later. (This syntax is not supported by any other MySQL storage engine, including `InnoDB`.) MySQL NDB Cluster 7.3 and later support the standard `ALTER TABLE` syntax employed by the MySQL Server (`ALGORITHM=DEFAULT|INPLACE|COPY`), and described elsewhere in this section. For these reasons, you are strongly encouraged to convert any applications using the old `ONLINE` and `OFFLINE` syntax as soon as possible.

Operations that add and drop indexes on variable-width columns of `NDB` tables occur online. Online operations are noncopying; that is, they do not require that indexes be re-created. They do not lock the table being altered from access by other API nodes in an NDB Cluster (but see Limitations of NDB online operations, later in this section). Such operations do not require single user mode for `NDB` table alterations made in an NDB cluster with multiple API nodes; transactions can continue uninterrupted during online DDL operations.

`ALGORITHM=INPLACE` can be used to perform online `ADD COLUMN`, `ADD INDEX` (including `CREATE INDEX` statements), and `DROP INDEX` operations on `NDB` tables. Online renaming of `NDB` tables is also supported.

Currently you cannot add disk-based columns to `NDB` tables online. This means that, if you wish to add an in-memory column to an `NDB` table that uses a table-level `STORAGE DISK` option, you must declare the new column as using memory-based storage explicitly. For example—assuming that you have already created tablespace `ts1`—suppose that you create table `t1` as follows:

```
mysql> CREATE TABLE t1 (
     >     c1 INT NOT NULL PRIMARY KEY,
     >     c2 VARCHAR(30)
     >     )
     >     TABLESPACE ts1 STORAGE DISK
     >     ENGINE NDB;
Query OK, 0 rows affected (1.73 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

You can add a new in-memory column to this table online as shown here:

```
mysql> ALTER TABLE t1
     >     ADD COLUMN c3 INT COLUMN_FORMAT DYNAMIC STORAGE MEMORY,
```

```
    >     ALGORITHM=INPLACE;
Query OK, 0 rows affected (1.25 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

This statement fails if the `STORAGE MEMORY` option is omitted:

```
mysql> ALTER TABLE t1
    >     ADD COLUMN c4 INT COLUMN_FORMAT DYNAMIC,
    >     ALGORITHM=INPLACE;
ERROR 1846 (0A000): ALGORITHM=INPLACE is not supported. Reason:
Adding column(s) or add/reorganize partition not supported online. Try
ALGORITHM=COPY.
```

If you omit the `COLUMN_FORMAT DYNAMIC` option, the dynamic column format is employed automatically, but a warning is issued, as shown here:

```
mysql> ALTER ONLINE TABLE t1 ADD COLUMN c4 INT STORAGE MEMORY;
Query OK, 0 rows affected, 1 warning (1.17 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1478
Message: DYNAMIC column c4 with STORAGE DISK is not supported, column will
become FIXED


mysql> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `c1` int(11) NOT NULL,
  `c2` varchar(30) DEFAULT NULL,
  `c3` int(11) /*!50606 STORAGE MEMORY */ /*!50606 COLUMN_FORMAT DYNAMIC */ DEFAULT NULL,
  `c4` int(11) /*!50606 STORAGE MEMORY */ DEFAULT NULL,
  PRIMARY KEY (`c1`)
) /*!50606 TABLESPACE ts_1 STORAGE DISK */ ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.03 sec)
```

Note

The `STORAGE` and `COLUMN_FORMAT` keywords are supported only in NDB Cluster; in any other version of MySQL, attempting to use either of these keywords in a `CREATE TABLE` or `ALTER TABLE` statement results in an error.

It is also possible to use the statement `ALTER TABLE ... REORGANIZE PARTITION, ALGORITHM=INPLACE` with no *`partition_names`* `INTO (`*`partition_definitions`*`)` option on `NDB` tables. This can be used to redistribute NDB Cluster data among new data nodes that have been added to the cluster online. This does *not* perform any defragmentation, which requires an `OPTIMIZE TABLE` or null `ALTER TABLE` statement. For more information, see Section 18.5.7, “Adding NDB Cluster Data Nodes Online”.

Limitations of NDB online operations

Online `DROP COLUMN` operations are not supported.

Online `ALTER TABLE`, `CREATE INDEX`, or `DROP INDEX` statements that add columns or add or drop indexes are subject to the following limitations:

- A given online `ALTER TABLE` can use only one of `ADD COLUMN`, `ADD INDEX`, or `DROP INDEX`. One or more columns can be added online in a single statement; only one index may be created or dropped online in a single statement.
- The table being altered is not locked with respect to API nodes other than the one on which an online `ALTER TABLE ADD COLUMN`, `ADD INDEX`, or `DROP INDEX` operation (or `CREATE INDEX` or `DROP INDEX` statement) is run. However, the table is locked against any other operations originating on the *same* API node while the online operation is being executed.

- The table to be altered must have an explicit primary key; the hidden primary key created by the NDB storage engine is not sufficient for this purpose.
- The storage engine used by the table cannot be changed online.
- The tablespace used by the table cannot be changed online. (Bug #99269, Bug #31180526)
- When used with NDB Cluster Disk Data tables, it is not possible to change the storage type (DISK or MEMORY) of a column online. This means, that when you add or drop an index in such a way that the operation would be performed online, and you want the storage type of the column or columns to be changed, you must use ALGORITHM=COPY in the statement that adds or drops the index.

Columns to be added online cannot use the BLOB or TEXT type, and must meet the following criteria:

- The columns must be dynamic; that is, it must be possible to create them using COLUMN_FORMAT DYNAMIC. If you omit the COLUMN_FORMAT DYNAMIC option, the dynamic column format is employed automatically.
- The columns must permit NULL values and not have any explicit default value other than NULL. Columns added online are automatically created as DEFAULT NULL, as can be seen here:

```
mysql> CREATE TABLE t2 (
     >     c1 INT NOT NULL AUTO_INCREMENT PRIMARY KEY
     >     ) ENGINE=NDB;
Query OK, 0 rows affected (1.44 sec)

mysql> ALTER TABLE t2
     >     ADD COLUMN c2 INT,
     >     ADD COLUMN c3 INT,
     >     ALGORITHM=INPLACE;
Query OK, 0 rows affected, 2 warnings (0.93 sec)

mysql> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t2` (
  `c1` int(11) NOT NULL AUTO_INCREMENT,
  `c2` int(11) DEFAULT NULL,
  `c3` int(11) DEFAULT NULL,
  PRIMARY KEY (`c1`)
) ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

- The columns must be added following any existing columns. If you attempt to add a column online before any existing columns or using the FIRST keyword, the statement fails with an error.
- Existing table columns cannot be reordered online.

For online ALTER TABLE operations on NDB tables, fixed-format columns are converted to dynamic when they are added online, or when indexes are created or dropped online, as shown here (repeating the CREATE TABLE and ALTER TABLE statements just shown for the sake of clarity):

```
mysql> CREATE TABLE t2 (
     >     c1 INT NOT NULL AUTO_INCREMENT PRIMARY KEY
     >     ) ENGINE=NDB;
Query OK, 0 rows affected (1.44 sec)

mysql> ALTER TABLE t2
     >     ADD COLUMN c2 INT,
     >     ADD COLUMN c3 INT,
     >     ALGORITHM=INPLACE;
Query OK, 0 rows affected, 2 warnings (0.93 sec)

mysql> SHOW WARNINGS;
*************************** 1. row ***************************
  Level: Warning
   Code: 1478
Message: Converted FIXED field 'c2' to DYNAMIC to enable online ADD COLUMN
*************************** 2. row ***************************
```

```
  Level: Warning
   Code: 1478
Message: Converted FIXED field 'c3' to DYNAMIC to enable online ADD COLUMN
2 rows in set (0.00 sec)
```

Only the column or columns to be added online must be dynamic. Existing columns need not be; this includes the table's primary key, which may also be `FIXED`, as shown here:

```
mysql> CREATE TABLE t3 (
     >     c1 INT NOT NULL AUTO_INCREMENT PRIMARY KEY COLUMN_FORMAT FIXED
     >     ) ENGINE=NDB;
Query OK, 0 rows affected (2.10 sec)

mysql> ALTER TABLE t3 ADD COLUMN c2 INT, ALGORITHM=INPLACE;
Query OK, 0 rows affected, 1 warning (0.78 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW WARNINGS;
*************************** 1. row ***************************
  Level: Warning
   Code: 1478
Message: Converted FIXED field 'c2' to DYNAMIC to enable online ADD COLUMN
1 row in set (0.00 sec)
```

Columns are not converted from `FIXED` to `DYNAMIC` column format by renaming operations. For more information about `COLUMN_FORMAT`, see Section 13.1.17, “CREATE TABLE Statement”.

The `KEY`, `CONSTRAINT`, and `IGNORE` keywords are supported in `ALTER TABLE` statements using `ALGORITHM=INPLACE`.

Beginning with NDB Cluster 7.3.18 and 7.4.16, setting `MAX_ROWS` to 0 using an online `ALTER TABLE` statement is disallowed. You must use a copying `ALTER TABLE` to perform this operation. (Bug #21960004)

18.5.12 Distributed Privileges Using Shared Grant Tables

NDB Cluster supports distribution of MySQL users and privileges across all SQL nodes in an NDB Cluster. This support is not enabled by default; you should follow the procedure outlined in this section in order to do so.

Normally, each MySQL server's user privilege tables in the `mysql` database must use the `MyISAM` storage engine, which means that a user account and its associated privileges created on one SQL node are not available on the cluster's other SQL nodes. An SQL file `ndb_dist_priv.sql` is provided with NDB Cluster 7.3 and later distributions. This file can be found in the `share` directory in the MySQL installation directory.

The first step in enabling distributed privileges is to load this script into a MySQL Server that functions as an SQL node (which we refer to after this as the *target* SQL node or MySQL Server). You can do this by executing the following command from the system shell on the target SQL node after changing to its MySQL installation directory (where *`options`* stands for any additional options needed to connect to this SQL node):

```
shell> mysql options -uroot < share/ndb_dist_priv.sql
```

Importing `ndb_dist_priv.sql` creates a number of stored routines (six stored procedures and one stored function) in the `mysql` database on the target SQL node. After connecting to the SQL node in the `mysql` client (as the MySQL `root` user), you can verify that these were created as shown here:

```
mysql> SELECT ROUTINE_NAME, ROUTINE_SCHEMA, ROUTINE_TYPE
    ->     FROM INFORMATION_SCHEMA.ROUTINES
    ->     WHERE ROUTINE_NAME LIKE 'mysql_cluster%'
    ->     ORDER BY ROUTINE_TYPE;
+-----------------------------------------+----------------+--------------+
| ROUTINE_NAME                            | ROUTINE_SCHEMA | ROUTINE_TYPE |
+-----------------------------------------+----------------+--------------+
| mysql_cluster_privileges_are_distributed | mysql          | FUNCTION     |
| mysql_cluster_backup_privileges          | mysql          | PROCEDURE    |
```

```
| mysql_cluster_move_grant_tables             | mysql          | PROCEDURE    |
| mysql_cluster_move_privileges               | mysql          | PROCEDURE    |
| mysql_cluster_restore_local_privileges      | mysql          | PROCEDURE    |
| mysql_cluster_restore_privileges            | mysql          | PROCEDURE    |
| mysql_cluster_restore_privileges_from_local | mysql          | PROCEDURE    |
+---------------------------------------------+----------------+--------------+
7 rows in set (0.01 sec)
```

The stored procedure named `mysql_cluster_move_privileges` creates backup copies of the existing privilege tables, then converts them to `NDB`.

`mysql_cluster_move_privileges` performs the backup and conversion in two steps. The first step is to call `mysql_cluster_backup_privileges`, which creates two sets of copies in the `mysql` database:

- A set of local copies that use the `MyISAM` storage engine. Their names are generated by adding the suffix `_backup` to the original privilege table names.
- A set of distributed copies that use the `NDBCLUSTER` storage engine. These tables are named by prefixing `ndb_` and appending `_backup` to the names of the original tables.

After the copies are created, `mysql_cluster_move_privileges` invokes `mysql_cluster_move_grant_tables`, which contains the `ALTER TABLE ... ENGINE = NDB` statements that convert the mysql system tables to `NDB`.

Normally, you should not invoke either `mysql_cluster_backup_privileges` or `mysql_cluster_move_grant_tables` manually; these stored procedures are intended only for use by `mysql_cluster_move_privileges`.

Although the original privilege tables are backed up automatically, it is always a good idea to create backups manually of the existing privilege tables on all affected SQL nodes before proceeding. You can do this using `mysqldump` in a manner similar to what is shown here:

```
shell> mysqldump options -uroot \
    mysql user db tables_priv columns_priv procs_priv proxies_priv > backup_file
```

To perform the conversion, you must be connected to the target SQL node using the `mysql` client (again, as the MySQL `root` user). Invoke the stored procedure like this:

```
mysql> CALL mysql.mysql_cluster_move_privileges();
Query OK, 0 rows affected (22.32 sec)
```

Depending on the number of rows in the privilege tables, this procedure may take some time to execute. If some of the privilege tables are empty, you may see one or more `No data - zero rows fetched, selected, or processed` warnings when `mysql_cluster_move_privileges` returns. In such cases, the warnings may be safely ignored. To verify that the conversion was successful, you can use the stored function `mysql_cluster_privileges_are_distributed` as shown here:

```
mysql> SELECT CONCAT(
    ->    'Conversion ',
    ->    IF(mysql.mysql_cluster_privileges_are_distributed(), 'succeeded', 'failed'),
    ->    '.')
    ->    AS Result;
+-----------------------+
| Result                |
+-----------------------+
| Conversion succeeded. |
+-----------------------+
1 row in set (0.00 sec)
```

`mysql_cluster_privileges_are_distributed` checks for the existence of the distributed privilege tables and returns `1` if all of the privilege tables are distributed; otherwise, it returns `0`.

You can verify that the backups have been created using a query such as this one:

```
mysql> SELECT TABLE_NAME, ENGINE FROM INFORMATION_SCHEMA.TABLES
    ->     WHERE TABLE_SCHEMA = 'mysql' AND TABLE_NAME LIKE '%backup'
    ->     ORDER BY ENGINE;
+-------------------------+------------+
| TABLE_NAME              | ENGINE     |
+-------------------------+------------+
| db_backup               | MyISAM     |
| user_backup             | MyISAM     |
| columns_priv_backup     | MyISAM     |
| tables_priv_backup      | MyISAM     |
| proxies_priv_backup     | MyISAM     |
| procs_priv_backup       | MyISAM     |
| ndb_user_backup         | ndbcluster |
| ndb_tables_priv_backup  | ndbcluster |
| ndb_proxies_priv_backup | ndbcluster |
| ndb_procs_priv_backup   | ndbcluster |
| ndb_db_backup           | ndbcluster |
| ndb_columns_priv_backup | ndbcluster |
+-------------------------+------------+
12 rows in set (0.00 sec)
```

Once the conversion to distributed privileges has been made, any time a MySQL user account is created, dropped, or has its privileges updated on any SQL node, the changes take effect immediately on all other MySQL servers attached to the cluster. Once privileges are distributed, any new MySQL Servers that connect to the cluster automatically participate in the distribution.

Note

For clients connected to SQL nodes at the time that `mysql_cluster_move_privileges` is executed, you may need to execute `FLUSH PRIVILEGES` on those SQL nodes, or to disconnect and then reconnect the clients, in order for those clients to be able to see the changes in privileges.

All MySQL user privileges are distributed across all connected MySQL Servers. This includes any privileges associated with views and stored routines, even though distribution of views and stored routines themselves is not currently supported.

In the event that an SQL node becomes disconnected from the cluster while `mysql_cluster_move_privileges` is running, you must drop its privilege tables after reconnecting to the cluster, using a statement such as `DROP TABLE IF EXISTS mysql.user mysql.db mysql.tables_priv mysql.columns_priv mysql.procs_priv`. This causes the SQL node to use the shared privilege tables rather than its own local versions of them. This is not needed when connecting a new SQL node to the cluster for the first time.

In the event of an initial restart of the entire cluster (all data nodes shut down, then started again with `--initial`), the shared privilege tables are lost. If this happens, you can restore them using the original target SQL node either from the backups made by `mysql_cluster_move_privileges` or from a dump file created with `mysqldump`. If you need to use a new MySQL Server to perform the restoration, you should start it with `--skip-grant-tables` when connecting to the cluster for the first time; after this, you can restore the privilege tables locally, then distribute them again using `mysql_cluster_move_privileges`. After restoring and distributing the tables, you should restart this MySQL Server without the `--skip-grant-tables` option.

You can also restore the distributed tables using `ndb_restore --restore-privilege-tables` from a backup made using `START BACKUP` in the `ndb_mgm` client. (The `MyISAM` tables created by `mysql_cluster_move_privileges` are not backed up by the `START BACKUP` command.) `ndb_restore` does not restore the privilege tables by default; the `--restore-privilege-tables` option causes it to do so.

You can restore the SQL node's local privileges using either of two procedures. `mysql_cluster_restore_privileges` works as follows:

1. If copies of the `mysql.ndb_*_backup` tables are available, attempt to restore the system tables from these.

2. Otherwise, attempt to restore the system tables from the local backups named `*_backup` (without the `ndb_` prefix).

The other procedure, named `mysql_cluster_restore_local_privileges`, restores the system tables from the local backups only, without checking the `ndb_*` backups.

The system tables re-created by `mysql_cluster_restore_privileges` or `mysql_cluster_restore_local_privileges` use the MySQL server default storage engine; they are not shared or distributed in any way, and do not use NDB Cluster's `NDB` storage engine.

The additional stored procedure `mysql_cluster_restore_privileges_from_local` is intended for the use of `mysql_cluster_restore_privileges` and `mysql_cluster_restore_local_privileges`. It should not be invoked directly.

Important

Applications that access NDB Cluster data directly, including NDB API and ClusterJ applications, are not subject to the MySQL privilege system. This means that, once you have distributed the grant tables, they can be freely accessed by such applications, just as they can any other `NDB` tables. In particular, you should keep in mind that *NDB API and ClusterJ applications can read and write user names, host names, password hashes, and any other contents of the distributed grant tables without any restrictions*.

18.5.13 NDB API Statistics Counters and Variables

A number of types of statistical counters relating to actions performed by or affecting `Ndb` objects are available. Such actions include starting and closing (or aborting) transactions; primary key and unique key operations; table, range, and pruned scans; threads blocked while waiting for the completion of various operations; and data and events sent and received by `NDBCLUSTER`. The counters are incremented inside the NDB kernel whenever NDB API calls are made or data is sent to or received by the data nodes. `mysqld` exposes these counters as system status variables; their values can be read in the output of `SHOW STATUS`, or by querying the `INFORMATION_SCHEMA.SESSION_STATUS` or `INFORMATION_SCHEMA.GLOBAL_STATUS` table. By comparing the values before and after statements operating on `NDB` tables, you can observe the corresponding actions taken on the API level, and thus the cost of performing the statement.

You can list all of these status variables using the following `SHOW STATUS` statement:

```
mysql> SHOW STATUS LIKE 'ndb_api%';
+-----------------------------------------------+----------+
| Variable_name                                 | Value    |
+-----------------------------------------------+----------+
| Ndb_api_wait_exec_complete_count_session      | 0        |
| Ndb_api_wait_scan_result_count_session        | 0        |
| Ndb_api_wait_meta_request_count_session       | 0        |
| Ndb_api_wait_nanos_count_session              | 0        |
| Ndb_api_bytes_sent_count_session              | 0        |
| Ndb_api_bytes_received_count_session          | 0        |
| Ndb_api_trans_start_count_session             | 0        |
| Ndb_api_trans_commit_count_session            | 0        |
| Ndb_api_trans_abort_count_session             | 0        |
| Ndb_api_trans_close_count_session             | 0        |
| Ndb_api_pk_op_count_session                   | 0        |
| Ndb_api_uk_op_count_session                   | 0        |
| Ndb_api_table_scan_count_session              | 0        |
| Ndb_api_range_scan_count_session              | 0        |
| Ndb_api_pruned_scan_count_session             | 0        |
| Ndb_api_scan_batch_count_session              | 0        |
| Ndb_api_read_row_count_session                | 0        |
| Ndb_api_trans_local_read_row_count_session    | 0        |
| Ndb_api_event_data_count_injector             | 0        |
| Ndb_api_event_nondata_count_injector          | 0        |
| Ndb_api_event_bytes_count_injector            | 0        |
| Ndb_api_wait_exec_complete_count_slave        | 0        |
```

```
| Ndb_api_wait_scan_result_count_slave       | 0        |
| Ndb_api_wait_meta_request_count_slave      | 0        |
| Ndb_api_wait_nanos_count_slave             | 0        |
| Ndb_api_bytes_sent_count_slave             | 0        |
| Ndb_api_bytes_received_count_slave         | 0        |
| Ndb_api_trans_start_count_slave            | 0        |
| Ndb_api_trans_commit_count_slave           | 0        |
| Ndb_api_trans_abort_count_slave            | 0        |
| Ndb_api_trans_close_count_slave            | 0        |
| Ndb_api_pk_op_count_slave                  | 0        |
| Ndb_api_uk_op_count_slave                  | 0        |
| Ndb_api_table_scan_count_slave             | 0        |
| Ndb_api_range_scan_count_slave             | 0        |
| Ndb_api_pruned_scan_count_slave            | 0        |
| Ndb_api_scan_batch_count_slave             | 0        |
| Ndb_api_read_row_count_slave               | 0        |
| Ndb_api_trans_local_read_row_count_slave   | 0        |
| Ndb_api_wait_exec_complete_count           | 2        |
| Ndb_api_wait_scan_result_count             | 3        |
| Ndb_api_wait_meta_request_count            | 27       |
| Ndb_api_wait_nanos_count                   | 45612023 |
| Ndb_api_bytes_sent_count                   | 992      |
| Ndb_api_bytes_received_count               | 9640     |
| Ndb_api_trans_start_count                  | 2        |
| Ndb_api_trans_commit_count                 | 1        |
| Ndb_api_trans_abort_count                  | 0        |
| Ndb_api_trans_close_count                  | 2        |
| Ndb_api_pk_op_count                        | 1        |
| Ndb_api_uk_op_count                        | 0        |
| Ndb_api_table_scan_count                   | 1        |
| Ndb_api_range_scan_count                   | 0        |
| Ndb_api_pruned_scan_count                  | 0        |
| Ndb_api_scan_batch_count                   | 0        |
| Ndb_api_read_row_count                     | 1        |
| Ndb_api_trans_local_read_row_count         | 1        |
| Ndb_api_event_data_count                   | 0        |
| Ndb_api_event_nondata_count                | 0        |
| Ndb_api_event_bytes_count                  | 0        |
+--------------------------------------------+----------+
60 rows in set (0.02 sec)
```

These status variables are also available from the `SESSION_STATUS` and `GLOBAL_STATUS` tables of the `INFORMATION_SCHEMA` database, as shown here:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.SESSION_STATUS
    ->   WHERE VARIABLE_NAME LIKE 'ndb_api%';
+--------------------------------------------+----------------+
| VARIABLE_NAME                              | VARIABLE_VALUE |
+--------------------------------------------+----------------+
| NDB_API_WAIT_EXEC_COMPLETE_COUNT_SESSION   | 2              |
| NDB_API_WAIT_SCAN_RESULT_COUNT_SESSION     | 0              |
| NDB_API_WAIT_META_REQUEST_COUNT_SESSION    | 1              |
| NDB_API_WAIT_NANOS_COUNT_SESSION           | 8144375        |
| NDB_API_BYTES_SENT_COUNT_SESSION           | 68             |
| NDB_API_BYTES_RECEIVED_COUNT_SESSION       | 84             |
| NDB_API_TRANS_START_COUNT_SESSION          | 1              |
| NDB_API_TRANS_COMMIT_COUNT_SESSION         | 1              |
| NDB_API_TRANS_ABORT_COUNT_SESSION          | 0              |
| NDB_API_TRANS_CLOSE_COUNT_SESSION          | 1              |
| NDB_API_PK_OP_COUNT_SESSION                | 1              |
| NDB_API_UK_OP_COUNT_SESSION                | 0              |
| NDB_API_TABLE_SCAN_COUNT_SESSION           | 0              |
| NDB_API_RANGE_SCAN_COUNT_SESSION           | 0              |
| NDB_API_PRUNED_SCAN_COUNT_SESSION          | 0              |
| NDB_API_SCAN_BATCH_COUNT_SESSION           | 0              |
| NDB_API_READ_ROW_COUNT_SESSION             | 1              |
| NDB_API_TRANS_LOCAL_READ_ROW_COUNT_SESSION | 1              |
| NDB_API_EVENT_DATA_COUNT_INJECTOR          | 0              |
| NDB_API_EVENT_NONDATA_COUNT_INJECTOR       | 0              |
| NDB_API_EVENT_BYTES_COUNT_INJECTOR         | 0              |
| NDB_API_WAIT_EXEC_COMPLETE_COUNT_SLAVE     | 0              |
| NDB_API_WAIT_SCAN_RESULT_COUNT_SLAVE       | 0              |
```

```
| NDB_API_WAIT_META_REQUEST_COUNT_SLAVE      | 0              |
| NDB_API_WAIT_NANOS_COUNT_SLAVE             | 0              |
| NDB_API_BYTES_SENT_COUNT_SLAVE             | 0              |
| NDB_API_BYTES_RECEIVED_COUNT_SLAVE         | 0              |
| NDB_API_TRANS_START_COUNT_SLAVE            | 0              |
| NDB_API_TRANS_COMMIT_COUNT_SLAVE           | 0              |
| NDB_API_TRANS_ABORT_COUNT_SLAVE            | 0              |
| NDB_API_TRANS_CLOSE_COUNT_SLAVE            | 0              |
| NDB_API_PK_OP_COUNT_SLAVE                  | 0              |
| NDB_API_UK_OP_COUNT_SLAVE                  | 0              |
| NDB_API_TABLE_SCAN_COUNT_SLAVE             | 0              |
| NDB_API_RANGE_SCAN_COUNT_SLAVE             | 0              |
| NDB_API_PRUNED_SCAN_COUNT_SLAVE            | 0              |
| NDB_API_SCAN_BATCH_COUNT_SLAVE             | 0              |
| NDB_API_READ_ROW_COUNT_SLAVE               | 0              |
| NDB_API_TRANS_LOCAL_READ_ROW_COUNT_SLAVE   | 0              |
| NDB_API_WAIT_EXEC_COMPLETE_COUNT           | 4              |
| NDB_API_WAIT_SCAN_RESULT_COUNT             | 3              |
| NDB_API_WAIT_META_REQUEST_COUNT            | 28             |
| NDB_API_WAIT_NANOS_COUNT                   | 53756398       |
| NDB_API_BYTES_SENT_COUNT                   | 1060           |
| NDB_API_BYTES_RECEIVED_COUNT               | 9724           |
| NDB_API_TRANS_START_COUNT                  | 3              |
| NDB_API_TRANS_COMMIT_COUNT                 | 2              |
| NDB_API_TRANS_ABORT_COUNT                  | 0              |
| NDB_API_TRANS_CLOSE_COUNT                  | 3              |
| NDB_API_PK_OP_COUNT                        | 2              |
| NDB_API_UK_OP_COUNT                        | 0              |
| NDB_API_TABLE_SCAN_COUNT                   | 1              |
| NDB_API_RANGE_SCAN_COUNT                   | 0              |
| NDB_API_PRUNED_SCAN_COUNT                  | 0              |
| NDB_API_SCAN_BATCH_COUNT                   | 0              |
| NDB_API_READ_ROW_COUNT                     | 2              |
| NDB_API_TRANS_LOCAL_READ_ROW_COUNT         | 2              |
| NDB_API_EVENT_DATA_COUNT                   | 0              |
| NDB_API_EVENT_NONDATA_COUNT                | 0              |
| NDB_API_EVENT_BYTES_COUNT                  | 0              |
+--------------------------------------------+----------------+
60 rows in set (0.00 sec)

mysql> SELECT * FROM INFORMATION_SCHEMA.GLOBAL_STATUS
    ->     WHERE VARIABLE_NAME LIKE 'ndb_api%';
+--------------------------------------------+----------------+
| VARIABLE_NAME                              | VARIABLE_VALUE |
+--------------------------------------------+----------------+
| NDB_API_WAIT_EXEC_COMPLETE_COUNT_SESSION   | 2              |
| NDB_API_WAIT_SCAN_RESULT_COUNT_SESSION     | 0              |
| NDB_API_WAIT_META_REQUEST_COUNT_SESSION    | 1              |
| NDB_API_WAIT_NANOS_COUNT_SESSION           | 8144375        |
| NDB_API_BYTES_SENT_COUNT_SESSION           | 68             |
| NDB_API_BYTES_RECEIVED_COUNT_SESSION       | 84             |
| NDB_API_TRANS_START_COUNT_SESSION          | 1              |
| NDB_API_TRANS_COMMIT_COUNT_SESSION         | 1              |
| NDB_API_TRANS_ABORT_COUNT_SESSION          | 0              |
| NDB_API_TRANS_CLOSE_COUNT_SESSION          | 1              |
| NDB_API_PK_OP_COUNT_SESSION                | 1              |
| NDB_API_UK_OP_COUNT_SESSION                | 0              |
| NDB_API_TABLE_SCAN_COUNT_SESSION           | 0              |
| NDB_API_RANGE_SCAN_COUNT_SESSION           | 0              |
| NDB_API_PRUNED_SCAN_COUNT_SESSION          | 0              |
| NDB_API_SCAN_BATCH_COUNT_SESSION           | 0              |
| NDB_API_READ_ROW_COUNT_SESSION             | 1              |
| NDB_API_TRANS_LOCAL_READ_ROW_COUNT_SESSION | 1              |
| NDB_API_EVENT_DATA_COUNT_INJECTOR          | 0              |
| NDB_API_EVENT_NONDATA_COUNT_INJECTOR       | 0              |
| NDB_API_EVENT_BYTES_COUNT_INJECTOR         | 0              |
| NDB_API_WAIT_EXEC_COMPLETE_COUNT_SLAVE     | 0              |
| NDB_API_WAIT_SCAN_RESULT_COUNT_SLAVE       | 0              |
| NDB_API_WAIT_META_REQUEST_COUNT_SLAVE      | 0              |
| NDB_API_WAIT_NANOS_COUNT_SLAVE             | 0              |
| NDB_API_BYTES_SENT_COUNT_SLAVE             | 0              |
| NDB_API_BYTES_RECEIVED_COUNT_SLAVE         | 0              |
```

```
| NDB_API_TRANS_START_COUNT_SLAVE            | 0              |
| NDB_API_TRANS_COMMIT_COUNT_SLAVE           | 0              |
| NDB_API_TRANS_ABORT_COUNT_SLAVE            | 0              |
| NDB_API_TRANS_CLOSE_COUNT_SLAVE            | 0              |
| NDB_API_PK_OP_COUNT_SLAVE                  | 0              |
| NDB_API_UK_OP_COUNT_SLAVE                  | 0              |
| NDB_API_TABLE_SCAN_COUNT_SLAVE             | 0              |
| NDB_API_RANGE_SCAN_COUNT_SLAVE             | 0              |
| NDB_API_PRUNED_SCAN_COUNT_SLAVE            | 0              |
| NDB_API_SCAN_BATCH_COUNT_SLAVE             | 0              |
| NDB_API_READ_ROW_COUNT_SLAVE               | 0              |
| NDB_API_TRANS_LOCAL_READ_ROW_COUNT_SLAVE   | 0              |
| NDB_API_WAIT_EXEC_COMPLETE_COUNT           | 4              |
| NDB_API_WAIT_SCAN_RESULT_COUNT             | 3              |
| NDB_API_WAIT_META_REQUEST_COUNT            | 28             |
| NDB_API_WAIT_NANOS_COUNT                   | 53756398       |
| NDB_API_BYTES_SENT_COUNT                   | 1060           |
| NDB_API_BYTES_RECEIVED_COUNT               | 9724           |
| NDB_API_TRANS_START_COUNT                  | 3              |
| NDB_API_TRANS_COMMIT_COUNT                 | 2              |
| NDB_API_TRANS_ABORT_COUNT                  | 0              |
| NDB_API_TRANS_CLOSE_COUNT                  | 3              |
| NDB_API_PK_OP_COUNT                        | 2              |
| NDB_API_UK_OP_COUNT                        | 0              |
| NDB_API_TABLE_SCAN_COUNT                   | 1              |
| NDB_API_RANGE_SCAN_COUNT                   | 0              |
| NDB_API_PRUNED_SCAN_COUNT                  | 0              |
| NDB_API_SCAN_BATCH_COUNT                   | 0              |
| NDB_API_READ_ROW_COUNT                     | 2              |
| NDB_API_TRANS_LOCAL_READ_ROW_COUNT         | 2              |
| NDB_API_EVENT_DATA_COUNT                   | 0              |
| NDB_API_EVENT_NONDATA_COUNT                | 0              |
| NDB_API_EVENT_BYTES_COUNT                  | 0              |
+--------------------------------------------+----------------+
60 rows in set (0.00 sec)
```

Each `Ndb` object has its own counters. NDB API applications can read the values of the counters for use in optimization or monitoring. For multithreaded clients which use more than one `Ndb` object concurrently, it is also possible to obtain a summed view of counters from all `Ndb` objects belonging to a given `Ndb_cluster_connection`.

Four sets of these counters are exposed. One set applies to the current session only; the other 3 are global. *This is in spite of the fact that their values can be obtained as either session or global status variables in the `mysql` client*. This means that specifying the `SESSION` or `GLOBAL` keyword with `SHOW STATUS` has no effect on the values reported for NDB API statistics status variables, and the value for each of these variables is the same whether the value is obtained from the equivalent column of the `SESSION_STATUS` or the `GLOBAL_STATUS` table.

- *Session counters (session specific)*

  Session counters relate to the `Ndb` objects in use by (only) the current session. Use of such objects by other MySQL clients does not influence these counts.

  In order to minimize confusion with standard MySQL session variables, we refer to the variables that correspond to these NDB API session counters as "`_session` variables", with a leading underscore.

- *Replica counters (global)*

  This set of counters relates to the `Ndb` objects used by the replica SQL thread, if any. If this `mysqld` does not act as a replica, or does not use `NDB` tables, then all of these counts are 0.

  We refer to the related status variables as "`_slave` variables" (with a leading underscore).

- *Injector counters (global)*

Injector counters relate to the `Ndb` object used to listen to cluster events by the binary log injector thread. Even when not writing a binary log, `mysqld` processes attached to an NDB Cluster continue to listen for some events, such as schema changes.

We refer to the status variables that correspond to NDB API injector counters as "`_injector` variables" (with a leading underscore).

- *Server (Global) counters (global)*

  This set of counters relates to all `Ndb` objects currently used by this `mysqld`. This includes all MySQL client applications, the replica SQL thread (if any), the binlog injector, and the `NDB` utility thread.

  We refer to the status variables that correspond to these counters as "global variables" or "`mysqld`-level variables".

You can obtain values for a particular set of variables by additionally filtering for the substring `session`, `slave`, or `injector` in the variable name (along with the common prefix `Ndb_api`). For `_session` variables, this can be done as shown here:

```
mysql> SHOW STATUS LIKE 'ndb_api%session';
+-----------------------------------------+---------+
| Variable_name                           | Value   |
+-----------------------------------------+---------+
| Ndb_api_wait_exec_complete_count_session   | 2       |
| Ndb_api_wait_scan_result_count_session     | 0       |
| Ndb_api_wait_meta_request_count_session    | 1       |
| Ndb_api_wait_nanos_count_session           | 8144375 |
| Ndb_api_bytes_sent_count_session           | 68      |
| Ndb_api_bytes_received_count_session       | 84      |
| Ndb_api_trans_start_count_session          | 1       |
| Ndb_api_trans_commit_count_session         | 1       |
| Ndb_api_trans_abort_count_session          | 0       |
| Ndb_api_trans_close_count_session          | 1       |
| Ndb_api_pk_op_count_session                | 1       |
| Ndb_api_uk_op_count_session                | 0       |
| Ndb_api_table_scan_count_session           | 0       |
| Ndb_api_range_scan_count_session           | 0       |
| Ndb_api_pruned_scan_count_session          | 0       |
| Ndb_api_scan_batch_count_session           | 0       |
| Ndb_api_read_row_count_session             | 1       |
| Ndb_api_trans_local_read_row_count_session | 1       |
+-----------------------------------------+---------+
18 rows in set (0.50 sec)
```

To obtain a listing of the NDB API `mysqld`-level status variables, filter for variable names beginning with `ndb_api` and ending in `_count`, like this:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.SESSION_STATUS
    ->     WHERE VARIABLE_NAME LIKE 'ndb_api%count';
+-----------------------------------+----------------+
| VARIABLE_NAME                     | VARIABLE_VALUE |
+-----------------------------------+----------------+
| NDB_API_WAIT_EXEC_COMPLETE_COUNT  | 4              |
| NDB_API_WAIT_SCAN_RESULT_COUNT    | 3              |
| NDB_API_WAIT_META_REQUEST_COUNT   | 28             |
| NDB_API_WAIT_NANOS_COUNT          | 53756398       |
| NDB_API_BYTES_SENT_COUNT          | 1060           |
| NDB_API_BYTES_RECEIVED_COUNT      | 9724           |
| NDB_API_TRANS_START_COUNT         | 3              |
| NDB_API_TRANS_COMMIT_COUNT        | 2              |
| NDB_API_TRANS_ABORT_COUNT         | 0              |
| NDB_API_TRANS_CLOSE_COUNT         | 3              |
| NDB_API_PK_OP_COUNT               | 2              |
| NDB_API_UK_OP_COUNT               | 0              |
| NDB_API_TABLE_SCAN_COUNT          | 1              |
| NDB_API_RANGE_SCAN_COUNT          | 0              |
| NDB_API_PRUNED_SCAN_COUNT         | 0              |
```

```
| NDB_API_SCAN_BATCH_COUNT           | 0              |
| NDB_API_READ_ROW_COUNT             | 2              |
| NDB_API_TRANS_LOCAL_READ_ROW_COUNT | 2              |
| NDB_API_EVENT_DATA_COUNT           | 0              |
| NDB_API_EVENT_NONDATA_COUNT        | 0              |
| NDB_API_EVENT_BYTES_COUNT          | 0              |
+------------------------------------+----------------+
21 rows in set (0.09 sec)
```

Not all counters are reflected in all 4 sets of status variables. For the event counters `DataEventsRecvdCount`, `NondataEventsRecvdCount`, and `EventBytesRecvdCount`, only `_injector` and `mysqld`-level NDB API status variables are available:

```
mysql> SHOW STATUS LIKE 'ndb_api%event%';
+-------------------------------------+-------+
| Variable_name                       | Value |
+-------------------------------------+-------+
| Ndb_api_event_data_count_injector   | 0     |
| Ndb_api_event_nondata_count_injector | 0     |
| Ndb_api_event_bytes_count_injector  | 0     |
| Ndb_api_event_data_count            | 0     |
| Ndb_api_event_nondata_count         | 0     |
| Ndb_api_event_bytes_count           | 0     |
+-------------------------------------+-------+
6 rows in set (0.00 sec)
```

`_injector` status variables are not implemented for any other NDB API counters, as shown here:

```
mysql> SHOW STATUS LIKE 'ndb_api%injector%';
+-------------------------------------+-------+
| Variable_name                       | Value |
+-------------------------------------+-------+
| Ndb_api_event_data_count_injector   | 0     |
| Ndb_api_event_nondata_count_injector | 0     |
| Ndb_api_event_bytes_count_injector  | 0     |
+-------------------------------------+-------+
3 rows in set (0.00 sec)
```

The names of the status variables can easily be associated with the names of the corresponding counters. Each NDB API statistics counter is listed in the following table with a description as well as the names of any MySQL server status variables corresponding to this counter.

Table 18.56 NDB API statistics counters

Counter Name	Description	Status Variables (by statistic type): • **Session** • **Slave (replica)** • **Injector** • **Server**
`WaitExecCompleteCount`	Number of times thread has been blocked while waiting for execution of an operation to complete. Includes all `execute()` calls as well as implicit executes for blob operations and auto-increment not visible to clients.	• `Ndb_api_wait_exec_complete_c` • `Ndb_api_wait_exec_complete_c` • [none] • `Ndb_api_wait_exec_complete_c`
`WaitScanResultCount`	Number of times thread has been blocked while waiting for a scan-based signal, such waiting for additional results, or for a scan to close.	• `Ndb_api_wait_scan_result_cou` • `Ndb_api_wait_scan_result_cou` • [none]

Counter Name	Description	Status Variables (by statistic type): • **Session** • **Slave (replica)** • **Injector** • **Server**
		• Ndb_api_wait_scan_result_count
WaitMetaRequestCount	Number of times thread has been blocked waiting for a metadata-based signal; this can occur when waiting for a DDL operation or for an epoch to be started (or ended).	• Ndb_api_wait_meta_request_count • Ndb_api_wait_meta_request_count • [none] • Ndb_api_wait_meta_request_count
WaitNanosCount	Total time (in nanoseconds) spent waiting for some type of signal from the data nodes.	• Ndb_api_wait_nanos_count_sessio • Ndb_api_wait_nanos_count_slave • [none] • Ndb_api_wait_nanos_count
BytesSentCount	Amount of data (in bytes) sent to the data nodes	• Ndb_api_bytes_sent_count_sessio • Ndb_api_bytes_sent_count_slave • [none] • Ndb_api_bytes_sent_count
BytesRecvdCount	Amount of data (in bytes) received from the data nodes	• Ndb_api_bytes_received_count_se • Ndb_api_bytes_received_count_sl • [none] • Ndb_api_bytes_received_count
TransStartCount	Number of transactions started.	• Ndb_api_trans_start_count_sessi • Ndb_api_trans_start_count_slave • [none] • Ndb_api_trans_start_count
TransCommitCount	Number of transactions committed.	• Ndb_api_trans_commit_count_sess • Ndb_api_trans_commit_count_slav • [none] • Ndb_api_trans_commit_count
TransAbortCount	Number of transactions aborted.	• Ndb_api_trans_abort_count_sessi • Ndb_api_trans_abort_count_slave • [none] • Ndb_api_trans_abort_count

Counter Name	Description	Status Variables (by statistic type): • **Session** • **Slave (replica)** • **Injector** • **Server**
TransCloseCount	Number of transactions aborted. (This value may be greater than the sum of TransCommitCount and TransAbortCount.)	• Ndb_api_trans_close_count_se • Ndb_api_trans_close_count_sl • [none] • Ndb_api_trans_close_count
PkOpCount	Number of operations based on or using primary keys. This count includes blob-part table operations, implicit unlocking operations, and auto-increment operations, as well as primary key operations normally visible to MySQL clients.	• Ndb_api_pk_op_count_session • Ndb_api_pk_op_count_slave • [none] • Ndb_api_pk_op_count
UkOpCount	Number of operations based on or using unique keys.	• Ndb_api_uk_op_count_session • Ndb_api_uk_op_count_slave • [none] • Ndb_api_uk_op_count
TableScanCount	Number of table scans that have been started. This includes scans of internal tables.	• Ndb_api_table_scan_count_ses • Ndb_api_table_scan_count_sla • [none] • Ndb_api_table_scan_count
RangeScanCount	Number of range scans that have been started.	• Ndb_api_range_scan_count_ses • Ndb_api_range_scan_count_sla • [none] • Ndb_api_range_scan_count
PrunedScanCount	Number of scans that have been pruned to a single partition.	• Ndb_api_pruned_scan_count_se • Ndb_api_pruned_scan_count_sl • [none] • Ndb_api_pruned_scan_count
ScanBatchCount	Number of batches of rows received. (A *batch* in this context is a set of scan results from a single fragment.)	• Ndb_api_scan_batch_count_ses • Ndb_api_scan_batch_count_sla • [none] • Ndb_api_scan_batch_count

Counter Name	Description	Status Variables (by statistic type): • Session • Slave (replica) • Injector • Server
ReadRowCount	Total number of rows that have been read. Includes rows read using primary key, unique key, and scan operations.	• Ndb_api_read_row_count_session • Ndb_api_read_row_count_slave • [none] • Ndb_api_read_row_count
TransLocalReadRowCount	Number of rows read from the data same node on which the transaction was being run.	• Ndb_api_trans_local_read_row_co • Ndb_api_trans_local_read_row_co • [none] • Ndb_api_trans_local_read_row_co
DataEventsRecvdCount	Number of row change events received.	• [none] • [none] • Ndb_api_event_data_count_injec • Ndb_api_event_data_count
NondataEventsRecvdCount	Number of events received, other than row change events.	• [none] • [none] • Ndb_api_event_nondata_count_inj • Ndb_api_event_nondata_count
EventBytesRecvdCount	Number of bytes of events received.	• [none] • [none] • Ndb_api_event_bytes_count_injec • Ndb_api_event_bytes_count

To see all counts of committed transactions—that is, all TransCommitCount counter status variables—you can filter the results of SHOW STATUS for the substring trans_commit_count, like this:

```
mysql> SHOW STATUS LIKE '%trans_commit_count%';
+-----------------------------------+-------+
| Variable_name                     | Value |
+-----------------------------------+-------+
| Ndb_api_trans_commit_count_session | 1     |
| Ndb_api_trans_commit_count_slave   | 0     |
| Ndb_api_trans_commit_count         | 2     |
+-----------------------------------+-------+
3 rows in set (0.00 sec)
```

From this you can determine that 1 transaction has been committed in the current mysql client session, and 2 transactions have been committed on this mysqld since it was last restarted.

You can see how various NDB API counters are incremented by a given SQL statement by comparing the values of the corresponding `_session` status variables immediately before and after performing the statement. In this example, after getting the initial values from `SHOW STATUS`, we create in the `test` database an `NDB` table, named `t`, that has a single column:

```
mysql> SHOW STATUS LIKE 'ndb_api%session%';
+-----------------------------------------+--------+
| Variable_name                           | Value  |
+-----------------------------------------+--------+
| Ndb_api_wait_exec_complete_count_session   | 2      |
| Ndb_api_wait_scan_result_count_session     | 0      |
| Ndb_api_wait_meta_request_count_session    | 3      |
| Ndb_api_wait_nanos_count_session           | 820705 |
| Ndb_api_bytes_sent_count_session           | 132    |
| Ndb_api_bytes_received_count_session       | 372    |
| Ndb_api_trans_start_count_session          | 1      |
| Ndb_api_trans_commit_count_session         | 1      |
| Ndb_api_trans_abort_count_session          | 0      |
| Ndb_api_trans_close_count_session          | 1      |
| Ndb_api_pk_op_count_session                | 1      |
| Ndb_api_uk_op_count_session                | 0      |
| Ndb_api_table_scan_count_session           | 0      |
| Ndb_api_range_scan_count_session           | 0      |
| Ndb_api_pruned_scan_count_session          | 0      |
| Ndb_api_scan_batch_count_session           | 0      |
| Ndb_api_read_row_count_session             | 1      |
| Ndb_api_trans_local_read_row_count_session | 1      |
+-----------------------------------------+--------+
18 rows in set (0.00 sec)

mysql> USE test;
Database changed
mysql> CREATE TABLE t (c INT) ENGINE NDBCLUSTER;
Query OK, 0 rows affected (0.85 sec)
```

Now you can execute a new `SHOW STATUS` statement and observe the changes, as shown here (with the changed rows highlighted in the output):

```
mysql> SHOW STATUS LIKE 'ndb_api%session%';
+-----------------------------------------+-----------+
| Variable_name                           | Value     |
+-----------------------------------------+-----------+
| Ndb_api_wait_exec_complete_count_session   | 8         |
| Ndb_api_wait_scan_result_count_session     | 0         |
| Ndb_api_wait_meta_request_count_session    | 17        |
| Ndb_api_wait_nanos_count_session           | 706871709 |
| Ndb_api_bytes_sent_count_session           | 2376      |
| Ndb_api_bytes_received_count_session       | 3844      |
| Ndb_api_trans_start_count_session          | 4         |
| Ndb_api_trans_commit_count_session         | 4         |
| Ndb_api_trans_abort_count_session          | 0         |
| Ndb_api_trans_close_count_session          | 4         |
| Ndb_api_pk_op_count_session                | 6         |
| Ndb_api_uk_op_count_session                | 0         |
| Ndb_api_table_scan_count_session           | 0         |
| Ndb_api_range_scan_count_session           | 0         |
| Ndb_api_pruned_scan_count_session          | 0         |
| Ndb_api_scan_batch_count_session           | 0         |
| Ndb_api_read_row_count_session             | 2         |
| Ndb_api_trans_local_read_row_count_session | 1         |
+-----------------------------------------+-----------+
18 rows in set (0.00 sec)
```

Similarly, you can see the changes in the NDB API statistics counters caused by inserting a row into `t`: Insert the row, then run the same `SHOW STATUS` statement used in the previous example, as shown here:

```
mysql> INSERT INTO t VALUES (100);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SHOW STATUS LIKE 'ndb_api%session%';
+-----------------------------------------+-----------+
| Variable_name                           | Value     |
+-----------------------------------------+-----------+
| Ndb_api_wait_exec_complete_count_session | 11        |
| Ndb_api_wait_scan_result_count_session   | 6         |
| Ndb_api_wait_meta_request_count_session  | 20        |
| Ndb_api_wait_nanos_count_session         | 707370418 |
| Ndb_api_bytes_sent_count_session         | 2724      |
| Ndb_api_bytes_received_count_session     | 4116      |
| Ndb_api_trans_start_count_session        | 7         |
| Ndb_api_trans_commit_count_session       | 6         |
| Ndb_api_trans_abort_count_session        | 0         |
| Ndb_api_trans_close_count_session        | 7         |
| Ndb_api_pk_op_count_session              | 8         |
| Ndb_api_uk_op_count_session              | 0         |
| Ndb_api_table_scan_count_session         | 1         |
| Ndb_api_range_scan_count_session         | 0         |
| Ndb_api_pruned_scan_count_session        | 0         |
| Ndb_api_scan_batch_count_session         | 0         |
| Ndb_api_read_row_count_session           | 3         |
| Ndb_api_trans_local_read_row_count_session | 2       |
+-----------------------------------------+-----------+
18 rows in set (0.00 sec)
```

We can make a number of observations from these results:

- Although we created `t` with no explicit primary key, 5 primary key operations were performed in doing so (the difference in the “before” and “after” values of `Ndb_api_pk_op_count_session`, or 6 minus 1). This reflects the creation of the hidden primary key that is a feature of all tables using the `NDB` storage engine.

- By comparing successive values for `Ndb_api_wait_nanos_count_session`, we can see that the NDB API operations implementing the `CREATE TABLE` statement waited much longer (706871709 - 820705 = 706051004 nanoseconds, or approximately 0.7 second) for responses from the data nodes than those executed by the `INSERT` (707370418 - 706871709 = 498709 ns or roughly .0005 second). The execution times reported for these statements in the `mysql` client correlate roughly with these figures.

  On platforms without sufficient (nanosecond) time resolution, small changes in the value of the `WaitNanosCount` NDB API counter due to SQL statements that execute very quickly may not always be visible in the values of `Ndb_api_wait_nanos_count_session`, `Ndb_api_wait_nanos_count_slave`, or `Ndb_api_wait_nanos_count`.

- The `INSERT` statement incremented both the `ReadRowCount` and `TransLocalReadRowCount` NDB API statistics counters, as reflected by the increased values of `Ndb_api_read_row_count_session` and `Ndb_api_trans_local_read_row_count_session`.

18.5.14 ndbinfo: The NDB Cluster Information Database

`ndbinfo` is a database containing information specific to NDB Cluster.

This database contains a number of tables, each providing a different sort of data about NDB Cluster node status, resource usage, and operations. You can find more detailed information about each of these tables in the next several sections.

`ndbinfo` is included with NDB Cluster support in the MySQL Server; no special compilation or configuration steps are required; the tables are created by the MySQL Server when it connects to the cluster. You can verify that `ndbinfo` support is active in a given MySQL Server instance using `SHOW PLUGINS`; if `ndbinfo` support is enabled, you should see a row containing `ndbinfo` in the `Name` column and `ACTIVE` in the `Status` column, as shown here (emphasized text):

```
mysql> SHOW PLUGINS;
+----------------------------------+--------+--------------------+---------+---------+
```

```
| Name                              | Status | Type               | Library | License |
+-----------------------------------+--------+--------------------+---------+---------+
| binlog                            | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| mysql_native_password             | ACTIVE | AUTHENTICATION     | NULL    | GPL     |
| mysql_old_password                | ACTIVE | AUTHENTICATION     | NULL    | GPL     |
| CSV                               | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| MEMORY                            | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| MRG_MYISAM                        | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| MyISAM                            | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| PERFORMANCE_SCHEMA                | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| BLACKHOLE                         | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| ARCHIVE                           | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| ndbcluster                        | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| ndbinfo                           | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| ndb_transid_mysql_connection_map  | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| InnoDB                            | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| INNODB_TRX                        | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_LOCKS                      | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_LOCK_WAITS                 | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMP                        | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMP_RESET                  | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMPMEM                     | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMPMEM_RESET               | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| partition                         | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
+-----------------------------------+--------+--------------------+---------+---------+
22 rows in set (0.00 sec)
```

You can also do this by checking the output of `SHOW ENGINES` for a line including `ndbinfo` in the `Engine` column and `YES` in the `Support` column, as shown here (emphasized text):

```
mysql> SHOW ENGINES\G
*************************** 1. row ***************************
      Engine: ndbcluster
     Support: YES
     Comment: Clustered, fault-tolerant tables
Transactions: YES
          XA: NO
  Savepoints: NO
*************************** 2. row ***************************
      Engine: MRG_MYISAM
     Support: YES
     Comment: Collection of identical MyISAM tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 3. row ***************************
      Engine: ndbinfo
     Support: YES
     Comment: NDB Cluster system information storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 4. row ***************************
      Engine: CSV
     Support: YES
     Comment: CSV storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 5. row ***************************
      Engine: MEMORY
     Support: YES
     Comment: Hash based, stored in memory, useful for temporary tables
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 6. row ***************************
      Engine: FEDERATED
     Support: NO
     Comment: Federated MySQL storage engine
Transactions: NULL
          XA: NULL
```

```
  Savepoints: NULL
*************************** 7. row ***************************
      Engine: ARCHIVE
     Support: YES
     Comment: Archive storage engine
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 8. row ***************************
      Engine: InnoDB
     Support: YES
     Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
          XA: YES
  Savepoints: YES
*************************** 9. row ***************************
      Engine: MyISAM
     Support: DEFAULT
     Comment: Default engine as of MySQL 3.23 with great performance
Transactions: NO
          XA: NO
  Savepoints: NO
*************************** 10. row ***************************
      Engine: BLACKHOLE
     Support: YES
     Comment: /dev/null storage engine (anything you write to it disappears)
Transactions: NO
          XA: NO
  Savepoints: NO
10 rows in set (0.00 sec)
```

If `ndbinfo` support is enabled, then you can access `ndbinfo` using SQL statements in `mysql` or another MySQL client. For example, you can see `ndbinfo` listed in the output of `SHOW DATABASES`, as shown here (emphasized text):

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| ndbinfo            |
| test               |
+--------------------+
4 rows in set (0.00 sec)
```

If the `mysqld` process was not started with the `--ndbcluster` option, `ndbinfo` is not available and is not displayed by `SHOW DATABASES`. If `mysqld` was formerly connected to an NDB Cluster but the cluster becomes unavailable (due to events such as cluster shutdown, loss of network connectivity, and so forth), `ndbinfo` and its tables remain visible, but an attempt to access any tables (other than `blocks` or `config_params`) fails with `Got error 157 'Connection to NDB failed' from NDBINFO`.

With the exception of the `blocks` and `config_params` tables, what we refer to as `ndbinfo` “tables” are actually views generated from internal `NDB` tables not normally visible to the MySQL Server.

All `ndbinfo` tables are read-only, and are generated on demand when queried. Because many of them are generated in parallel by the data nodes while other are specific to a given SQL node, they are not guaranteed to provide a consistent snapshot.

In addition, pushing down of joins is not supported on `ndbinfo` tables; so joining large `ndbinfo` tables can require transfer of a large amount of data to the requesting API node, even when the query makes use of a `WHERE` clause.

`ndbinfo` tables are not included in the query cache. (Bug #59831)

You can select the `ndbinfo` database with a `USE` statement, and then issue a `SHOW TABLES` statement to obtain a list of tables, just as for any other database, like this:

```
mysql> USE ndbinfo;
Database changed

mysql> SHOW TABLES;
+---------------------------------+
| Tables_in_ndbinfo               |
+---------------------------------+
| arbitrator_validity_detail      |
| arbitrator_validity_summary     |
| blocks                          |
| cluster_operations              |
| cluster_transactions            |
| config_params                   |
| counters                        |
| dict_obj_types                  |
| disk_write_speed_aggregate      |
| disk_write_speed_aggregate_node |
| disk_write_speed_base           |
| diskpagebuffer                  |
| logbuffers                      |
| logspaces                       |
| membership                      |
| memory_per_fragment             |
| memoryusage                     |
| nodes                           |
| operations_per_fragment         |
| resources                       |
| restart_info                    |
| server_operations               |
| server_transactions             |
| threadblocks                    |
| threadstat                      |
| transporters                    |
+---------------------------------+
26 rows in set (0.00 sec)
```

The `dict_obj_types`, `disk_write_speed_aggregate`, `disk_write_speed_aggregate_node`, `disk_write_speed_base`, and `memory_per_fragment` tables were added in NDB 7.4.1. The `restart_info` table was added in NDB 7.4.2. The `operations_per_fragment` table was added in NDB 7.4.3.

You can execute `SELECT` statements against these tables, just as you would normally expect:

```
mysql> SELECT * FROM memoryusage;
+---------+--------------------+--------+------------+------------+-------------+
| node_id | memory_type        | used   | used_pages | total      | total_pages |
+---------+--------------------+--------+------------+------------+-------------+
|       5 | Data memory        | 753664 |         23 | 1073741824 |       32768 |
|       5 | Index memory       | 163840 |         20 | 1074003968 |      131104 |
|       5 | Long message buffer |   2304 |          9 |   67108864 |      262144 |
|       6 | Data memory        | 753664 |         23 | 1073741824 |       32768 |
|       6 | Index memory       | 163840 |         20 | 1074003968 |      131104 |
|       6 | Long message buffer |   2304 |          9 |   67108864 |      262144 |
+---------+--------------------+--------+------------+------------+-------------+
6 rows in set (0.02 sec)
```

More complex queries, such as the two following `SELECT` statements using the `memoryusage` table, are possible:

```
mysql> SELECT SUM(used) as 'Data Memory Used, All Nodes'
    >     FROM memoryusage
    >     WHERE memory_type = 'Data memory';
+----------------------------+
| Data Memory Used, All Nodes |
+----------------------------+
|                       6460 |
+----------------------------+
1 row in set (0.37 sec)

mysql> SELECT SUM(max) as 'Total IndexMemory Available'
```

```
     >     FROM memoryusage
     >     WHERE memory_type = 'Index memory';
+------------------------------+
| Total IndexMemory Available |
+------------------------------+
|                        25664 |
+------------------------------+
1 row in set (0.33 sec)
```

`ndbinfo` table and column names are case-sensitive (as is the name of the `ndbinfo` database itself). These identifiers are in lowercase. Trying to use the wrong lettercase results in an error, as shown in this example:

```
mysql> SELECT * FROM nodes;
+---------+--------+---------+-------------+
| node_id | uptime | status  | start_phase |
+---------+--------+---------+-------------+
|       1 |  13602 | STARTED |           0 |
|       2 |     16 | STARTED |           0 |
+---------+--------+---------+-------------+
2 rows in set (0.04 sec)

mysql> SELECT * FROM Nodes;
ERROR 1146 (42S02): Table 'ndbinfo.Nodes' doesn't exist
```

`mysqldump` ignores the `ndbinfo` database entirely, and excludes it from any output. This is true even when using the `--databases` or `--all-databases` option.

NDB Cluster also maintains tables in the `INFORMATION_SCHEMA` information database, including the `FILES` table which contains information about files used for NDB Cluster Disk Data storage, and the `ndb_transid_mysql_connection_map` table, which shows the relationships between transactions, transaction coordinators, and NDB Cluster API nodes. For more information, see the descriptions of the tables or Section 21.31, “INFORMATION_SCHEMA NDB Cluster Tables”.

18.5.14.1 The ndbinfo arbitrator_validity_detail Table

The `arbitrator_validity_detail` table shows the view that each data node in the cluster has of the arbitrator. It is a subset of the `membership` table.

The `arbitrator_validity_detail` table contains the following columns:

- `node_id`

  This node's node ID

- `arbitrator`

  Node ID of arbitrator

- `arb_ticket`

  Internal identifier used to track arbitration

- `arb_connected`

  Whether this node is connected to the arbitrator

- `arb_state`

  Arbitration state

Notes

The node ID is the same as that reported by `ndb_mgm -e "SHOW"`.

All nodes should show the same `arbitrator` and `arb_ticket` values as well as the same `arb_state` value. Possible `arb_state` values are `ARBIT_NULL`, `ARBIT_INIT`, `ARBIT_FIND`, `ARBIT_PREP1`, `ARBIT_PREP2`, `ARBIT_START`, `ARBIT_RUN`, `ARBIT_CHOOSE`, `ARBIT_CRASH`, and `UNKNOWN`.

`arb_connected` shows whether the current node is connected to the `arbitrator`.

18.5.14.2 The ndbinfo arbitrator_validity_summary Table

The `arbitrator_validity_summary` table provides a composite view of the arbitrator with regard to the cluster's data nodes.

The `arbitrator_validity_summary` table contains the following columns:

- `arbitrator`

  Node ID of arbitrator

- `arb_ticket`

  Internal identifier used to track arbitration

- `arb_connected`

  Whether this arbitrator is connected to the cluster

- `consensus_count`

  Number of data nodes that see this node as arbitrator

Notes

In normal operations, this table should have only 1 row for any appreciable length of time. If it has more than 1 row for longer than a few moments, then either not all nodes are connected to the arbitrator, or all nodes are connected, but do not agree on the same arbitrator.

The `arbitrator` column shows the arbitrator's node ID.

`arb_ticket` is the internal identifier used by this arbitrator.

`arb_connected` shows whether this node is connected to the cluster as an arbitrator.

18.5.14.3 The ndbinfo blocks Table

The `blocks` table is a static table which simply contains the names and internal IDs of all NDB kernel blocks (see NDB Kernel Blocks). It is for use by the other `ndbinfo` tables (most of which are actually views) in mapping block numbers to block names for producing human-readable output.

The `blocks` table contains the following columns:

- `block_number`

  Block number

- `block_name`

  Block name

Notes

To obtain a list of all block names, simply execute `SELECT block_name FROM ndbinfo.blocks`. Although this is a static table, its content can vary between different NDB Cluster releases.

18.5.14.4 The ndbinfo cluster_operations Table

The `cluster_operations` table provides a per-operation (stateful primary key op) view of all activity in the NDB Cluster from the point of view of the local data management (LQH) blocks (see The DBLQH Block).

The `cluster_operations` table contains the following columns:

- `node_id`

  Node ID of reporting LQH block

- `block_instance`

  LQH block instance

- `transid`

  Transaction ID

- `operation_type`

  Operation type (see text for possible values)

- `state`

  Operation state (see text for possible values)

- `tableid`

  Table ID

- `fragmentid`

  Fragment ID

- `client_node_id`

  Client node ID

- `client_block_ref`

  Client block reference

- `tc_node_id`

  Transaction coordinator node ID

- `tc_block_no`

  Transaction coordinator block number

- `tc_block_instance`

  Transaction coordinator block instance

Notes

The transaction ID is a unique 64-bit number which can be obtained using the NDB API's `getTransactionId()` method. (Currently, the MySQL Server does not expose the NDB API transaction ID of an ongoing transaction.)

The `operation_type` column can take any one of the values `READ`, `READ-SH`, `READ-EX`, `INSERT`, `UPDATE`, `DELETE`, `WRITE`, `UNLOCK`, `REFRESH`, `SCAN`, `SCAN-SH`, `SCAN-EX`, or `<unknown>`.

The `state` column can have any one of the values `ABORT_QUEUED`, `ABORT_STOPPED`, `COMMITTED`, `COMMIT_QUEUED`, `COMMIT_STOPPED`, `COPY_CLOSE_STOPPED`, `COPY_FIRST_STOPPED`, `COPY_STOPPED`, `COPY_TUPKEY`, `IDLE`, `LOG_ABORT_QUEUED`, `LOG_COMMIT_QUEUED`, `LOG_COMMIT_QUEUED_WAIT_SIGNAL`, `LOG_COMMIT_WRITTEN`, `LOG_COMMIT_WRITTEN_WAIT_SIGNAL`, `LOG_QUEUED`, `PREPARED`, `PREPARED_RECEIVED_COMMIT`, `SCAN_CHECK_STOPPED`, `SCAN_CLOSE_STOPPED`, `SCAN_FIRST_STOPPED`, `SCAN_RELEASE_STOPPED`, `SCAN_STATE_USED`, `SCAN_STOPPED`, `SCAN_TUPKEY`, `STOPPED`, `TC_NOT_CONNECTED`, `WAIT_ACC`, `WAIT_ACC_ABORT`, `WAIT_AI_AFTER_ABORT`, `WAIT_ATTR`, `WAIT_SCAN_AI`, `WAIT_TUP`, `WAIT_TUPKEYINFO`, `WAIT_TUP_COMMIT`, or `WAIT_TUP_TO_ABORT`. (If the MySQL Server is running with `ndbinfo_show_hidden` enabled, you can view this list of states by selecting from the `ndb$dblqh_tcconnect_state` table, which is normally hidden.)

You can obtain the name of an `NDB` table from its table ID by checking the output of `ndb_show_tables`.

The `fragid` is the same as the partition number seen in the output of `ndb_desc --extra-partition-info` (short form `-p`).

In `client_node_id` and `client_block_ref`, `client` refers to an NDB Cluster API or SQL node (that is, an NDB API client or a MySQL Server attached to the cluster).

The `block_instance` and `tc_block_instance` column provide, respectively, the `DBLQH` and `DBTC` block instance numbers. You can use these along with the block names to obtain information about specific threads from the `threadblocks` table.

18.5.14.5 The ndbinfo cluster_transactions Table

The `cluster_transactions` table shows information about all ongoing transactions in an NDB Cluster.

The `cluster_transactions` table contains the following columns:

- `node_id`

  Node ID of transaction coordinator

- `block_instance`

  TC block instance

- `transid`

  Transaction ID

- `state`

  Operation state (see text for possible values)

- `count_operations`

  Number of stateful primary key operations in transaction (includes reads with locks, as well as DML operations)

- `outstanding_operations`

  Operations still being executed in local data management blocks

- `inactive_seconds`

  Time spent waiting for API

- `client_node_id`

Client node ID

- `client_block_ref`

  Client block reference

Notes

The transaction ID is a unique 64-bit number which can be obtained using the NDB API's `getTransactionId()` method. (Currently, the MySQL Server does not expose the NDB API transaction ID of an ongoing transaction.)

`block_instance` refers to an instance of a kernel block. Together with the block name, this number can be used to look up a given instance in the `threadblocks` table.

The `state` column can have any one of the values `CS_ABORTING`, `CS_COMMITTING`, `CS_COMMIT_SENT`, `CS_COMPLETE_SENT`, `CS_COMPLETING`, `CS_CONNECTED`, `CS_DISCONNECTED`, `CS_FAIL_ABORTED`, `CS_FAIL_ABORTING`, `CS_FAIL_COMMITTED`, `CS_FAIL_COMMITTING`, `CS_FAIL_COMPLETED`, `CS_FAIL_PREPARED`, `CS_PREPARE_TO_COMMIT`, `CS_RECEIVING`, `CS_REC_COMMITTING`, `CS_RESTART`, `CS_SEND_FIRE_TRIG_REQ`, `CS_STARTED`, `CS_START_COMMITTING`, `CS_START_SCAN`, `CS_WAIT_ABORT_CONF`, `CS_WAIT_COMMIT_CONF`, `CS_WAIT_COMPLETE_CONF`, `CS_WAIT_FIRE_TRIG_REQ`. (If the MySQL Server is running with `ndbinfo_show_hidden` enabled, you can view this list of states by selecting from the `ndb$dbtc_apiconnect_state` table, which is normally hidden.)

In `client_node_id` and `client_block_ref`, `client` refers to an NDB Cluster API or SQL node (that is, an NDB API client or a MySQL Server attached to the cluster).

The `tc_block_instance` column provides the `DBTC` block instance number. You can use this along with the block name to obtain information about specific threads from the `threadblocks` table.

18.5.14.6 The ndbinfo config_params Table

The `config_params` table is a static table which provides the names and internal ID numbers of and other information about NDB Cluster configuration parameters.

The `config_params` table contains the following columns:

- `param_number`

  The parameter's internal ID number

- `param_name`

  The name of the parameter

Notes

Although this is a static table, its content can vary between NDB Cluster installations, since supported parameters can vary due to differences between software releases, cluster hardware configurations, and other factors.

18.5.14.7 The ndbinfo counters Table

The `counters` table provides running totals of events such as reads and writes for specific kernel blocks and data nodes. Counts are kept from the most recent node start or restart; a node start or restart resets all counters on that node. Not all kernel blocks have all types of counters.

The `counters` table contains the following columns:

- `node_id`

The data node ID

- `block_name`

  Name of the associated NDB kernel block (see NDB Kernel Blocks).

- `block_instance`

  Block instance

- `counter_id`

  The counter's internal ID number; normally an integer between 1 and 10, inclusive.

- `counter_name`

  The name of the counter. See text for names of individual counters and the NDB kernel block with which each counter is associated.

- `val`

  The counter's value

Notes

Each counter is associated with a particular NDB kernel block.

The `OPERATIONS` counter is associated with the `DBLQH` (local query handler) kernel block (see The DBLQH Block). A primary-key read counts as one operation, as does a primary-key update. For reads, there is one operation in `DBLQH` per operation in `DBTC`. For writes, there is one operation counted per fragment replica.

The `ATTRINFO`, `TRANSACTIONS`, `COMMITS`, `READS`, `LOCAL_READS`, `SIMPLE_READS`, `WRITES`, `LOCAL_WRITES`, `ABORTS`, `TABLE_SCANS`, and `RANGE_SCANS` counters are associated with the DBTC (transaction co-ordinator) kernel block (see The DBTC Block).

`LOCAL_WRITES` and `LOCAL_READS` are primary-key operations using a transaction coordinator in a node that also holds the primary fragment replica of the record.

The `READS` counter includes all reads. `LOCAL_READS` includes only those reads of the primary fragment replica on the same node as this transaction coordinator. `SIMPLE_READS` includes only those reads in which the read operation is the beginning and ending operation for a given transaction. Simple reads do not hold locks but are part of a transaction, in that they observe uncommitted changes made by the transaction containing them but not of any other uncommitted transactions. Such reads are “simple” from the point of view of the TC block; since they hold no locks they are not durable, and once `DBTC` has routed them to the relevant LQH block, it holds no state for them.

`ATTRINFO` keeps a count of the number of times an interpreted program is sent to the data node. See NDB Protocol Messages, for more information about `ATTRINFO` messages in the `NDB` kernel.

The `LOCAL_TABLE_SCANS_SENT`, `READS_RECEIVED`, `PRUNED_RANGE_SCANS_RECEIVED`, `RANGE_SCANS_RECEIVED`, `LOCAL_READS_SENT`, `CONST_PRUNED_RANGE_SCANS_RECEIVED`, `LOCAL_RANGE_SCANS_SENT`, `REMOTE_READS_SENT`, `REMOTE_RANGE_SCANS_SENT`, `READS_NOT_FOUND`, `SCAN_BATCHES_RETURNED`, `TABLE_SCANS_RECEIVED`, and `SCAN_ROWS_RETURNED` counters are associated with the `DBSPJ` (select push-down join) kernel block (see The DBSPJ Block).

The `block_name` and `block_instance` columns provide, respectively, the applicable NDB kernel block name and instance number. You can use these to obtain information about specific threads from the `threadblocks` table.

A number of counters provide information about transporter overload and send buffer sizing when troubleshooting such issues. For each LQH instance, there is one instance of each counter in the following list:

- `LQHKEY_OVERLOAD`: Number of primary key requests rejected at the LQH block instance due to transporter overload
- `LQHKEY_OVERLOAD_TC`: Count of instances of `LQHKEY_OVERLOAD` where the TC node transporter was overloaded
- `LQHKEY_OVERLOAD_READER`: Count of instances of `LQHKEY_OVERLOAD` where the API reader (reads only) node was overloaded.
- `LQHKEY_OVERLOAD_NODE_PEER`: Count of instances of `LQHKEY_OVERLOAD` where the next backup data node (writes only) was overloaded
- `LQHKEY_OVERLOAD_SUBSCRIBER`: Count of instances of `LQHKEY_OVERLOAD` where a event subscriber (writes only) was overloaded.
- `LQHSCAN_SLOWDOWNS`: Count of instances where a fragment scan batch size was reduced due to scanning API transporter overload.

18.5.14.8 The ndbinfo dict_obj_types Table

The `dict_obj_types` table is a static table listing possible dictionary object types used in the NDB kernel. These are the same types defined by `Object::Type` in the NDB API.

The `dict_obj_types` table contains the following columns:

- `type_id`

  The type ID for this type

- `type_name`

  The name of this type

Notes

This table was added in NDB 7.4.1.

18.5.14.9 The ndbinfo disk_write_speed_base Table

The `disk_write_speed_base` table provides base information about the speed of disk writes during LCP, backup, and restore operations.

The `disk_write_speed_base` table contains the following columns:

- `node_id`

  Node ID of this node

- `thr_no`

  Thread ID of this LDM thread

- `millis_ago`

  Milliseconds since this reporting period ended

- `millis_passed`

  Milliseconds elapsed in this reporting period

- `backup_lcp_bytes_written`

  Number of bytes written to disk by local checkpoints and backup processes during this period

- `redo_bytes_written`

  Number of bytes written to REDO log during this period

- `target_disk_write_speed`

  Actual speed of disk writes per LDM thread (base data)

Notes

This table was added in NDB 7.4.1.

18.5.14.10 The ndbinfo disk_write_speed_aggregate Table

The `disk_write_speed_aggregate` table provides aggregated information about the speed of disk writes during LCP, backup, and restore operations.

The `disk_write_speed_aggregate` table in NDB 7.4.3 and later contains the following columns:

- `node_id`

  Node ID of this node

- `thr_no`

  Thread ID of this LDM thread

- `backup_lcp_speed_last_sec`

  Number of bytes written to disk by backup and LCP processes in the last second

- `redo_speed_last_sec`

  Number of bytes written to REDO log in the last second

- `backup_lcp_speed_last_10sec`

  Number of bytes written to disk by backup and LCP processes per second, averaged over the last 10 seconds

- `redo_speed_last_10sec`

  Number of bytes written to REDO log per second, averaged over the last 10 seconds

- `std_dev_backup_lcp_speed_last_10sec`

  Standard deviation in number of bytes written to disk by backup and LCP processes per second, averaged over the last 10 seconds

- `std_dev_redo_speed_last_10sec`

  Standard deviation in number of bytes written to REDO log per second, averaged over the last 10 seconds

- `backup_lcp_speed_last_60sec`

  Number of bytes written to disk by backup and LCP processes per second, averaged over the last 60 seconds

- `redo_speed_last_60sec`

Number of bytes written to REDO log per second, averaged over the last 10 seconds

- `std_dev_backup_lcp_speed_last_60sec`

  Standard deviation in number of bytes written to disk by backup and LCP processes per second, averaged over the last 60 seconds

- `std_dev_redo_speed_last_60sec`

  Standard deviation in number of bytes written to REDO log per second, averaged over the last 60 seconds

- `slowdowns_due_to_io_lag`

  Number of seconds since last node start that disk writes were slowed due to REDO log I/O lag

- `slowdowns_due_to_high_cpu`

  Number of seconds since last node start that disk writes were slowed due to high CPU usage

- `disk_write_speed_set_to_min`

  Number of seconds since last node start that disk write speed was set to minimum

- `current_target_disk_write_speed`

  Actual speed of disk writes per LDM thread (aggregated)

Notes

Prior to NDB 7.4.3, the standard deviation used in obtaining the values shown in the `std_dev_backup_lcp_speed_last_10sec`, `std_dev_redo_speed_last_10sec`, `std_dev_backup_lcp_speed_last_60sec`, and `std_dev_redo_speed_last_60sec` columns of this table was not calculated correctly. (Bug #74317, Bug #19795072)

Prior to NDB 7.4.3, some columns of this table showed values in kilobytes or other multiples of bytes. (In NDB 7.4.3 and later, all such columns display values in bytes, as shown previously.) (Bug #74317, Bug #19795072) In NDB 7.4.2 and earlier, this table contained the following columns:

- `node_id`

  Node ID of this node

- `thr_no`

  Thread ID of this LDM thread

- `backup_lcp_speed_last_sec`

  Number of kilobytes written to disk by backup and LCP processes in the last second

- `redo_speed_last_sec`

  Number of kilobytes written to REDO log in the last second

- `backup_lcp_speed_last_10sec`

  Number of kilobytes written to disk by backup and LCP processes per second, averaged over the last 10 seconds

- `redo_speed_last_10sec`

  Number of kilobytes written to REDO log per second, averaged over the last 10 seconds

- `std_dev_backup_lcp_speed_last_10sec`

  Standard deviation in number of kilobytes written to disk by backup and LCP processes per second, averaged over the last 10 seconds

- `std_dev_redo_speed_last_10sec`

  Standard deviation in number of kilobytes written to REDO log per second, averaged over the last 10 seconds

- `backup_lcp_speed_last_60sec`

  Number of kilobytes written to disk by backup and LCP processes per second, averaged over the last 60 seconds

- `redo_speed_last_60sec`

  Number of kilobytes written to REDO log per second, averaged over the last 10 seconds

- `std_dev_backup_lcp_speed_last_60sec`

  Standard deviation in number of kilobytes written to disk by backup and LCP processes per second, averaged over the last 60 seconds

- `std_dev_redo_speed_last_60sec`

  Standard deviation in number of kilobytes written to REDO log per second, averaged over the last 60 seconds

- `slowdowns_due_to_io_lag`

  Number of seconds since last node start that disk writes were slowed due to REDO log I/O lag

- `slowdowns_due_to_high_cpu`

  Number of seconds since last node start that disk writes were slowed due to high CPU usage

- `disk_write_speed_set_to_min`

  Number of seconds since last node start that disk write speed was set to minimum

- `current_target_disk_write_speed`

  Actual speed of disk writes per LDM thread (aggregated)

The `disk_write_speed_aggregate` table was added in NDB 7.4.1.

18.5.14.11 The ndbinfo disk_write_speed_aggregate_node Table

The `disk_write_speed_aggregate_node` table provides aggregated information per node about the speed of disk writes during LCP, backup, and restore operations.

As of NDB 7.4.3, the `disk_write_speed_aggregate_node` contains the following columns (information relating to NDB 7.4.2 and earlier can be found later in this section):

- `node_id`

  Node ID of this node

- `backup_lcp_speed_last_sec`

  Number of bytes written to disk by backup and LCP processes in the last second

- `redo_speed_last_sec`

Number of bytes written to REDO log in the last second

- `backup_lcp_speed_last_10sec`

  Number of bytes written to disk by backup and LCP processes per second, averaged over the last 10 seconds

- `redo_speed_last_10sec`

  Number of bytes written to REDO log per second, averaged over the last 10 seconds

- `backup_lcp_speed_last_60sec`

  Number of bytes written to disk by backup and LCP processes per second, averaged over the last 60 seconds

- `redo_speed_last_60sec`

  Number of bytes written to disk by backup and LCP processes per second, averaged over the last 60 seconds

Notes

Prior to NDB 7.4.3, columns of this table showed values in kilobytes. (In NDB 7.4.3 and later, the columns display all such values in bytes.) (Bug #74317, Bug #19795072) In NDB 7.4.2 and earlier, the `disk_write_speed_aggregate` contains the following columns:

- `node_id`

  Node ID of this node

- `backup_lcp_speed_last_sec`

  Number of kilobytes written to disk by backup and LCP processes in the last second

- `redo_speed_last_sec`

  Number of kilobytes written to REDO log in the last second

- `backup_lcp_speed_last_10sec`

  Number of kilobytes written to disk by backup and LCP processes per second, averaged over the last 10 seconds

- `redo_speed_last_10sec`

  Number of kilobytes written to REDO log per second, averaged over the last 10 seconds

- `backup_lcp_speed_last_60sec`

  Number of kilobytes written to disk by backup and LCP processes per second, averaged over the last 60 seconds

- `redo_speed_last_60sec`

  Number of kilobytes written to disk by backup and LCP processes per second, averaged over the last 60 seconds

The `disk_write_speed_aggregate_node` table was added in NDB 7.4.1.

18.5.14.12 The ndbinfo diskpagebuffer Table

The `diskpagebuffer` table provides statistics about disk page buffer usage by NDB Cluster Disk Data tables.

The `diskpagebuffer` table contains the following columns:

- `node_id`

  The data node ID

- `block_instance`

  Block instance

- `pages_written`

  Number of pages written to disk.

- `pages_written_lcp`

  Number of pages written by local checkpoints.

- `pages_read`

  Number of pages read from disk

- `log_waits`

  Number of page writes waiting for log to be written to disk

- `page_requests_direct_return`

  Number of requests for pages that were available in buffer

- `page_requests_wait_queue`

  Number of requests that had to wait for pages to become available in buffer

- `page_requests_wait_io`

  Number of requests that had to be read from pages on disk (pages were unavailable in buffer)

Notes

You can use this table with NDB Cluster Disk Data tables to determine whether `DiskPageBufferMemory` is sufficiently large to allow data to be read from the buffer rather from disk; minimizing disk seeks can help improve performance of such tables.

You can determine the proportion of reads from `DiskPageBufferMemory` to the total number of reads using a query such as this one, which obtains this ratio as a percentage:

```
SELECT
  node_id,
  100 * page_requests_direct_return /
    (page_requests_direct_return + page_requests_wait_io)
      AS hit_ratio
FROM ndbinfo.diskpagebuffer;
```

The result from this query should be similar to what is shown here, with one row for each data node in the cluster (in this example, the cluster has 4 data nodes):

```
+---------+-----------+
| node_id | hit_ratio |
+---------+-----------+
|       5 |   97.6744 |
|       6 |   97.6879 |
|       7 |   98.1776 |
```

```
|       8 |   98.1343 |
+---------+-----------+
4 rows in set (0.00 sec)
```

`hit_ratio` values approaching 100% indicate that only a very small number of reads are being made from disk rather than from the buffer, which means that Disk Data read performance is approaching an optimum level. If any of these values are less than 95%, this is a strong indicator that the setting for `DiskPageBufferMemory` needs to be increased in the `config.ini` file.

Note

A change in `DiskPageBufferMemory` requires a rolling restart of all of the cluster's data nodes before it takes effect.

The `block_instance` column provides the NDB kernel block instance number. You can use this to obtain information about specific threads from the `threadblocks` table.

18.5.14.13 The ndbinfo logbuffers Table

The `logbuffer` table provides information on NDB Cluster log buffer usage.

The `logbuffers` table contains the following columns:

- `node_id`

  The ID of this data node.

- `log_type`

  Type of log, one of: `REDO` or `DD-UNDO`.

- `log_id`

  The log ID.

- `log_part`

  The log part number.

- `total`

  Total space available for this log.

- `used`

  Space used by this log.

Notes

Beginning with NDB 7.6.5, `logbuffers` table rows reflecting two additional log types are available when performing an NDB backup. One of these rows has the log type `BACKUP-DATA`, which shows the amount of data buffer used during backup to copy fragments to backup files. The other row has the log type `BACKUP-LOG`, which displays the amount of log buffer used during the backup to record changes made after the backup has started. One each of these `log_type` rows is shown in the `logbuffers` table for each data node in the cluster. These rows are not present unless an NDB backup is currently being performed. (Bug #25822988)

18.5.14.14 The ndbinfo logspaces Table

This table provides information about NDB Cluster log space usage.

The `logspaces` table contains the following columns:

- `node_id`

The ID of this data node.

- `log_type`

  Type of log; one of: `REDO` or `DD-UNDO`.

- `log_id`

  The log ID.

- `log_part`

  The log part number.

- `total`

  Total space available for this log.

- `used`

  Space used by this log.

18.5.14.15 The ndbinfo membership Table

The `membership` table describes the view that each data node has of all the others in the cluster, including node group membership, president node, arbitrator, arbitrator successor, arbitrator connection states, and other information.

The `membership` table contains the following columns:

- `node_id`

  This node's node ID

- `group_id`

  Node group to which this node belongs

- `left node`

  Node ID of the previous node

- `right_node`

  Node ID of the next node

- `president`

  President's node ID

- `successor`

  Node ID of successor to president

- `succession_order`

  Order in which this node succeeds to presidency

- `Conf_HB_order`

  -

- `arbitrator`

Node ID of arbitrator

- `arb_ticket`

  Internal identifier used to track arbitration

- `arb_state`

  Arbitration state

- `arb_connected`

  Whether this node is connected to the arbitrator

- `connected_rank1_arbs`

  Connected arbitrators of rank 1

- `connected_rank2_arbs`

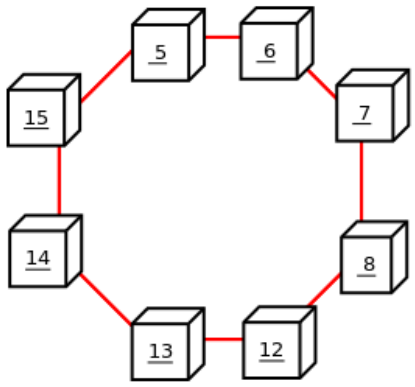
  Connected arbitrators of rank 1

Notes

The node ID and node group ID are the same as reported by `ndb_mgm -e "SHOW"`.

`left_node` and `right_node` are defined in terms of a model that connects all data nodes in a circle, in order of their node IDs, similar to the ordering of the numbers on a clock dial, as shown here:

Figure 18.18 Circular Arrangement of NDB Cluster Nodes



In this example, we have 8 data nodes, numbered 5, 6, 7, 8, 12, 13, 14, and 15, ordered clockwise in a circle. We determine “left” and “right” from the interior of the circle. The node to the left of node 5 is node 15, and the node to the right of node 5 is node 6. You can see all these relationships by running the following query and observing the output:

```
mysql> SELECT node_id,left_node,right_node
    -> FROM ndbinfo.membership;
+---------+-----------+------------+
| node_id | left_node | right_node |
+---------+-----------+------------+
|       5 |        15 |          6 |
|       6 |         5 |          7 |
|       7 |         6 |          8 |
|       8 |         7 |         12 |
|      12 |         8 |         13 |
|      13 |        12 |         14 |
|      14 |        13 |         15 |
|      15 |        14 |          5 |
+---------+-----------+------------+
8 rows in set (0.00 sec)
```

The designations “left” and “right” are used in the event log in the same way.

The `president` node is the node viewed by the current node as responsible for setting an arbitrator (see NDB Cluster Start Phases). If the president fails or becomes disconnected, the current node expects the node whose ID is shown in the `successor` column to become the new president. The `succession_order` column shows the place in the succession queue that the current node views itself as having.

In a normal NDB Cluster, all data nodes should see the same node as `president`, and the same node (other than the president) as its `successor`. In addition, the current president should see itself as `1` in the order of succession, the `successor` node should see itself as `2`, and so on.

All nodes should show the same `arb_ticket` values as well as the same `arb_state` values. Possible `arb_state` values are `ARBIT_NULL`, `ARBIT_INIT`, `ARBIT_FIND`, `ARBIT_PREP1`, `ARBIT_PREP2`, `ARBIT_START`, `ARBIT_RUN`, `ARBIT_CHOOSE`, `ARBIT_CRASH`, and `UNKNOWN`.

`arb_connected` shows whether this node is connected to the node shown as this node's `arbitrator`.

The `connected_rank1_arbs` and `connected_rank2_arbs` columns each display a list of 0 or more arbitrators having an `ArbitrationRank` equal to 1, or to 2, respectively.

Note

Both management nodes and API nodes are eligible to become arbitrators.

18.5.14.16 The ndbinfo memoryusage Table

Querying this table provides information similar to that provided by the `ALL REPORT MemoryUsage` command in the `ndb_mgm` client, or logged by `ALL DUMP 1000`.

The `memoryusage` table contains the following columns:

- `node_id`

  The node ID of this data node.

- `memory_type`

  One of `Data memory`, `Index memory`, or `Long message buffer`.

- `used`

  Number of bytes currently used for data memory or index memory by this data node.

- `used_pages`

  Number of pages currently used for data memory or index memory by this data node; see text.

- `total`

  Total number of bytes of data memory or index memory available for this data node; see text.

- `total_pages`

  Total number of memory pages available for data memory or index memory on this data node; see text.

Notes

The `total` column represents the total amount of memory in bytes available for the given resource (data memory or index memory) on a particular data node. This number should be approximately equal to the setting of the corresponding configuration parameter in the `config.ini` file.

Suppose that the cluster has 2 data nodes having node IDs `5` and `6`, and the `config.ini` file contains the following:

```
[ndbd default]
DataMemory = 1G
IndexMemory = 1G
```

Suppose also that the value of the `LongMessageBuffer` configuration parameter is allowed to assume its default (64 MB in NDB 7.3.5 and later).

The following query shows approximately the same values:

```
mysql> SELECT node_id, memory_type, total
     > FROM ndbinfo.memoryusage;
+---------+--------------------+------------+
| node_id | memory_type        | total      |
+---------+--------------------+------------+
|       5 | Data memory        | 1073741824 |
|       5 | Index memory       | 1074003968 |
|       5 | Long message buffer |   67108864 |
|       6 | Data memory        | 1073741824 |
|       6 | Index memory       | 1074003968 |
|       6 | Long message buffer |   67108864 |
+---------+--------------------+------------+
6 rows in set (0.00 sec)
```

In this case, the `total` column values for index memory are slightly higher than the value set of `IndexMemory` due to internal rounding.

For the `used_pages` and `total_pages` columns, resources are measured in pages, which are 32K in size for `DataMemory` and 8K for `IndexMemory`. For long message buffer memory, the page size is 256 bytes.

Long message buffer information can be found in this table beginning with NDB 7.3.5; in earlier versions of NDB Cluster, only data memory and index memory were included.

18.5.14.17 The ndbinfo memory_per_fragment Table

The `memory_per_fragment` table provides information about the usage of memory by individual fragments.

The `memory_per_fragment` table contains the following columns:

- `fq_name`

  Name of this fragment

- `parent_fq_name`

  Name of this fragment's parent

- `type`

  Type of object; see text for possible values

- `table_id`

  Table ID for this table

- `node_id`

  Node ID for this node

- `block_instance`

  Kernel block instance ID

- `fragment_num`

  Fragment ID (number)

- `fixed_elem_alloc_bytes`

  Number of bytes allocated for fixed-sized elements

- `fixed_elem_free_bytes`

  Free bytes remaining in pages allocated to fixed-size elements

- `fixed_elem_size_bytes`

  Length of each fixed-size element in bytes

- `fixed_elem_count`

  Number of fixed-size elements

- `fixed_elem_free_count`

  Number of free rows for fixed-size elements

- `var_elem_alloc_bytes`

  Number of bytes allocated for variable-size elements

- `var_elem_free_bytes`

  Free bytes remaining in pages allocated to variable-size elements

- `var_elem_count`

  Number of variable-size elements

- `hash_index_alloc_bytes`

  Number of bytes allocated to hash indexes

Notes

The `type` column from this table shows the dictionary object type used for this fragment (`Object::Type`, in the NDB API), and can take any one of the values shown in the following list:

- System table
- User table
- Unique hash index
- Hash index
- Unique ordered index
- Ordered index
- Hash index trigger
- Subscription trigger
- Read only constraint
- Index trigger

- Reorganize trigger
- Tablespace
- Log file group
- Data file
- Undo file
- Hash map
- Foreign key definition
- Foreign key parent trigger
- Foreign key child trigger
- Schema transaction

You can also obtain this list by executing `SELECT * FROM ndbinfo.dict_obj_types` in the `mysql` client.

The `block_instance` column provides the NDB kernel block instance number. You can use this to obtain information about specific threads from the `threadblocks` table.

This table was added in NDB 7.4.1.

18.5.14.18 The ndbinfo nodes Table

This table contains information on the status of data nodes. For each data node that is running in the cluster, a corresponding row in this table provides the node's node ID, status, and uptime. For nodes that are starting, it also shows the current start phase.

The `nodes` table contains the following columns:

- `node_id`

  The data node's unique node ID in the cluster.

- `uptime`

  Time since the node was last started, in seconds.

- `status`

  Current status of the data node; see text for possible values.

- `start_phase`

  If the data node is starting, the current start phase.

- `config_generation`

  The version of the cluster configuration file in use on this data node.

Notes

The `uptime` column shows the time in seconds that this node has been running since it was last started or restarted. This is a `BIGINT` value. This figure includes the time actually needed to start the node; in other words, this counter starts running the moment that `ndbd` or `ndbmtd` is first invoked; thus, even for a node that has not yet finished starting, `uptime` may show a nonzero value.

The `status` column shows the node's current status. This is one of: `NOTHING`, `CMVMI`, `STARTING`, `STARTED`, `SINGLEUSER`, `STOPPING_1`, `STOPPING_2`, `STOPPING_3`, or `STOPPING_4`. When the

status is STARTING, you can see the current start phase in the start_phase column (see later in this section). SINGLEUSER is displayed in the status column for all data nodes when the cluster is in single user mode (see Section 18.5.6, “NDB Cluster Single User Mode”). Seeing one of the STOPPING states does not necessarily mean that the node is shutting down but can mean rather that it is entering a new state. For example, if you put the cluster in single user mode, you can sometimes see data nodes report their state briefly as STOPPING_2 before the status changes to SINGLEUSER.

The start_phase column uses the same range of values as those used in the output of the ndb_mgm client *node_id* STATUS command (see Section 18.5.1, “Commands in the NDB Cluster Management Client”). If the node is not currently starting, then this column shows 0. For a listing of NDB Cluster start phases with descriptions, see Section 18.5.4, “Summary of NDB Cluster Start Phases”.

The config_generation column shows which version of the cluster configuration is in effect on each data node. This can be useful when performing a rolling restart of the cluster in order to make changes in configuration parameters. For example, from the output of the following SELECT statement, you can see that node 3 is not yet using the latest version of the cluster configuration (6) although nodes 1, 2, and 4 are doing so:

```
mysql> USE ndbinfo;
Database changed
mysql> SELECT * FROM nodes;
+---------+--------+---------+-------------+-------------------+
| node_id | uptime | status  | start_phase | config_generation |
+---------+--------+---------+-------------+-------------------+
|       1 |  10462 | STARTED |           0 |                 6 |
|       2 |  10460 | STARTED |           0 |                 6 |
|       3 |  10457 | STARTED |           0 |                 5 |
|       4 |  10455 | STARTED |           0 |                 6 |
+---------+--------+---------+-------------+-------------------+
2 rows in set (0.04 sec)
```

Therefore, for the case just shown, you should restart node 3 to complete the rolling restart of the cluster.

Nodes that are stopped are not accounted for in this table. Suppose that you have an NDB Cluster with 4 data nodes (node IDs 1, 2, 3 and 4), and all nodes are running normally, then this table contains 4 rows, 1 for each data node:

```
mysql> USE ndbinfo;
Database changed
mysql> SELECT * FROM nodes;
+---------+--------+---------+-------------+-------------------+
| node_id | uptime | status  | start_phase | config_generation |
+---------+--------+---------+-------------+-------------------+
|       1 |  11776 | STARTED |           0 |                 6 |
|       2 |  11774 | STARTED |           0 |                 6 |
|       3 |  11771 | STARTED |           0 |                 6 |
|       4 |  11769 | STARTED |           0 |                 6 |
+---------+--------+---------+-------------+-------------------+
4 rows in set (0.04 sec)
```

If you shut down one of the nodes, only the nodes that are still running are represented in the output of this SELECT statement, as shown here:

```
ndb_mgm> 2 STOP
Node 2: Node shutdown initiated
Node 2: Node shutdown completed.
Node 2 has shutdown.
```

```
mysql> SELECT * FROM nodes;
+---------+--------+---------+-------------+-------------------+
| node_id | uptime | status  | start_phase | config_generation |
+---------+--------+---------+-------------+-------------------+
|       1 |  11807 | STARTED |           0 |                 6 |
|       3 |  11802 | STARTED |           0 |                 6 |
|       4 |  11800 | STARTED |           0 |                 6 |
+---------+--------+---------+-------------+-------------------+
```

```
3 rows in set (0.02 sec)
```

18.5.14.19 The ndbinfo operations_per_fragment Table

The `operations_per_fragment` table provides information about the operations performed on individual fragments and fragment replicas, as well as about some of the results from these operations.

The `operations_per_fragment` table contains the following columns:

- `fq_name`

  Name of this fragment

- `parent_fq_name`

  Name of this fragment's parent

- `type`

  Type of object; see text for possible values

- `table_id`

  Table ID for this table

- `node_id`

  Node ID for this node

- `block_instance`

  Kernel block instance ID

- `fragment_num`

  Fragment ID (number)

- `tot_key_reads`

  Total number of key reads for this fragment replica

- `tot_key_inserts`

  Total number of key inserts for this fragment replica

- `tot_key_updates`

  total number of key updates for this fragment replica

- `tot_key_writes`

  Total number of key writes for this fragment replica

- `tot_key_deletes`

  Total number of key deletes for this fragment replica

- `tot_key_refs`

  Number of key operations refused

- `tot_key_attrinfo_bytes`

  Total size of all `attrinfo` attributes

- `tot_key_keyinfo_bytes`

  Total size of all `keyinfo` attributes

- `tot_key_prog_bytes`

  Total size of all interpreted programs carried by `attrinfo` attributes

- `tot_key_inst_exec`

  Total number of instructions executed by interpreted programs for key operations

- `tot_key_bytes_returned`

  Total size of all data and metadata returned from key read operations

- `tot_frag_scans`

  Total number of scans performed on this fragment replica

- `tot_scan_rows_examined`

  Total number of rows examined by scans

- `tot_scan_rows_returned`

  Total number of rows returned to client

- `tot_scan_bytes_returned`

  Total size of data and metadata returned to the client

- `tot_scan_prog_bytes`

  Total size of interpreted programs for scan operations

- `tot_scan_bound_bytes`

  Total size of all bounds used in ordered index scans

- `tot_scan_inst_exec`

  Total number of instructions executed for scans

- `tot_qd_frag_scans`

  Number of times that scans of this fragment replica have been queued

- `conc_frag_scans`

  Number of scans currently active on this fragment replica (excluding queued scans)

- `conc_qd_frag_scans`

  Number of scans currently queued for this fragment replica

- `tot_commits`

  Total number of row changes committed to this fragment replica

Notes

The `fq_name` contains the fully qualified name of the schema object to which this fragment replica belongs. This currently has the following formats:

- Base table: *DbName*/def/*TblName*
- BLOB table: *DbName*/def/NDB$BLOB_*BaseTblId*_*ColNo*
- Ordered index: sys/def/*BaseTblId*/*IndexName*
- Unique index: sys/def/*BaseTblId*/*IndexName*$unique

The $unique suffix shown for unique indexes is added by mysqld; for an index created by a different NDB API client application, this may differ, or not be present.

The syntax just shown for fully qualified object names is an internal interface which is subject to change in future releases.

Consider a table t1 created and modified by the following SQL statements:

```
CREATE DATABASE mydb;

USE mydb;

CREATE TABLE t1 (
  a INT NOT NULL,
  b INT NOT NULL,
  t TEXT NOT NULL,
  PRIMARY KEY (b)
) ENGINE=ndbcluster;

CREATE UNIQUE INDEX ix1 ON t1(b) USING HASH;
```

If t1 is assigned table ID 11, this yields the fq_name values shown here:

- Base table: mydb/def/t1
- BLOB table: mydb/def/NDB$BLOB_11_2
- Ordered index (primary key): sys/def/11/PRIMARY
- Unique index: sys/def/11/ix1$unique

For indexes or BLOB tables, the parent_fq_name column contains the fq_name of the corresponding base table. For base tables, this column is always NULL.

The type column shows the schema object type used for this fragment, which can take any one of the values System table, User table, Unique hash index, or Ordered index. BLOB tables are shown as User table.

The table_id column value is unique at any given time, but can be reused if the corresponding object has been deleted. The same ID can be seen using the ndb_show_tables utility.

The block_instance column shows which LDM instance this fragment replica belongs to. You can use this to obtain information about specific threads from the threadblocks table. The first such instance is always numbered 0.

Since there are typically two fragment replicas, and assuming that this is so, each fragment_num value should appear twice in the table, on two different data nodes from the same node group.

Since NDB does not use single-key access for ordered indexes, the counts for tot_key_reads, tot_key_inserts, tot_key_updates, tot_key_writes, and tot_key_deletes are not incremented by ordered index operations.

Note

When using tot_key_writes, you should keep in mind that a write operation in this context updates the row if the key exists, and inserts a new row

otherwise. (One use of this is in the NDB implementation of the REPLACE SQL statement.)

The tot_key_refs column shows the number of key operations refused by the LDM. Generally, such a refusal is due to duplicate keys (inserts), Key not found errors (updates, deletes, and reads), or the operation was rejected by an interpreted program used as a predicate on the row matching the key.

The attrinfo and keyinfo attributes counted by the tot_key_attrinfo_bytes and tot_key_keyinfo_bytes columns are attributes of an LQHKEYREQ signal (see The NDB Communication Protocol) used to initiate a key operation by the LDM. An attrinfo typically contains tuple field values (inserts and updates) or projection specifications (for reads); keyinfo contains the primary or unique key needed to locate a given tuple in this schema object.

The value shown by tot_frag_scans includes both full scans (that examine every row) and scans of subsets. Unique indexes and BLOB tables are never scanned, so this value, like other scan-related counts, is 0 for fragment replicas of these.

tot_scan_rows_examined may display less than the total number of rows in a given fragment replica, since ordered index scans can limited by bounds. In addition, a client may choose to end a scan before all potentially matching rows have been examined; this occurs when using an SQL statement containing a LIMIT or EXISTS clause, for example. tot_scan_rows_returned is always less than or equal to tot_scan_rows_examined.

tot_scan_bytes_returned includes, in the case of pushed joins, projections returned to the DBSPJ block in the NDB kernel.

tot_qd_frag_scans can be effected by the setting for the MaxParallelScansPerFragment data node configuration parameter, which limits the number of scans that may execute concurrently on a single fragment replica.

This table was added in NDB 7.4.3.

18.5.14.20 The ndbinfo resources Table

This table provides information about data node resource availability and usage.

These resources are sometimes known as *super-pools*.

The resources table contains the following columns:

- node_id

  The unique node ID of this data node.

- resource_name

  Name of the resource; see text.

- reserved

  The amount reserved for this resource.

- used

  The amount actually used by this resource.

- max

  The maximum amount of this resource used, since the node was last started.

Notes

The resource_name can be any one of the names shown in the following table:

- `RESERVED`: Reserved by the system; cannot be overridden.
- `DISK_OPERATIONS`: If a log file group is allocated, the size of the undo log buffer is used to set the size of this resource. This resource is used only to allocate the undo log buffer for an undo log file group; there can only be one such group. Overallocation occurs as needed by `CREATE LOGFILE GROUP`.
- `DISK_RECORDS`: Records allocated for Disk Data operations.
- `DATA_MEMORY`: Used for main memory tuples, indexes, and hash indexes. Sum of DataMemory and IndexMemory, plus 8 pages of 32 KB each if IndexMemory has been set. Cannot be overallocated.
- `JOBBUFFER`: Used for allocating job buffers by the NDB scheduler; cannot be overallocated. This is approximately 2 MB per thread plus a 1 MB buffer in both directions for all threads that can communicate. For large configurations this consume several GB.
- `FILE_BUFFERS`: Used by the redo log handler in the `DBLQH` kernel block; cannot be overallocated. Size is `NoOfFragmentLogParts` * `RedoBuffer`, plus 1 MB per log file part.
- `TRANSPORTER_BUFFERS`: Used for send buffers by `ndbmtd`; the sum of `TotalSendBufferMemory` and `ExtraSendBufferMemory`. This resource that can be overallocated by up to 25 percent. `TotalSendBufferMemory` is calculated by summing the send buffer memory per node, the default value of which is 2 MB. Thus, in a system having four data nodes and eight API nodes, the data nodes have 12 * 2 MB send buffer memory. `ExtraSendBufferMemory` is used by `ndbmtd` and amounts to 2 MB extra memory per thread. Thus, with 4 LDM threads, 2 TC threads, 1 main thread, 1 replication thread, and 2 receive threads, `ExtraSendBufferMemory` is 10 * 2 MB. Overallocation of this resource can be performed by setting the `SharedGlobalMemory` data node configuration parameter.
- `DISK_PAGE_BUFFER`: Used for the disk page buffer; determined by the `DiskPageBufferMemory` configuration parameter. Cannot be overallocated.
- `QUERY_MEMORY`: Used by the `DBSPJ` kernel block.
- `SCHEMA_TRANS_MEMORY`: Minimum is 2 MB; can be overallocated to use any remaining available memory.

18.5.14.21 The ndbinfo restart_info Table

The `restart_info` table contains information about node restart operations. Each entry in the table corresponds to a node restart status report in real time from a data node with the given node ID. Only the most recent report for any given node is shown.

The `restart_info` table contains the following columns:

- `node_id`

  Node ID in the cluster

- `node_restart_status`

  Node status; see text for values. Each of these corresponds to a possible value of `node_restart_status_int`.

- `node_restart_status_int`

  Node status code; see text for values.

- `secs_to_complete_node_failure`

  Time in seconds to complete node failure handling

- `secs_to_allocate_node_id`

Time in seconds from node failure completion to allocation of node ID

- `secs_to_include_in_heartbeat_protocol`

  Time in seconds from allocation of node ID to inclusion in heartbeat protocol

- `secs_until_wait_for_ndbcntr_master`

  Time in seconds from being included in heartbeat protocol until waiting for `NDBCNTR` master began

- `secs_wait_for_ndbcntr_master`

  Time in seconds spent waiting to be accepted by `NDBCNTR` master for starting

- `secs_to_get_start_permitted`

  Time in seconds elapsed from receiving of permission for start from master until all nodes have accepted start of this node

- `secs_to_wait_for_lcp_for_copy_meta_data`

  Time in seconds spent waiting for LCP completion before copying metadata

- `secs_to_copy_meta_data`

  Time in seconds required to copy metadata from master to newly starting node

- `secs_to_include_node`

  Time in seconds waited for GCP and inclusion of all nodes into protocols

- `secs_starting_node_to_request_local_recovery`

  Time in seconds that the node just starting spent waiting to request local recovery

- `secs_for_local_recovery`

  Time in seconds required for local recovery by node just starting

- `secs_restore_fragments`

  Time in seconds required to restore fragments from LCP files

- `secs_undo_disk_data`

  Time in seconds required to execute undo log on disk data part of records

- `secs_exec_redo_log`

  Time in seconds required to execute redo log on all restored fragments

- `secs_index_rebuild`

  Time in seconds required to rebuild indexes on restored fragments

- `secs_to_synchronize_starting_node`

  Time in seconds required to synchronize starting node from live nodes

- `secs_wait_lcp_for_restart`

  Time in seconds required for LCP start and completion before restart was completed

- `secs_wait_subscription_handover`

Time in seconds spent waiting for handover of replication subscriptions

- `total_restart_secs`

  Total number of seconds from node failure until node is started again

Notes

The following list contains values defined for the `node_restart_status_int` column with their internal status names (in parentheses), and the corresponding messages shown in the `node_restart_status` column:

- `0` (`ALLOCATED_NODE_ID`)

  `Allocated node id`

- `1` (`INCLUDED_IN_HB_PROTOCOL`)

  `Included in heartbeat protocol`

- `2` (`NDBCNTR_START_WAIT`)

  `Wait for NDBCNTR master to permit us to start`

- `3` (`NDBCNTR_STARTED`)

  `NDBCNTR master permitted us to start`

- `4` (`START_PERMITTED`)

  `All nodes permitted us to start`

- `5` (`WAIT_LCP_TO_COPY_DICT`)

  `Wait for LCP completion to start copying metadata`

- `6` (`COPY_DICT_TO_STARTING_NODE`)

  `Copying metadata to starting node`

- `7` (`INCLUDE_NODE_IN_LCP_AND_GCP`)

  `Include node in LCP and GCP protocols`

- `8` (`LOCAL_RECOVERY_STARTED`)

  `Restore fragments ongoing`

- `9` (`COPY_FRAGMENTS_STARTED`)

  `Synchronizing starting node with live nodes`

- `10` (`WAIT_LCP_FOR_RESTART`)

  `Wait for LCP to ensure durability`

- `11` (`WAIT_SUMA_HANDOVER`)

  `Wait for handover of subscriptions`

- `12` (`RESTART_COMPLETED`)

  `Restart completed`

- 13 (NODE_FAILED)

  Node failed, failure handling in progress

- 14 (NODE_FAILURE_COMPLETED)

  Node failure handling completed

- 15 (NODE_GETTING_PERMIT)

  All nodes permitted us to start

- 16 (NODE_GETTING_INCLUDED)

  Include node in LCP and GCP protocols

- 17 (NODE_GETTING_SYNCHED)

  Synchronizing starting node with live nodes

- 18 (NODE_GETTING_LCP_WAITED)

  [none]

- 19 (NODE_ACTIVE)

  Restart completed

- 20 (NOT_DEFINED_IN_CLUSTER)

  [none]

- 21 (NODE_NOT_RESTARTED_YET)

  Initial state

Status numbers 0 through 12 apply on master nodes only; the remainder of those shown in the table apply to all restarting data nodes. Status numbers 13 and 14 define node failure states; 20 and 21 occur when no information about the restart of a given node is available.

The restart_info table was added in NDB 7.4.2.

See also Section 18.5.4, “Summary of NDB Cluster Start Phases”.

18.5.14.22 The ndbinfo server_operations Table

The server_operations table contains entries for all ongoing NDB operations that the current SQL node (MySQL Server) is currently involved in. It effectively is a subset of the cluster_operations table, in which operations for other SQL and API nodes are not shown.

The server_operations table contains the following columns:

- mysql_connection_id

  MySQL Server connection ID

- node_id

  Node ID

- block_instance

  Block instance

- `transid`

  Transaction ID

- `operation_type`

  Operation type (see text for possible values)

- `state`

  Operation state (see text for possible values)

- `tableid`

  Table ID

- `fragmentid`

  Fragment ID

- `client_node_id`

  Client node ID

- `client_block_ref`

  Client block reference

- `tc_node_id`

  Transaction coordinator node ID

- `tc_block_no`

  Transaction coordinator block number

- `tc_block_instance`

  Transaction coordinator block instance

Notes

The `mysql_connection_id` is the same as the connection or session ID shown in the output of `SHOW PROCESSLIST`. It is obtained from the `INFORMATION_SCHEMA` table `NDB_TRANSID_MYSQL_CONNECTION_MAP`.

`block_instance` refers to an instance of a kernel block. Together with the block name, this number can be used to look up a given instance in the `threadblocks` table.

The transaction ID (`transid`) is a unique 64-bit number which can be obtained using the NDB API's `getTransactionId()` method. (Currently, the MySQL Server does not expose the NDB API transaction ID of an ongoing transaction.)

The `operation_type` column can take any one of the values `READ`, `READ-SH`, `READ-EX`, `INSERT`, `UPDATE`, `DELETE`, `WRITE`, `UNLOCK`, `REFRESH`, `SCAN`, `SCAN-SH`, `SCAN-EX`, or `<unknown>`.

The `state` column can have any one of the values `ABORT_QUEUED`, `ABORT_STOPPED`, `COMMITTED`, `COMMIT_QUEUED`, `COMMIT_STOPPED`, `COPY_CLOSE_STOPPED`, `COPY_FIRST_STOPPED`, `COPY_STOPPED`, `COPY_TUPKEY`, `IDLE`, `LOG_ABORT_QUEUED`, `LOG_COMMIT_QUEUED`, `LOG_COMMIT_QUEUED_WAIT_SIGNAL`, `LOG_COMMIT_WRITTEN`, `LOG_COMMIT_WRITTEN_WAIT_SIGNAL`, `LOG_QUEUED`, `PREPARED`, `PREPARED_RECEIVED_COMMIT`, `SCAN_CHECK_STOPPED`, `SCAN_CLOSE_STOPPED`, `SCAN_FIRST_STOPPED`, `SCAN_RELEASE_STOPPED`, `SCAN_STATE_USED`, `SCAN_STOPPED`, `SCAN_TUPKEY`, `STOPPED`,

TC_NOT_CONNECTED, WAIT_ACC, WAIT_ACC_ABORT, WAIT_AI_AFTER_ABORT, WAIT_ATTR, WAIT_SCAN_AI, WAIT_TUP, WAIT_TUPKEYINFO, WAIT_TUP_COMMIT, or WAIT_TUP_TO_ABORT. (If the MySQL Server is running with ndbinfo_show_hidden enabled, you can view this list of states by selecting from the ndb$dblqh_tcconnect_state table, which is normally hidden.)

You can obtain the name of an NDB table from its table ID by checking the output of ndb_show_tables.

The fragid is the same as the partition number seen in the output of ndb_desc --extra-partition-info (short form -p).

In client_node_id and client_block_ref, client refers to an NDB Cluster API or SQL node (that is, an NDB API client or a MySQL Server attached to the cluster).

The block_instance and tc_block_instance column provide NDB kernel block instance numbers. You can use these to obtain information about specific threads from the threadblocks table.

18.5.14.23 The ndbinfo server_transactions Table

The server_transactions table is subset of the cluster_transactions table, but includes only those transactions in which the current SQL node (MySQL Server) is a participant, while including the relevant connection IDs.

The server_transactions table contains the following columns:

- mysql_connection_id

  MySQL Server connection ID

- node_id

  Transaction coordinator node ID

- block_instance

  Transaction coordinator block instance

- transid

  Transaction ID

- state

  Operation state (see text for possible values)

- count_operations

  Number of stateful operations in the transaction

- outstanding_operations

  Operations still being executed by local data management layer (LQH blocks)

- inactive_seconds

  Time spent waiting for API

- client_node_id

  Client node ID

- client_block_ref

Client block reference

Notes

The `mysql_connection_id` is the same as the connection or session ID shown in the output of `SHOW PROCESSLIST`. It is obtained from the `INFORMATION_SCHEMA` table `NDB_TRANSID_MYSQL_CONNECTION_MAP`.

`block_instance` refers to an instance of a kernel block. Together with the block name, this number can be used to look up a given instance in the `threadblocks` table.

The transaction ID (`transid`) is a unique 64-bit number which can be obtained using the NDB API's `getTransactionId()` method. (Currently, the MySQL Server does not expose the NDB API transaction ID of an ongoing transaction.)

The `state` column can have any one of the values `CS_ABORTING`, `CS_COMMITTING`, `CS_COMMIT_SENT`, `CS_COMPLETE_SENT`, `CS_COMPLETING`, `CS_CONNECTED`, `CS_DISCONNECTED`, `CS_FAIL_ABORTED`, `CS_FAIL_ABORTING`, `CS_FAIL_COMMITTED`, `CS_FAIL_COMMITTING`, `CS_FAIL_COMPLETED`, `CS_FAIL_PREPARED`, `CS_PREPARE_TO_COMMIT`, `CS_RECEIVING`, `CS_REC_COMMITTING`, `CS_RESTART`, `CS_SEND_FIRE_TRIG_REQ`, `CS_STARTED`, `CS_START_COMMITTING`, `CS_START_SCAN`, `CS_WAIT_ABORT_CONF`, `CS_WAIT_COMMIT_CONF`, `CS_WAIT_COMPLETE_CONF`, `CS_WAIT_FIRE_TRIG_REQ`. (If the MySQL Server is running with `ndbinfo_show_hidden` enabled, you can view this list of states by selecting from the `ndb$dbtc_apiconnect_state` table, which is normally hidden.)

In `client_node_id` and `client_block_ref`, `client` refers to an NDB Cluster API or SQL node (that is, an NDB API client or a MySQL Server attached to the cluster).

The `block_instance` column provides the `DBTC` kernel block instance number. You can use this to obtain information about specific threads from the `threadblocks` table.

18.5.14.24 The ndbinfo tc_time_track_stats Table

The `tc_time_track_stats` table provides time-tracking information obtained from the `DBTC` block (TC) instances in the data nodes, through API nodes access `NDB`. Each TC instance tracks latencies for a set of activities it undertakes on behalf of API nodes or other data nodes; these activities include transactions, transaction errors, key reads, key writes, unique index operations, failed key operations of any type, scans, failed scans, fragment scans, and failed fragment scans.

A set of counters is maintained for each activity, each counter covering a range of latencies less than or equal to an upper bound. At the conclusion of each activity, its latency is determined and the appropriate counter incremented. `tc_time_track_stats` presents this information as rows, with a row for each instance of the following:

- Data node, using its ID
- TC block instance
- Other communicating data node or API node, using its ID
- Upper bound value

Notes

Each row contains a value for each activity type. This is the number of times that this activity occurred with a latency within the range specified by the row (that is, where the latency does not exceed the upper bound).

The `tc_time_track_stats` table contains the following columns:

- `node_id`

Requesting node ID

- `block_number`

  TC block number

- `block_instance`

  TC block instance number

- `comm_node_id`

  Node ID of communicating API or data node

- `upper_bound`

  Upper bound of interval (in microseconds)

- `scans`

  Based on duration of successful scans from opening to closing, tracked against the API or data nodes requesting them.

- `scan_errors`

  Based on duration of failed scans from opening to closing, tracked against the API or data nodes requesting them.

- `scan_fragments`

  Based on duration of successful fragment scans from opening to closing, tracked against the data nodes executing them

- `scan_fragment_errors`

  Based on duration of failed fragment scans from opening to closing, tracked against the data nodes executing them

- `transactions`

  Based on duration of successful transactions from beginning until sending of commit `ACK`, tracked against the API or data nodes requesting them. Stateless transactions are not included.

- `transaction_errors`

  Based on duration of failing transactions from start to point of failure, tracked against the API or data nodes requesting them.

- `read_key_ops`

  Based on duration of successful primary key reads with locks. Tracked against both the API or data node requesting them and the data node executing them.

- `write_key_ops`

  Based on duration of successful primary key writes, tracked against both the API or data node requesting them and the data node executing them.

- `index_key_ops`

  Based on duration of successful unique index key operations, tracked against both the API or data node requesting them and the data node executing reads of base tables.

- `key_op_errors`

  Based on duration of all unsuccessful key read or write operations, tracked against both the API or data node requesting them and the data node executing them.

The `block_instance` column provides the `DBTC` kernel block instance number. You can use this together with the block name to obtain information about specific threads from the `threadblocks` table.

The `tc_time_track_stats` table was added in NDB 7.4.9 (Bug #78533, Bug #21889652).

18.5.14.25 The ndbinfo threadblocks Table

The `threadblocks` table associates data nodes, threads, and instances of `NDB` kernel blocks.

The `threadblocks` table contains the following columns:

- `node_id`

  Node ID

- `thr_no`

  Thread ID

- `block_name`

  Block name

- `block_instance`

  Block instance number

Notes

The value of the `block_name` in this table is one of the values found in the `block_name` column when selecting from the `ndbinfo.blocks` table. Although the list of possible values is static for a given NDB Cluster release, the list may vary between releases.

The `block_instance` column provides the kernel block instance number.

18.5.14.26 The ndbinfo threadstat Table

The `threadstat` table provides a rough snapshot of statistics for threads running in the `NDB` kernel.

The `threadstat` table contains the following columns:

- `node_id`

  Node ID

- `thr_no`

  Thread ID

- `thr_nm`

  Thread name

- `c_loop`

  Number of loops in main loop

- `c_exec`

Number of signals executed

- `c_wait`

  Number of times waiting for additional input

- `c_l_sent_prioa`

  Number of priority A signals sent to own node

- `c_l_sent_priob`

  Number of priority B signals sent to own node

- `c_r_sent_prioa`

  Number of priority A signals sent to remote node

- `c_r_sent_priob`

  Number of priority B signals sent to remote node

- `os_tid`

  OS thread ID

- `os_now`

  OS time (ms)

- `os_ru_utime`

  OS user CPU time (µs)

- `os_ru_stime`

  OS system CPU time (µs)

- `os_ru_minflt`

  OS page reclaims (soft page faults)

- `os_ru_majflt`

  OS page faults (hard page faults)

- `os_ru_nvcsw`

  OS voluntary context switches

- `os_ru_nivcsw`

  OS involuntary context switches

Notes

`os_time` uses the system `gettimeofday()` call.

The values of the `os_ru_utime`, `os_ru_stime`, `os_ru_minflt`, `os_ru_majflt`, `os_ru_nvcsw`, and `os_ru_nivcsw` columns are obtained using the system `getrusage()` call, or the equivalent.

Since this table contains counts taken at a given point in time, for best results it is necessary to query this table periodically and store the results in an intermediate table or tables. The MySQL

Server's Event Scheduler can be employed to automate such monitoring. For more information, see Section 20.4, “Using the Event Scheduler”.

18.5.14.27 The ndbinfo transporters Table

This table contains information about NDB transporters.

The `transporters` table contains the following columns:

- `node_id`

  This data node's unique node ID in the cluster

- `remote_node_id`

  The remote data node's node ID

- `status`

  Status of the connection

- `remote_address`

  Name or IP address of the remote host

- `bytes_sent`

  Number of bytes sent using this connection

- `bytes_received`

  Number of bytes received using this connection

- `connect_count`

  Number of times connection established on this transporter

- `overloaded`

  1 if this transporter is currently overloaded, otherwise 0

- `overload_count`

  Number of times this transporter has entered overload state since connecting

- `slowdown`

  1 if this transporter is in slowdown state, otherwise 0

- `slowdown_count`

  Number of times this transporter has entered slowdown state since connecting

Notes

For each running data node in the cluster, the `transporters` table displays a row showing the status of each of that node's connections with all nodes in the cluster, *including itself*. This information is shown in the table's *status* column, which can have any one of the following values: `CONNECTING`, `CONNECTED`, `DISCONNECTING`, or `DISCONNECTED`.

Connections to API and management nodes which are configured but not currently connected to the cluster are shown with status `DISCONNECTED`. Rows where the `node_id` is that of a data node which

is not currently connected are not shown in this table. (This is similar omission of disconnected nodes in the `ndbinfo.nodes` table.

The `remote_address` is the host name or address for the node whose ID is shown in the `remote_node_id` column. The `bytes_sent` from this node and `bytes_received` by this node are the numbers, respectively, of bytes sent and received by the node using this connection since it was established. For nodes whose status is `CONNECTING` or `DISCONNECTED`, these columns always display `0`.

Assume you have a 5-node cluster consisting of 2 data nodes, 2 SQL nodes, and 1 management node, as shown in the output of the `SHOW` command in the `ndb_mgm` client:

```
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=1    @10.100.10.1  (5.6.51-ndb-7.4.31, Nodegroup: 0, *)
id=2    @10.100.10.2  (5.6.51-ndb-7.4.31, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=10   @10.100.10.10  (5.6.51-ndb-7.4.31)

[mysqld(API)]   2 node(s)
id=20   @10.100.10.20  (5.6.51-ndb-7.4.31)
id=21   @10.100.10.21  (5.6.51-ndb-7.4.31)
```

There are 10 rows in the `transporters` table—5 for the first data node, and 5 for the second—assuming that all data nodes are running, as shown here:

```
mysql> SELECT node_id, remote_node_id, status
    ->   FROM ndbinfo.transporters;
+---------+----------------+---------------+
| node_id | remote_node_id | status        |
+---------+----------------+---------------+
|       1 |              1 | DISCONNECTED  |
|       1 |              2 | CONNECTED     |
|       1 |             10 | CONNECTED     |
|       1 |             20 | CONNECTED     |
|       1 |             21 | CONNECTED     |
|       2 |              1 | CONNECTED     |
|       2 |              2 | DISCONNECTED  |
|       2 |             10 | CONNECTED     |
|       2 |             20 | CONNECTED     |
|       2 |             21 | CONNECTED     |
+---------+----------------+---------------+
10 rows in set (0.04 sec)
```

If you shut down one of the data nodes in this cluster using the command `2 STOP` in the `ndb_mgm` client, then repeat the previous query (again using the `mysql` client), this table now shows only 5 rows—1 row for each connection from the remaining management node to another node, including both itself and the data node that is currently offline—and displays `CONNECTING` for the status of each remaining connection to the data node that is currently offline, as shown here:

```
mysql> SELECT node_id, remote_node_id, status
    ->   FROM ndbinfo.transporters;
+---------+----------------+---------------+
| node_id | remote_node_id | status        |
+---------+----------------+---------------+
|       1 |              1 | DISCONNECTED  |
|       1 |              2 | CONNECTING    |
|       1 |             10 | CONNECTED     |
|       1 |             20 | CONNECTED     |
|       1 |             21 | CONNECTED     |
+---------+----------------+---------------+
5 rows in set (0.02 sec)
```

The `connect_count`, `overloaded`, `overload_count` ,`slowdown`, and `slowdown_count` counters are reset on connection, and retain their values after the remote node disconnects. The `bytes_sent`

and `bytes_received` counters are also reset on connection, and so retain their values following disconnection (until the next connection resets them).

The *overload* state referred to by the `overloaded` and `overload_count` columns occurs when this transporter's send buffer contains more than `OVerloadLimit` bytes (default is 80% of `SendBufferMemory`, that is, 0.8 * 2097152 = 1677721 bytes). When a given transporter is in a state of overload, any new transaction that tries to use this transporter fails with Error 1218 (`Send Buffers overloaded in NDB kernel`). This affects both scans and primary key operations.

The *slowdown* state referenced by the `slowdown` and `slowdown_count` columns of this table occurs when the transporter's send buffer contains more than 60% of the overload limit (equal to 0.6 * 2097152 = 1258291 bytes by default). In this state, any new scan using this transporter has its batch size reduced to minimize the load on the transporter.

Common causes of send buffer slowdown or overloading include the following:

- Data size, in particular the quantity of data stored in `TEXT` columns or `BLOB` columns (or both types of columns)
- Having a data node (ndbd or ndbmtd) on the same host as an SQL node that is engaged in binary logging
- Large number of rows per transaction or transaction batch
- Configuration issues such as insufficient `SendBufferMemory`
- Hardware issues such as insufficient RAM or poor network connectivity

See also Section 18.3.3.12, “Configuring NDB Cluster Send Buffer Parameters”.

18.5.15 Quick Reference: NDB Cluster SQL Statements

This section discusses several SQL statements that can prove useful in managing and monitoring a MySQL server that is connected to an NDB Cluster, and in some cases provide information about the cluster itself.

- `SHOW ENGINE NDB STATUS`, `SHOW ENGINE NDBCLUSTER STATUS`

  The output of this statement contains information about the server's connection to the cluster, creation and usage of NDB Cluster objects, and binary logging for NDB Cluster replication.

  See Section 13.7.5.16, “SHOW ENGINE Statement”, for a usage example and more detailed information.

- `SHOW ENGINES`

  This statement can be used to determine whether or not clustering support is enabled in the MySQL server, and if so, whether it is active.

  See Section 13.7.5.17, “SHOW ENGINES Statement”, for more detailed information.

  Note

  This statement does not support a `LIKE` clause. However, you can use `LIKE` to filter queries against the `INFORMATION_SCHEMA.ENGINES` table, as discussed in the next item.

- `SELECT * FROM INFORMATION_SCHEMA.ENGINES [WHERE ENGINE LIKE 'NDB%']`

  This is the equivalent of `SHOW ENGINES`, but uses the `ENGINES` table of the `INFORMATION_SCHEMA` database. Unlike the case with the `SHOW ENGINES` statement, it is possible

to filter the results using a `LIKE` clause, and to select specific columns to obtain information that may be of use in scripts. For example, the following query shows whether the server was built with `NDB` support and, if so, whether it is enabled:

```
mysql> SELECT SUPPORT FROM INFORMATION_SCHEMA.ENGINES
    ->   WHERE ENGINE LIKE 'NDB%';
+---------+
| support |
+---------+
| ENABLED |
+---------+
```

See Section 21.7, “The INFORMATION_SCHEMA ENGINES Table”, for more information.

- `SHOW VARIABLES LIKE 'NDB%'`

  This statement provides a list of most server system variables relating to the `NDB` storage engine, and their values, as shown here:

```
mysql> SHOW VARIABLES LIKE 'NDB%';
+-----------------------------------+-----------------------------------+
| Variable_name                     | Value                             |
+-----------------------------------+-----------------------------------+
| ndb_autoincrement_prefetch_sz     | 1                                 |
| ndb_batch_size                    | 32768                             |
| ndb_blob_read_batch_bytes         | 65536                             |
| ndb_blob_write_batch_bytes        | 65536                             |
| ndb_cache_check_time              | 0                                 |
| ndb_clear_apply_status            | ON                                |
| ndb_cluster_connection_pool       | 1                                 |
| ndb_connectstring                 | 127.0.0.1                         |
| ndb_deferred_constraints          | 0                                 |
| ndb_distribution                  | KEYHASH                           |
| ndb_eventbuffer_free_percent      | 20                                |
| ndb_eventbuffer_max_alloc         | 0                                 |
| ndb_extra_logging                 | 1                                 |
| ndb_force_send                    | ON                                |
| ndb_index_stat_enable             | ON                                |
| ndb_index_stat_option             | loop_enable=1000ms,loop_idle=1000ms,
loop_busy=100ms,update_batch=1,read_batch=4,idle_batch=32,check_batch=8,
check_delay=10m,delete_batch=8,clean_delay=1m,error_batch=4,error_delay=1m,
evict_batch=8,evict_delay=1m,cache_limit=32M,cache_lowpct=90,zero_total=0 |
| ndb_join_pushdown                 | ON                                |
| ndb_log_apply_status              | OFF                               |
| ndb_log_bin                       | ON                                |
| ndb_log_binlog_index              | ON                                |
| ndb_log_empty_epochs              | OFF                               |
| ndb_log_empty_update              | OFF                               |
| ndb_log_exclusive_reads           | OFF                               |
| ndb_log_fail_terminate            | OFF                               |
| ndb_log_orig                      | OFF                               |
| ndb_log_transaction_id            | OFF                               |
| ndb_log_update_as_write           | ON                                |
| ndb_log_update_minimal            | OFF                               |
| ndb_log_updated_only              | ON                                |
| ndb_mgmd_host                     | 127.0.0.1                         |
| ndb_nodeid                        | 0                                 |
| ndb_optimization_delay            | 10                                |
| ndb_optimized_node_selection      | 3                                 |
| ndb_recv_thread_activation_threshold | 8                                 |
| ndb_recv_thread_cpu_mask          |                                   |
| ndb_report_thresh_binlog_epoch_slip | 3                                 |
| ndb_report_thresh_binlog_mem_usage | 10                                |
| ndb_show_foreign_key_mock_tables  | OFF                               |
| ndb_slave_conflict_role           | NONE                              |
| ndb_table_no_logging              | OFF                               |
| ndb_table_temporary               | OFF                               |
| ndb_use_copying_alter_table       | OFF                               |
| ndb_use_exact_count               | OFF                               |
| ndb_use_transactions              | ON                                |
| ndb_version                       | 459808                            |
```

```
| ndb_version_string                  | ndb-7.4.31                          |
| ndb_wait_connected                  | 30                                  |
| ndb_wait_setup                      | 30                                  |
| ndbinfo_database                    | ndbinfo                             |
| ndbinfo_max_bytes                   | 0                                   |
| ndbinfo_max_rows                    | 10                                  |
| ndbinfo_offline                     | OFF                                 |
| ndbinfo_show_hidden                 | OFF                                 |
| ndbinfo_table_prefix                | ndb$                                |
| ndbinfo_version                     | 459808                              |
+-------------------------------------+-------------------------------------+
```

See Section 5.1.7, “Server System Variables”, for more information.

- `SELECT * FROM INFORMATION_SCHEMA.GLOBAL_VARIABLES WHERE VARIABLE_NAME LIKE 'NDB%';`

  This statement is the equivalent of the `SHOW VARIABLES` statement described in the previous item, and provides almost identical output, as shown here:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.GLOBAL_VARIABLES
    ->   WHERE VARIABLE_NAME LIKE 'NDB%';
+-------------------------------------+-------------------------------------+
| VARIABLE_NAME                       | VARIABLE_VALUE                      |
+-------------------------------------+-------------------------------------+
| NDB_CACHE_CHECK_TIME                | 0                                   |
| NDB_TABLE_TEMPORARY                 | OFF                                 |
| NDB_AUTOINCREMENT_PREFETCH_SZ       | 1                                   |
| NDB_LOG_BINLOG_INDEX                | ON                                  |
| NDB_FORCE_SEND                      | ON                                  |
| NDB_WAIT_SETUP                      | 30                                  |
| NDB_CLUSTER_CONNECTION_POOL         | 1                                   |
| NDB_WAIT_CONNECTED                  | 30                                  |
| NDB_USE_EXACT_COUNT                 | OFF                                 |
| NDB_LOG_EMPTY_UPDATE                | OFF                                 |
| NDB_RECV_THREAD_ACTIVATION_THRESHOLD | 8                                   |
| NDB_EVENTBUFFER_MAX_ALLOC           | 0                                   |
| NDB_OPTIMIZED_NODE_SELECTION        | 3                                   |
| NDBINFO_DATABASE                    | ndbinfo                             |
| NDB_LOG_APPLY_STATUS                | OFF                                 |
| NDB_JOIN_PUSHDOWN                   | ON                                  |
| NDB_RECV_THREAD_CPU_MASK            |                                     |
| NDBINFO_VERSION                     | 459808                              |
| NDB_CONNECTSTRING                   | 127.0.0.1                           |
| NDB_TABLE_NO_LOGGING                | OFF                                 |
| NDB_LOG_UPDATED_ONLY                | ON                                  |
| NDB_VERSION                         | 459808                              |
| NDB_LOG_UPDATE_MINIMAL              | OFF                                 |
| NDB_OPTIMIZATION_DELAY              | 10                                  |
| NDB_LOG_UPDATE_AS_WRITE             | ON                                  |
| NDB_SHOW_FOREIGN_KEY_MOCK_TABLES    | OFF                                 |
| NDB_VERSION_STRING                  | ndb-7.4.31                          |
| NDBINFO_OFFLINE                     | OFF                                 |
| NDB_INDEX_STAT_OPTION               | loop_enable=1000ms,loop_idle=1000ms,
loop_busy=100ms,update_batch=1,read_batch=4,idle_batch=32,check_batch=8,
check_delay=10m,delete_batch=8,clean_delay=1m,error_batch=4,error_delay=1m,
evict_batch=8,evict_delay=1m,cache_limit=32M,cache_lowpct=90,zero_total=0 |
| NDBINFO_MAX_ROWS                    | 10                                  |
| NDB_BATCH_SIZE                      | 32768                               |
| NDB_USE_TRANSACTIONS                | ON                                  |
| NDB_NODEID                          | 0                                   |
| NDB_SLAVE_CONFLICT_ROLE             | NONE                                |
| NDB_REPORT_THRESH_BINLOG_MEM_USAGE  | 10                                  |
| NDB_MGMD_HOST                       | 127.0.0.1                           |
| NDB_REPORT_THRESH_BINLOG_EPOCH_SLIP | 3                                   |
| NDBINFO_MAX_BYTES                   | 0                                   |
| NDB_LOG_BIN                         | ON                                  |
| NDBINFO_TABLE_PREFIX                | ndb$                                |
| NDB_LOG_EMPTY_EPOCHS                | OFF                                 |
| NDB_LOG_ORIG                        | OFF                                 |
| NDB_LOG_EXCLUSIVE_READS             | OFF                                 |
```

```
| NDB_LOG_TRANSACTION_ID               | OFF                                  |
| NDB_LOG_FAIL_TERMINATE               | OFF                                  |
| NDB_EVENTBUFFER_FREE_PERCENT         | 20                                   |
| NDB_CLEAR_APPLY_STATUS               | ON                                   |
| NDBINFO_SHOW_HIDDEN                  | OFF                                  |
| NDB_INDEX_STAT_ENABLE                | ON                                   |
| NDB_DISTRIBUTION                     | KEYHASH                              |
| NDB_EXTRA_LOGGING                    | 1                                    |
| NDB_BLOB_WRITE_BATCH_BYTES           | 65536                                |
| NDB_USE_COPYING_ALTER_TABLE          | OFF                                  |
| NDB_DEFERRED_CONSTRAINTS             | 0                                    |
| NDB_BLOB_READ_BATCH_BYTES            | 65536                                |
+--------------------------------------+--------------------------------------+
```

Unlike the case with the `SHOW VARIABLES` statement, it is possible to select individual columns. For example:

```
mysql> SELECT VARIABLE_VALUE
    ->   FROM INFORMATION_SCHEMA.GLOBAL_VARIABLES
    ->   WHERE VARIABLE_NAME = 'ndb_force_send';
+----------------+
| VARIABLE_VALUE |
+----------------+
| ON             |
+----------------+
```

See Section 21.10, “The INFORMATION_SCHEMA GLOBAL_VARIABLES and SESSION_VARIABLES Tables”, and Section 5.1.7, “Server System Variables”, for more information.

- `SHOW STATUS LIKE 'NDB%'`

  This statement shows at a glance whether or not the MySQL server is acting as a cluster SQL node, and if so, it provides the MySQL server's cluster node ID, the host name and port for the cluster management server to which it is connected, and the number of data nodes in the cluster, as shown here:

```
mysql> SHOW STATUS LIKE 'NDB%';
+-------------------------------------------+--------------------------------+
| Variable_name                             | Value                          |
+-------------------------------------------+--------------------------------+
| Ndb_api_wait_exec_complete_count          | 2                              |
| Ndb_api_wait_scan_result_count            | 3                              |
| Ndb_api_wait_meta_request_count           | 109                            |
| Ndb_api_wait_nanos_count                  | 830671169345                   |
| Ndb_api_bytes_sent_count                  | 13284                          |
| Ndb_api_bytes_received_count              | 112672                         |
| Ndb_api_trans_start_count                 | 2                              |
| Ndb_api_trans_commit_count                | 1                              |
| Ndb_api_trans_abort_count                 | 0                              |
| Ndb_api_trans_close_count                 | 2                              |
| Ndb_api_pk_op_count                       | 1                              |
| Ndb_api_uk_op_count                       | 0                              |
| Ndb_api_table_scan_count                  | 1                              |
| Ndb_api_range_scan_count                  | 0                              |
| Ndb_api_pruned_scan_count                 | 0                              |
| Ndb_api_scan_batch_count                  | 0                              |
| Ndb_api_read_row_count                    | 1                              |
| Ndb_api_trans_local_read_row_count        | 1                              |
| Ndb_api_adaptive_send_forced_count        | 0                              |
| Ndb_api_adaptive_send_unforced_count      | 3                              |
| Ndb_api_adaptive_send_deferred_count      | 0                              |
| Ndb_api_event_data_count                  | 0                              |
| Ndb_api_event_nondata_count               | 0                              |
| Ndb_api_event_bytes_count                 | 3122560                        |
| Ndb_cluster_node_id                       | 100                            |
| Ndb_config_from_host                      | 127.0.0.1                      |
| Ndb_config_from_port                      | 1186                           |
| Ndb_number_of_data_nodes                  | 2                              |
| Ndb_number_of_ready_data_nodes            | 2                              |
| Ndb_connect_count                         | 0                              |
```

```
| Ndb_execute_count                             | 0                                |
| Ndb_scan_count                                | 0                                |
| Ndb_pruned_scan_count                         | 0                                |
| Ndb_schema_locks_count                        | 0                                |
| Ndb_api_wait_exec_complete_count_session      | 0                                |
| Ndb_api_wait_scan_result_count_session        | 0                                |
| Ndb_api_wait_meta_request_count_session       | 0                                |
| Ndb_api_wait_nanos_count_session              | 0                                |
| Ndb_api_bytes_sent_count_session              | 0                                |
| Ndb_api_bytes_received_count_session          | 0                                |
| Ndb_api_trans_start_count_session             | 0                                |
| Ndb_api_trans_commit_count_session            | 0                                |
| Ndb_api_trans_abort_count_session             | 0                                |
| Ndb_api_trans_close_count_session             | 0                                |
| Ndb_api_pk_op_count_session                   | 0                                |
| Ndb_api_uk_op_count_session                   | 0                                |
| Ndb_api_table_scan_count_session              | 0                                |
| Ndb_api_range_scan_count_session              | 0                                |
| Ndb_api_pruned_scan_count_session             | 0                                |
| Ndb_api_scan_batch_count_session              | 0                                |
| Ndb_api_read_row_count_session                | 0                                |
| Ndb_api_trans_local_read_row_count_session    | 0                                |
| Ndb_api_adaptive_send_forced_count_session    | 0                                |
| Ndb_api_adaptive_send_unforced_count_session  | 0                                |
| Ndb_api_adaptive_send_deferred_count_session  | 0                                |
| Ndb_sorted_scan_count                         | 0                                |
| Ndb_pushed_queries_defined                    | 0                                |
| Ndb_pushed_queries_dropped                    | 0                                |
| Ndb_pushed_queries_executed                   | 0                                |
| Ndb_pushed_reads                              | 0                                |
| Ndb_last_commit_epoch_server                  | 73014444043                      |
| Ndb_last_commit_epoch_session                 | 0                                |
| Ndb_api_wait_exec_complete_count_slave        | 0                                |
| Ndb_api_wait_scan_result_count_slave          | 0                                |
| Ndb_api_wait_meta_request_count_slave         | 0                                |
| Ndb_api_wait_nanos_count_slave                | 0                                |
| Ndb_api_bytes_sent_count_slave                | 0                                |
| Ndb_api_bytes_received_count_slave            | 0                                |
| Ndb_api_trans_start_count_slave               | 0                                |
| Ndb_api_trans_commit_count_slave              | 0                                |
| Ndb_api_trans_abort_count_slave               | 0                                |
| Ndb_api_trans_close_count_slave               | 0                                |
| Ndb_api_pk_op_count_slave                     | 0                                |
| Ndb_api_uk_op_count_slave                     | 0                                |
| Ndb_api_table_scan_count_slave                | 0                                |
| Ndb_api_range_scan_count_slave                | 0                                |
| Ndb_api_pruned_scan_count_slave               | 0                                |
| Ndb_api_scan_batch_count_slave                | 0                                |
| Ndb_api_read_row_count_slave                  | 0                                |
| Ndb_api_trans_local_read_row_count_slave      | 0                                |
| Ndb_api_adaptive_send_forced_count_slave      | 0                                |
| Ndb_api_adaptive_send_unforced_count_slave    | 0                                |
| Ndb_api_adaptive_send_deferred_count_slave    | 0                                |
| Ndb_slave_max_replicated_epoch                | 0                                |
| Ndb_api_event_data_count_injector             | 0                                |
| Ndb_api_event_nondata_count_injector          | 0                                |
| Ndb_api_event_bytes_count_injector            | 2080768                          |
| Ndb_conflict_fn_max                           | 0                                |
| Ndb_conflict_fn_old                           | 0                                |
| Ndb_conflict_fn_max_del_win                   | 0                                |
| Ndb_conflict_fn_epoch                         | 0                                |
| Ndb_conflict_fn_epoch_trans                   | 0                                |
| Ndb_conflict_fn_epoch2                        | 0                                |
| Ndb_conflict_fn_epoch2_trans                  | 0                                |
| Ndb_conflict_trans_row_conflict_count         | 0                                |
| Ndb_conflict_trans_row_reject_count           | 0                                |
| Ndb_conflict_trans_reject_count               | 0                                |
| Ndb_conflict_trans_detect_iter_count          | 0                                |
| Ndb_conflict_trans_conflict_commit_count      | 0                                |
| Ndb_conflict_epoch_delete_delete_count        | 0                                |
| Ndb_conflict_reflected_op_prepare_count       | 0                                |
| Ndb_conflict_reflected_op_discard_count       | 0                                |
```

```
| Ndb_conflict_refresh_op_count                | 0                              |
| Ndb_conflict_last_conflict_epoch             | 0                              |
| Ndb_conflict_last_stable_epoch               | 0                              |
| Ndb_index_stat_status                        | allow:1,enable:1,busy:0,
loop:1000,list:(new:0,update:0,read:0,idle:0,check:0,delete:0,error:0,total:0),
analyze:(queue:0,wait:0),stats:(nostats:0,wait:0),
total:(analyze:(all:0,error:0),query:(all:0,nostats:0,error:0),
event:(act:0,skip:0,miss:0),cache:(refresh:0,clean:0,pinned:0,drop:0,evict:0)),
cache:(query:0,clean:0,drop:0,evict:0,usedpct:0.00,highpct:0.00)               |
| Ndb_index_stat_cache_query                   | 0                              |
| Ndb_index_stat_cache_clean                   | 0                              |
+----------------------------------------------+--------------------------------+
```

If the MySQL server was built with clustering support, but it is not connected to a cluster, all rows in the output of this statement contain a zero or an empty string.

See also Section 13.7.5.36, “SHOW STATUS Statement”.

- `SELECT * FROM INFORMATION_SCHEMA.GLOBAL_STATUS WHERE VARIABLE_NAME LIKE 'NDB%';`

  This statement provides similar output to the `SHOW STATUS` statement discussed in the previous item. However, unlike the case with `SHOW STATUS`, it is possible using the `SELECT` to extract values in SQL for use in scripts for monitoring and automation purposes.

  See Section 21.9, “The INFORMATION_SCHEMA GLOBAL_STATUS and SESSION_STATUS Tables”, for more information.

You can also query the tables in the `ndbinfo` information database for real-time data about many NDB Cluster operations. See Section 18.5.14, “ndbinfo: The NDB Cluster Information Database”.

18.5.16 NDB Cluster Security Issues

This section discusses security considerations to take into account when setting up and running NDB Cluster.

Topics covered in this section include the following:

- NDB Cluster and network security issues
- Configuration issues relating to running NDB Cluster securely
- NDB Cluster and the MySQL privilege system
- MySQL standard security procedures as applicable to NDB Cluster

18.5.16.1 NDB Cluster Security and Networking Issues

In this section, we discuss basic network security issues as they relate to NDB Cluster. It is extremely important to remember that NDB Cluster “out of the box” is not secure; you or your network administrator must take the proper steps to ensure that your cluster cannot be compromised over the network.

Cluster communication protocols are inherently insecure, and no encryption or similar security measures are used in communications between nodes in the cluster. Because network speed and latency have a direct impact on the cluster's efficiency, it is also not advisable to employ SSL or other encryption to network connections between nodes, as such schemes effectively slow communications.

It is also true that no authentication is used for controlling API node access to an NDB Cluster. As with encryption, the overhead of imposing authentication requirements would have an adverse impact on Cluster performance.

In addition, there is no checking of the source IP address for either of the following when accessing the cluster:

- SQL or API nodes using “free slots” created by empty `[mysqld]` or `[api]` sections in the `config.ini` file

  This means that, if there are any empty `[mysqld]` or `[api]` sections in the `config.ini` file, then any API nodes (including SQL nodes) that know the management server's host name (or IP address) and port can connect to the cluster and access its data without restriction. (See Section 18.5.16.2, “NDB Cluster and MySQL Privileges”, for more information about this and related issues.)

  Note

  You can exercise some control over SQL and API node access to the cluster by specifying a `HostName` parameter for all `[mysqld]` and `[api]` sections in the `config.ini` file. However, this also means that, should you wish to connect an API node to the cluster from a previously unused host, you need to add an `[api]` section containing its host name to the `config.ini` file.

  More information is available elsewhere in this chapter about the `HostName` parameter. Also see Section 18.3.1, “Quick Test Setup of NDB Cluster”, for configuration examples using `HostName` with API nodes.

- Any `ndb_mgm` client

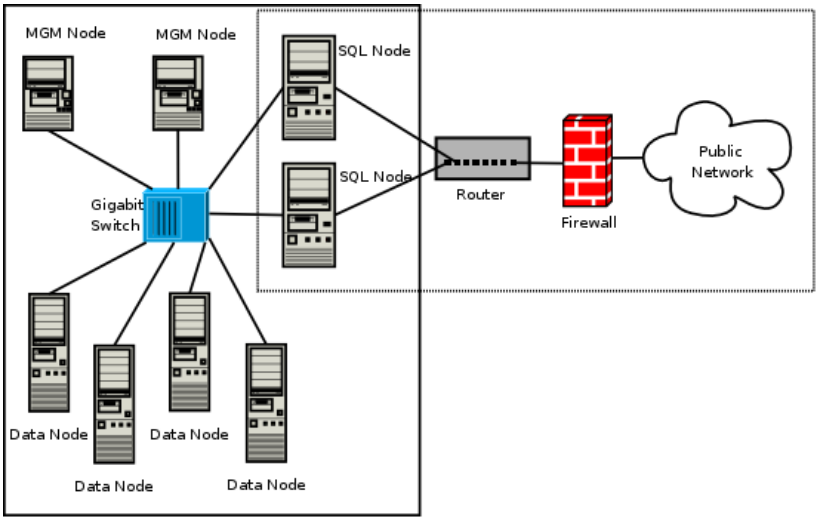
  This means that any cluster management client that is given the management server's host name (or IP address) and port (if not the standard port) can connect to the cluster and execute any management client command. This includes commands such as `ALL STOP` and `SHUTDOWN`.

For these reasons, it is necessary to protect the cluster on the network level. The safest network configuration for Cluster is one which isolates connections between Cluster nodes from any other network communications. This can be accomplished by any of the following methods:

1. Keeping Cluster nodes on a network that is physically separate from any public networks. This option is the most dependable; however, it is the most expensive to implement.

   We show an example of an NDB Cluster setup using such a physically segregated network here:

   Figure 18.19 NDB Cluster with Hardware Firewall

   This setup has two networks, one private (solid box) for the Cluster management servers and data nodes, and one public (dotted box) where the SQL nodes reside. (We show the management and data nodes connected using a gigabit switch since this provides the best performance.) Both networks are protected from the outside by a hardware firewall, sometimes also known as a *network-based firewall*.

This network setup is safest because no packets can reach the cluster's management or data nodes from outside the network—and none of the cluster's internal communications can reach the outside—without going through the SQL nodes, as long as the SQL nodes do not permit any packets to be forwarded. This means, of course, that all SQL nodes must be secured against hacking attempts.

Important

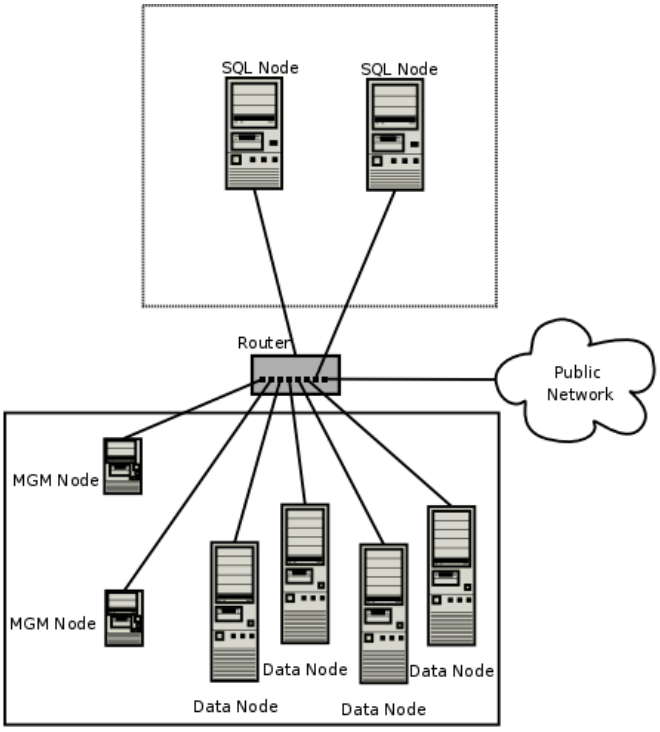
With regard to potential security vulnerabilities, an SQL node is no different from any other MySQL server. See Section 6.1.3, “Making MySQL Secure Against Attackers”, for a description of techniques you can use to secure MySQL servers.

2. Using one or more software firewalls (also known as *host-based firewalls*) to control which packets pass through to the cluster from portions of the network that do not require access to it. In this type of setup, a software firewall must be installed on every host in the cluster which might otherwise be accessible from outside the local network.

   The host-based option is the least expensive to implement, but relies purely on software to provide protection and so is the most difficult to keep secure.

   This type of network setup for NDB Cluster is illustrated here:

   Figure 18.20 NDB Cluster with Software Firewalls



   Using this type of network setup means that there are two zones of NDB Cluster hosts. Each cluster host must be able to communicate with all of the other machines in the cluster, but only those hosting SQL nodes (dotted box) can be permitted to have any contact with the outside, while those in the zone containing the data nodes and management nodes (solid box) must be isolated

from any machines that are not part of the cluster. Applications using the cluster and user of those applications must *not* be permitted to have direct access to the management and data node hosts.

To accomplish this, you must set up software firewalls that limit the traffic to the type or types shown in the following table, according to the type of node that is running on each cluster host computer:

Table 18.57 Node types in a host-based firewall cluster configuration

Node Type	Permitted Traffic
SQL or API node	• It originates from the IP address of a management or data node (using any TCP or UDP port). • It originates from within the network in which the cluster resides and is on the port that your application is using.
Data node or Management node	• It originates from the IP address of a management or data node (using any TCP or UDP port). • It originates from the IP address of an SQL or API node.

Any traffic other than that shown in the table for a given node type should be denied.

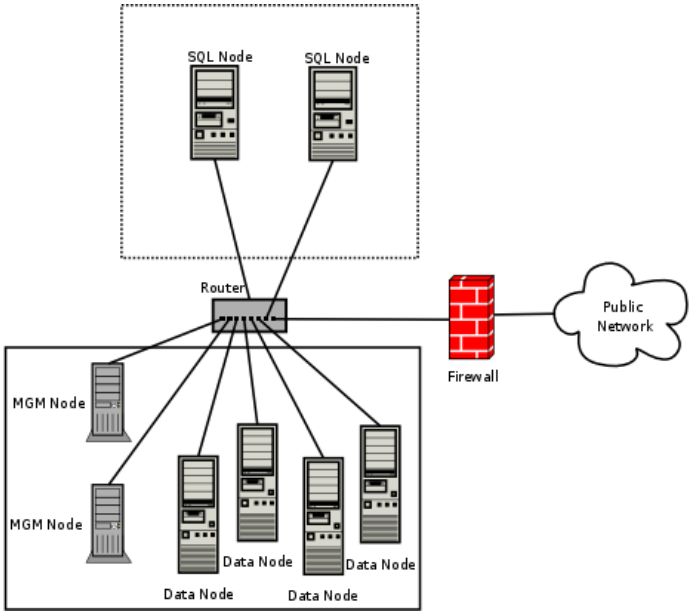
The specifics of configuring a firewall vary from firewall application to firewall application, and are beyond the scope of this Manual. `iptables` is a very common and reliable firewall application, which is often used with APF as a front end to make configuration easier. You can (and should) consult the documentation for the software firewall that you employ, should you choose to implement an NDB Cluster network setup of this type, or of a “mixed” type as discussed under the next item.

3. It is also possible to employ a combination of the first two methods, using both hardware and software to secure the cluster—that is, using both network-based and host-based firewalls. This is between the first two schemes in terms of both security level and cost. This type of network setup

keeps the cluster behind the hardware firewall, but permits incoming packets to travel beyond the router connecting all cluster hosts to reach the SQL nodes.

One possible network deployment of an NDB Cluster using hardware and software firewalls in combination is shown here:

Figure 18.21 NDB Cluster with a Combination of Hardware and Software Firewalls



In this case, you can set the rules in the hardware firewall to deny any external traffic except to SQL nodes and API nodes, and then permit traffic to them only on the ports required by your application.

Whatever network configuration you use, remember that your objective from the viewpoint of keeping the cluster secure remains the same—to prevent any unessential traffic from reaching the cluster while ensuring the most efficient communication between the nodes in the cluster.

Because NDB Cluster requires large numbers of ports to be open for communications between nodes, the recommended option is to use a segregated network. This represents the simplest way to prevent unwanted traffic from reaching the cluster.

Note

If you wish to administer an NDB Cluster remotely (that is, from outside the local network), the recommended way to do this is to use `ssh` or another secure login shell to access an SQL node host. From this host, you can then run the management client to access the management server safely, from within the cluster's own local network.

Even though it is possible to do so in theory, it is *not* recommended to use `ndb_mgm` to manage a Cluster directly from outside the local network on which the Cluster is running. Since neither authentication nor encryption takes place between the management client and the management server, this represents an extremely insecure means of managing the cluster, and is almost certain to be compromised sooner or later.

18.5.16.2 NDB Cluster and MySQL Privileges

In this section, we discuss how the MySQL privilege system works in relation to NDB Cluster and the implications of this for keeping an NDB Cluster secure.

Standard MySQL privileges apply to NDB Cluster tables. This includes all MySQL privilege types (`SELECT` privilege, `UPDATE` privilege, `DELETE` privilege, and so on) granted on the database, table, and column level. As with any other MySQL Server, user and privilege information is stored in the `mysql` system database. The SQL statements used to grant and revoke privileges on `NDB` tables, databases containing such tables, and columns within such tables are identical in all respects with the `GRANT` and `REVOKE` statements used in connection with database objects involving any (other) MySQL storage engine. The same thing is true with respect to the `CREATE USER` and `DROP USER` statements.

It is important to keep in mind that, by default, the MySQL grant tables use the `MyISAM` storage engine. Because of this, those tables are not normally duplicated or shared among MySQL servers acting as SQL nodes in an NDB Cluster. In other words, changes in users and their privileges do not automatically propagate between SQL nodes by default. If you wish, you can enable automatic distribution of MySQL users and privileges across NDB Cluster SQL nodes; see Section 18.5.12, “Distributed Privileges Using Shared Grant Tables”, for details.

Conversely, because there is no way in MySQL to deny privileges (privileges can either be revoked or not granted in the first place, but not denied as such), there is no special protection for `NDB` tables on one SQL node from users that have privileges on another SQL node; (This is true even if you are not using automatic distribution of user privileges. The definitive example of this is the MySQL `root` account, which can perform any action on any database object. In combination with empty `[mysqld]` or `[api]` sections of the `config.ini` file, this account can be especially dangerous. To understand why, consider the following scenario:

- The `config.ini` file contains at least one empty `[mysqld]` or `[api]` section. This means that the NDB Cluster management server performs no checking of the host from which a MySQL Server (or other API node) accesses the NDB Cluster.
- There is no firewall, or the firewall fails to protect against access to the NDB Cluster from hosts external to the network.
- The host name or IP address of the NDB Cluster management server is known or can be determined from outside the network.

If these conditions are true, then anyone, anywhere can start a MySQL Server with `--ndbcluster --ndb-connectstring=`*`management_host`* and access this NDB Cluster. Using the MySQL `root` account, this person can then perform the following actions:

- Execute metadata statements such as `SHOW DATABASES` statement (to obtain a list of all `NDB` databases on the server) or `SHOW TABLES FROM` *`some_ndb_database`* statement to obtain a list of all `NDB` tables in a given database
- Run any legal MySQL statements on any of the discovered tables, such as:
  - `SELECT * FROM` *`some_table`* to read all the data from any table
  - `DELETE FROM` *`some_table`* to delete all the data from a table
  - `DESCRIBE` *`some_table`* or `SHOW CREATE TABLE` *`some_table`* to determine the table schema
  - `UPDATE` *`some_table`* `SET` *`column1 = some_value`* to fill a table column with “garbage” data; this could actually cause much greater damage than simply deleting all the data

    More insidious variations might include statements like these:

    ```
    UPDATE some_table SET an_int_column = an_int_column + 1
    ```

    or

    ```
    UPDATE some_table SET a_varchar_column = REVERSE(a_varchar_column)
    ```

    Such malicious statements are limited only by the imagination of the attacker.

The only tables that would be safe from this sort of mayhem would be those tables that were created using storage engines other than `NDB`, and so not visible to a “rogue” SQL node.

A user who can log in as `root` can also access the `INFORMATION_SCHEMA` database and its tables, and so obtain information about databases, tables, stored routines, scheduled events, and any other database objects for which metadata is stored in `INFORMATION_SCHEMA`.

It is also a very good idea to use different passwords for the `root` accounts on different NDB Cluster SQL nodes unless you are using distributed privileges.

In sum, you cannot have a safe NDB Cluster if it is directly accessible from outside your local network.

Important

Never leave the MySQL root account password empty. This is just as true when running MySQL as an NDB Cluster SQL node as it is when running it as a standalone (non-Cluster) MySQL Server, and should be done as part of the MySQL installation process before configuring the MySQL Server as an SQL node in an NDB Cluster.

If you wish to employ NDB Cluster's distributed privilege capabilities, you should not simply convert the system tables in the `mysql` database to use the `NDB` storage engine manually. Use the stored procedure provided for this purpose instead; see Section 18.5.12, “Distributed Privileges Using Shared Grant Tables”.

Otherwise, if you need to synchronize `mysql` system tables between SQL nodes, you can use standard MySQL replication to do so, or employ a script to copy table entries between the MySQL servers.

Summary. The most important points to remember regarding the MySQL privilege system with regard to NDB Cluster are listed here:

1. Users and privileges established on one SQL node do not automatically exist or take effect on other SQL nodes in the cluster. Conversely, removing a user or privilege on one SQL node in the cluster does not remove the user or privilege from any other SQL nodes.
2. You can distribute MySQL users and privileges among SQL nodes using the SQL script, and the stored procedures it contains, that are supplied for this purpose in the NDB Cluster distribution.
3. Once a MySQL user is granted privileges on an `NDB` table from one SQL node in an NDB Cluster, that user can “see” any data in that table regardless of the SQL node from which the data originated, even if you are not using privilege distribution.

18.5.16.3 NDB Cluster and MySQL Security Procedures

In this section, we discuss MySQL standard security procedures as they apply to running NDB Cluster.

In general, any standard procedure for running MySQL securely also applies to running a MySQL Server as part of an NDB Cluster. First and foremost, you should always run a MySQL Server as the `mysql` operating system user; this is no different from running MySQL in a standard (non-Cluster) environment. The `mysql` system account should be uniquely and clearly defined. Fortunately, this is the default behavior for a new MySQL installation. You can verify that the `mysqld` process is running as the `mysql` operating system user by using the system command such as the one shown here:

```
shell> ps aux | grep mysql
root     10467  0.0  0.1   3616  1380 pts/3    S    11:53   0:00 \
  /bin/sh ./mysqld_safe --ndbcluster --ndb-connectstring=localhost:1186
mysql    10512  0.2  2.5  58528 26636 pts/3    Sl   11:53   0:00 \
  /usr/local/mysql/libexec/mysqld --basedir=/usr/local/mysql \
  --datadir=/usr/local/mysql/var --user=mysql --ndbcluster \
  --ndb-connectstring=localhost:1186 --pid-file=/usr/local/mysql/var/mothra.pid \
  --log-error=/usr/local/mysql/var/mothra.err
```

```
jon      10579  0.0  0.0   2736   688 pts/0    S+   11:54   0:00 grep mysql
```

If the `mysqld` process is running as any other user than `mysql`, you should immediately shut it down and restart it as the `mysql` user. If this user does not exist on the system, the `mysql` user account should be created, and this user should be part of the `mysql` user group; in this case, you should also make sure that the MySQL data directory on this system (as set using the `--datadir` option for `mysqld`) is owned by the `mysql` user, and that the SQL node's `my.cnf` file includes `user=mysql` in the `[mysqld]` section. Alternatively, you can start the MySQL server process with `--user=mysql` on the command line, but it is preferable to use the `my.cnf` option, since you might forget to use the command-line option and so have `mysqld` running as another user unintentionally. The `mysqld_safe` startup script forces MySQL to run as the `mysql` user.

Important

Never run `mysqld` as the system root user. Doing so means that potentially any file on the system can be read by MySQL, and thus—should MySQL be compromised—by an attacker.

As mentioned in the previous section (see Section 18.5.16.2, “NDB Cluster and MySQL Privileges”), you should always set a root password for the MySQL Server as soon as you have it running. You should also delete the anonymous user account that is installed by default. You can accomplish these tasks using the following statements:

```
shell> mysql -u root

mysql> UPDATE mysql.user
    ->     SET Password=PASSWORD('secure_password')
    ->     WHERE User='root';

mysql> DELETE FROM mysql.user
    ->     WHERE User='';

mysql> FLUSH PRIVILEGES;
```

Be very careful when executing the `DELETE` statement not to omit the `WHERE` clause, or you risk deleting *all* MySQL users. Be sure to run the `FLUSH PRIVILEGES` statement as soon as you have modified the `mysql.user` table, so that the changes take immediate effect. Without `FLUSH PRIVILEGES`, the changes do not take effect until the next time that the server is restarted.

Note

Many of the NDB Cluster utilities such as `ndb_show_tables`, `ndb_desc`, and `ndb_select_all` also work without authentication and can reveal table names, schemas, and data. By default these are installed on Unix-style systems with the permissions `wxr-xr-x` (755), which means they can be executed by any user that can access the `mysql/bin` directory.

See Section 18.4, “NDB Cluster Programs”, for more information about these utilities.

18.6 NDB Cluster Replication

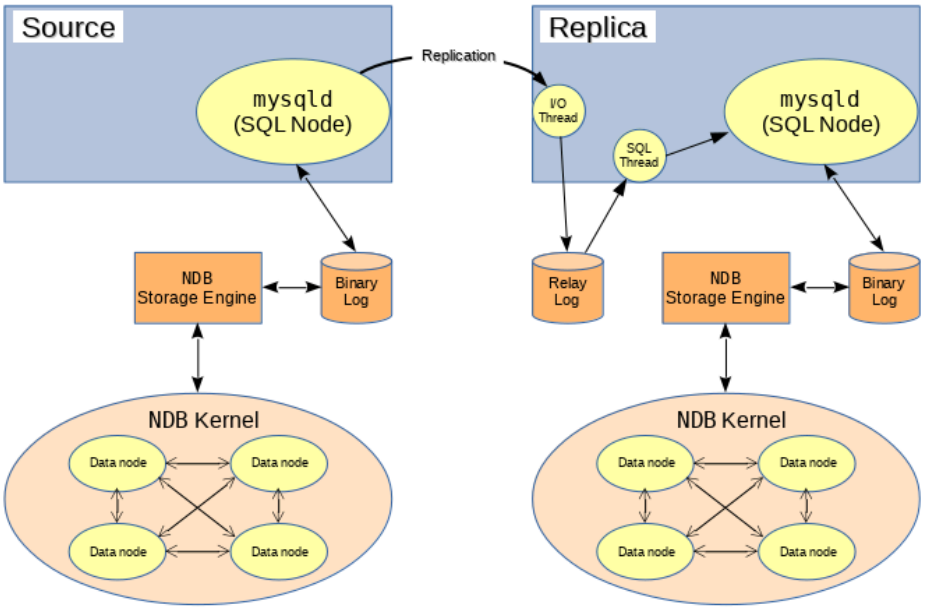
NDB Cluster supports *asynchronous replication*, more usually referred to simply as “replication”. This section explains how to set up and manage a configuration in which one group of computers operating as an NDB Cluster replicates to a second computer or group of computers. We assume some familiarity on the part of the reader with standard MySQL replication as discussed elsewhere in this Manual. (See Chapter 17, *Replication*).

Note

NDB Cluster does not support replication using GTIDs; semisynchronous replication is also not supported by the `NDB` storage engine.

Normal (non-clustered) replication involves a source server (formerly called a “master”) and a replica server (formerly referred to as a “slave”), the source being so named because operations and data to be replicated originate with it, and the replica being the recipient of these. In NDB Cluster, replication is conceptually very similar but can be more complex in practice, as it may be extended to cover a number of different configurations including replicating between two complete clusters. Although an NDB Cluster itself depends on the `NDB` storage engine for clustering functionality, it is not necessary to use `NDB` as the storage engine for the replica's copies of the replicated tables (see Replication from NDB to other storage engines). However, for maximum availability, it is possible (and preferable) to replicate from one NDB Cluster to another, and it is this scenario that we discuss, as shown in the following figure:

Figure 18.22 NDB Cluster-to-Cluster Replication Layout



In this scenario, the replication process is one in which successive states of a source cluster are logged and saved to a replica cluster. This process is accomplished by a special thread known as the NDB binary log injector thread, which runs on each MySQL server and produces a binary log (`binlog`). This thread ensures that all changes in the cluster producing the binary log—and not just those changes that are effected through the MySQL Server—are inserted into the binary log with the correct serialization order. We refer to the MySQL source and replica servers as replication servers or replication nodes, and the data flow or line of communication between them as a *replication channel*.

For information about performing point-in-time recovery with NDB Cluster and NDB Cluster Replication, see Section 18.6.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”.

NDB API replica status variables. NDB API counters can provide enhanced monitoring capabilities on replica clusters. These counters are implemented as NDB statistics `_slave` status variables, as seen in the output of `SHOW STATUS`, or in the results of queries against the `SESSION_STATUS` or `GLOBAL_STATUS` table in a `mysql` client session connected to a MySQL Server that is acting as a replica in NDB Cluster Replication. By comparing the values of these status variables before and after the execution of statements affecting replicated `NDB` tables, you can observe the corresponding actions taken on the NDB API level by the replica, which can be useful when monitoring or troubleshooting NDB Cluster Replication. Section 18.5.13, “NDB API Statistics Counters and Variables”, provides additional information.

Replication from NDB to non-NDB tables. It is possible to replicate `NDB` tables from an NDB Cluster acting as the replication source to tables using other MySQL storage engines such as `InnoDB`

or MyISAM on a replica mysqld. This is subject to a number of conditions; see Replication from NDB to other storage engines, and Replication from NDB to a nontransactional storage engine, for more information.

18.6.1 NDB Cluster Replication: Abbreviations and Symbols

Throughout this section, we use the following abbreviations or symbols for referring to the source and replica clusters, and to processes and commands run on the clusters or cluster nodes:

Table 18.58 Abbreviations used throughout this section referring to source and replica clusters, and to processes and commands run on cluster nodes

Symbol or Abbreviation	Description (Refers to...)
S	The cluster serving as the (primary) replication source
R	The cluster acting as the (primary) replica
shell*S*>	Shell command to be issued on the source cluster
mysql*S*>	MySQL client command issued on a single MySQL server running as an SQL node on the source cluster
mysql*S**>	MySQL client command to be issued on all SQL nodes participating in the replication source cluster
shell*R*>	Shell command to be issued on the replica cluster
mysql*R*>	MySQL client command issued on a single MySQL server running as an SQL node on the replica cluster
mysql*R**>	MySQL client command to be issued on all SQL nodes participating in the replica cluster
C	Primary replication channel
C'	Secondary replication channel
S'	Secondary replication source
R'	Secondary replica

18.6.2 General Requirements for NDB Cluster Replication

A replication channel requires two MySQL servers acting as replication servers (one each for the source and replica). For example, this means that in the case of a replication setup with two replication channels (to provide an extra channel for redundancy), there are a total of four replication nodes, two per cluster.

Replication of an NDB Cluster as described in this section and those following is dependent on row-based replication. This means that the replication source MySQL server must be running with --binlog-format=ROW or --binlog-format=MIXED, as described in Section 18.6.6, “Starting NDB Cluster Replication (Single Replication Channel)”. For general information about row-based replication, see Section 17.1.2, “Replication Formats”.

Important

If you attempt to use NDB Cluster Replication with --binlog-format=STATEMENT, replication fails to work properly because the ndb_binlog_index table on the source cluster and the epoch column of the ndb_apply_status table on the replica cluster are not updated (see Section 18.6.4, “NDB Cluster Replication Schema and Tables”). Instead, only updates on the MySQL server acting as the replication source propagate to the replica, and no updates from any other SQL nodes in the source cluster are replicated.

- The default value for the `--binlog-format` option is `MIXED`.

Each MySQL server used for replication in either cluster must be uniquely identified among all the MySQL replication servers participating in either cluster (you cannot have replication servers on both the source and replica clusters sharing the same ID). This can be done by starting each SQL node using the `--server-id=id` option, where *`id`* is a unique integer. Although it is not strictly necessary, we assume for purposes of this discussion that all NDB Cluster binaries are of the same release version.

It is generally true in MySQL Replication that both MySQL servers (`mysqld` processes) involved must be compatible with one another with respect to both the version of the replication protocol used and the SQL feature sets which they support (see Section 17.4.2, “Replication Compatibility Between MySQL Versions”). It is due to such differences between the binaries in the NDB Cluster and MySQL Server 5.6 distributions that NDB Cluster Replication has the additional requirement that both `mysqld` binaries come from an NDB Cluster distribution. The simplest and easiest way to assure that the `mysqld` servers are compatible is to use the same NDB Cluster distribution for all source and replica `mysqld` binaries.

We assume that the replica server or cluster is dedicated to replication of the source cluster, and that no other data is being stored on it.

All `NDB` tables being replicated must be created using a MySQL server and client. Tables and other database objects created using the NDB API (with, for example, `Dictionary::createTable()`) are not visible to a MySQL server and so are not replicated. Updates by NDB API applications to existing tables that were created using a MySQL server can be replicated.

Note

It is possible to replicate an NDB Cluster using statement-based replication. However, in this case, the following restrictions apply:

- All updates to data rows on the cluster acting as the source must be directed to a single MySQL server.
- It is not possible to replicate a cluster using multiple simultaneous MySQL replication processes.
- Only changes made at the SQL level are replicated.

These are in addition to the other limitations of statement-based replication as opposed to row-based replication; see Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”, for more specific information concerning the differences between the two replication formats.

18.6.3 Known Issues in NDB Cluster Replication

This section discusses known problems or issues when using replication with NDB Cluster.

Loss of connection between source and replica. A loss of connection can occur either between the source cluster SQL node and the replica cluster SQL node, or between the source SQL node and the data nodes of the source cluster. In the latter case, this can occur not only as a result of loss of physical connection (for example, a broken network cable), but due to the overflow of data node event buffers; if the SQL node is too slow to respond, it may be dropped by the cluster (this is controllable to some degree by adjusting the `MaxBufferedEpochs` and `TimeBetweenEpochs` configuration parameters). If this occurs, *it is entirely possible for new data to be inserted into the source cluster without being recorded in the source SQL node's binary log*. For this reason, to guarantee high availability, it is extremely important to maintain a backup replication channel, to monitor the primary

channel, and to fail over to the secondary replication channel when necessary to keep the replica cluster synchronized with the source. NDB Cluster is not designed to perform such monitoring on its own; for this, an external application is required.

The source SQL node issues a “gap” event when connecting or reconnecting to the source cluster. (A gap event is a type of “incident event,” which indicates an incident that occurs that affects the contents of the database but that cannot easily be represented as a set of changes. Examples of incidents are server failures, database resynchronization, some software updates, and some hardware changes.) When the replica encounters a gap in the replication log, it stops with an error message. This message is available in the output of `SHOW SLAVE STATUS`, and indicates that the SQL thread has stopped due to an incident registered in the replication stream, and that manual intervention is required. See Section 18.6.8, “Implementing Failover with NDB Cluster Replication”, for more information about what to do in such circumstances.

Important

Because NDB Cluster is not designed on its own to monitor replication status or provide failover, if high availability is a requirement for the replica server or cluster, then you must set up multiple replication lines, monitor the source `mysqld` on the primary replication line, and be prepared fail over to a secondary line if and as necessary. This must be done manually, or possibly by means of a third-party application. For information about implementing this type of setup, see Section 18.6.7, “Using Two Replication Channels for NDB Cluster Replication”, and Section 18.6.8, “Implementing Failover with NDB Cluster Replication”.

If you are replicating from a standalone MySQL server to an NDB Cluster, one channel is usually sufficient.

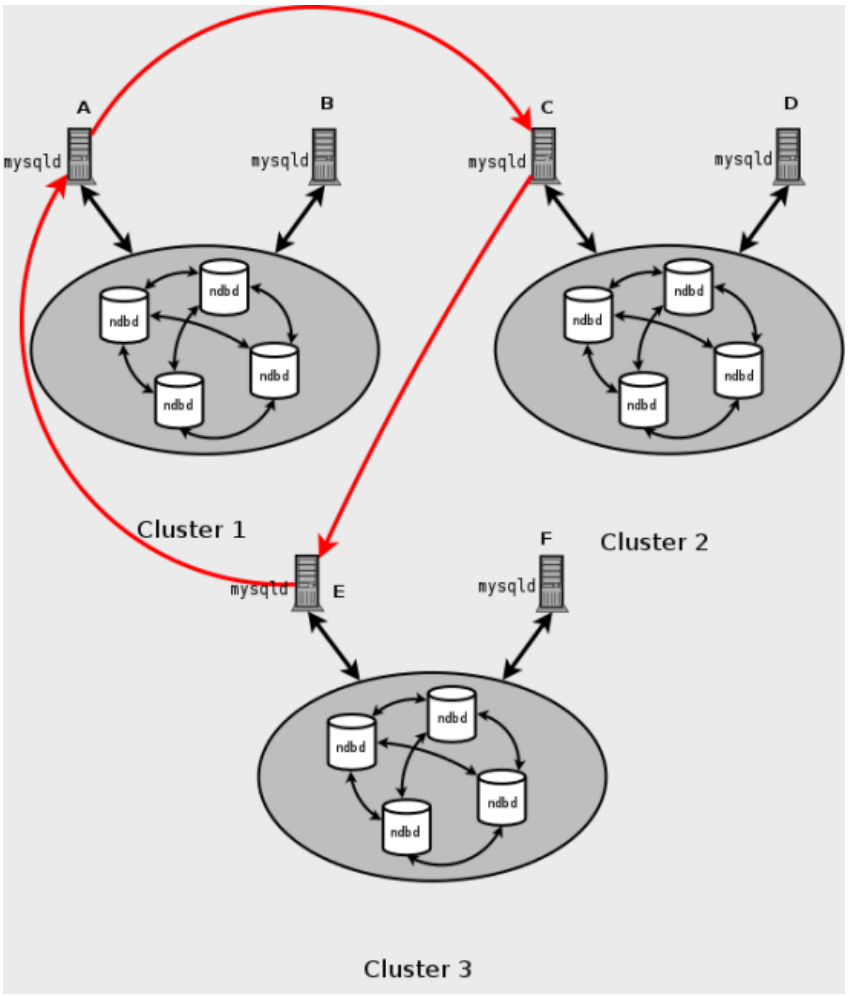
Circular replication. NDB Cluster Replication supports circular replication, as shown in the next example. The replication setup involves three NDB Clusters numbered 1, 2, and 3, in which Cluster 1 acts as the replication source for Cluster 2, Cluster 2 acts as the source for Cluster 3, and Cluster 3 acts as the source for Cluster 1, thus completing the circle. Each NDB Cluster has two SQL nodes, with SQL nodes A and B belonging to Cluster 1, SQL nodes C and D belonging to Cluster 2, and SQL nodes E and F belonging to Cluster 3.

Circular replication using these clusters is supported as long as the following conditions are met:

- The SQL nodes on all source and replica clusters are the same.
- All SQL nodes acting as sources and replicas are started with the `log_slave_updates` system variable enabled.

This type of circular replication setup is shown in the following diagram:

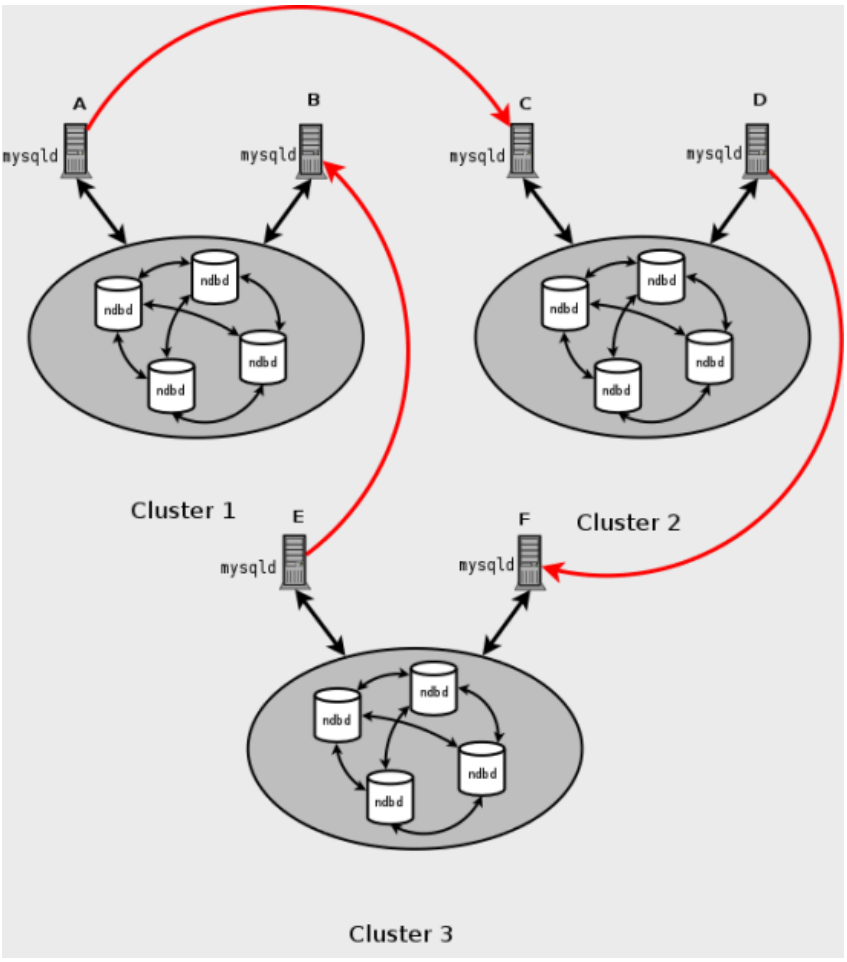
Figure 18.23 NDB Cluster Circular Replication With All Sources As Replicas



In this scenario, SQL node A in Cluster 1 replicates to SQL node C in Cluster 2; SQL node C replicates to SQL node E in Cluster 3; SQL node E replicates to SQL node A. In other words, the replication line (indicated by the curved arrows in the diagram) directly connects all SQL nodes used as sources and replicas.

It should also be possible to set up circular replication in which not all source SQL nodes are also replicas, as shown here:

Figure 18.24 NDB Cluster Circular Replication Where Not All Sources Are Replicas



In this case, different SQL nodes in each cluster are used as sources and replicas. However, you must *not* start any of the SQL nodes with the `log_slave_updates` system variable enabled. This type of circular replication scheme for NDB Cluster, in which the line of replication (again indicated by the curved arrows in the diagram) is discontinuous, should be possible, but it should be noted that it has not yet been thoroughly tested and must therefore still be considered experimental.

Note

The `NDB` storage engine uses *idempotent execution mode*, which suppresses duplicate-key and other errors that otherwise break circular replication of NDB Cluster. This is equivalent to setting the global `slave_exec_mode` system variable to `IDEMPOTENT`, although this is not necessary in NDB Cluster replication, since NDB Cluster sets this variable automatically and ignores any attempts to set it explicitly.

NDB Cluster replication and primary keys. In the event of a node failure, errors in replication of `NDB` tables without primary keys can still occur, due to the possibility of duplicate rows being inserted in such cases. For this reason, it is highly recommended that all `NDB` tables being replicated have explicit primary keys.

NDB Cluster Replication and Unique Keys. In older versions of NDB Cluster, operations that updated values of unique key columns of `NDB` tables could result in duplicate-key errors when replicated. This issue is solved for replication between `NDB` tables by deferring unique key checks until after all table row updates have been performed.

Deferring constraints in this way is currently supported only by NDB. Thus, updates of unique keys when replicating from NDB to a different storage engine such as InnoDB or MyISAM are still not supported.

The problem encountered when replicating without deferred checking of unique key updates can be illustrated using NDB table such as t, is created and populated on the source (and transmitted to a replica that does not support deferred unique key updates) as shown here:

```
CREATE TABLE t (
    p INT PRIMARY KEY,
    c INT,
    UNIQUE KEY u (c)
)   ENGINE NDB;

INSERT INTO t
    VALUES (1,1), (2,2), (3,3), (4,4), (5,5);
```

The following UPDATE statement on t succeeds on the source, since the rows affected are processed in the order determined by the ORDER BY option, performed over the entire table:

```
UPDATE t SET c = c - 1 ORDER BY p;
```

The same statement fails with a duplicate key error or other constraint violation on the replica, because the ordering of the row updates is performed for one partition at a time, rather than for the table as a whole.

Note

Every NDB table is implicitly partitioned by key when it is created. See Section 19.2.5, “KEY Partitioning”, for more information.

GTIDs not supported. Replication using global transaction IDs is not compatible with the NDB storage engine, and is not supported. Enabling GTIDs is likely to cause NDB Cluster Replication to fail.

Multithreaded replicas not supported. NDB Cluster does not support multithreaded replicas, and setting related system variables such as slave_parallel_workers, slave_checkpoint_group, and slave_checkpoint_group (or the equivalent mysqld startup options) has no effect.

This is because the replica may not be able to separate transactions occurring in one database from those in another if they are written within the same epoch. In addition, every transaction handled by the NDB storage engine involves at least two databases—the target database and the mysql system database—due to the requirement for updating the mysql.ndb_apply_status table (see Section 18.6.4, “NDB Cluster Replication Schema and Tables”). This in turn breaks the requirement for multithreading that the transaction is specific to a given database.

Restarting with --initial. Restarting the cluster with the --initial option causes the sequence of GCI and epoch numbers to start over from 0. (This is generally true of NDB Cluster and not limited to replication scenarios involving Cluster.) The MySQL servers involved in replication should in this case be restarted. After this, you should use the RESET MASTER and RESET SLAVE statements to clear the invalid ndb_binlog_index and ndb_apply_status tables, respectively.

Replication from NDB to other storage engines. It is possible to replicate an NDB table on the source to a table using a different storage engine on the replica, taking into account the restrictions listed here:

- Multi-source and circular replication are not supported (tables on both the source and the replica must use the NDB storage engine for this to work).
- Using a storage engine which does not perform binary logging for tables on the replica requires special handling.
- Use of a nontransactional storage engine for tables on the replica also requires special handling.

- The source `mysqld` must be started with `--ndb-log-update-as-write=0` or `--ndb-log-update-as-write=OFF`.

The next few paragraphs provide additional information about each of the issues just described.

Multiple sources not supported when replicating NDB to other storage engines. For replication from `NDB` to a different storage engine, the relationship between the two databases must be one-to-one. This means that bidirectional or circular replication is not supported between NDB Cluster and other storage engines.

In addition, it is not possible to configure more than one replication channel when replicating between `NDB` and a different storage engine. (An NDB Cluster database *can* simultaneously replicate to multiple NDB Cluster databases.) If the source uses `NDB` tables, it is still possible to have more than one MySQL Server maintain a binary log of all changes, but for the replica to change sources (fail over), the new source-replica relationship must be explicitly defined on the replica.

Replicating NDB tables to a storage engine that does not perform binary logging. If you attempt to replicate from an NDB Cluster to a replica that uses a storage engine that does not handle its own binary logging, the replication process aborts with the error `Binary logging not possible ... Statement cannot be written atomically since more than one engine involved and at least one engine is self-logging` (Error `1595`). It is possible to work around this issue in one of the following ways:

- **Turn off binary logging on the replica.** This can be accomplished by setting `sql_log_bin = 0`.
- **Change the storage engine used for the mysql.ndb_apply_status table.** Causing this table to use an engine that does not handle its own binary logging can also eliminate the conflict. This can be done by issuing a statement such as `ALTER TABLE mysql.ndb_apply_status ENGINE=MyISAM` on the replica. It is safe to do this when using a storage engine other than `NDB` on the replica, since you do not need to worry about keeping multiple replicas synchronized.
- **Filter out changes to the mysql.ndb_apply_status table on the replica.** This can be done by starting the replica with `--replicate-ignore-table=mysql.ndb_apply_status`. If you need for other tables to be ignored by replication, you might wish to use an appropriate `--replicate-wild-ignore-table` option instead.

Important

You should *not* disable replication or binary logging of `mysql.ndb_apply_status` or change the storage engine used for this table when replicating from one NDB Cluster to another. See Replication and binary log filtering rules with replication between NDB Clusters, for details.

Replication from NDB to a nontransactional storage engine. When replicating from `NDB` to a nontransactional storage engine such as `MyISAM`, you may encounter unnecessary duplicate key errors when replicating `INSERT ... ON DUPLICATE KEY UPDATE` statements. You can suppress these by using `--ndb-log-update-as-write=0`, which forces updates to be logged as writes, rather than as updates.

Replication and binary log filtering rules with replication between NDB Clusters. If you are using any of the options `--replicate-do-*`, `--replicate-ignore-*`, `--binlog-do-db`, or `--binlog-ignore-db` to filter databases or tables being replicated, you must take care not to block replication or binary logging of the `mysql.ndb_apply_status`, which is required for replication between NDB Clusters to operate properly. In particular, you must keep in mind the following:

1. Using `--replicate-do-db=`*`db_name`* (and no other `--replicate-do-*` or `--replicate-ignore-*` options) means that *only* tables in database *`db_name`* are replicated. In this case, you should also use `--replicate-do-db=mysql`, `--binlog-do-db=mysql`, or `--replicate-`

`do-table=mysql.ndb_apply_status` to ensure that `mysql.ndb_apply_status` is populated on replicas.

Using `--binlog-do-db=db_name` (and no other `--binlog-do-db` options) means that changes *only* to tables in database *`db_name`* are written to the binary log. In this case, you should also use `--replicate-do-db=mysql`, `--binlog-do-db=mysql`, or `--replicate-do-table=mysql.ndb_apply_status` to ensure that `mysql.ndb_apply_status` is populated on replicas.

2. Using `--replicate-ignore-db=mysql` means that no tables in the `mysql` database are replicated. In this case, you should also use `--replicate-do-table=mysql.ndb_apply_status` to ensure that `mysql.ndb_apply_status` is replicated.

   Using `--binlog-ignore-db=mysql` means that no changes to tables in the `mysql` database are written to the binary log. In this case, you should also use `--replicate-do-table=mysql.ndb_apply_status` to ensure that `mysql.ndb_apply_status` is replicated.

You should also remember that each replication rule requires the following:

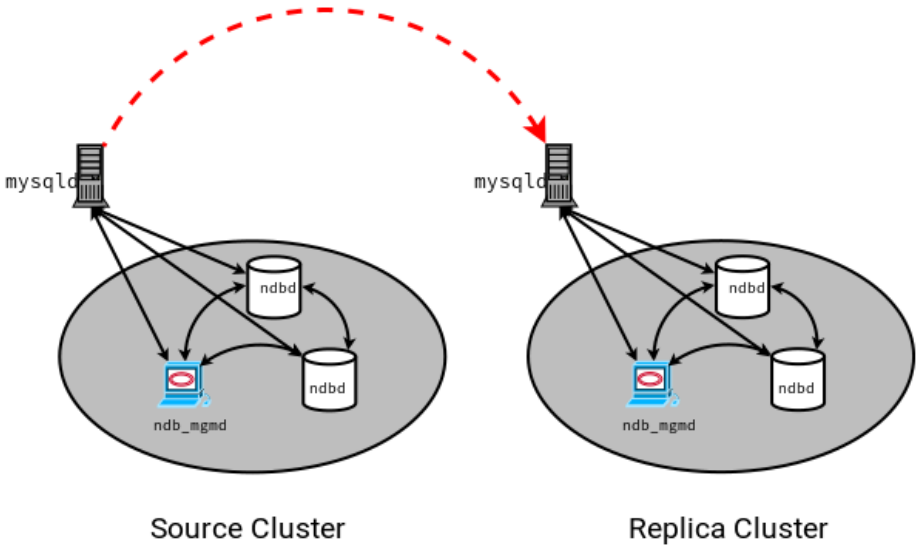
1. Its own `--replicate-do-*` or `--replicate-ignore-*` option, and that multiple rules cannot be expressed in a single replication filtering option. For information about these rules, see Section 17.1.4, “Replication and Binary Logging Options and Variables”.

2. Its own `--binlog-do-db` or `--binlog-ignore-db` option, and that multiple rules cannot be expressed in a single binary log filtering option. For information about these rules, see Section 5.4.4, “The Binary Log”.

If you are replicating an NDB Cluster to a replica that uses a storage engine other than `NDB`, the considerations just given previously may not apply, as discussed elsewhere in this section.

NDB Cluster Replication and IPv6. While the NDB API and MGM API (and thus data nodes and management nodes) do not support IPv6 in NDB 7.3 and 7.4, MySQL Servers—including those acting as SQL nodes in an NDB Cluster—can use IPv6 to contact other MySQL Servers. This means that you can replicate between NDB Clusters using IPv6 to connect the source and replica SQL nodes as shown by the dotted arrow in the following diagram:

Figure 18.25 Replication Between SQL Nodes Connected Using IPv6



All connections originating *within* the NDB Cluster —represented in the preceding diagram by solid arrows—must use IPv4. In other words, all NDB Cluster data nodes, management servers, and

management clients must be accessible from one another using IPv4. In addition, SQL nodes must use IPv4 to communicate with the cluster.

Since there is currently no support in the NDB and MGM APIs for IPv6, any applications written using these APIs must also make all connections using IPv4.

Attribute promotion and demotion. NDB Cluster Replication includes support for attribute promotion and demotion. The implementation of the latter distinguishes between lossy and non-lossy type conversions, and their use on the replica can be controlled by setting the `slave_type_conversions` global server system variable.

For more information about attribute promotion and demotion in NDB Cluster, see Row-based replication: attribute promotion and demotion.

18.6.4 NDB Cluster Replication Schema and Tables

Replication in NDB Cluster makes use of a number of dedicated tables in the `mysql` database on each MySQL Server instance acting as an SQL node in both the cluster being replicated and in the replica. This is true regardless of whether the replica is a single server or a cluster. These tables are created during the MySQL installation process by the `mysql_install_db` script, and include a table for storing the binary log's indexing data. Since the `ndb_binlog_index` table is local to each MySQL server and does not participate in clustering, it uses the `MyISAM` storage engine. This means that it must be created separately on each `mysqld` participating in the source cluster. (The binary log itself contains updates from all MySQL servers in the cluster to be replicated.) This table is defined as follows:

```
CREATE TABLE `ndb_binlog_index` (
    `Position` BIGINT(20) UNSIGNED NOT NULL,
    `File` VARCHAR(255) NOT NULL,
    `epoch` BIGINT(20) UNSIGNED NOT NULL,
    `inserts` INT(10) UNSIGNED NOT NULL,
    `updates` INT(10) UNSIGNED NOT NULL,
    `deletes` INT(10) UNSIGNED NOT NULL,
    `schemaops` INT(10) UNSIGNED NOT NULL,
    `orig_server_id` INT(10) UNSIGNED NOT NULL,
    `orig_epoch` BIGINT(20) UNSIGNED NOT NULL,
    `gci` INT(10) UNSIGNED NOT NULL,
    `next_position` bigint(20) unsigned NOT NULL,
    `next_file` varchar(255) NOT NULL,
    PRIMARY KEY (`epoch`,`orig_server_id`,`orig_epoch`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

The size of this table is dependent on the number of epochs per binary log file and the number of binary log files. The number of epochs per binary log file normally depends on the amount of binary log generated per epoch and the size of the binary log file, with smaller epochs resulting in more epochs per file. You should be aware that empty epochs produce inserts to the `ndb_binlog_index` table, even when the `--ndb-log-empty-epochs` option is `OFF`, meaning that the number of entries per file depends on the length of time that the file is in use; this relationship can be represented by the formula shown here:

```
[number of epochs per file] = [time spent per file] / TimeBetweenEpochs
```

A busy NDB Cluster writes to the binary log regularly and presumably rotates binary log files more quickly than a quiet one. This means that a “quiet” NDB Cluster with `--ndb-log-empty-epochs=ON` can actually have a much higher number of `ndb_binlog_index` rows per file than one with a great deal of activity.

When `mysqld` is started with the `--ndb-log-orig` option, the `orig_server_id` and `orig_epoch` columns store, respectively, the ID of the server on which the event originated and the epoch in which the event took place on the originating server, which is useful in NDB Cluster replication setups employing multiple sources. The `SELECT` statement used to find the closest binary log position to the highest applied epoch on the replica in a multi-source setup (see Section 18.6.10, “NDB Cluster Replication: Bidirectional and Circular Replication”) employs these two columns, which are not indexed.

This can lead to performance issues when trying to fail over, since the query must perform a table scan, especially when the source has been running with `--ndb-log-empty-epochs=ON`. You can improve multi-source failover times by adding an index to these columns, as shown here:

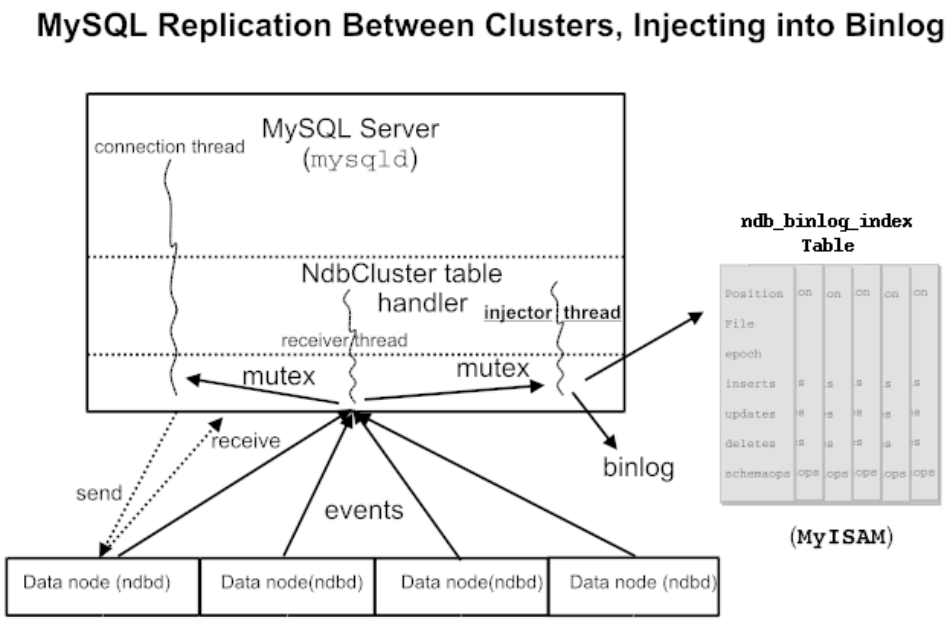
```
ALTER TABLE mysql.ndb_binlog_index
    ADD INDEX orig_lookup USING BTREE (orig_server_id, orig_epoch);
```

Adding this index provides no benefit when replicating from a single source to a single replica, since the query used to get the binary log position in such cases makes no use of `orig_server_id` or `orig_epoch`.

See Section 18.6.8, “Implementing Failover with NDB Cluster Replication”, for more information about using the `next_position` and `next_file` columns.

The following figure shows the relationship of the NDB Cluster replication source server, its binary log injector thread, and the `mysql.ndb_binlog_index` table.

Figure 18.26 The Replication Source Cluster



An additional table, named `ndb_apply_status`, is used to keep a record of the operations that have been replicated from the source to the replica. Unlike the case with `ndb_binlog_index`, the data in this table is not specific to any one SQL node in the (replica) cluster, and so `ndb_apply_status` can use the `NDBCLUSTER` storage engine, as shown here:

```
CREATE TABLE `ndb_apply_status` (
    `server_id`   INT(10) UNSIGNED NOT NULL,
    `epoch`       BIGINT(20) UNSIGNED NOT NULL,
    `log_name`    VARCHAR(255) CHARACTER SET latin1 COLLATE latin1_bin NOT NULL,
    `start_pos`   BIGINT(20) UNSIGNED NOT NULL,
    `end_pos`     BIGINT(20) UNSIGNED NOT NULL,
    PRIMARY KEY (`server_id`) USING HASH
) ENGINE=NDBCLUSTER   DEFAULT CHARSET=latin1;
```

The `ndb_apply_status` table is populated only on replicas, which means that, on the source, this table never contains any rows; thus, there is no need to allot any `DataMemory` or `IndexMemory` to `ndb_apply_status` there.

Because this table is populated from data originating on the source, it should be allowed to replicate; any replication filtering or binary log filtering rules that inadvertently prevent the replica from updating `ndb_apply_status`, or that prevent the source from writing into the binary log may prevent replication between clusters from operating properly. For more information about potential problems arising from such filtering rules, see Replication and binary log filtering rules with replication between NDB Clusters.

The `ndb_binlog_index` and `ndb_apply_status` tables are created in the `mysql` database because they should not be explicitly replicated by the user. User intervention is normally not required to create or maintain either of these tables, since both are maintained by the `NDB` binary log (binlog) injector thread. This keeps the source `mysqld` process updated to changes performed by the `NDB` storage engine. The `NDB` *binlog injector thread* receives events directly from the `NDB` storage engine. The `NDB` injector is responsible for capturing all the data events within the cluster, and ensures that all events which change, insert, or delete data are recorded in the `ndb_binlog_index` table. The replica I/O thread transfers the events from the source's binary log to the replica's relay log.

Even though `ndb_binlog_index` and `ndb_apply_status` are created and maintained automatically, it is advisable to check for the existence and integrity of these tables as an initial step in preparing an NDB Cluster for replication. It is possible to view event data recorded in the binary log by querying the `mysql.ndb_binlog_index` table directly on the source. This can be also be accomplished using the `SHOW BINLOG EVENTS` statement on either the source or replica SQL node. (See Section 13.7.5.3, “SHOW BINLOG EVENTS Statement”.)

You can also obtain useful information from the output of `SHOW ENGINE NDB STATUS`.

Note

When performing schema changes on `NDB` tables, applications should wait until the `ALTER TABLE` statement has returned in the MySQL client connection that issued the statement before attempting to use the updated definition of the table.

If the `ndb_apply_status` table does not exist on the replica, `ndb_restore` re-creates it.

Conflict resolution for NDB Cluster Replication requires the presence of an additional `mysql.ndb_replication` table. Currently, this table must be created manually. For information about how to do this, see Section 18.6.11, “NDB Cluster Replication Conflict Resolution”.

18.6.5 Preparing the NDB Cluster for Replication

Preparing the NDB Cluster for replication consists of the following steps:

1. Check all MySQL servers for version compatibility (see Section 18.6.2, “General Requirements for NDB Cluster Replication”).
2. Create a replication account on the source Cluster with the appropriate privileges, using the following SQL statement:

```
mysqlM> GRANT REPLICATION SLAVE
     -> ON *.* TO 'replica_user'@'replica_host'
     -> IDENTIFIED BY 'replica_password';
```

In the previous statement, *`replica_user`* is the replication account user name, *`replica_host`* is the host name or IP address of the replica, and *`replica_password`* is the password to assign to this account.

For example, to create a replica user account with the name `myreplica`, logging in from the host named `replica-host`, and using the password `53cr37`, use the following `GRANT` statement:

```
mysqlM> GRANT REPLICATION SLAVE
     -> ON *.* TO 'myreplica'@'replica-host'
     -> IDENTIFIED BY '53cr37';
```

For security reasons, it is preferable to use a unique user account—not employed for any other purpose—for the replication account.

3. Set up the replica to use the source. Using the `mysql` client, this can be accomplished with the the following `CHANGE MASTER TO` statement:

```
mysqlR> CHANGE MASTER TO
     -> MASTER_HOST='source_host',
     -> MASTER_PORT=source_port,
     -> MASTER_USER='replica_user',
     -> MASTER_PASSWORD='replica_password';
```

In the previous statement, *`source_host`* is the host name or IP address of the replication source, *`source_port`* is the port for the replica to use when connecting to the source, *`replica_user`* is the user name set up for the replica on the source, and *`replica_password`* is the password set for that user account in the previous step.

For example, to tell the replica to use the MySQL server whose host name is `rep-source` with the replication account created in the previous step, use the following statement:

```
mysqlR> CHANGE MASTER TO
     -> MASTER_HOST='rep-source',
     -> MASTER_PORT=3306,
     -> MASTER_USER='myreplica',
     -> MASTER_PASSWORD='53cr37';
```

For a complete list of options that can be used with this statement, see Section 13.4.2.1, “CHANGE MASTER TO Statement”.

To provide replication backup capability, you also need to add an `--ndb-connectstring` option to the replica's `my.cnf` file prior to starting the replication process. See Section 18.6.9, “NDB Cluster Backups With NDB Cluster Replication”, for details.

For additional options that can be set in `my.cnf` for replicas, see Section 17.1.4, “Replication and Binary Logging Options and Variables”.

4. If the source cluster is already in use, you can create a backup of the source and load this onto the replica to cut down on the amount of time required for the replica to synchronize itself with the source. If the replica is also running NDB Cluster, this can be accomplished using the backup and restore procedure described in Section 18.6.9, “NDB Cluster Backups With NDB Cluster Replication”.

```
ndb-connectstring=management_host[:port]
```

In the event that you are *not* using NDB Cluster on the replica, you can create a backup with this command on the source:

```
shellS> mysqldump --master-data=1
```

Then import the resulting data dump onto the replica by copying the dump file over to it. After this, you can use the `mysql` client to import the data from the dumpfile into the replica database as shown here, where *`dump_file`* is the name of the file that was generated using `mysqldump` on the source, and *`db_name`* is the name of the database to be replicated:

```
shellR> mysql -u root -p db_name < dump_file
```

For a complete list of options to use with `mysqldump`, see Section 4.5.4, “`mysqldump` — A Database Backup Program”.

Note

If you copy the data to the replica in this fashion, you should make sure that the replica is started with the `--skip-slave-start` option on the

command line, or else include `skip-slave-start` in the replica's `my.cnf` file to keep it from trying to connect to the source to begin replicating before all the data has been loaded. Once the data loading has completed, follow the additional steps outlined in the next two sections.

5. Ensure that each MySQL server acting as a replication source is assigned a unique server ID, and has binary logging enabled, using the row-based format. (See Section 17.1.2, “Replication Formats”.) In addition, we recommend enabling the `slave_allow_batching` system variable, and possibly increasing the values used with the `--ndb-batch-size` and `--ndb-blob-write-batch-bytes` options as well. All of these options can be set either in the source server's `my.cnf` file, or on the command line when starting the source `mysqld` process. See Section 18.6.6, “Starting NDB Cluster Replication (Single Replication Channel)”, for more information.

18.6.6 Starting NDB Cluster Replication (Single Replication Channel)

This section outlines the procedure for starting NDB Cluster replication using a single replication channel.

1. Start the MySQL replication source server by issuing this command:

```
shellS> mysqld --ndbcluster --server-id=id \
        --log-bin --ndb-log-bin &
```

In the previous statement, *`id`* is this server's unique ID (see Section 18.6.2, “General Requirements for NDB Cluster Replication”). This starts the server's `mysqld` process with binary logging enabled using the proper logging format.

Note

You can also start the source with `--binlog-format=MIXED`, in which case row-based replication is used automatically when replicating between clusters. Statement-based binary logging is not supported for NDB Cluster Replication (see Section 18.6.2, “General Requirements for NDB Cluster Replication”).

2. Start the MySQL replica server as shown here:

```
shellR> mysqld --ndbcluster --server-id=id &
```

In the command just shown, *`id`* is the replica server's unique ID. It is not necessary to enable logging on the replica.

Note

You should use the `--skip-slave-start` option with this command or else you should include `skip-slave-start` in the replica server's `my.cnf` file, unless you want replication to begin immediately. With the use of this option, the start of replication is delayed until the appropriate `START SLAVE` statement has been issued, as explained in Step 4 below.

3. It is necessary to synchronize the replica server with the source server's replication binary log. If binary logging has not previously been running on the source, run the following statement on the replica:

```
mysqlR> CHANGE MASTER TO
     -> MASTER_LOG_FILE='',
     -> MASTER_LOG_POS=4;
```

This instructs the replica to begin reading the source server's binary log from the log's starting point. Otherwise—that is, if you are loading data from the source using a backup—see Section 18.6.8, “Implementing Failover with NDB Cluster Replication”, for information on how to obtain the correct values to use for `MASTER_LOG_FILE` and `MASTER_LOG_POS` in such cases.

4. Finally, instruct the replica to begin applying replication by issuing this command from the `mysql` client on the replica:

   ```
   mysqlR> START SLAVE;
   ```

   This also initiates the transmission of data and changes from the source to the replica.

It is also possible to use two replication channels, in a manner similar to the procedure described in the next section; the differences between this and using a single replication channel are covered in Section 18.6.7, “Using Two Replication Channels for NDB Cluster Replication”.

It is also possible to improve cluster replication performance by enabling *batched updates*. This can be accomplished by setting the `slave_allow_batching` system variable on the replicas' `mysqld` processes. Normally, updates are applied as soon as they are received. However, the use of batching causes updates to be applied in batches of 32 KB each; this can result in higher throughput and less CPU usage, particularly where individual updates are relatively small.

Note

Batching works on a per-epoch basis; updates belonging to more than one transaction can be sent as part of the same batch.

All outstanding updates are applied when the end of an epoch is reached, even if the updates total less than 32 KB.

Batching can be turned on and off at runtime. To activate it at runtime, you can use either of these two statements:

```
SET GLOBAL slave_allow_batching = 1;
SET GLOBAL slave_allow_batching = ON;
```

If a particular batch causes problems (such as a statement whose effects do not appear to be replicated correctly), batching can be deactivated using either of the following statements:

```
SET GLOBAL slave_allow_batching = 0;
SET GLOBAL slave_allow_batching = OFF;
```

You can check whether batching is currently being used by means of an appropriate `SHOW VARIABLES` statement, like this one:

```
mysql> SHOW VARIABLES LIKE 'slave%';
+---------------------------+-------+
| Variable_name             | Value |
+---------------------------+-------+
| slave_allow_batching      | ON    |
| slave_compressed_protocol | OFF   |
| slave_load_tmpdir         | /tmp  |
| slave_net_timeout         | 3600  |
| slave_skip_errors         | OFF   |
| slave_transaction_retries | 10    |
+---------------------------+-------+
6 rows in set (0.00 sec)
```

18.6.7 Using Two Replication Channels for NDB Cluster Replication

In a more complete example scenario, we envision two replication channels to provide redundancy and thereby guard against possible failure of a single replication channel. This requires a total of four replication servers, two source servers on the source cluster and two replica servers on the replica cluster. For purposes of the discussion that follows, we assume that unique identifiers are assigned as shown here:

Table 18.59 NDB Cluster replication servers described in the text

Server ID	Description
1	Source - primary replication channel (*S*)

Server ID	Description
2	Source - secondary replication channel (*S'*)
3	Replica - primary replication channel (*R*)
4	replica - secondary replication channel (*R'*)

Setting up replication with two channels is not radically different from setting up a single replication channel. First, the `mysqld` processes for the primary and secondary replication source servers must be started, followed by those for the primary and secondary replicas. The replication processes can be initiated by issuing the `START SLAVE` statement on each of the replicas. The commands and the order in which they need to be issued are shown here:

1. Start the primary replication source:

```
shellS> mysqld --ndbcluster --server-id=1 \
               --log-bin &
```

2. Start the secondary replication source:

```
shellS'> mysqld --ndbcluster --server-id=2 \
               --log-bin &
```

3. Start the primary replica server:

```
shellR> mysqld --ndbcluster --server-id=3 \
               --skip-slave-start &
```

4. Start the secondary replica server:

```
shellR'> mysqld --ndbcluster --server-id=4 \
                --skip-slave-start &
```

5. Finally, initiate replication on the primary channel by executing the `START SLAVE` statement on the primary replica as shown here:

```
mysqlR> START SLAVE;
```

Warning

Only the primary channel must be started at this point. The secondary replication channel needs to be started only in the event that the primary replication channel fails, as described in Section 18.6.8, “Implementing Failover with NDB Cluster Replication”. Running multiple replication channels simultaneously can result in unwanted duplicate records being created on the replicas.

As mentioned previously, it is not necessary to enable binary logging on the replicas.

18.6.8 Implementing Failover with NDB Cluster Replication

In the event that the primary Cluster replication process fails, it is possible to switch over to the secondary replication channel. The following procedure describes the steps required to accomplish this.

1. Obtain the time of the most recent global checkpoint (GCP). That is, you need to determine the most recent epoch from the `ndb_apply_status` table on the replica cluster, which can be found using the following query:

```
mysqlR'> SELECT @latest:=MAX(epoch)
      ->        FROM mysql.ndb_apply_status;
```

In a circular replication topology, with a source and a replica running on each host, when you are using `ndb_log_apply_status=1`, NDB Cluster epochs are written in the replicas' binary logs. This means that the `ndb_apply_status` table contains information for the replica on this host as

well as for any other host which acts as a replica of the replication source server running on this host.

In this case, you need to determine the latest epoch on this replica to the exclusion of any epochs from any other replicas in this replica's binary log that were not listed in the `IGNORE_SERVER_IDS` options of the `CHANGE MASTER TO` statement used to set up this replica. The reason for excluding such epochs is that rows in the `mysql.ndb_apply_status` table whose server IDs have a match in the `IGNORE_SERVER_IDS` list from the `CHANGE MASTER TO` statement used to prepare this replicas's source are also considered to be from local servers, in addition to those having the replica's own server ID. You can retrieve this list as `Replicate_Ignore_Server_Ids` from the output of `SHOW SLAVE STATUS`. We assume that you have obtained this list and are substituting it for *`ignore_server_ids`* in the query shown here, which like the previous version of the query, selects the greatest epoch into a variable named `@latest`:

```
mysqlR'> SELECT @latest:=MAX(epoch)
      ->        FROM mysql.ndb_apply_status
      ->        WHERE server_id NOT IN (ignore_server_ids);
```

In some cases, it may be simpler or more efficient (or both) to use a list of the server IDs to be included and `server_id IN server_id_list` in the `WHERE` condition of the preceding query.

2. Using the information obtained from the query shown in Step 1, obtain the corresponding records from the `ndb_binlog_index` table on the source cluster.

   You can use the following query to obtain the needed records from the `ndb_binlog_index` table on the source:

   ```
   mysqlS'> SELECT
         ->     @file:=SUBSTRING_INDEX(next_file, '/', -1),
         ->     @pos:=next_position
         -> FROM mysql.ndb_binlog_index
         -> WHERE epoch >= @latest
         -> ORDER BY epoch ASC LIMIT 1;
   ```

   These are the records saved on the source since the failure of the primary replication channel. We have employed a user variable `@latest` here to represent the value obtained in Step 1. Of course, it is not possible for one `mysqld` instance to access user variables set on another server instance directly. These values must be “plugged in” to the second query manually or by an application.

   Important

   You must ensure that the replica `mysqld` is started with `--slave-skip-errors=ddl_exist_errors` before executing `START SLAVE`. Otherwise, replication may stop with duplicate DDL errors.

3. Now it is possible to synchronize the secondary channel by running the following query on the secondary replica server:

   ```
   mysqlR'> CHANGE MASTER TO
         ->     MASTER_LOG_FILE='@file',
         ->     MASTER_LOG_POS=@pos;
   ```

   Again we have employed user variables (in this case `@file` and `@pos`) to represent the values obtained in Step 2 and applied in Step 3; in practice these values must be inserted manually or using an application that can access both of the servers involved.

   Note

   `@file` is a string value such as `'/var/log/mysql/replication-source-bin.00001'`, and so must be quoted when used in SQL or application code. However, the value represented by `@pos` must *not* be quoted. Although MySQL normally attempts to convert strings to numbers, this case is an exception.

4. You can now initiate replication on the secondary channel by issuing the appropriate statement on the secondary replica `mysqld`:

```
mysqlR'> START SLAVE;
```

Once the secondary replication channel is active, you can investigate the failure of the primary and effect repairs. The precise actions required to do this depend upon the reasons for which the primary channel failed.

Warning

The secondary replication channel is to be started only if and when the primary replication channel has failed. Running multiple replication channels simultaneously can result in unwanted duplicate records being created on the replicas.

If the failure is limited to a single server, it should in theory be possible to replicate from *`S`* to *`R'`*, or from *`S'`* to *`R`*.

18.6.9 NDB Cluster Backups With NDB Cluster Replication

This section discusses making backups and restoring from them using NDB Cluster replication. We assume that the replication servers have already been configured as covered previously (see Section 18.6.5, “Preparing the NDB Cluster for Replication”, and the sections immediately following). This having been done, the procedure for making a backup and then restoring from it is as follows:

1. There are two different methods by which the backup may be started.

   - **Method A.** This method requires that the cluster backup process was previously enabled on the source server, prior to starting the replication process. This can be done by including the following line in a `[mysql_cluster]` section in the `my.cnf file`, where *`management_host`* is the IP address or host name of the `NDB` management server for the source cluster, and *`port`* is the management server's port number:

     ```
     ndb-connectstring=management_host[:port]
     ```

     Note

     The port number needs to be specified only if the default port (1186) is not being used. See Section 18.2.3, “Initial Configuration of NDB Cluster”, for more information about ports and port allocation in NDB Cluster.

     In this case, the backup can be started by executing this statement on the replication source:

     ```
     shellS> ndb_mgm -e "START BACKUP"
     ```

   - **Method B.** If the `my.cnf` file does not specify where to find the management host, you can start the backup process by passing this information to the `NDB` management client as part of the `START BACKUP` command. This can be done as shown here, where *`management_host`* and *`port`* are the host name and port number of the management server:

     ```
     shellS> ndb_mgm management_host:port -e "START BACKUP"
     ```

     In our scenario as outlined earlier (see Section 18.6.5, “Preparing the NDB Cluster for Replication”), this would be executed as follows:

     ```
     shellS> ndb_mgm rep-source:1186 -e "START BACKUP"
     ```

2. Copy the cluster backup files to the replica that is being brought on line. Each system running an `ndbd` process for the source cluster has cluster backup files located on it, and *all* of these files must be copied to the replica to ensure a successful restore. The backup files can be copied into any directory on the computer where the replica's management host resides, as long as the MySQL and

NDB binaries have read permissions in that directory. In this case, we assume that these files have been copied into the directory `/var/BACKUPS/BACKUP-1`.

While it is not necessary that the replica cluster have the same number of `ndbd` processes (data nodes) as the source, it is highly recommended this number be the same. It *is* necessary that the replica be started with the `--skip-slave-start` option, to prevent premature startup of the replication process.

3. Create any databases on the replica cluster that are present on the source cluster and that are to be replicated.

Important

A `CREATE DATABASE` (or `CREATE SCHEMA`) statement corresponding to each database to be replicated must be executed on each SQL node in the replica cluster.

4. Reset the replica cluster using this statement in the `mysql` client:

```
mysqlR> RESET SLAVE;
```

5. You can now start the cluster restoration process on the replica using the `ndb_restore` command for each backup file in turn. For the first of these, it is necessary to include the `-m` option to restore the cluster metadata, as shown here:

```
shellR> ndb_restore -c replica_host:port -n node-id \
        -b backup-id -m -r dir
```

`dir` is the path to the directory where the backup files have been placed on the replica. For the `ndb_restore` commands corresponding to the remaining backup files, the `-m` option should *not* be used.

For restoring from a source cluster with four data nodes (as shown in the figure in Section 18.6, “NDB Cluster Replication”) where the backup files have been copied to the directory `/var/BACKUPS/BACKUP-1`, the proper sequence of commands to be executed on the replica might look like this:

```
shellR> ndb_restore -c replica-host:1186 -n 2 -b 1 -m \
        -r ./var/BACKUPS/BACKUP-1
shellR> ndb_restore -c replica-host:1186 -n 3 -b 1 \
        -r ./var/BACKUPS/BACKUP-1
shellR> ndb_restore -c replica-host:1186 -n 4 -b 1 \
        -r ./var/BACKUPS/BACKUP-1
shellR> ndb_restore -c replica-host:1186 -n 5 -b 1 -e \
        -r ./var/BACKUPS/BACKUP-1
```

Important

The `-e` (or `--restore-epoch`) option in the final invocation of `ndb_restore` in this example is required to make sure that the epoch is written to the replica's `mysql.ndb_apply_status` table. Without this information, the replica cannot synchronize properly with the source. (See Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”.)

6. Now you need to obtain the most recent epoch from the `ndb_apply_status` table on the replica (as discussed in Section 18.6.8, “Implementing Failover with NDB Cluster Replication”):

```
mysqlR> SELECT @latest:=MAX(epoch)
        FROM mysql.ndb_apply_status;
```

7. Using `@latest` as the epoch value obtained in the previous step, you can obtain the correct starting position `@pos` in the correct binary log file `@file` from the `mysql.ndb_binlog_index` table on the source using the query shown here:

```
mysqlS> SELECT
     ->     @file:=SUBSTRING_INDEX(File, '/', -1),
     ->     @pos:=Position
     -> FROM mysql.ndb_binlog_index
     -> WHERE epoch >= @latest
     -> ORDER BY epoch ASC LIMIT 1;
```

   In the event that there is currently no replication traffic, you can get this information by running `SHOW MASTER STATUS` on the source and using the value shown in the `Position` column of the output for the file whose name has the suffix with the greatest value for all files shown in the `File` column. In this case, you must determine which file this is and supply the name in the next step manually or by parsing the output with a script.

8. Using the values obtained in the previous step, you can now issue the appropriate `CHANGE MASTER TO` statement in the replica's `mysql` client:

```
mysqlR> CHANGE MASTER TO
     ->     MASTER_LOG_FILE='@file',
     ->     MASTER_LOG_POS=@pos;
```

9. Now that the replica knows from what point in which binary log file to start reading data from the source, you can cause the replica to begin replicating with this statement:

```
mysqlR> START SLAVE;
```

To perform a backup and restore on a second replication channel, it is necessary only to repeat these steps, substituting the host names and IDs of the secondary source and replica for those of the primary source and replica servers where appropriate, and running the preceding statements on them.

For additional information on performing Cluster backups and restoring Cluster from backups, see Section 18.5.8, “Online Backup of NDB Cluster”.

18.6.9.1 NDB Cluster Replication: Automating Synchronization of the Replica to the Source Binary Log

It is possible to automate much of the process described in the previous section (see Section 18.6.9, “NDB Cluster Backups With NDB Cluster Replication”). The following Perl script `reset-replica.pl` serves as an example of how you can do this.

```
#!/user/bin/perl -w

#  file: reset-replica.pl

#  Copyright (c) 2005, 2020, Oracle and/or its affiliates. All rights reserved.

#  This program is free software; you can redistribute it and/or modify
#  it under the terms of the GNU General Public License as published by
#  the Free Software Foundation; either version 2 of the License, or
#  (at your option) any later version.

#  This program is distributed in the hope that it will be useful,
#  but WITHOUT ANY WARRANTY; without even the implied warranty of
#  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
#  GNU General Public License for more details.

#  You should have received a copy of the GNU General Public License
#  along with this program; if not, write to:
#  Free Software Foundation, Inc.
#  59 Temple Place, Suite 330
#  Boston, MA 02111-1307 USA
#
#  Version 1.1
```

```
######################### Includes ###############################

use DBI;

######################### Globals ################################

my  $m_host='';
my  $m_port='';
my  $m_user='';
my  $m_pass='';
my  $s_host='';
my  $s_port='';
my  $s_user='';
my  $s_pass='';
my  $dbhM='';
my  $dbhS='';

######################### Sub Prototypes #########################

sub CollectCommandPromptInfo;
sub ConnectToDatabases;
sub DisconnectFromDatabases;
sub GetReplicaEpoch;
sub GetSourceInfo;
sub UpdateReplica;

######################### Program Main ###########################

CollectCommandPromptInfo;
ConnectToDatabases;
GetReplicaEpoch;
GetSourceInfo;
UpdateReplica;
DisconnectFromDatabases;

################### Collect Command Prompt Info ##################

sub CollectCommandPromptInfo
{
  ### Check that user has supplied correct number of command line args
  die "Usage:\n
       reset-replica >source MySQL host< >source MySQL port< \n
                     >source user< >source pass< >replica MySQL host< \n
                     >replica MySQL port< >replica user< >replica pass< \n
       All 8 arguments must be passed. Use BLANK for NULL passwords\n"
       unless @ARGV == 8;

  $m_host  =  $ARGV[0];
  $m_port  =  $ARGV[1];
  $m_user  =  $ARGV[2];
  $m_pass  =  $ARGV[3];
  $s_host  =  $ARGV[4];
  $s_port  =  $ARGV[5];
  $s_user  =  $ARGV[6];
  $s_pass  =  $ARGV[7];

  if ($m_pass eq "BLANK") { $m_pass = '';}
  if ($s_pass eq "BLANK") { $s_pass = '';}
}

###############  Make connections to both databases #############

sub ConnectToDatabases
{
  ### Connect to both source and replica cluster databases

  ### Connect to source
  $dbhM
    = DBI->connect(
    "dbi:mysql:database=mysql;host=$m_host;port=$m_port",
    "$m_user", "$m_pass")
```

```
      or die "Can't connect to source cluster MySQL process!
              Error: $DBI::errstr\n";

  ### Connect to replica
  $dbhS
    = DBI->connect(
          "dbi:mysql:database=mysql;host=$s_host",
          "$s_user", "$s_pass")
    or die "Can't connect to replica cluster MySQL process!
            Error: $DBI::errstr\n";
}

################  Disconnect from both databases ################

sub DisconnectFromDatabases
{
  ### Disconnect from source

  $dbhM->disconnect
  or warn " Disconnection failed: $DBI::errstr\n";

  ### Disconnect from replica

  $dbhS->disconnect
  or warn " Disconnection failed: $DBI::errstr\n";
}

######################  Find the last good GCI ##################

sub GetReplicaEpoch
{
  $sth = $dbhS->prepare("SELECT MAX(epoch)
                         FROM mysql.ndb_apply_status;")
      or die "Error while preparing to select epoch from replica: ",
             $dbhS->errstr;

  $sth->execute
      or die "Selecting epoch from replica error: ", $sth->errstr;

  $sth->bind_col (1, \$epoch);
  $sth->fetch;
  print "\tReplica epoch =  $epoch\n";
  $sth->finish;
}

#######  Find the position of the last GCI in the binary log ########

sub GetSourceInfo
{
  $sth = $dbhM->prepare("SELECT
                           SUBSTRING_INDEX(File, '/', -1), Position
                         FROM mysql.ndb_binlog_index
                         WHERE epoch > $epoch
                         ORDER BY epoch ASC LIMIT 1;")
      or die "Prepare to select from source error: ", $dbhM->errstr;

  $sth->execute
      or die "Selecting from source error: ", $sth->errstr;

  $sth->bind_col (1, \$binlog);
  $sth->bind_col (2, \$binpos);
  $sth->fetch;
  print "\tSource binary log file =  $binlog\n";
  print "\tSource binary log position =  $binpos\n";
  $sth->finish;
}

##########  Set the replica to process from that location #########

sub UpdateReplica
{
  $sth = $dbhS->prepare("CHANGE MASTER TO
```

```
                        MASTER_LOG_FILE='$binlog',
                        MASTER_LOG_POS=$binpos;")
      or die "Prepare to CHANGE MASTER error: ", $dbhS->errstr;

  $sth->execute
       or die "CHANGE MASTER on replica error: ", $sth->errstr;
  $sth->finish;
  print "\tReplica has been updated. You may now start the replica.\n";
}

# end reset-replica.pl
```

18.6.9.2 Point-In-Time Recovery Using NDB Cluster Replication

Point-in-time recovery—that is, recovery of data changes made since a given point in time—is performed after restoring a full backup that returns the server to its state when the backup was made. Performing point-in-time recovery of NDB Cluster tables with NDB Cluster and NDB Cluster Replication can be accomplished using a native `NDB` data backup (taken by issuing `CREATE BACKUP` in the `ndb_mgm` client) and restoring the `ndb_binlog_index` table (from a dump made using `mysqldump`).

To perform point-in-time recovery of NDB Cluster, it is necessary to follow the steps shown here:

1. Back up all `NDB` databases in the cluster, using the `START BACKUP` command in the `ndb_mgm` client (see Section 18.5.8, “Online Backup of NDB Cluster”).
2. At some later point, prior to restoring the cluster, make a backup of the `mysql.ndb_binlog_index` table. It is probably simplest to use `mysqldump` for this task. Also back up the binary log files at this time.

   This backup should be updated regularly—perhaps even hourly—depending on your needs.
3. (*Catastrophic failure or error occurs*.)
4. Locate the last known good backup.
5. Clear the data node file systems (using `ndbd --initial` or `ndbmtd --initial`).

   Note

   NDB Cluster Disk Data tablespace and log files are not removed by `--initial`. You must delete these manually.
6. Use `DROP TABLE` or `TRUNCATE TABLE` with the `mysql.ndb_binlog_index` table.
7. Execute `ndb_restore`, restoring all data. You must include the `--restore-epoch` option when you run `ndb_restore`, so that the `ndb_apply_status` table is populated correctly. (See Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”, for more information.)
8. Restore the `ndb_binlog_index` table from the output of `mysqldump` and restore the binary log files from backup, if necessary.
9. Find the epoch applied most recently—that is, the maximum `epoch` column value in the `ndb_apply_status` table—as the user variable `@LATEST_EPOCH` (emphasized):

   ```
   SELECT @LATEST_EPOCH:=MAX(epoch)
       FROM mysql.ndb_apply_status;
   ```
10. Find the latest binary log file (`@FIRST_FILE`) and position (`Position` column value) within this file that correspond to `@LATEST_EPOCH` in the `ndb_binlog_index` table:

    ```
    SELECT Position, @FIRST_FILE:=File
        FROM mysql.ndb_binlog_index
        WHERE epoch > @LATEST_EPOCH ORDER BY epoch ASC LIMIT 1;
    ```
11. Using `mysqlbinlog`, replay the binary log events from the given file and position up to the point of the failure. (See Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”.)

See also Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”, for more information about the binary log, replication, and incremental recovery.

18.6.10 NDB Cluster Replication: Bidirectional and Circular Replication

It is possible to use NDB Cluster for bidirectional replication between two clusters, as well as for circular replication between any number of clusters.

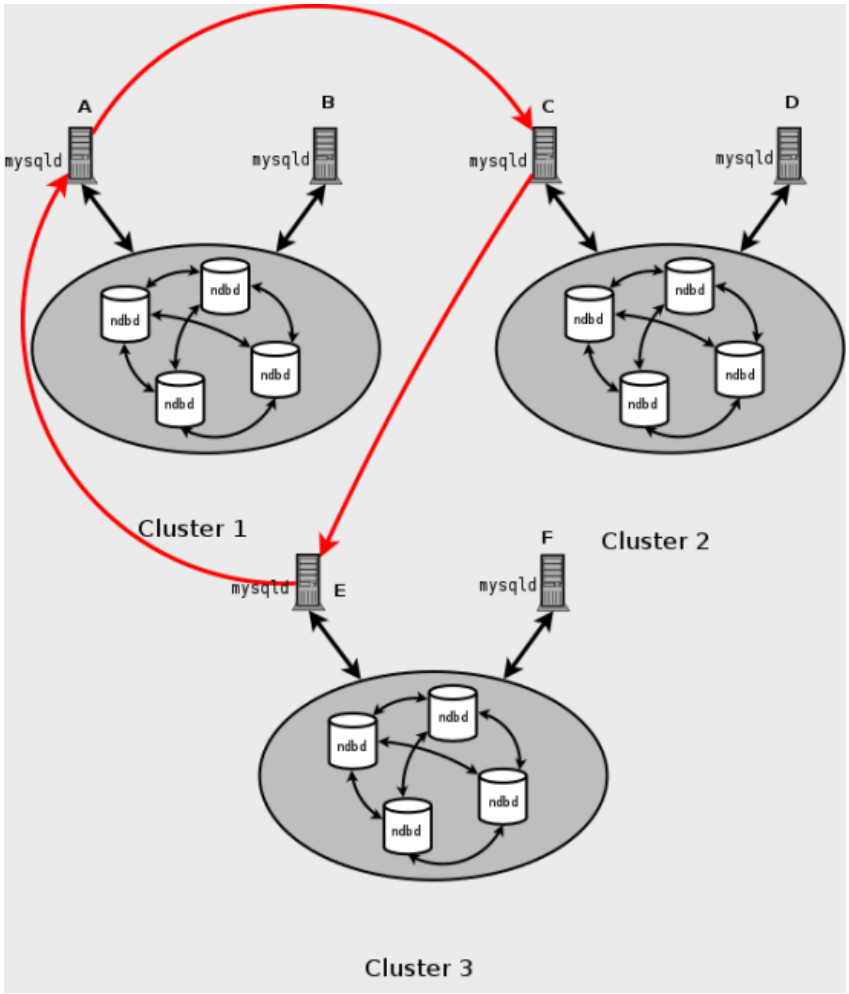
Circular replication example. In the next few paragraphs we consider the example of a replication setup involving three NDB Clusters numbered 1, 2, and 3, in which Cluster 1 acts as the replication source for Cluster 2, Cluster 2 acts as the source for Cluster 3, and Cluster 3 acts as the source for Cluster 1. Each cluster has two SQL nodes, with SQL nodes A and B belonging to Cluster 1, SQL nodes C and D belonging to Cluster 2, and SQL nodes E and F belonging to Cluster 3.

Circular replication using these clusters is supported as long as the following conditions are met:

- The SQL nodes on all sources and replicas are the same.
- All SQL nodes acting as sources and replicas are started with the `log_slave_updates` system variable enabled.

This type of circular replication setup is shown in the following diagram:

Figure 18.27 NDB Cluster Circular Replication with All Sources As Replicas

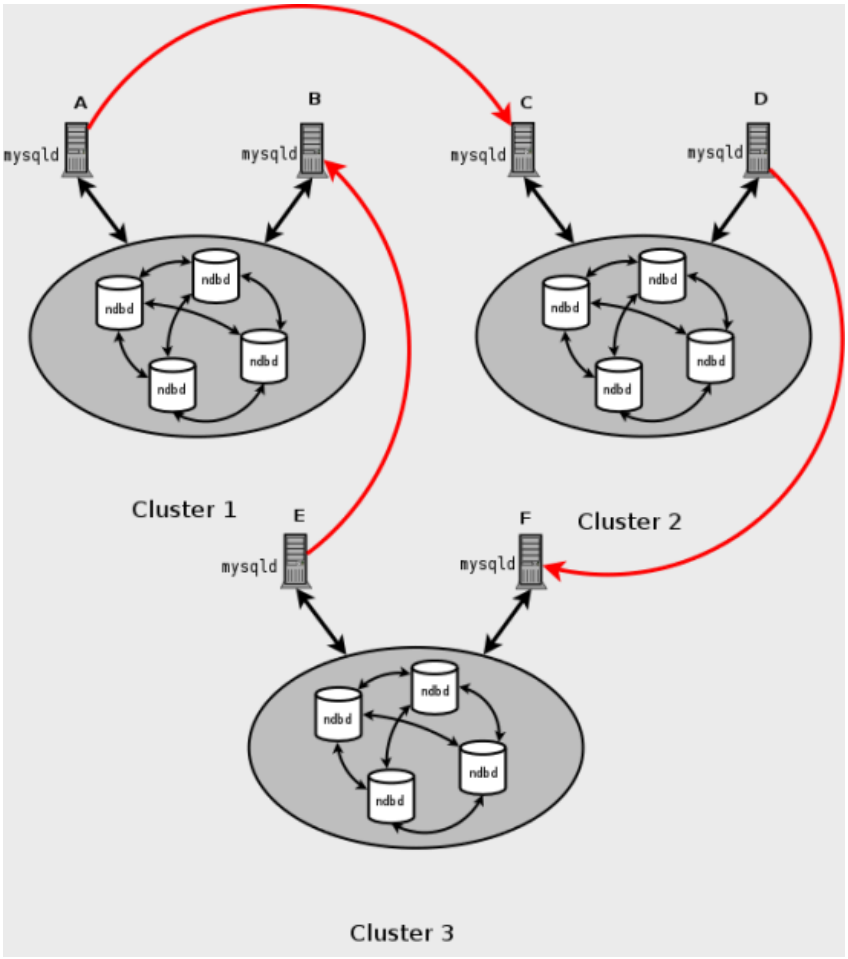


In this scenario, SQL node A in Cluster 1 replicates to SQL node C in Cluster 2; SQL node C replicates to SQL node E in Cluster 3; SQL node E replicates to SQL node A. In other words, the replication line

(indicated by the curved arrows in the diagram) directly connects all SQL nodes used as replication sources and replicas.

It is also possible to set up circular replication in such a way that not all source SQL nodes are also replicas, as shown here:

Figure 18.28 NDB Cluster Circular Replication Where Not All Sources Are Replicas

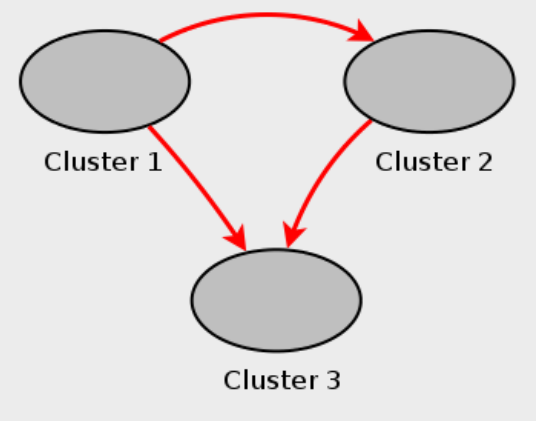


In this case, different SQL nodes in each cluster are used as replication sources and replicas. You must *not* start any of the SQL nodes with the `log_slave_updates` system variable enabled. This type of circular replication scheme for NDB Cluster, in which the line of replication (again indicated by the curved arrows in the diagram) is discontinuous, should be possible, but it should be noted that it has not yet been thoroughly tested and must therefore still be considered experimental.

Using NDB-native backup and restore to initialize a replica cluster. When setting up circular replication, it is possible to initialize the replica cluster by using the management client `START BACKUP` command on one NDB Cluster to create a backup and then applying this backup on another NDB Cluster using `ndb_restore`. This does not automatically create binary logs on the second NDB Cluster's SQL node acting as the replica; in order to cause the binary logs to be created, you must issue a `SHOW TABLES` statement on that SQL node; this should be done prior to running `START SLAVE`. This is a known issue.

Multi-source failover example. In this section, we discuss failover in a multi-source NDB Cluster replication setup with three NDB Clusters having server IDs 1, 2, and 3. In this scenario, Cluster 1 replicates to Clusters 2 and 3; Cluster 2 also replicates to Cluster 3. This relationship is shown here:

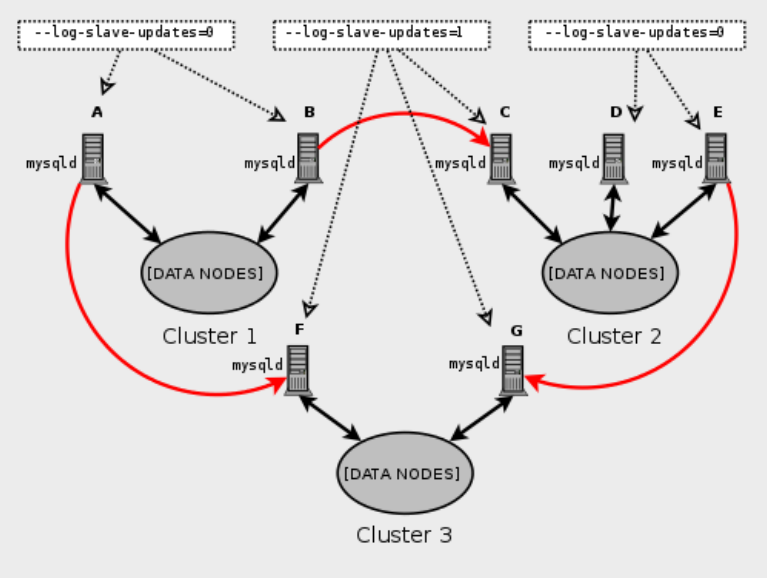
Figure 18.29 NDB Cluster Multi-Master Replication With 3 Sources



In other words, data replicates from Cluster 1 to Cluster 3 through 2 different routes: directly, and by way of Cluster 2.

Not all MySQL servers taking part in multi-source replication must act as both source and replica, and a given NDB Cluster might use different SQL nodes for different replication channels. Such a case is shown here:

Figure 18.30 NDB Cluster Multi-Source Replication, With MySQL Servers



MySQL servers acting as replicas must be run with the `log_slave_updates` system variable enabled. Which `mysqld` processes require this option is also shown in the preceding diagram.

Note

Using the `log_slave_updates` system variable has no effect on servers not being run as replicas.

The need for failover arises when one of the replicating clusters goes down. In this example, we consider the case where Cluster 1 is lost to service, and so Cluster 3 loses 2 sources of updates from Cluster 1. Because replication between NDB Clusters is asynchronous, there is no guarantee that Cluster 3's updates originating directly from Cluster 1 are more recent than those received through Cluster 2. You can handle this by ensuring that Cluster 3 catches up to Cluster 2 with regard to updates from Cluster 1. In terms of MySQL servers, this means that you need to replicate any outstanding updates from MySQL server C to server F.

On server C, perform the following queries:

```
mysqlC> SELECT @latest:=MAX(epoch)
     ->     FROM mysql.ndb_apply_status
     ->     WHERE server_id=1;

mysqlC> SELECT
     ->     @file:=SUBSTRING_INDEX(File, '/', -1),
     ->     @pos:=Position
     ->     FROM mysql.ndb_binlog_index
     ->     WHERE orig_epoch >= @latest
     ->     AND orig_server_id = 1
     ->     ORDER BY epoch ASC LIMIT 1;
```

Note

You can improve the performance of this query, and thus likely speed up failover times significantly, by adding the appropriate index to the `ndb_binlog_index` table. See Section 18.6.4, “NDB Cluster Replication Schema and Tables”, for more information.

Copy over the values for *`@file`* and *`@pos`* manually from server C to server F (or have your application perform the equivalent). Then, on server F, execute the following `CHANGE MASTER TO` statement:

```
mysqlF> CHANGE MASTER TO
     ->     MASTER_HOST = 'serverC'
     ->     MASTER_LOG_FILE='@file',
     ->     MASTER_LOG_POS=@pos;
```

Once this has been done, you can issue a `START SLAVE` statement on MySQL server F; this causes any missing updates originating from server B to be replicated to server F.

The `CHANGE MASTER TO` statement also supports an `IGNORE_SERVER_IDS` option which takes a comma-separated list of server IDs and causes events originating from the corresponding servers to be ignored. For more information, see Section 13.4.2.1, “CHANGE MASTER TO Statement”, and Section 13.7.5.35, “SHOW SLAVE STATUS Statement”. For information about how this option intereacts with the `ndb_log_apply_status` variable, see Section 18.6.8, “Implementing Failover with NDB Cluster Replication”.

18.6.11 NDB Cluster Replication Conflict Resolution

When using a replication setup involving multiple sources (including circular replication), it is possible that different sources may try to update the same row on the replica with different data. Conflict resolution in NDB Cluster Replication provides a means of resolving such conflicts by permitting a user-defined resolution column to be used to determine whether or not an update on a given source should be applied on the replica.

Some types of conflict resolution supported by NDB Cluster (`NDB$OLD()`, `NDB$MAX()`, `NDB$MAX_DELETE_WIN()`) implement this user-defined column as a “timestamp” column (although its type cannot be `TIMESTAMP`, as explained later in this section). These types of conflict resolution are always applied a row-by-row basis rather than a transactional basis. The epoch-based conflict resolution functions `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` compare the order in which epochs are replicated (and thus these functions are transactional). Different methods can be used to compare resolution column values on the replica when conflicts occur, as explained later in this section; the method used can be set on a per-table basis.

You should also keep in mind that it is the application's responsibility to ensure that the resolution column is correctly populated with relevant values, so that the resolution function can make the appropriate choice when determining whether to apply an update.

Requirements. Preparations for conflict resolution must be made on both the source and the replica. These tasks are described in the following list:

- On the source writing the binary logs, you must determine which columns are sent (all columns or only those that have been updated). This is done for the MySQL Server as a whole by applying the `mysqld` startup option `--ndb-log-updated-only` (described later in this section) or on a per-table basis by entries in the `mysql.ndb_replication` table (see The ndb_replication system table).

  Note

  If you are replicating tables with very large columns (such as `TEXT` or `BLOB` columns), `--ndb-log-updated-only` can also be useful for reducing the size of the binary logs and avoiding possible replication failures due to exceeding `max_allowed_packet`.

  See Section 17.4.1.19, "Replication and max_allowed_packet", for more information about this issue.

- On the replica, you must determine which type of conflict resolution to apply ("latest timestamp wins", "same timestamp wins", "primary wins", "primary wins, complete transaction", or none). This is done using the `mysql.ndb_replication` system table, on a per-table basis (see The ndb_replication system table).

- NDB 7.4.1 and later also supports read conflict detection, that is, detecting conflicts between reads of a given row in one cluster and updates or deletes of the same row in another cluster. This requires exclusive read locks obtained by setting `ndb_log_exclusive_reads` equal to 1 on the replica. All rows read by a conflicting read are logged in the exceptions table. For more information, see Read conflict detection and resolution.

When using the functions `NDB$OLD()`, `NDB$MAX()`, and `NDB$MAX_DELETE_WIN()` for timestamp-based conflict resolution, we often refer to the column used for determining updates as a "timestamp" column. However, the data type of this column is never `TIMESTAMP`; instead, its data type should be `INT` (`INTEGER`) or `BIGINT`. The "timestamp" column should also be `UNSIGNED` and `NOT NULL`.

The `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` functions discussed later in this section work by comparing the relative order of replication epochs applied on a primary and secondary NDB Cluster, and do not make use of timestamps.

Source column control. We can see update operations in terms of "before" and "after" images—that is, the states of the table before and after the update is applied. Normally, when updating a table with a primary key, the "before" image is not of great interest; however, when we need to determine on a per-update basis whether or not to use the updated values on a replica, we need to make sure that both images are written to the source's binary log. This is done with the `--ndb-log-update-as-write` option for `mysqld`, as described later in this section.

Important

Whether logging of complete rows or of updated columns only is done is decided when the MySQL server is started, and cannot be changed online; you must either restart `mysqld`, or start a new `mysqld` instance with different logging options.

Logging Full or Partial Rows (--ndb-log-updated-only Option)

Command-Line Format	`--ndb-log-updated-only[={OFF|ON}]`
System Variable	`ndb_log_updated_only`
Scope	Global
Dynamic	Yes
Type	Boolean

Default Value	ON

For purposes of conflict resolution, there are two basic methods of logging rows, as determined by the setting of the --ndb-log-updated-only option for mysqld:

- Log complete rows
- Log only column data that has been updated—that is, column data whose value has been set, regardless of whether or not this value was actually changed. This is the default behavior.

It is usually sufficient—and more efficient—to log updated columns only; however, if you need to log full rows, you can do so by setting --ndb-log-updated-only to 0 or OFF.

--ndb-log-update-as-write Option: Logging Changed Data as Updates

Command-Line Format	--ndb-log-update-as-write[={OFF|ON}]
System Variable	ndb_log_update_as_write
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	ON

The setting of the MySQL Server's --ndb-log-update-as-write option determines whether logging is performed with or without the “before” image. Because conflict resolution is done in the MySQL Server's update handler, it is necessary to control logging performed by the replication source such that updates are updates and not writes; that is, such that updates are treated as changes in existing rows rather than the writing of new rows, even though these replace existing rows. This option is turned on by default; in other words, updates are treated as writes. That is, updates are by default written as write_row events in the binary log, rather than as update_row events.

To disable the option, start the source mysqld with --ndb-log-update-as-write=0 or --ndb-log-update-as-write=OFF. You must do this when replicating from NDB tables to tables using a different storage engine; see Replication from NDB to other storage engines, and Replication from NDB to a nontransactional storage engine, for more information.

Conflict resolution control. Conflict resolution is usually enabled on the server where conflicts can occur. Like logging method selection, it is enabled by entries in the mysql.ndb_replication table.

The ndb_replication system table. To enable conflict resolution, it is necessary to create an ndb_replication table in the mysql system database on the source, the replica, or both, depending on the conflict resolution type and method to be employed. This table is used to control logging and conflict resolution functions on a per-table basis, and has one row per table involved in replication. ndb_replication is created and filled with control information on the server where the conflict is to be resolved. In a simple source-replica setup where data can also be changed locally on the replica this is typically the replica. In a more complex replication scheme, such as bidirectional replication, this is usually all of the sources involved. Each row in mysql.ndb_replication corresponds to a table being replicated, and specifies how to log and resolve conflicts (that is, which conflict resolution function, if any, to use) for that table. The definition of the mysql.ndb_replication table is shown here:

```
CREATE TABLE mysql.ndb_replication  (
    db VARBINARY(63),
    table_name VARBINARY(63),
    server_id INT UNSIGNED,
    binlog_type INT UNSIGNED,
    conflict_fn VARBINARY(128),
    PRIMARY KEY USING HASH (db, table_name, server_id)
)   ENGINE=NDB
```

```
PARTITION BY KEY(db,table_name);
```

The columns in this table are described in the next few paragraphs.

db. The name of the database containing the table to be replicated. You may employ either or both of the wildcards `_` and `%` as part of the database name. Matching is similar to what is implemented for the `LIKE` operator.

table_name. The name of the table to be replicated. The table name may include either or both of the wildcards `_` and `%`. Matching is similar to what is implemented for the `LIKE` operator.

server_id. The unique server ID of the MySQL instance (SQL node) where the table resides.

binlog_type. The type of binary logging to be employed. This is determined as shown in the following table:

Table 18.60 binlog_type values, with internal values and descriptions

Value	Internal Value	Description
0	`NBT_DEFAULT`	Use server default
1	`NBT_NO_LOGGING`	Do not log this table in the binary log
2	`NBT_UPDATED_ONLY`	Only updated attributes are logged
3	`NBT_FULL`	Log full row, even if not updated (MySQL server default behavior)
4	`NBT_USE_UPDATE`	(For generating `NBT_UPDATED_ONLY_USE_UPDATE` and `NBT_FULL_USE_UPDATE` values only—not intended for separate use)
5	[*Not used*]	---
6	`NBT_UPDATED_ONLY_USE_UPDATE` (equal to `NBT_UPDATED_ONLY | NBT_USE_UPDATE`)	Use updated attributes, even if values are unchanged
7	`NBT_FULL_USE_UPDATE` (equal to `NBT_FULL | NBT_USE_UPDATE`)	Use full row, even if values are unchanged

conflict_fn. The conflict resolution function to be applied. This function must be specified as one of those shown in the following list:

- NDB$OLD(column_name)
- NDB$MAX(column_name)
- NDB$MAX_DELETE_WIN()
- NDB$EPOCH() and NDB$EPOCH_TRANS()
- NDB$EPOCH_TRANS()
- NDB$EPOCH2() (NDB 7.4.2 and later)
- NDB$EPOCH2_TRANS() (NDB 7.4.2 and later)
- `NULL`: Indicates that conflict resolution is not to be used for the corresponding table.

These functions are described in the next few paragraphs.

NDB$OLD(column_name). If the value of *`column_name`* is the same on both the source and the replica, then the update is applied; otherwise, the update is not applied on the replica and an exception is written to the log. This is illustrated by the following pseudocode:

```
if (source_old_column_value == replica_current_column_value)
  apply_update();
else
  log_exception();
```

This function can be used for “same value wins” conflict resolution. This type of conflict resolution ensures that updates are not applied on the replica from the wrong source.

Important

The column value from the source's “before” image is used by this function.

NDB$MAX(column_name). If the “timestamp” column value for a given row coming from the source is higher than that on the replica, it is applied; otherwise it is not applied on the replica. This is illustrated by the following pseudocode:

```
if (source_new_column_value > replica_current_column_value)
  apply_update();
```

This function can be used for “greatest timestamp wins” conflict resolution. This type of conflict resolution ensures that, in the event of a conflict, the version of the row that was most recently updated is the version that persists.

Important

The column value from the sources's “after” image is used by this function.

NDB$MAX_DELETE_WIN(). This is a variation on `NDB$MAX()`. Due to the fact that no timestamp is available for a delete operation, a delete using `NDB$MAX()` is in fact processed as `NDB$OLD`, but for some use cases, this is not optimal. For `NDB$MAX_DELETE_WIN()`, if the “timestamp” column value for a given row adding or updating an existing row coming from the source is higher than that on the replica, it is applied. However, delete operations are treated as always having the higher value. This is illustrated by the following pseudocode:

```
if ( (source_new_column_value > replica_current_column_value)
       ||
     operation.type == "delete")
  apply_update();
```

This function can be used for “greatest timestamp, delete wins” conflict resolution. This type of conflict resolution ensures that, in the event of a conflict, the version of the row that was deleted or (otherwise) most recently updated is the version that persists.

Note

As with `NDB$MAX()`, the column value from the source's “after” image is the value used by this function.

NDB$EPOCH() and NDB$EPOCH_TRANS(). The `NDB$EPOCH()` function tracks the order in which replicated epochs are applied on a replica cluster relative to changes originating on the replica. This relative ordering is used to determine whether changes originating on the replica are concurrent with any changes that originate locally, and are therefore potentially in conflict.

Most of what follows in the description of `NDB$EPOCH()` also applies to `NDB$EPOCH_TRANS()`. Any exceptions are noted in the text.

`NDB$EPOCH()` is asymmetric, operating on one NDB Cluster in a bidirectional replication configuration (sometimes referred to as “active-active” replication). We refer here to cluster on which it operates as

the primary, and the other as the secondary. The replica on the primary is responsible for detecting and handling conflicts, while the replica on the secondary is not involved in any conflict detection or handling.

When the replica on the primary detects conflicts, it injects events into its own binary log to compensate for these; this ensures that the secondary NDB Cluster eventually realigns itself with the primary and so keeps the primary and secondary from diverging. This compensation and realignment mechanism requires that the primary NDB Cluster always wins any conflicts with the secondary—that is, that the primary's changes are always used rather than those from the secondary in event of a conflict. This “primary always wins” rule has the following implications:

- Operations that change data, once committed on the primary, are fully persistent and are not undone or rolled back by conflict detection and resolution.
- Data read from the primary is fully consistent. Any changes committed on the Primary (locally or from the replica) are not reverted later.
- Operations that change data on the secondary may later be reverted if the primary determines that they are in conflict.
- Individual rows read on the secondary are self-consistent at all times, each row always reflecting either a state committed by the secondary, or one committed by the primary.
- Sets of rows read on the secondary may not necessarily be consistent at a given single point in time. For `NDB$EPOCH_TRANS()`, this is a transient state; for `NDB$EPOCH()`, it can be a persistent state.
- Assuming a period of sufficient length without any conflicts, all data on the secondary NDB Cluster (eventually) becomes consistent with the primary's data.

`NDB$EPOCH()` and `NDB$EPOCH_TRANS()` do not require any user schema modifications, or application changes to provide conflict detection. However, careful thought must be given to the schema used, and the access patterns used, to verify that the complete system behaves within specified limits.

Each of the `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` functions can take an optional parameter; this is the number of bits to use to represent the lower 32 bits of the epoch, and should be set to no less than the value calculated as shown here:

```
CEIL( LOG2( TimeBetweenGlobalCheckpoints / TimeBetweenEpochs ), 1)
```

For the default values of these configuration parameters (2000 and 100 milliseconds, respectively), this gives a value of 5 bits, so the default value (6) should be sufficient, unless other values are used for `TimeBetweenGlobalCheckpoints`, `TimeBetweenEpochs`, or both. A value that is too small can result in false positives, while one that is too large could lead to excessive wasted space in the database.

Both `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` insert entries for conflicting rows into the relevant exceptions tables, provided that these tables have been defined according to the same exceptions table schema rules as described elsewhere in this section (see NDB$OLD(column_name)). You must create any exceptions table before creating the data table with which it is to be used.

As with the other conflict detection functions discussed in this section, `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` are activated by including relevant entries in the `mysql.ndb_replication` table (see The ndb_replication system table). The roles of the primary and secondary NDB Clusters in this scenario are fully determined by `mysql.ndb_replication` table entries.

Because the conflict detection algorithms employed by `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` are asymmetric, you must use different values for the `server_id` entries of the primary and secondary replicas.

Prior to NDB 7.3.6, conflicts between `DELETE` operations were handled like those for `UPDATE` operations, and within the same epoch were considered in conflict. In NDB 7.3.6 and later, a conflict

between DELETE operations alone is not sufficient to trigger a conflict using NDB$EPOCH() or NDB$EPOCH_TRANS(), and the relative placement within epochs does not matter. (Bug #18459944)

Conflict detection status variables. Several status variables can be used to monitor conflict detection. You can see how many rows have been found in conflict by NDB$EPOCH() since this replica was last restarted from the current value of the Ndb_conflict_fn_epoch system status variable.

Ndb_conflict_fn_epoch_trans provides the number of rows that have been found directly in conflict by NDB$EPOCH_TRANS(). Ndb_conflict_fn_epoch2 and Ndb_conflict_fn_epoch2_trans, added in NDB 7.4.2, show the number of rows found in conflict by NDB$EPOCH2() and NDB$EPOCH2_TRANS(), respectively. The number of rows actually realigned, including those affected due to their membership in or dependency on the same transactions as other conflicting rows, is given by Ndb_conflict_trans_row_reject_count.

For more information, see NDB Cluster Status Variables.

Limitations on NDB$EPOCH(). The following limitations currently apply when using NDB$EPOCH() to perform conflict detection:

- Conflicts are detected using NDB Cluster epoch boundaries, with granularity proportional to TimeBetweenEpochs (default: 100 milliseconds). The minimum conflict window is the minimum time during which concurrent updates to the same data on both clusters always report a conflict. This is always a nonzero length of time, and is roughly proportional to 2 * (latency + queueing + TimeBetweenEpochs). This implies that—assuming the default for TimeBetweenEpochs and ignoring any latency between clusters (as well as any queuing delays)—the minimum conflict window size is approximately 200 milliseconds. This minimum window should be considered when looking at expected application “race” patterns.
- Additional storage is required for tables using the NDB$EPOCH() and NDB$EPOCH_TRANS() functions; from 1 to 32 bits extra space per row is required, depending on the value passed to the function.
- Conflicts between delete operations may result in divergence between the primary and secondary. When a row is deleted on both clusters concurrently, the conflict can be detected, but is not recorded, since the row is deleted. This means that further conflicts during the propagation of any subsequent realignment operations are not detected, which can lead to divergence.

  Deletes should be externally serialized, or routed to one cluster only. Alternatively, a separate row should be updated transactionally with such deletes and any inserts that follow them, so that conflicts can be tracked across row deletes. This may require changes in applications.
- Only two NDB Clusters in a birectional “active-active” configuration are currently supported when using NDB$EPOCH() or NDB$EPOCH_TRANS() for conflict detection.
- Tables having BLOB or TEXT columns are not currently supported with NDB$EPOCH() or NDB$EPOCH_TRANS().

NDB$EPOCH_TRANS(). NDB$EPOCH_TRANS() extends the NDB$EPOCH() function. Conflicts are detected and handled in the same way using the “primary wins all” rule (see NDB$EPOCH() and NDB$EPOCH_TRANS()) but with the extra condition that any other rows updated in the same transaction in which the conflict occurred are also regarded as being in conflict. In other words, where NDB$EPOCH() realigns individual conflicting rows on the secondary, NDB$EPOCH_TRANS() realigns conflicting transactions.

In addition, any transactions which are detectably dependent on a conflicting transaction are also regarded as being in conflict, these dependencies being determined by the contents of the secondary cluster's binary log. Since the binary log contains only data modification operations (inserts, updates, and deletes), only overlapping data modifications are used to determine dependencies between transactions.

NDB$EPOCH_TRANS() is subject to the same conditions and limitations as NDB$EPOCH(), and in addition requires that Version 2 binary log row events are used (log_bin_use_v1_row_events

equal to 0), which adds a storage overhead of 2 bytes per event in the binary log. In addition, all transaction IDs must be recorded in the secondary's binary log (`--ndb-log-transaction-id` option), which adds a further variable overhead (up to 13 bytes per row).

See NDB$EPOCH() and NDB$EPOCH_TRANS().

Status information. A server status variable `Ndb_conflict_fn_max` provides a count of the number of times that a row was not applied on the current SQL node due to “greatest timestamp wins” conflict resolution since the last time that `mysqld` was started.

The number of times that a row was not applied as the result of “same timestamp wins” conflict resolution on a given `mysqld` since the last time it was restarted is given by the global status variable `Ndb_conflict_fn_old`. In addition to incrementing `Ndb_conflict_fn_old`, the primary key of the row that was not used is inserted into an *exceptions table*, as explained later in this section.

NDB$EPOCH2(). The `NDB$EPOCH2()` function, added in NDB 7.4.2, is similar to `NDB$EPOCH()`, except that `NDB$EPOCH2()` provides for delete-delete handling with a bidirectional replication topology. In this scenario, primary and secondary roles are assigned to the two sources by setting the `ndb_slave_conflict_role` system variable to the appropriate value on each source (usually one each of `PRIMARY`, `SECONDARY`). When this is done, modifications made by the secondary are reflected by the primary back to the secondary which then conditionally applies them.

NDB$EPOCH2_TRANS(). In NDB 7.4.2 and later, `NDB$EPOCH2_TRANS()` extends the `NDB$EPOCH2()` function. Conflicts are detected and handled in the same way, and assigning primary and secondary roles to the replicating clusters, but with the extra condition that any other rows updated in the same transaction in which the conflict occurred are also regarded as being in conflict. That is, `NDB$EPOCH2()` realigns individual conflicting rows on the secondary, while `NDB$EPOCH_TRANS()` realigns conflicting transactions.

Where `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` use metadata that is specified per row, per last modified epoch, to determine on the primary whether an incoming replicated row change from the secondary is concurrent with a locally committed change; concurrent changes are regarded as conflicting, with subesequent exceptions table updates and realignment of the secondary. A problem arises when a row is deleted on the primary so there is no longer any last-modified epoch available to determine whether any replicated operations conflict, which means that conflicting delete operationss are not detected. This can result in divergence, an example being a delete on one cluster which is concurrent with a delete and insert on the other; this why delete operations can be routed to only one cluster when using `NDB$EPOCH()` and `NDB$EPOCH_TRANS()`.

`NDB$EPOCH2()` bypasses the issue just described—storing information about deleted rows on the PRIMARY—by ignoring any delete-delete conflict, and by avoiding any potential resultant divergence as well. This is accomplished by reflecting any operation successfully applied on and replicated from the secondary back to the secondary. On its return to the secondary, it can be used to reapply an operation on the secondary which was deleted by an operation originating from the primary.

When using `NDB$EPOCH2()`, you should keep in mind that the secondary applies the delete from the primary, removing the new row until it is restored by a reflected operation. In theory, the subsequent insert or update on the secondary conflicts with the delete from the primary, but in this case, we choose to ignore this and allow the secondary to “win”, in the interest of preventing divergence between the clusters. In other words, after a delete, the primary does not detect conflicts, and instead adopts the secondary's following changes immediately. Because of this, the secondary's state can revisit multiple previous committed states as it progresses to a final (stable) state, and some of these may be visible.

You should also be aware that reflecting all operations from the secondary back to the primary increases the size of the primary's logbinary log, as well as demands on bandwidth, CPU usage, and disk I/O.

Application of reflected operations on the secondary depends on the state of the target row on the secondary. Whether or not reflected changes are applied on the secondary

can be tracked by checking the `Ndb_conflict_reflected_op_prepare_count` and `Ndb_conflict_reflected_op_discard_count` status variables (both added in NDB 7.4.2). The number of changes applied is simply the difference between these two values (note that `Ndb_conflict_reflected_op_prepare_count` is always greater than or equal to `Ndb_conflict_reflected_op_discard_count`).

Events are applied if and only if both of the following conditions are true:

- The existence of the row—that is, whether or not it exists—is in accordance with the type of event. For delete and update operations, the row must already exist. For insert operations, the row must *not* exist.
- The row was last modified by the primary. It is possible that the modification was accomplished through the execution of a reflected operation.

If both of the conditions are not met, the reflected operation is discarded by the secondary.

Conflict resolution exceptions table. To use the `NDB$OLD()` conflict resolution function, it is also necessary to create an exceptions table corresponding to each `NDB` table for which this type of conflict resolution is to be employed. This is also true when using `NDB$EPOCH()` or `NDB$EPOCH_TRANS()`. The name of this table is that of the table for which conflict resolution is to be applied, with the string `$EX` appended. (For example, if the name of the original table is `mytable`, the name of the corresponding exceptions table name should be `mytable$EX`.) Prior to NDB 7.4.1, the syntax for creating the exceptions table is as shown here:

```
CREATE TABLE original_table$EX  (
    server_id INT UNSIGNED,
    source_server_id INT UNSIGNED,
    source_epoch BIGINT UNSIGNED,
    count INT UNSIGNED,

    original_table_pk_columns,

    [additional_columns,]

    PRIMARY KEY(server_id, source_server_id, source_epoch, count)
) ENGINE=NDB;
```

NDB 7.4.1 and later support an extended exceptions table definition that includes optional columns providing information about an exception's type, cause, and originating transaction. In these versions, the syntax for creating the exceptions table is as shown here:

```
CREATE TABLE original_table$EX  (
    [NDB$]server_id INT UNSIGNED,
    [NDB$]source_server_id INT UNSIGNED,
    [NDB$]source_epoch BIGINT UNSIGNED,
    [NDB$]count INT UNSIGNED,

    [NDB$OP_TYPE ENUM('WRITE_ROW','UPDATE_ROW', 'DELETE_ROW',
      'REFRESH_ROW', 'READ_ROW') NOT NULL,]
    [NDB$CFT_CAUSE ENUM('ROW_DOES_NOT_EXIST', 'ROW_ALREADY_EXISTS',
      'DATA_IN_CONFLICT', 'TRANS_IN_CONFLICT') NOT NULL,]
    [NDB$ORIG_TRANSID BIGINT UNSIGNED NOT NULL,]

    original_table_pk_columns,

    [orig_table_column|orig_table_column$OLD|orig_table_column$NEW,]

    [additional_columns,]

    PRIMARY KEY([NDB$]server_id, [NDB$]source_server_id, [NDB$]source_epoch, [NDB$]count)
) ENGINE=NDB;
```

The first four columns are required. The names of the first four columns and the columns matching the original table's primary key columns are not critical; however, we suggest for reasons of clarity and consistency, that you use the names shown here for the `server_id`, `source_server_id`,

`source_epoch`, and `count` columns, and that you use the same names as in the original table for the columns matching those in the original table's primary key.

Starting with NDB 7.4.1, if the exceptions table uses one or more of the optional columns `NDB$OP_TYPE`, `NDB$CFT_CAUSE`, or `NDB$ORIG_TRANSID` discussed later in this section, then each of the required columns must also be named using the prefix `NDB$`. If desired, you can use the `NDB$` prefix to name the required columns even if you do not define any optional columns, but in this case, all four of the required columns must be named using the prefix.

Following these columns, the columns making up the original table's primary key should be copied in the order in which they are used to define the primary key of the original table. The data types for the columns duplicating the primary key columns of the original table should be the same as (or larger than) those of the original columns. In NDB Cluster 7.3 and earlier, the exceptions table's primary key must be reproduced column for column. Beginning with NDB 7.4.1, a subset of the primary key columns may be used instead.

Regardless of the NDB Cluster version employed, the exceptions table must use the `NDB` storage engine. (An example that uses `NDB$OLD()` with an exceptions table is shown later in this section.)

Additional columns may optionally be defined following the copied primary key columns, but not before any of them; any such extra columns cannot be `NOT NULL`. In NDB 7.4.1 and later, support is provided for three additional, predefined optional columns `NDB$OP_TYPE`, `NDB$CFT_CAUSE`, and `NDB$ORIG_TRANSID`, which are described in the next few paragraphs.

`NDB$OP_TYPE`: This column can be used to obtain the type of operation causing the conflict. If you use this column, define it as shown here:

```
NDB$OP_TYPE ENUM('WRITE_ROW', 'UPDATE_ROW', 'DELETE_ROW',
    'REFRESH_ROW', 'READ_ROW') NOT NULL
```

The `WRITE_ROW`, `UPDATE_ROW`, and `DELETE_ROW` operation types represent user-initiated operations. `REFRESH_ROW` operations are operations generated by conflict resolution in compensating transactions sent back to the originating cluster from the cluster that detected the conflict. `READ_ROW` operations are user-initiated read tracking operations defined with exclusive row locks.

`NDB$CFT_CAUSE`: You can define an optional column `NDB$CFT_CAUSE` which provides the cause of the registered conflict. This column, if used, is defined as shown here:

```
NDB$CFT_CAUSE ENUM('ROW_DOES_NOT_EXIST', 'ROW_ALREADY_EXISTS',
    'DATA_IN_CONFLICT', 'TRANS_IN_CONFLICT') NOT NULL
```

`ROW_DOES_NOT_EXIST` can be reported as the cause for `UPDATE_ROW` and `WRITE_ROW` operations; `ROW_ALREADY_EXISTS` can be reported for `WRITE_ROW` events. `DATA_IN_CONFLICT` is reported when a row-based conflict function detects a conflict; `TRANS_IN_CONFLICT` is reported when a transactional conflict function rejects all of the operations belonging to a complete transaction.

`NDB$ORIG_TRANSID`: The `NDB$ORIG_TRANSID` column, if used, contains the ID of the originating transaction. This column should be defined as follows:

```
NDB$ORIG_TRANSID BIGINT UNSIGNED NOT NULL
```

`NDB$ORIG_TRANSID` is a 64-bit value generated by `NDB`. This value can be used to correlate multiple exceptions table entries belonging to the same conflicting transaction from the same or different exceptions tables.

In NDB 7.4.1 and later, additional reference columns which are not part of the original table's primary key can be named *`colname`*`$OLD` or *`colname`*`$NEW`. *`colname`*`$OLD` references old values in update and delete operations—that is, operations containing `DELETE_ROW` events. *`colname`*`$NEW` can be used to reference new values in insert and update operations—in other words, operations using `WRITE_ROW` events, `UPDATE_ROW` events, or both types of events. Where a conflicting operation does

not supply a value for a given reference column that is not a primary key, the exceptions table row contains either `NULL`, or a defined default value for that column.

Important

The `mysql.ndb_replication` table is read when a data table is set up for replication, so the row corresponding to a table to be replicated must be inserted into `mysql.ndb_replication` *before* the table to be replicated is created.

Examples

The following examples assume that you have already a working NDB Cluster replication setup, as described in Section 18.6.5, “Preparing the NDB Cluster for Replication”, and Section 18.6.6, “Starting NDB Cluster Replication (Single Replication Channel)”.

NDB$MAX() example. Suppose you wish to enable “greatest timestamp wins” conflict resolution on table `test.t1`, using column `mycol` as the “timestamp”. This can be done using the following steps:

1. Make sure that you have started the source `mysqld` with `--ndb-log-update-as-write=OFF`.
2. On the source, perform this `INSERT` statement:

   ```
   INSERT INTO mysql.ndb_replication
       VALUES ('test', 't1', 0, NULL, 'NDB$MAX(mycol)');
   ```

   Inserting a 0 into the `server_id` indicates that all SQL nodes accessing this table should use conflict resolution. If you want to use conflict resolution on a specific `mysqld` only, use the actual server ID.

   Inserting `NULL` into the `binlog_type` column has the same effect as inserting 0 (`NBT_DEFAULT`); the server default is used.
3. Create the `test.t1` table:

   ```
   CREATE TABLE test.t1 (
       columns
       mycol INT UNSIGNED,
       columns
   ) ENGINE=NDB;
   ```

   Now, when updates are performed on this table, conflict resolution is applied, and the version of the row having the greatest value for `mycol` is written to the replica.

Note

Other `binlog_type` options—such as `NBT_UPDATED_ONLY_USE_UPDATE` should be used to control logging on the source using the `ndb_replication` table rather than by using command-line options.

NDB$OLD() example. Suppose an `NDB` table such as the one defined here is being replicated, and you wish to enable “same timestamp wins” conflict resolution for updates to this table:

```
CREATE TABLE test.t2  (
    a INT UNSIGNED NOT NULL,
    b CHAR(25) NOT NULL,
    columns,
    mycol INT UNSIGNED NOT NULL,
    columns,
    PRIMARY KEY pk (a, b)
)   ENGINE=NDB;
```

The following steps are required, in the order shown:

1. First—and *prior* to creating `test.t2`—you must insert a row into the `mysql.ndb_replication` table, as shown here:

```
INSERT INTO mysql.ndb_replication
    VALUES ('test', 't2', 0, NULL, 'NDB$OLD(mycol)');
```

   Possible values for the `binlog_type` column are shown earlier in this section. The value `'NDB$OLD(mycol)'` should be inserted into the `conflict_fn` column.

2. Create an appropriate exceptions table for `test.t2`. The table creation statement shown here includes all required columns; any additional columns must be declared following these columns, and before the definition of the table's primary key.

```
CREATE TABLE test.t2$EX  (
    server_id INT UNSIGNED,
    source_server_id INT UNSIGNED,
    source_epoch BIGINT UNSIGNED,
    count INT UNSIGNED,
    a INT UNSIGNED NOT NULL,
    b CHAR(25) NOT NULL,

    [additional_columns,]

    PRIMARY KEY(server_id, source_server_id, source_epoch, count)
)   ENGINE=NDB;
```

   In NDB 7.4.1 and later, we can include additional columns for information about the type, cause, and originating transaction ID for a given conflict. We are also not required to supply matching columns for all primary key columns in the original table. In these versions, you can create the exceptions table like this:

```
CREATE TABLE test.t2$EX  (
    NDB$server_id INT UNSIGNED,
    NDB$source_server_id INT UNSIGNED,
    NDB$source_epoch BIGINT UNSIGNED,
    NDB$count INT UNSIGNED,
    a INT UNSIGNED NOT NULL,

    NDB$OP_TYPE ENUM('WRITE_ROW','UPDATE_ROW', 'DELETE_ROW',
      'REFRESH_ROW', 'READ_ROW') NOT NULL,
    NDB$CFT_CAUSE ENUM('ROW_DOES_NOT_EXIST', 'ROW_ALREADY_EXISTS',
      'DATA_IN_CONFLICT', 'TRANS_IN_CONFLICT') NOT NULL,
    NDB$ORIG_TRANSID BIGINT UNSIGNED NOT NULL,

    [additional_columns,]

    PRIMARY KEY(NDB$server_id, NDB$source_server_id, NDB$source_epoch, NDB$count)
)   ENGINE=NDB;
```

Note

The `NDB$` prefix is required for the four required columns since we included at least one of the columns `NDB$OP_TYPE`, `NDB$CFT_CAUSE`, or `NDB$ORIG_TRANSID` in the table definition.

3. Create the table `test.t2` as shown previously.

These steps must be followed for every table for which you wish to perform conflict resolution using `NDB$OLD()`. For each such table, there must be a corresponding row in `mysql.ndb_replication`, and there must be an exceptions table in the same database as the table being replicated.

Read conflict detection and resolution. NDB 7.4.1 and later support tracking of read operations, which makes it possible in circular replication setups to manage conflicts between reads of a given row in one cluster and updates or deletes of the same row in another. This example uses `employee` and `department` tables to model a scenario in which an employee is moved from one department to another on the source cluster (which we refer to hereafter as cluster *A*) while the replica cluster

(hereafter *B*) updates the employee count of the employee's former department in an interleaved transaction.

The data tables have been created using the following SQL statements:

```
# Employee table
CREATE TABLE employee (
    id INT PRIMARY KEY,
    name VARCHAR(2000),
    dept INT NOT NULL
)   ENGINE=NDB;

# Department table
CREATE TABLE department (
    id INT PRIMARY KEY,
    name VARCHAR(2000),
    members INT
)   ENGINE=NDB;
```

The contents of the two tables include the rows shown in the (partial) output of the following `SELECT` statements:

```
mysql> SELECT id, name, dept FROM employee;
+---------------+------+
| id   | name   | dept |
+------+--------+------+
...
| 998  |  Mike  | 3    |
| 999  |  Joe   | 3    |
| 1000 |  Mary  | 3    |
...
+------+--------+------+

mysql> SELECT id, name, members FROM department;
+-----+-------------+---------+
| id  | name        | members |
+-----+-------------+---------+
...
| 3   | Old project | 24      |
...
+-----+-------------+---------+
```

We assume that we are already using an exceptions table that includes the four required columns (and these are used for this table's primary key), the optional columns for operation type and cause, and the original table's primary key column, created using the SQL statement shown here:

```
CREATE TABLE employee$EX  (
    NDB$server_id INT UNSIGNED,
    NDB$source_server_id INT UNSIGNED,
    NDB$source_epoch BIGINT UNSIGNED,
    NDB$count INT UNSIGNED,

    NDB$OP_TYPE ENUM( 'WRITE_ROW','UPDATE_ROW', 'DELETE_ROW',
                      'REFRESH_ROW','READ_ROW') NOT NULL,
    NDB$CFT_CAUSE ENUM( 'ROW_DOES_NOT_EXIST',
                        'ROW_ALREADY_EXISTS',
                        'DATA_IN_CONFLICT',
                        'TRANS_IN_CONFLICT') NOT NULL,

    id INT NOT NULL,

    PRIMARY KEY(NDB$server_id, NDB$source_server_id, NDB$source_epoch, NDB$count)
)   ENGINE=NDB;
```

Suppose there occur the two simultaneous transactions on the two clusters. On cluster *A*, we create a new department, then move employee number 999 into that department, using the following SQL statements:

```
BEGIN;
  INSERT INTO department VALUES (4, "New project", 1);
```

```
  UPDATE employee SET dept = 4 WHERE id = 999;
COMMIT;
```

At the same time, on cluster *B*, another transaction reads from `employee`, as shown here:

```
BEGIN;
  SELECT name FROM employee WHERE id = 999;
  UPDATE department SET members = members - 1  WHERE id = 3;
commit;
```

The conflicting transactions are not normally detected by the conflict resolution mechanism, since the conflict is between a read (`SELECT`) and an update operation. Beginning with NDB 7.4.1, we can circumvent this issue by executing `SET ndb_log_exclusive_reads = 1` on the replica cluster. Acquiring exclusive read locks in this way causes any rows read on the source to be flagged as needing conflict resolution on the replica cluster. If we enable exclusive reads in this way prior to the logging of these transactions, the read on cluster *B* is tracked and sent to cluster *A* for resolution; the conflict on the employee row is subsequently detected and the transaction on cluster *B* is aborted.

The conflict is registered in the exceptions table (on cluster *A*) as a `READ_ROW` operation (see Conflict resolution exceptions table, for a description of operation types), as shown here:

```
mysql> SELECT id, NDB$OP_TYPE, NDB$CFT_CAUSE FROM employee$EX;
+-------+-------------+------------------+
| id    | NDB$OP_TYPE | NDB$CFT_CAUSE    |
+-------+-------------+------------------+
...
| 999   | READ_ROW    | TRANS_IN_CONFLICT |
+-------+-------------+------------------+
```

Any existing rows found in the read operation are flagged. This means that multiple rows resulting from the same conflict may be logged in the exception table, as shown by examining the effects a conflict between an update on cluster *A* and a read of multiple rows on cluster *B* from the same table in simultaneous transactions. The transaction executed on cluster *A* is shown here:

```
BEGIN;
  INSERT INTO department VALUES (4, "New project", 0);
  UPDATE employee SET dept = 4 WHERE dept = 3;
  SELECT COUNT(*) INTO @count FROM employee WHERE dept = 4;
  UPDATE department SET members = @count WHERE id = 4;
COMMIT;
```

Concurrently a transaction containing the statements shown here runs on cluster *B*:

```
SET ndb_log_exclusive_reads = 1;  # Must be set if not already enabled
...
BEGIN;
  SELECT COUNT(*) INTO @count FROM employee WHERE dept = 3 FOR UPDATE;
  UPDATE department SET members = @count WHERE id = 3;
COMMIT;
```

In this case, all three rows matching the `WHERE` condition in the second transaction's `SELECT` are read, and are thus flagged in the exceptions table, as shown here:

```
mysql> SELECT id, NDB$OP_TYPE, NDB$CFT_CAUSE FROM employee$EX;
+-------+-------------+------------------+
| id    | NDB$OP_TYPE | NDB$CFT_CAUSE    |
+-------+-------------+------------------+
...
| 998   | READ_ROW    | TRANS_IN_CONFLICT |
| 999   | READ_ROW    | TRANS_IN_CONFLICT |
| 1000  | READ_ROW    | TRANS_IN_CONFLICT |
...
+-------+-------------+------------------+
```

Read tracking is performed on the basis of existing rows only. A read based on a given condition track conflicts only of any rows that are *found* and not of any rows that are inserted in an interleaved

transaction. This is similar to how exclusive row locking is performed in a single instance of NDB Cluster.

18.7 NDB Cluster Release Notes

Changes in NDB Cluster releases are documented separately from this reference manual; you can find release notes for the changes in each NDB Cluster 7.4 release at NDB Cluster 7.4 Release Notes, and for NDB Cluster 7.3 releases at NDB Cluster 7.3 Release Notes.

You can obtain release notes for older versions of NDB Cluster from NDB Cluster Release Notes.

Chapter 19 Partitioning

Table of Contents

19.1 Overview of Partitioning in MySQL 2869
19.2 Partitioning Types 2871
19.2.1 RANGE Partitioning 2873
19.2.2 LIST Partitioning 2877
19.2.3 COLUMNS Partitioning 2879
19.2.4 HASH Partitioning 2887
19.2.5 KEY Partitioning 2890
19.2.6 Subpartitioning 2891
19.2.7 How MySQL Partitioning Handles NULL 2894
19.3 Partition Management 2899
19.3.1 Management of RANGE and LIST Partitions 2900
19.3.2 Management of HASH and KEY Partitions 2905
19.3.3 Exchanging Partitions and Subpartitions with Tables 2906
19.3.4 Maintenance of Partitions 2911
19.3.5 Obtaining Information About Partitions 2912
19.4 Partition Pruning 2914
19.5 Partition Selection 2918
19.6 Restrictions and Limitations on Partitioning 2923
19.6.1 Partitioning Keys, Primary Keys, and Unique Keys 2930
19.6.2 Partitioning Limitations Relating to Storage Engines 2933
19.6.3 Partitioning Limitations Relating to Functions 2934
19.6.4 Partitioning and Locking 2936

This chapter discusses MySQL's implementation of *user-defined partitioning*. You can determine whether your MySQL Server supports partitioning by checking the output of the `SHOW PLUGINS` statement, like this:

Note

Previous versions of MySQL had the `have_partitioning` variable, which was deprecated and removed in MySQL 5.6.1.

```
mysql> SHOW PLUGINS;
+------------+----------+----------------+---------+---------+
| Name       | Status   | Type           | Library | License |
+------------+----------+----------------+---------+---------+
| binlog     | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| partition  | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| ARCHIVE    | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| BLACKHOLE  | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| CSV        | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| FEDERATED  | DISABLED | STORAGE ENGINE | NULL    | GPL     |
| MEMORY     | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| InnoDB     | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| MRG_MYISAM | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| MyISAM     | ACTIVE   | STORAGE ENGINE | NULL    | GPL     |
| ndbcluster | DISABLED | STORAGE ENGINE | NULL    | GPL     |
+------------+----------+----------------+---------+---------+
11 rows in set (0.00 sec)
```

You can also check the `INFORMATION_SCHEMA.PLUGINS` table with a query similar to this one:

```
mysql> SELECT
    ->     PLUGIN_NAME as Name,
    ->     PLUGIN_VERSION as Version,
```

```
    ->     PLUGIN_STATUS as Status
    -> FROM INFORMATION_SCHEMA.PLUGINS
    -> WHERE PLUGIN_TYPE='STORAGE ENGINE';
+--------------------+---------+--------+
| Name               | Version | Status |
+--------------------+---------+--------+
| binlog             | 1.0     | ACTIVE |
| CSV                | 1.0     | ACTIVE |
| MEMORY             | 1.0     | ACTIVE |
| MRG_MYISAM         | 1.0     | ACTIVE |
| MyISAM             | 1.0     | ACTIVE |
| PERFORMANCE_SCHEMA | 0.1     | ACTIVE |
| BLACKHOLE          | 1.0     | ACTIVE |
| ARCHIVE            | 3.0     | ACTIVE |
| InnoDB             | 5.6     | ACTIVE |
| partition          | 1.0     | ACTIVE |
+--------------------+---------+--------+
10 rows in set (0.00 sec)
```

In either case, if you do not see the `partition` plugin listed with the value `ACTIVE` for the `Status` column in the output (shown in bold text in each of the examples just given), then your version of MySQL was not built with partitioning support.

MySQL 5.6 Community binaries provided by Oracle include partitioning support. For information about partitioning support offered in MySQL Enterprise Edition binaries, see Chapter 24, *MySQL Enterprise Edition*.

To enable partitioning if you are compiling MySQL 5.6 from source, the build must be configured with the `-DWITH_PARTITION_STORAGE_ENGINE` option. For more information, see Section 2.9, “Installing MySQL from Source”.

If your MySQL binary is built with partitioning support, nothing further needs to be done to enable it (for example, no special entries are required in your `my.cnf` file).

If you want to disable partitioning support, you can start the MySQL Server with the `--skip-partition` option. When partitioning support is disabled, you can see any existing partitioned tables and drop them (although doing this is not advised), but you cannot otherwise manipulate them or access their data.

See Section 19.1, “Overview of Partitioning in MySQL”, for an introduction to partitioning and partitioning concepts.

MySQL supports several types of partitioning as well as subpartitioning; see Section 19.2, “Partitioning Types”, and Section 19.2.6, “Subpartitioning”.

Section 19.3, “Partition Management”, covers methods of adding, removing, and altering partitions in existing partitioned tables.

Section 19.3.4, “Maintenance of Partitions”, discusses table maintenance commands for use with partitioned tables.

The `PARTITIONS` table in the `INFORMATION_SCHEMA` database provides information about partitions and partitioned tables. See Section 21.14, “The INFORMATION_SCHEMA PARTITIONS Table”, for more information; for some examples of queries against this table, see Section 19.2.7, “How MySQL Partitioning Handles NULL”.

For known issues with partitioning in MySQL 5.6, see Section 19.6, “Restrictions and Limitations on Partitioning”.

You may also find the following resources to be useful when working with partitioned tables.

Additional Resources. Other sources of information about user-defined partitioning in MySQL include the following:

- MySQL Partitioning Forum

  This is the official discussion forum for those interested in or experimenting with MySQL Partitioning technology. It features announcements and updates from MySQL developers and others. It is monitored by members of the Partitioning Development and Documentation Teams.

- Mikael Ronström's Blog

  MySQL Partitioning Architect and Lead Developer Mikael Ronström frequently posts articles here concerning his work with MySQL Partitioning and NDB Cluster.

- PlanetMySQL

  A MySQL news site featuring MySQL-related blogs, which should be of interest to anyone using my MySQL. We encourage you to check here for links to blogs kept by those working with MySQL Partitioning, or to have your own blog added to those covered.

MySQL 5.6 binaries are available from https://dev.mysql.com/downloads/mysql/5.6.html. However, for the latest partitioning bugfixes and feature additions, you can obtain the source from our GitHub repository. To enable partitioning, the build must be configured with the `-DWITH_PARTITION_STORAGE_ENGINE` option. For more information about building MySQL, see Section 2.9, “Installing MySQL from Source”. If you have problems compiling a partitioning-enabled MySQL 5.6 build, check the MySQL Partitioning Forum and ask for assistance there if you do not find a solution to your problem already posted.

19.1 Overview of Partitioning in MySQL

This section provides a conceptual overview of partitioning in MySQL 5.6.

For information on partitioning restrictions and feature limitations, see Section 19.6, “Restrictions and Limitations on Partitioning”.

The SQL standard does not provide much in the way of guidance regarding the physical aspects of data storage. The SQL language itself is intended to work independently of any data structures or media underlying the schemas, tables, rows, or columns with which it works. Nonetheless, most advanced database management systems have evolved some means of determining the physical location to be used for storing specific pieces of data in terms of the file system, hardware or even both. In MySQL, the `InnoDB` storage engine has long supported the notion of a tablespace, and the MySQL Server, even prior to the introduction of partitioning, could be configured to employ different physical directories for storing different databases (see Section 8.12.3, “Using Symbolic Links”, for an explanation of how this is done).

Partitioning takes this notion a step further, by enabling you to distribute portions of individual tables across a file system according to rules which you can set largely as needed. In effect, different portions of a table are stored as separate tables in different locations. The user-selected rule by which the division of data is accomplished is known as a *partitioning function*, which in MySQL can be the modulus, simple matching against a set of ranges or value lists, an internal hashing function, or a linear hashing function. The function is selected according to the partitioning type specified by the user, and takes as its parameter the value of a user-supplied expression. This expression can be a column value, a function acting on one or more column values, or a set of one or more column values, depending on the type of partitioning that is used.

In the case of `RANGE`, `LIST`, and [`LINEAR`] `HASH` partitioning, the value of the partitioning column is passed to the partitioning function, which returns an integer value representing the number of the partition in which that particular record should be stored. This function must be nonconstant and nonrandom. It may not contain any queries, but may use an SQL expression that is valid in MySQL, as long as that expression returns either `NULL` or an integer *`intval`* such that

```
-MAXVALUE <= intval <= MAXVALUE
```

(MAXVALUE is used to represent the least upper bound for the type of integer in question. -MAXVALUE represents the greatest lower bound.)

For [LINEAR] KEY, RANGE COLUMNS, and LIST COLUMNS partitioning, the partitioning expression consists of a list of one or more columns.

For [LINEAR] KEY partitioning, the partitioning function is supplied by MySQL.

For more information about permitted partitioning column types and partitioning functions, see Section 19.2, “Partitioning Types”, as well as Section 13.1.17, “CREATE TABLE Statement”, which provides partitioning syntax descriptions and additional examples. For information about restrictions on partitioning functions, see Section 19.6.3, “Partitioning Limitations Relating to Functions”.

This is known as *horizontal partitioning*—that is, different rows of a table may be assigned to different physical partitions. MySQL 5.6 does not support *vertical partitioning*, in which different columns of a table are assigned to different physical partitions. There are no plans at this time to introduce vertical partitioning into MySQL.

For information about determining whether your MySQL Server binary supports user-defined partitioning, see Chapter 19, *Partitioning*.

For creating partitioned tables, you can use most storage engines that are supported by your MySQL server; the MySQL partitioning engine runs in a separate layer and can interact with any of these. In MySQL 5.6, all partitions of the same partitioned table must use the same storage engine; for example, you cannot use MyISAM for one partition and InnoDB for another. However, there is nothing preventing you from using different storage engines for different partitioned tables on the same MySQL server or even in the same database.

MySQL partitioning cannot be used with the MERGE, CSV, or FEDERATED storage engines.

Partitioning by KEY or LINEAR KEY is possible with NDB, but other types of user-defined partitioning are not supported for tables using this storage engine. In addition, an NDB table that employs user-defined partitioning must have an explicit primary key, and any columns referenced in the table's partitioning expression must be part of the primary key. However, if no columns are listed in the PARTITION BY KEY or PARTITION BY LINEAR KEY clause of the CREATE TABLE or ALTER TABLE statement used to create or modify a user-partitioned NDB table, then the table is not required to have an explicit primary key. For more information, see Section 18.1.7.1, “Noncompliance with SQL Syntax in NDB Cluster”.

To employ a particular storage engine for a partitioned table, it is necessary only to use the [STORAGE] ENGINE option just as you would for a nonpartitioned table. However, you should keep in mind that [STORAGE] ENGINE (and other table options) need to be listed *before* any partitioning options are used in a CREATE TABLE statement. This example shows how to create a table that is partitioned by hash into 6 partitions and which uses the InnoDB storage engine:

```
CREATE TABLE ti (id INT, amount DECIMAL(7,2), tr_date DATE)
    ENGINE=INNODB
    PARTITION BY HASH( MONTH(tr_date) )
    PARTITIONS 6;
```

Each PARTITION clause can include a [STORAGE] ENGINE option, but in MySQL 5.6 this has no effect.

Important

Partitioning applies to all data and indexes of a table; you cannot partition only the data and not the indexes, or vice versa, nor can you partition only a portion of the table.

Data and indexes for each partition can be assigned to a specific directory using the DATA DIRECTORY and INDEX DIRECTORY options for the PARTITION clause of the CREATE TABLE statement used to create the partitioned table.

`DATA DIRECTORY` and `INDEX DIRECTORY` are not supported for individual partitions or subpartitions of `MyISAM` tables on Windows.

Only the `DATA DIRECTORY` option is supported for individual partitions and subpartitions of `InnoDB` tables.

All columns used in the table's partitioning expression must be part of every unique key that the table may have, including any primary key. This means that a table such as this one, created by the following SQL statement, cannot be partitioned:

```
CREATE TABLE tnp (
    id INT NOT NULL AUTO_INCREMENT,
    ref BIGINT NOT NULL,
    name VARCHAR(255),
    PRIMARY KEY pk (id),
    UNIQUE KEY uk (name)
);
```

Because the keys `pk` and `uk` have no columns in common, there are no columns available for use in a partitioning expression. Possible workarounds in this situation include adding the `name` column to the table's primary key, adding the `id` column to `uk`, or simply removing the unique key altogether. See Section 19.6.1, “Partitioning Keys, Primary Keys, and Unique Keys”, for more information.

In addition, `MAX_ROWS` and `MIN_ROWS` can be used to determine the maximum and minimum numbers of rows, respectively, that can be stored in each partition. The `MAX_ROWS` option can be useful for causing NDB Cluster tables to be created with extra partitions, thus allowing for greater storage of hash indexes. See the documentation for the `DataMemory` data node configuration parameter, as well as Section 18.1.2, “NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions”, for more information.

Some advantages of partitioning are listed here:

- Partitioning makes it possible to store more data in one table than can be held on a single disk or file system partition.
- Data that loses its usefulness can often be easily removed from a partitioned table by dropping the partition (or partitions) containing only that data. Conversely, the process of adding new data can in some cases be greatly facilitated by adding one or more new partitions for storing specifically that data.
- Some queries can be greatly optimized in virtue of the fact that data satisfying a given `WHERE` clause can be stored only on one or more partitions, which automatically excludes any remaining partitions from the search. Because partitions can be altered after a partitioned table has been created, you can reorganize your data to enhance frequent queries that may not have been often used when the partitioning scheme was first set up. This ability to exclude non-matching partitions (and thus any rows they contain) is often referred to as *partition pruning*. For more information, see Section 19.4, “Partition Pruning”.

  In addition, MySQL 5.6 supports explicit partition selection for queries. For example, `SELECT * FROM t PARTITION (p0,p1) WHERE c < 5` selects only those rows in partitions `p0` and `p1` that match the `WHERE` condition. In this case, MySQL does not check any other partitions of table `t`; this can greatly speed up queries when you already know which partition or partitions you wish to examine. Partition selection is also supported for the data modification statements `DELETE`, `INSERT`, `REPLACE`, `UPDATE`, and `LOAD DATA`, `LOAD XML`. See the descriptions of these statements for more information and examples.

19.2 Partitioning Types

This section discusses the types of partitioning which are available in MySQL 5.6. These include the types listed here:

- **RANGE partitioning.** This type of partitioning assigns rows to partitions based on column values falling within a given range. See Section 19.2.1, “RANGE Partitioning”. For information about an extension to this type, `RANGE COLUMNS`, see Section 19.2.3.1, “RANGE COLUMNS partitioning”.
- **LIST partitioning.** Similar to partitioning by `RANGE`, except that the partition is selected based on columns matching one of a set of discrete values. See Section 19.2.2, “LIST Partitioning”. For information about an extension to this type, `LIST COLUMNS`, see Section 19.2.3.2, “LIST COLUMNS partitioning”.
- **HASH partitioning.** With this type of partitioning, a partition is selected based on the value returned by a user-defined expression that operates on column values in rows to be inserted into the table. The function may consist of any expression valid in MySQL that yields a nonnegative integer value. An extension to this type, `LINEAR HASH`, is also available. See Section 19.2.4, “HASH Partitioning”.
- **KEY partitioning.** This type of partitioning is similar to partitioning by `HASH`, except that only one or more columns to be evaluated are supplied, and the MySQL server provides its own hashing function. These columns can contain other than integer values, since the hashing function supplied by MySQL guarantees an integer result regardless of the column data type. An extension to this type, `LINEAR KEY`, is also available. See Section 19.2.5, “KEY Partitioning”.

A very common use of database partitioning is to segregate data by date. Some database systems support explicit date partitioning, which MySQL does not implement in 5.6. However, it is not difficult in MySQL to create partitioning schemes based on `DATE`, `TIME`, or `DATETIME` columns, or based on expressions making use of such columns.

When partitioning by `KEY` or `LINEAR KEY`, you can use a `DATE`, `TIME`, or `DATETIME` column as the partitioning column without performing any modification of the column value. For example, this table creation statement is perfectly valid in MySQL:

```
CREATE TABLE members (
    firstname VARCHAR(25) NOT NULL,
    lastname VARCHAR(25) NOT NULL,
    username VARCHAR(16) NOT NULL,
    email VARCHAR(35),
    joined DATE NOT NULL
)
PARTITION BY KEY(joined)
PARTITIONS 6;
```

In MySQL 5.6, it is also possible to use a `DATE` or `DATETIME` column as the partitioning column using `RANGE COLUMNS` and `LIST COLUMNS` partitioning.

MySQL's other partitioning types, however, require a partitioning expression that yields an integer value or `NULL`. If you wish to use date-based partitioning by `RANGE`, `LIST`, `HASH`, or `LINEAR HASH`, you can simply employ a function that operates on a `DATE`, `TIME`, or `DATETIME` column and returns such a value, as shown here:

```
CREATE TABLE members (
    firstname VARCHAR(25) NOT NULL,
    lastname VARCHAR(25) NOT NULL,
    username VARCHAR(16) NOT NULL,
    email VARCHAR(35),
    joined DATE NOT NULL
)
PARTITION BY RANGE( YEAR(joined) ) (
    PARTITION p0 VALUES LESS THAN (1960),
    PARTITION p1 VALUES LESS THAN (1970),
    PARTITION p2 VALUES LESS THAN (1980),
    PARTITION p3 VALUES LESS THAN (1990),
    PARTITION p4 VALUES LESS THAN MAXVALUE
);
```

Additional examples of partitioning using dates may be found in the following sections of this chapter:

- Section 19.2.1, “RANGE Partitioning”
- Section 19.2.4, “HASH Partitioning”
- Section 19.2.4.1, “LINEAR HASH Partitioning”

For more complex examples of date-based partitioning, see the following sections:

- Section 19.4, “Partition Pruning”
- Section 19.2.6, “Subpartitioning”

MySQL partitioning is optimized for use with the `TO_DAYS()`, `YEAR()`, and `TO_SECONDS()` functions. However, you can use other date and time functions that return an integer or `NULL`, such as `WEEKDAY()`, `DAYOFYEAR()`, or `MONTH()`. See Section 12.7, “Date and Time Functions”, for more information about such functions.

It is important to remember—regardless of the type of partitioning that you use—that partitions are always numbered automatically and in sequence when created, starting with `0`. When a new row is inserted into a partitioned table, it is these partition numbers that are used in identifying the correct partition. For example, if your table uses 4 partitions, these partitions are numbered `0`, `1`, `2`, and `3`. For the `RANGE` and `LIST` partitioning types, it is necessary to ensure that there is a partition defined for each partition number. For `HASH` partitioning, the user-supplied expression must evaluate to an integer value greater than `0`. For `KEY` partitioning, this issue is taken care of automatically by the hashing function which the MySQL server employs internally.

Names of partitions generally follow the rules governing other MySQL identifiers, such as those for tables and databases. However, partition names are not case-sensitive. For example, the following `CREATE TABLE` statement fails as shown:

```
mysql> CREATE TABLE t2 (val INT)
    -> PARTITION BY LIST(val)(
    ->     PARTITION mypart VALUES IN (1,3,5),
    ->     PARTITION MyPart VALUES IN (2,4,6)
    -> );
ERROR 1488 (HY000): Duplicate partition name mypart
```

Failure occurs because MySQL sees no difference between the partition names `mypart` and `MyPart`.

When you specify the number of partitions for the table, this must be expressed as a positive, nonzero integer literal with no leading zeros, and may not be an expression such as `0.8E+01` or `6-2`, even if it evaluates to an integer value. Decimal fractions are not permitted.

In the sections that follow, we do not necessarily provide all possible forms for the syntax that can be used for creating each partition type; this information may be found in Section 13.1.17, “CREATE TABLE Statement”.

19.2.1 RANGE Partitioning

A table that is partitioned by range is partitioned in such a way that each partition contains rows for which the partitioning expression value lies within a given range. Ranges should be contiguous but not overlapping, and are defined using the `VALUES LESS THAN` operator. For the next few examples, suppose that you are creating a table such as the following to hold personnel records for a chain of 20 video stores, numbered 1 through 20:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
);
```

Note

The `employees` table used here has no primary or unique keys. While the examples work as shown for purposes of the present discussion, you should keep in mind that tables are extremely likely in practice to have primary keys, unique keys, or both, and that allowable choices for partitioning columns depend on the columns used for these keys, if any are present. For a discussion of these issues, see Section 19.6.1, "Partitioning Keys, Primary Keys, and Unique Keys".

This table can be partitioned by range in a number of ways, depending on your needs. One way would be to use the `store_id` column. For instance, you might decide to partition the table 4 ways by adding a `PARTITION BY RANGE` clause as shown here:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
)
PARTITION BY RANGE (store_id) (
    PARTITION p0 VALUES LESS THAN (6),
    PARTITION p1 VALUES LESS THAN (11),
    PARTITION p2 VALUES LESS THAN (16),
    PARTITION p3 VALUES LESS THAN (21)
);
```

In this partitioning scheme, all rows corresponding to employees working at stores 1 through 5 are stored in partition `p0`, to those employed at stores 6 through 10 are stored in partition `p1`, and so on. Each partition is defined in order, from lowest to highest. This is a requirement of the `PARTITION BY RANGE` syntax; you can think of it as being analogous to a series of `if ... elseif ...` statements in C or Java in this regard.

It is easy to determine that a new row containing the data `(72, 'Mitchell', 'Wilson', '1998-06-25', NULL, 13)` is inserted into partition `p2`, but what happens when your chain adds a 21st store? Under this scheme, there is no rule that covers a row whose `store_id` is greater than 20, so an error results because the server does not know where to place it. You can keep this from occurring by using a "catchall" `VALUES LESS THAN` clause in the `CREATE TABLE` statement that provides for all values greater than the highest value explicitly named:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
)
PARTITION BY RANGE (store_id) (
    PARTITION p0 VALUES LESS THAN (6),
    PARTITION p1 VALUES LESS THAN (11),
    PARTITION p2 VALUES LESS THAN (16),
    PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

Note

Another way to avoid an error when no matching value is found is to use the `IGNORE` keyword as part of the `INSERT` statement. For an example, see Section 19.2.2, "LIST Partitioning". Also see Section 13.2.5, "INSERT Statement", for general information about `IGNORE`.

MAXVALUE represents an integer value that is always greater than the largest possible integer value (in mathematical language, it serves as a *least upper bound*). Now, any rows whose store_id column value is greater than or equal to 16 (the highest value defined) are stored in partition p3. At some point in the future—when the number of stores has increased to 25, 30, or more—you can use an ALTER TABLE statement to add new partitions for stores 21-25, 26-30, and so on (see Section 19.3, "Partition Management", for details of how to do this).

In much the same fashion, you could partition the table based on employee job codes—that is, based on ranges of job_code column values. For example—assuming that two-digit job codes are used for regular (in-store) workers, three-digit codes are used for office and support personnel, and four-digit codes are used for management positions—you could create the partitioned table using the following statement:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
)
PARTITION BY RANGE (job_code) (
    PARTITION p0 VALUES LESS THAN (100),
    PARTITION p1 VALUES LESS THAN (1000),
    PARTITION p2 VALUES LESS THAN (10000)
);
```

In this instance, all rows relating to in-store workers would be stored in partition p0, those relating to office and support staff in p1, and those relating to managers in partition p2.

It is also possible to use an expression in VALUES LESS THAN clauses. However, MySQL must be able to evaluate the expression's return value as part of a LESS THAN (<) comparison.

Rather than splitting up the table data according to store number, you can use an expression based on one of the two DATE columns instead. For example, let us suppose that you wish to partition based on the year that each employee left the company; that is, the value of YEAR(separated). An example of a CREATE TABLE statement that implements such a partitioning scheme is shown here:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY RANGE ( YEAR(separated) ) (
    PARTITION p0 VALUES LESS THAN (1991),
    PARTITION p1 VALUES LESS THAN (1996),
    PARTITION p2 VALUES LESS THAN (2001),
    PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

In this scheme, for all employees who left before 1991, the rows are stored in partition p0; for those who left in the years 1991 through 1995, in p1; for those who left in the years 1996 through 2000, in p2; and for any workers who left after the year 2000, in p3.

It is also possible to partition a table by RANGE, based on the value of a TIMESTAMP column, using the UNIX_TIMESTAMP() function, as shown in this example:

```
CREATE TABLE quarterly_report_status (
    report_id INT NOT NULL,
    report_status VARCHAR(20) NOT NULL,
    report_updated TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)
```

```
PARTITION BY RANGE ( UNIX_TIMESTAMP(report_updated) ) (
    PARTITION p0 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-01-01 00:00:00') ),
    PARTITION p1 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-04-01 00:00:00') ),
    PARTITION p2 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-07-01 00:00:00') ),
    PARTITION p3 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-10-01 00:00:00') ),
    PARTITION p4 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-01-01 00:00:00') ),
    PARTITION p5 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-04-01 00:00:00') ),
    PARTITION p6 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-07-01 00:00:00') ),
    PARTITION p7 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-10-01 00:00:00') ),
    PARTITION p8 VALUES LESS THAN ( UNIX_TIMESTAMP('2010-01-01 00:00:00') ),
    PARTITION p9 VALUES LESS THAN (MAXVALUE)
);
```

Any other expressions involving `TIMESTAMP` values are not permitted. (See Bug #42849.)

Range partitioning is particularly useful when one or more of the following conditions is true:

- You want or need to delete “old” data. If you are using the partitioning scheme shown previously for the `employees` table, you can simply use `ALTER TABLE employees DROP PARTITION p0;` to delete all rows relating to employees who stopped working for the firm prior to 1991. (See Section 13.1.7, “ALTER TABLE Statement”, and Section 19.3, “Partition Management”, for more information.) For a table with a great many rows, this can be much more efficient than running a `DELETE` query such as `DELETE FROM employees WHERE YEAR(separated) <= 1990;`.
- You want to use a column containing date or time values, or containing values arising from some other series.
- You frequently run queries that depend directly on the column used for partitioning the table. For example, when executing a query such as `EXPLAIN PARTITIONS SELECT COUNT(*) FROM employees WHERE separated BETWEEN '2000-01-01' AND '2000-12-31' GROUP BY store_id;`, MySQL can quickly determine that only partition `p2` needs to be scanned because the remaining partitions cannot contain any records satisfying the `WHERE` clause. See Section 19.4, “Partition Pruning”, for more information about how this is accomplished.

A variant on this type of partitioning is `RANGE COLUMNS` partitioning. Partitioning by `RANGE COLUMNS` makes it possible to employ multiple columns for defining partitioning ranges that apply both to placement of rows in partitions and for determining the inclusion or exclusion of specific partitions when performing partition pruning. See Section 19.2.3.1, “RANGE COLUMNS partitioning”, for more information.

Partitioning schemes based on time intervals. If you wish to implement a partitioning scheme based on ranges or intervals of time in MySQL 5.6, you have two options:

1. Partition the table by `RANGE`, and for the partitioning expression, employ a function operating on a `DATE`, `TIME`, or `DATETIME` column and returning an integer value, as shown here:

   ```
   CREATE TABLE members (
       firstname VARCHAR(25) NOT NULL,
       lastname VARCHAR(25) NOT NULL,
       username VARCHAR(16) NOT NULL,
       email VARCHAR(35),
       joined DATE NOT NULL
   )
   PARTITION BY RANGE( YEAR(joined) ) (
       PARTITION p0 VALUES LESS THAN (1960),
       PARTITION p1 VALUES LESS THAN (1970),
       PARTITION p2 VALUES LESS THAN (1980),
       PARTITION p3 VALUES LESS THAN (1990),
       PARTITION p4 VALUES LESS THAN MAXVALUE
   );
   ```

   In MySQL 5.6, it is also possible to partition a table by `RANGE` based on the value of a `TIMESTAMP` column, using the `UNIX_TIMESTAMP()` function, as shown in this example:

   ```
   CREATE TABLE quarterly_report_status (
       report_id INT NOT NULL,
   ```

```
    report_status VARCHAR(20) NOT NULL,
    report_updated TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRENT_TIMESTAMP
)
PARTITION BY RANGE ( UNIX_TIMESTAMP(report_updated) ) (
    PARTITION p0 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-01-01 00:00:00') ),
    PARTITION p1 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-04-01 00:00:00') ),
    PARTITION p2 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-07-01 00:00:00') ),
    PARTITION p3 VALUES LESS THAN ( UNIX_TIMESTAMP('2008-10-01 00:00:00') ),
    PARTITION p4 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-01-01 00:00:00') ),
    PARTITION p5 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-04-01 00:00:00') ),
    PARTITION p6 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-07-01 00:00:00') ),
    PARTITION p7 VALUES LESS THAN ( UNIX_TIMESTAMP('2009-10-01 00:00:00') ),
    PARTITION p8 VALUES LESS THAN ( UNIX_TIMESTAMP('2010-01-01 00:00:00') ),
    PARTITION p9 VALUES LESS THAN (MAXVALUE)
);
```

In MySQL 5.6, any other expressions involving `TIMESTAMP` values are not permitted. (See Bug #42849.)

Note

It is also possible in MySQL 5.6 to use `UNIX_TIMESTAMP(timestamp_column)` as a partitioning expression for tables that are partitioned by `LIST`. However, it is usually not practical to do so.

2. Partition the table by `RANGE COLUMNS`, using a `DATE` or `DATETIME` column as the partitioning column. For example, the `members` table could be defined using the `joined` column directly, as shown here:

```
CREATE TABLE members (
    firstname VARCHAR(25) NOT NULL,
    lastname VARCHAR(25) NOT NULL,
    username VARCHAR(16) NOT NULL,
    email VARCHAR(35),
    joined DATE NOT NULL
)
PARTITION BY RANGE COLUMNS(joined) (
    PARTITION p0 VALUES LESS THAN ('1960-01-01'),
    PARTITION p1 VALUES LESS THAN ('1970-01-01'),
    PARTITION p2 VALUES LESS THAN ('1980-01-01'),
    PARTITION p3 VALUES LESS THAN ('1990-01-01'),
    PARTITION p4 VALUES LESS THAN MAXVALUE
);
```

Note

The use of partitioning columns employing date or time types other than `DATE` or `DATETIME` is not supported with `RANGE COLUMNS`.

19.2.2 LIST Partitioning

List partitioning in MySQL is similar to range partitioning in many ways. As in partitioning by `RANGE`, each partition must be explicitly defined. The chief difference between the two types of partitioning is that, in list partitioning, each partition is defined and selected based on the membership of a column value in one of a set of value lists, rather than in one of a set of contiguous ranges of values. This is done by using `PARTITION BY LIST(expr)` where *`expr`* is a column value or an expression based on a column value and returning an integer value, and then defining each partition by means of a `VALUES IN (value_list)`, where *`value_list`* is a comma-separated list of integers.

Note

In MySQL 5.6, it is possible to match against only a list of integers (and possibly `NULL`—see Section 19.2.7, “How MySQL Partitioning Handles NULL”) when partitioning by `LIST`.

> However, other column types may be used in value lists when employing `LIST COLUMN` partitioning, which is described later in this section.

Unlike the case with partitions defined by range, list partitions do not need to be declared in any particular order. For more detailed syntactical information, see Section 13.1.17, “CREATE TABLE Statement”.

For the examples that follow, we assume that the basic definition of the table to be partitioned is provided by the `CREATE TABLE` statement shown here:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
);
```

(This is the same table used as a basis for the examples in Section 19.2.1, “RANGE Partitioning”.)

Suppose that there are 20 video stores distributed among 4 franchises as shown in the following table.

Region	Store ID Numbers
North	3, 5, 6, 9, 17
East	1, 2, 10, 11, 19, 20
West	4, 12, 13, 14, 18
Central	7, 8, 15, 16

To partition this table in such a way that rows for stores belonging to the same region are stored in the same partition, you could use the `CREATE TABLE` statement shown here:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY LIST(store_id) (
    PARTITION pNorth VALUES IN (3,5,6,9,17),
    PARTITION pEast VALUES IN (1,2,10,11,19,20),
    PARTITION pWest VALUES IN (4,12,13,14,18),
    PARTITION pCentral VALUES IN (7,8,15,16)
);
```

This makes it easy to add or drop employee records relating to specific regions to or from the table. For instance, suppose that all stores in the West region are sold to another company. In MySQL 5.6, all rows relating to employees working at stores in that region can be deleted with the query `ALTER TABLE employees TRUNCATE PARTITION pWest`, which can be executed much more efficiently than the equivalent `DELETE` statement `DELETE FROM employees WHERE store_id IN (4,12,13,14,18);`. (Using `ALTER TABLE employees DROP PARTITION pWest` would also delete all of these rows, but would also remove the partition `pWest` from the definition of the table; you would need to use an `ALTER TABLE ... ADD PARTITION` statement to restore the table's original partitioning scheme.)

As with `RANGE` partitioning, it is possible to combine `LIST` partitioning with partitioning by hash or key to produce a composite partitioning (subpartitioning). See Section 19.2.6, “Subpartitioning”.

Unlike the case with RANGE partitioning, there is no “catch-all” such as MAXVALUE; all expected values for the partitioning expression should be covered in PARTITION ... VALUES IN (...) clauses. An INSERT statement containing an unmatched partitioning column value fails with an error, as shown in this example:

```
mysql> CREATE TABLE h2 (
    ->   c1 INT,
    ->   c2 INT
    -> )
    -> PARTITION BY LIST(c1) (
    ->   PARTITION p0 VALUES IN (1, 4, 7),
    ->   PARTITION p1 VALUES IN (2, 5, 8)
    -> );
Query OK, 0 rows affected (0.11 sec)

mysql> INSERT INTO h2 VALUES (3, 5);
ERROR 1525 (HY000): Table has no partition for value 3
```

When inserting multiple rows using a single INSERT statement the behavior depends on whether the table uses a transactional storage engine. For an InnoDB table, the statement is considered a single transaction, so the presence of any unmatched values causes the statement to fail completely, and no rows are inserted. For a table using a nontransactional storage engine such as MyISAM, any rows coming before the row containing the unmatched value are inserted, but any coming after it are not.

You can cause this type of error to be ignored by using the IGNORE keyword. If you do so, rows containing unmatched partitioning column values are not inserted, but any rows with matching values *are* inserted, and no errors are reported:

```
mysql> TRUNCATE h2;
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM h2;
Empty set (0.00 sec)

mysql> INSERT IGNORE INTO h2 VALUES (2, 5), (6, 10), (7, 5), (3, 1), (1, 9);
Query OK, 3 rows affected (0.00 sec)
Records: 5  Duplicates: 2  Warnings: 0

mysql> SELECT * FROM h2;
+------+------+
| c1   | c2   |
+------+------+
|    7 |    5 |
|    1 |    9 |
|    2 |    5 |
+------+------+
3 rows in set (0.00 sec)
```

MySQL 5.6 provides support for LIST COLUMNS partitioning. This is a variant of LIST partitioning that enables you to use columns of types other than integer types for partitioning columns, as well as to use multiple columns as partitioning keys. For more information, see Section 19.2.3.2, “LIST COLUMNS partitioning”.

19.2.3 COLUMNS Partitioning

The next two sections discuss *COLUMNS partitioning*, which are variants on RANGE and LIST partitioning. COLUMNS partitioning enables the use of multiple columns in partitioning keys. All of these columns are taken into account both for the purpose of placing rows in partitions and for the determination of which partitions are to be checked for matching rows in partition pruning.

In addition, both RANGE COLUMNS partitioning and LIST COLUMNS partitioning support the use of non-integer columns for defining value ranges or list members. The permitted data types are shown in the following list:

- All integer types: TINYINT, SMALLINT, MEDIUMINT, INT (INTEGER), and BIGINT. (This is the same as with partitioning by RANGE and LIST.)

Other numeric data types (such as `DECIMAL` or `FLOAT`) are not supported as partitioning columns.

- `DATE` and `DATETIME`.

  Columns using other data types relating to dates or times are not supported as partitioning columns.

- The following string types: `CHAR`, `VARCHAR`, `BINARY`, and `VARBINARY`.

  `TEXT` and `BLOB` columns are not supported as partitioning columns.

The discussions of `RANGE COLUMNS` and `LIST COLUMNS` partitioning in the next two sections assume that you are already familiar with partitioning based on ranges and lists as supported in MySQL 5.1 and later; for more information about these, see Section 19.2.1, “RANGE Partitioning”, and Section 19.2.2, “LIST Partitioning”, respectively.

19.2.3.1 RANGE COLUMNS partitioning

Range columns partitioning is similar to range partitioning, but enables you to define partitions using ranges based on multiple column values. In addition, you can define the ranges using columns of types other than integer types.

`RANGE COLUMNS` partitioning differs significantly from `RANGE` partitioning in the following ways:

- `RANGE COLUMNS` does not accept expressions, only names of columns.
- `RANGE COLUMNS` accepts a list of one or more columns.

  `RANGE COLUMNS` partitions are based on comparisons between *tuples* (lists of column values) rather than comparisons between scalar values. Placement of rows in `RANGE COLUMNS` partitions is also based on comparisons between tuples; this is discussed further later in this section.

- `RANGE COLUMNS` partitioning columns are not restricted to integer columns; string, `DATE` and `DATETIME` columns can also be used as partitioning columns. (See Section 19.2.3, “COLUMNS Partitioning”, for details.)

The basic syntax for creating a table partitioned by `RANGE COLUMNS` is shown here:

```
CREATE TABLE table_name
PARTITIONED BY RANGE COLUMNS(column_list) (
    PARTITION partition_name VALUES LESS THAN (value_list)[,
    PARTITION partition_name VALUES LESS THAN (value_list)][,
    ...]
)

column_list:
    column_name[, column_name][, ...]

value_list:
    value[, value][, ...]
```

Note

Not all `CREATE TABLE` options that can be used when creating partitioned tables are shown here. For complete information, see Section 13.1.17, “CREATE TABLE Statement”.

In the syntax just shown, *`column_list`* is a list of one or more columns (sometimes called a *partitioning column list*), and *`value_list`* is a list of values (that is, it is a *partition definition value list*). A *`value_list`* must be supplied for each partition definition, and each *`value_list`* must have the same number of values as the *`column_list`* has columns. Generally speaking, if you use *`N`* columns in the `COLUMNS` clause, then each `VALUES LESS THAN` clause must also be supplied with a list of *`N`* values.

The elements in the partitioning column list and in the value list defining each partition must occur in the same order. In addition, each element in the value list must be of the same data type as the corresponding element in the column list. However, the order of the column names in the partitioning column list and the value lists does not have to be the same as the order of the table column definitions in the main part of the `CREATE TABLE` statement. As with table partitioned by `RANGE`, you can use `MAXVALUE` to represent a value such that any legal value inserted into a given column is always less than this value. Here is an example of a `CREATE TABLE` statement that helps to illustrate all of these points:

```
mysql> CREATE TABLE rcx (
    ->     a INT,
    ->     b INT,
    ->     c CHAR(3),
    ->     d INT
    -> )
    -> PARTITION BY RANGE COLUMNS(a,d,c) (
    ->     PARTITION p0 VALUES LESS THAN (5,10,'ggg'),
    ->     PARTITION p1 VALUES LESS THAN (10,20,'mmm'),
    ->     PARTITION p2 VALUES LESS THAN (15,30,'sss'),
    ->     PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE,MAXVALUE)
    -> );
Query OK, 0 rows affected (0.15 sec)
```

Table `rcx` contains the columns `a`, `b`, `c`, `d`. The partitioning column list supplied to the `COLUMNS` clause uses 3 of these columns, in the order `a`, `d`, `c`. Each value list used to define a partition contains 3 values in the same order; that is, each value list tuple has the form (`INT`, `INT`, `CHAR(3)`), which corresponds to the data types used by columns `a`, `d`, and `c` (in that order).

Placement of rows into partitions is determined by comparing the tuple from a row to be inserted that matches the column list in the `COLUMNS` clause with the tuples used in the `VALUES LESS THAN` clauses to define partitions of the table. Because we are comparing tuples (that is, lists or sets of values) rather than scalar values, the semantics of `VALUES LESS THAN` as used with `RANGE COLUMNS` partitions differs somewhat from the case with simple `RANGE` partitions. In `RANGE` partitioning, a row generating an expression value that is equal to a limiting value in a `VALUES LESS THAN` is never placed in the corresponding partition; however, when using `RANGE COLUMNS` partitioning, it is sometimes possible for a row whose partitioning column list's first element is equal in value to the that of the first element in a `VALUES LESS THAN` value list to be placed in the corresponding partition.

Consider the `RANGE` partitioned table created by this statement:

```
CREATE TABLE r1 (
    a INT,
    b INT
)
PARTITION BY RANGE (a)  (
    PARTITION p0 VALUES LESS THAN (5),
    PARTITION p1 VALUES LESS THAN (MAXVALUE)
);
```

If we insert 3 rows into this table such that the column value for `a` is `5` for each row, all 3 rows are stored in partition `p1` because the `a` column value is in each case not less than 5, as we can see by executing the proper query against the `INFORMATION_SCHEMA.PARTITIONS` table:

```
mysql> INSERT INTO r1 VALUES (5,10), (5,11), (5,12);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT PARTITION_NAME,TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'r1';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |          0 |
| p1             |          3 |
+----------------+------------+
```

```
2 rows in set (0.00 sec)
```

Now consider a similar table `rc1` that uses `RANGE COLUMNS` partitioning with both columns `a` and `b` referenced in the `COLUMNS` clause, created as shown here:

```
CREATE TABLE rc1 (
    a INT,
    b INT
)
PARTITION BY RANGE COLUMNS(a, b) (
    PARTITION p0 VALUES LESS THAN (5, 12),
    PARTITION p3 VALUES LESS THAN (MAXVALUE, MAXVALUE)
);
```

If we insert exactly the same rows into `rc1` as we just inserted into `r1`, the distribution of the rows is quite different:

```
mysql> INSERT INTO rc1 VALUES (5,10), (5,11), (5,12);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT PARTITION_NAME,TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'rc1';
+--------------+----------------+------------+
| TABLE_SCHEMA | PARTITION_NAME | TABLE_ROWS |
+--------------+----------------+------------+
| p            | p0             |          2 |
| p            | p1             |          1 |
+--------------+----------------+------------+
2 rows in set (0.00 sec)
```

This is because we are comparing rows rather than scalar values. We can compare the row values inserted with the limiting row value from the `VALUES THAN LESS THAN` clause used to define partition `p0` in table `rc1`, like this:

```
mysql> SELECT (5,10) < (5,12), (5,11) < (5,12), (5,12) < (5,12);
+----------------+----------------+----------------+
| (5,10) < (5,12) | (5,11) < (5,12) | (5,12) < (5,12) |
+----------------+----------------+----------------+
|              1 |              1 |              0 |
+----------------+----------------+----------------+
1 row in set (0.00 sec)
```

The 2 tuples `(5,10)` and `(5,11)` evaluate as less than `(5,12)`, so they are stored in partition `p0`. Since 5 is not less than 5 and 12 is not less than 12, `(5,12)` is considered not less than `(5,12)`, and is stored in partition `p1`.

The `SELECT` statement in the preceding example could also have been written using explicit row constructors, like this:

```
SELECT ROW(5,10) < ROW(5,12), ROW(5,11) < ROW(5,12), ROW(5,12) < ROW(5,12);
```

For more information about the use of row constructors in MySQL, see Section 13.2.10.5, “Row Subqueries”.

For a table partitioned by `RANGE COLUMNS` using only a single partitioning column, the storing of rows in partitions is the same as that of an equivalent table that is partitioned by `RANGE`. The following `CREATE TABLE` statement creates a table partitioned by `RANGE COLUMNS` using 1 partitioning column:

```
CREATE TABLE rx (
    a INT,
    b INT
)
PARTITION BY RANGE COLUMNS (a)  (
    PARTITION p0 VALUES LESS THAN (5),
    PARTITION p1 VALUES LESS THAN (MAXVALUE)
```

```
);
```

If we insert the rows `(5,10)`, `(5,11)`, and `(5,12)` into this table, we can see that their placement is the same as it is for the table `r` we created and populated earlier:

```
mysql> INSERT INTO rx VALUES (5,10), (5,11), (5,12);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql> SELECT PARTITION_NAME,TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'rx';
+--------------+----------------+------------+
| TABLE_SCHEMA | PARTITION_NAME | TABLE_ROWS |
+--------------+----------------+------------+
| p            | p0             |          0 |
| p            | p1             |          3 |
+--------------+----------------+------------+
2 rows in set (0.00 sec)
```

It is also possible to create tables partitioned by `RANGE COLUMNS` where limiting values for one or more columns are repeated in successive partition definitions. You can do this as long as the tuples of column values used to define the partitions are strictly increasing. For example, each of the following `CREATE TABLE` statements is valid:

```
CREATE TABLE rc2 (
    a INT,
    b INT
)
PARTITION BY RANGE COLUMNS(a,b) (
    PARTITION p0 VALUES LESS THAN (0,10),
    PARTITION p1 VALUES LESS THAN (10,20),
    PARTITION p2 VALUES LESS THAN (10,30),
    PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE)
 );

CREATE TABLE rc3 (
    a INT,
    b INT
)
PARTITION BY RANGE COLUMNS(a,b) (
    PARTITION p0 VALUES LESS THAN (0,10),
    PARTITION p1 VALUES LESS THAN (10,20),
    PARTITION p2 VALUES LESS THAN (10,30),
    PARTITION p3 VALUES LESS THAN (10,35),
    PARTITION p4 VALUES LESS THAN (20,40),
    PARTITION p5 VALUES LESS THAN (MAXVALUE,MAXVALUE)
 );
```

The following statement also succeeds, even though it might appear at first glance that it would not, since the limiting value of column `b` is 25 for partition `p0` and 20 for partition `p1`, and the limiting value of column `c` is 100 for partition `p1` and 50 for partition `p2`:

```
CREATE TABLE rc4 (
    a INT,
    b INT,
    c INT
)
PARTITION BY RANGE COLUMNS(a,b,c) (
    PARTITION p0 VALUES LESS THAN (0,25,50),
    PARTITION p1 VALUES LESS THAN (10,20,100),
    PARTITION p2 VALUES LESS THAN (10,30,50)
    PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE,MAXVALUE)
 );
```

When designing tables partitioned by `RANGE COLUMNS`, you can always test successive partition definitions by comparing the desired tuples using the `mysql` client, like this:

```
mysql> SELECT (0,25,50) < (10,20,100), (10,20,100) < (10,30,50);
+------------------------+-------------------------+
```

```
| (0,25,50) < (10,20,100) | (10,20,100) < (10,30,50) |
+-------------------------+--------------------------+
|                       1 |                        1 |
+-------------------------+--------------------------+
1 row in set (0.00 sec)
```

If a `CREATE TABLE` statement contains partition definitions that are not in strictly increasing order, it fails with an error, as shown in this example:

```
mysql> CREATE TABLE rcf (
    ->     a INT,
    ->     b INT,
    ->     c INT
    -> )
    -> PARTITION BY RANGE COLUMNS(a,b,c) (
    ->     PARTITION p0 VALUES LESS THAN (0,25,50),
    ->     PARTITION p1 VALUES LESS THAN (20,20,100),
    ->     PARTITION p2 VALUES LESS THAN (10,30,50),
    ->     PARTITION p3 VALUES LESS THAN (MAXVALUE,MAXVALUE,MAXVALUE)
    ->  );
ERROR 1493 (HY000): VALUES LESS THAN value must be strictly increasing for each partition
```

When you get such an error, you can deduce which partition definitions are invalid by making “less than” comparisons between their column lists. In this case, the problem is with the definition of partition `p2` because the tuple used to define it is not less than the tuple used to define partition `p3`, as shown here:

```
mysql> SELECT (0,25,50) < (20,20,100), (20,20,100) < (10,30,50);
+-------------------------+--------------------------+
| (0,25,50) < (20,20,100) | (20,20,100) < (10,30,50) |
+-------------------------+--------------------------+
|                       1 |                        0 |
+-------------------------+--------------------------+
1 row in set (0.00 sec)
```

It is also possible for `MAXVALUE` to appear for the same column in more than one `VALUES LESS THAN` clause when using `RANGE COLUMNS`. However, the limiting values for individual columns in successive partition definitions should otherwise be increasing, there should be no more than one partition defined where `MAXVALUE` is used as the upper limit for all column values, and this partition definition should appear last in the list of `PARTITION ... VALUES LESS THAN` clauses. In addition, you cannot use `MAXVALUE` as the limiting value for the first column in more than one partition definition.

As stated previously, it is also possible with `RANGE COLUMNS` partitioning to use non-integer columns as partitioning columns. (See Section 19.2.3, “COLUMNS Partitioning”, for a complete listing of these.) Consider a table named `employees` (which is not partitioned), created using the following statement:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
);
```

Using `RANGE COLUMNS` partitioning, you can create a version of this table that stores each row in one of four partitions based on the employee's last name, like this:

```
CREATE TABLE employees_by_lname (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT NOT NULL,
    store_id INT NOT NULL
)
```

```
PARTITION BY RANGE COLUMNS (lname)  (
    PARTITION p0 VALUES LESS THAN ('g'),
    PARTITION p1 VALUES LESS THAN ('m'),
    PARTITION p2 VALUES LESS THAN ('t'),
    PARTITION p3 VALUES LESS THAN (MAXVALUE)
);
```

Alternatively, you could cause the `employees` table as created previously to be partitioned using this scheme by executing the following `ALTER TABLE` statement:

```
ALTER TABLE employees PARTITION BY RANGE COLUMNS (lname)  (
    PARTITION p0 VALUES LESS THAN ('g'),
    PARTITION p1 VALUES LESS THAN ('m'),
    PARTITION p2 VALUES LESS THAN ('t'),
    PARTITION p3 VALUES LESS THAN (MAXVALUE)
);
```

Note

Because different character sets and collations have different sort orders, the character sets and collations in use may effect which partition of a table partitioned by `RANGE COLUMNS` a given row is stored in when using string columns as partitioning columns. In addition, changing the character set or collation for a given database, table, or column after such a table is created may cause changes in how rows are distributed. For example, when using a case-sensitive collation, `'and'` sorts before `'Andersen'`, but when using a collation that is case-insensitive, the reverse is true.

For information about how MySQL handles character sets and collations, see Chapter 10, *Character Sets, Collations, Unicode*.

Similarly, you can cause the `employees` table to be partitioned in such a way that each row is stored in one of several partitions based on the decade in which the corresponding employee was hired using the `ALTER TABLE` statement shown here:

```
ALTER TABLE employees PARTITION BY RANGE COLUMNS (hired)  (
    PARTITION p0 VALUES LESS THAN ('1970-01-01'),
    PARTITION p1 VALUES LESS THAN ('1980-01-01'),
    PARTITION p2 VALUES LESS THAN ('1990-01-01'),
    PARTITION p3 VALUES LESS THAN ('2000-01-01'),
    PARTITION p4 VALUES LESS THAN ('2010-01-01'),
    PARTITION p5 VALUES LESS THAN (MAXVALUE)
);
```

See Section 13.1.17, “CREATE TABLE Statement”, for additional information about `PARTITION BY RANGE COLUMNS` syntax.

19.2.3.2 LIST COLUMNS partitioning

MySQL 5.6 provides support for `LIST COLUMNS` partitioning. This is a variant of `LIST` partitioning that enables the use of multiple columns as partition keys, and for columns of data types other than integer types to be used as partitioning columns; you can use string types, `DATE`, and `DATETIME` columns. (For more information about permitted data types for `COLUMNS` partitioning columns, see Section 19.2.3, “COLUMNS Partitioning”.)

Suppose that you have a business that has customers in 12 cities which, for sales and marketing purposes, you organize into 4 regions of 3 cities each as shown in the following table:

Region	Cities
1	Oskarshamn, Högsby, Mönsterås
2	Vimmerby, Hultsfred, Västervik
3	Nässjö, Eksjö, Vetlanda

Region	Cities
4	Uppvidinge, Alvesta, Växjo

With `LIST COLUMNS` partitioning, you can create a table for customer data that assigns a row to any of 4 partitions corresponding to these regions based on the name of the city where a customer resides, as shown here:

```
CREATE TABLE customers_1 (
    first_name VARCHAR(25),
    last_name VARCHAR(25),
    street_1 VARCHAR(30),
    street_2 VARCHAR(30),
    city VARCHAR(15),
    renewal DATE
)
PARTITION BY LIST COLUMNS(city) (
    PARTITION pRegion_1 VALUES IN('Oskarshamn', 'Högsby', 'Mönsterås'),
    PARTITION pRegion_2 VALUES IN('Vimmerby', 'Hultsfred', 'Västervik'),
    PARTITION pRegion_3 VALUES IN('Nässjö', 'Eksjö', 'Vetlanda'),
    PARTITION pRegion_4 VALUES IN('Uppvidinge', 'Alvesta', 'Växjo')
);
```

As with partitioning by `RANGE COLUMNS`, you do not need to use expressions in the `COLUMNS()` clause to convert column values into integers. (In fact, the use of expressions other than column names is not permitted with `COLUMNS()`.)

It is also possible to use `DATE` and `DATETIME` columns, as shown in the following example that uses the same name and columns as the `customers_1` table shown previously, but employs `LIST COLUMNS` partitioning based on the `renewal` column to store rows in one of 4 partitions depending on the week in February 2010 the customer's account is scheduled to renew:

```
CREATE TABLE customers_2 (
    first_name VARCHAR(25),
    last_name VARCHAR(25),
    street_1 VARCHAR(30),
    street_2 VARCHAR(30),
    city VARCHAR(15),
    renewal DATE
)
PARTITION BY LIST COLUMNS(renewal) (
    PARTITION pWeek_1 VALUES IN('2010-02-01', '2010-02-02', '2010-02-03',
        '2010-02-04', '2010-02-05', '2010-02-06', '2010-02-07'),
    PARTITION pWeek_2 VALUES IN('2010-02-08', '2010-02-09', '2010-02-10',
        '2010-02-11', '2010-02-12', '2010-02-13', '2010-02-14'),
    PARTITION pWeek_3 VALUES IN('2010-02-15', '2010-02-16', '2010-02-17',
        '2010-02-18', '2010-02-19', '2010-02-20', '2010-02-21'),
    PARTITION pWeek_4 VALUES IN('2010-02-22', '2010-02-23', '2010-02-24',
        '2010-02-25', '2010-02-26', '2010-02-27', '2010-02-28')
);
```

This works, but becomes cumbersome to define and maintain if the number of dates involved grows very large; in such cases, it is usually more practical to employ `RANGE` or `RANGE COLUMNS` partitioning instead. In this case, since the column we wish to use as the partitioning key is a `DATE` column, we use `RANGE COLUMNS` partitioning, as shown here:

```
CREATE TABLE customers_3 (
    first_name VARCHAR(25),
    last_name VARCHAR(25),
    street_1 VARCHAR(30),
    street_2 VARCHAR(30),
    city VARCHAR(15),
    renewal DATE
)
PARTITION BY RANGE COLUMNS(renewal) (
    PARTITION pWeek_1 VALUES LESS THAN('2010-02-09'),
    PARTITION pWeek_2 VALUES LESS THAN('2010-02-15'),
    PARTITION pWeek_3 VALUES LESS THAN('2010-02-22'),
    PARTITION pWeek_4 VALUES LESS THAN('2010-03-01')
```

```
);
```

See Section 19.2.3.1, “RANGE COLUMNS partitioning”, for more information.

In addition (as with `RANGE COLUMNS` partitioning), you can use multiple columns in the `COLUMNS()` clause.

See Section 13.1.17, “CREATE TABLE Statement”, for additional information about `PARTITION BY LIST COLUMNS()` syntax.

19.2.4 HASH Partitioning

Partitioning by `HASH` is used primarily to ensure an even distribution of data among a predetermined number of partitions. With range or list partitioning, you must specify explicitly into which partition a given column value or set of column values is to be stored; with hash partitioning, MySQL takes care of this for you, and you need only specify a column value or expression based on a column value to be hashed and the number of partitions into which the partitioned table is to be divided.

To partition a table using `HASH` partitioning, it is necessary to append to the `CREATE TABLE` statement a `PARTITION BY HASH (expr)` clause, where *`expr`* is an expression that returns an integer. This can simply be the name of a column whose type is one of MySQL's integer types. In addition, you most likely want to follow this with `PARTITIONS num`, where *`num`* is a positive integer representing the number of partitions into which the table is to be divided.

Note

For simplicity, the tables in the examples that follow do not use any keys. You should be aware that, if a table has any unique keys, every column used in the partitioning expression for this table must be part of every unique key, including the primary key. See Section 19.6.1, “Partitioning Keys, Primary Keys, and Unique Keys”, for more information.

The following statement creates a table that uses hashing on the `store_id` column and is divided into 4 partitions:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY HASH(store_id)
PARTITIONS 4;
```

If you do not include a `PARTITIONS` clause, the number of partitions defaults to `1`.

Using the `PARTITIONS` keyword without a number following it results in a syntax error.

You can also use an SQL expression that returns an integer for *`expr`*. For instance, you might want to partition based on the year in which an employee was hired. This can be done as shown here:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY HASH( YEAR(hired) )
```

```
PARTITIONS 4;
```

expr must return a nonconstant, nonrandom integer value (in other words, it should be varying but deterministic), and must not contain any prohibited constructs as described in Section 19.6, “Restrictions and Limitations on Partitioning”. You should also keep in mind that this expression is evaluated each time a row is inserted or updated (or possibly deleted); this means that very complex expressions may give rise to performance issues, particularly when performing operations (such as batch inserts) that affect a great many rows at one time.

The most efficient hashing function is one which operates upon a single table column and whose value increases or decreases consistently with the column value, as this allows for “pruning” on ranges of partitions. That is, the more closely that the expression varies with the value of the column on which it is based, the more efficiently MySQL can use the expression for hash partitioning.

For example, where `date_col` is a column of type `DATE`, then the expression `TO_DAYS(date_col)` is said to vary directly with the value of `date_col`, because for every change in the value of `date_col`, the value of the expression changes in a consistent manner. The variance of the expression `YEAR(date_col)` with respect to `date_col` is not quite as direct as that of `TO_DAYS(date_col)`, because not every possible change in `date_col` produces an equivalent change in `YEAR(date_col)`. Even so, `YEAR(date_col)` is a good candidate for a hashing function, because it varies directly with a portion of `date_col` and there is no possible change in `date_col` that produces a disproportionate change in `YEAR(date_col)`.

By way of contrast, suppose that you have a column named `int_col` whose type is `INT`. Now consider the expression `POW(5-int_col,3) + 6`. This would be a poor choice for a hashing function because a change in the value of `int_col` is not guaranteed to produce a proportional change in the value of the expression. Changing the value of `int_col` by a given amount can produce widely differing changes in the value of the expression. For example, changing `int_col` from `5` to `6` produces a change of `-1` in the value of the expression, but changing the value of `int_col` from `6` to `7` produces a change of `-7` in the expression value.

In other words, the more closely the graph of the column value versus the value of the expression follows a straight line as traced by the equation `y=cx` where *c* is some nonzero constant, the better the expression is suited to hashing. This has to do with the fact that the more nonlinear an expression is, the more uneven the distribution of data among the partitions it tends to produce.

In theory, pruning is also possible for expressions involving more than one column value, but determining which of such expressions are suitable can be quite difficult and time-consuming. For this reason, the use of hashing expressions involving multiple columns is not particularly recommended.

When `PARTITION BY HASH` is used, MySQL determines which partition of *num* partitions to use based on the modulus of the result of the expression. In other words, for a given expression *expr*, the partition in which the record is stored is partition number *N*, where *N = MOD(expr, num)*. Suppose that table `t1` is defined as follows, so that it has 4 partitions:

```
CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATE)
    PARTITION BY HASH( YEAR(col3) )
    PARTITIONS 4;
```

If you insert a record into `t1` whose `col3` value is `'2005-09-15'`, then the partition in which it is stored is determined as follows:

```
MOD(YEAR('2005-09-01'),4)
=  MOD(2005,4)
=  1
```

MySQL 5.6 also supports a variant of `HASH` partitioning known as *linear hashing* which employs a more complex algorithm for determining the placement of new rows inserted into the partitioned table. See Section 19.2.4.1, “LINEAR HASH Partitioning”, for a description of this algorithm.

The user-supplied expression is evaluated each time a record is inserted or updated. It may also—depending on the circumstances—be evaluated when records are deleted.

19.2.4.1 LINEAR HASH Partitioning

MySQL also supports linear hashing, which differs from regular hashing in that linear hashing utilizes a linear powers-of-two algorithm whereas regular hashing employs the modulus of the hashing function's value.

Syntactically, the only difference between linear-hash partitioning and regular hashing is the addition of the `LINEAR` keyword in the `PARTITION BY` clause, as shown here:

```
CREATE TABLE employees (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30),
    hired DATE NOT NULL DEFAULT '1970-01-01',
    separated DATE NOT NULL DEFAULT '9999-12-31',
    job_code INT,
    store_id INT
)
PARTITION BY LINEAR HASH( YEAR(hired) )
PARTITIONS 4;
```

Given an expression *`expr`*, the partition in which the record is stored when linear hashing is used is partition number *`N`* from among *`num`* partitions, where *`N`* is derived according to the following algorithm:

1. Find the next power of 2 greater than *`num`*. We call this value *`V`*; it can be calculated as:

   ```
   V = POWER(2, CEILING(LOG(2, num)))
   ```

   (Suppose that *`num`* is 13. Then `LOG(2,13)` is 3.7004397181411. `CEILING(3.7004397181411)` is 4, and *`V`* = `POWER(2,4)`, which is 16.)

2. Set *`N`* = *`F`*(*`column_list`*) & (*`V`* - 1).

3. While *`N`* >= *`num`*:

   - Set *`V`* = *`V`* / 2
   - Set *`N`* = *`N`* & (*`V`* - 1)

Suppose that the table `t1`, using linear hash partitioning and having 6 partitions, is created using this statement:

```
CREATE TABLE t1 (col1 INT, col2 CHAR(5), col3 DATE)
    PARTITION BY LINEAR HASH( YEAR(col3) )
    PARTITIONS 6;
```

Now assume that you want to insert two records into `t1` having the `col3` column values `'2003-04-14'` and `'1998-10-19'`. The partition number for the first of these is determined as follows:

```
V = POWER(2, CEILING( LOG(2,6) )) = 8
N = YEAR('2003-04-14') & (8 - 1)
   = 2003 & 7
   = 3

(3 >= 6 is FALSE: record stored in partition #3)
```

The number of the partition where the second record is stored is calculated as shown here:

```
V = 8
N = YEAR('1998-10-19') & (8 - 1)
  = 1998 & 7
  = 6

(6 >= 6 is TRUE: additional step required)
```

```
N = 6 & ((8 / 2) - 1)
  = 6 & 3
  = 2

(2 >= 6 is FALSE: record stored in partition #2)
```

The advantage in partitioning by linear hash is that the adding, dropping, merging, and splitting of partitions is made much faster, which can be beneficial when dealing with tables containing extremely large amounts (terabytes) of data. The disadvantage is that data is less likely to be evenly distributed between partitions as compared with the distribution obtained using regular hash partitioning.

19.2.5 KEY Partitioning

Partitioning by key is similar to partitioning by hash, except that where hash partitioning employs a user-defined expression, the hashing function for key partitioning is supplied by the MySQL server. NDB Cluster uses `MD5()` for this purpose; for tables using other storage engines, the server employs its own internal hashing function which is based on the same algorithm as `PASSWORD()`.

The syntax rules for `CREATE TABLE ... PARTITION BY KEY` are similar to those for creating a table that is partitioned by hash. The major differences are listed here:

- `KEY` is used rather than `HASH`.
- `KEY` takes only a list of zero or more column names. Any columns used as the partitioning key must comprise part or all of the table's primary key, if the table has one. Where no column name is specified as the partitioning key, the table's primary key is used, if there is one. For example, the following `CREATE TABLE` statement is valid in MySQL 5.6:

  ```
  CREATE TABLE k1 (
      id INT NOT NULL PRIMARY KEY,
      name VARCHAR(20)
  )
  PARTITION BY KEY()
  PARTITIONS 2;
  ```

  If there is no primary key but there is a unique key, then the unique key is used for the partitioning key:

  ```
  CREATE TABLE k1 (
      id INT NOT NULL,
      name VARCHAR(20),
      UNIQUE KEY (id)
  )
  PARTITION BY KEY()
  PARTITIONS 2;
  ```

  However, if the unique key column were not defined as `NOT NULL`, then the previous statement would fail.

  In both of these cases, the partitioning key is the `id` column, even though it is not shown in the output of `SHOW CREATE TABLE` or in the `PARTITION_EXPRESSION` column of the `INFORMATION_SCHEMA.PARTITIONS` table.

  Unlike the case with other partitioning types, columns used for partitioning by `KEY` are not restricted to integer or `NULL` values. For example, the following `CREATE TABLE` statement is valid:

  ```
  CREATE TABLE tm1 (
      s1 CHAR(32) PRIMARY KEY
  )
  PARTITION BY KEY(s1)
  PARTITIONS 10;
  ```

  The preceding statement would *not* be valid, were a different partitioning type to be specified. (In this case, simply using `PARTITION BY KEY()` would also be valid and have the same effect as `PARTITION BY KEY(s1)`, since `s1` is the table's primary key.)

For additional information about this issue, see Section 19.6, “Restrictions and Limitations on Partitioning”.

Columns with index prefixes are not supported in partitioning keys. This means that `CHAR`, `VARCHAR`, `BINARY`, and `VARBINARY` columns can be used in a partitioning key, as long as they do not employ prefixes; because a prefix must be specified for `BLOB` and `TEXT` columns in index definitions, it is not possible to use columns of these two types in partitioning keys. In MySQL 5.6, columns using prefixes are permitted when creating, altering, or upgrading partitioned tables, even though they are not included in the table's partitioning key. This is a known issue in MySQL 5.6 which is addressed in MySQL 8.0, where this permissive behavior is deprecated, and the server displays appropriate warnings or errors when attempting to use such columns in these cases. See Column index prefixes not supported for key partitioning, for more information and examples.

Note

Tables using the `NDB` storage engine are implicitly partitioned by `KEY`, using the table's primary key as the partitioning key (as with other MySQL storage engines). In the event that the NDB Cluster table has no explicit primary key, the “hidden” primary key generated by the `NDB` storage engine for each NDB Cluster table is used as the partitioning key.

If you define an explicit partitioning scheme for an `NDB` table, the table must have an explicit primary key, and any columns used in the partitioning expression must be part of this key. However, if the table uses an “empty” partitioning expression—that is, `PARTITION BY KEY()` with no column references—then no explicit primary key is required.

You can observe this partitioning using the `ndb_desc` utility (with the `-p` option).

Important

For a key-partitioned table, you cannot execute an `ALTER TABLE DROP PRIMARY KEY`, as doing so generates the error `ERROR 1466 (HY000): Field in list of fields for partition function not found in table`. This is not an issue for NDB Cluster tables which are partitioned by `KEY`; in such cases, the table is reorganized using the “hidden” primary key as the table's new partitioning key. See Chapter 18, *MySQL NDB Cluster 7.3 and NDB Cluster 7.4*.

It is also possible to partition a table by linear key. Here is a simple example:

```
CREATE TABLE tk (
    col1 INT NOT NULL,
    col2 CHAR(5),
    col3 DATE
)
PARTITION BY LINEAR KEY (col1)
PARTITIONS 3;
```

Using `LINEAR` has the same effect on `KEY` partitioning as it does on `HASH` partitioning, with the partition number being derived using a powers-of-two algorithm rather than modulo arithmetic. See Section 19.2.4.1, “LINEAR HASH Partitioning”, for a description of this algorithm and its implications.

19.2.6 Subpartitioning

Subpartitioning—also known as *composite partitioning*—is the further division of each partition in a partitioned table. Consider the following `CREATE TABLE` statement:

```
CREATE TABLE ts (id INT, purchased DATE)
```

```
    PARTITION BY RANGE( YEAR(purchased) )
    SUBPARTITION BY HASH( TO_DAYS(purchased) )
    SUBPARTITIONS 2 (
        PARTITION p0 VALUES LESS THAN (1990),
        PARTITION p1 VALUES LESS THAN (2000),
        PARTITION p2 VALUES LESS THAN MAXVALUE
    );
```

Table `ts` has 3 `RANGE` partitions. Each of these partitions—`p0`, `p1`, and `p2`—is further divided into 2 subpartitions. In effect, the entire table is divided into `3 * 2 = 6` partitions. However, due to the action of the `PARTITION BY RANGE` clause, the first 2 of these store only those records with a value less than 1990 in the `purchased` column.

In MySQL 5.6, it is possible to subpartition tables that are partitioned by `RANGE` or `LIST`. Subpartitions may use either `HASH` or `KEY` partitioning. This is also known as *composite partitioning*.

Note

`SUBPARTITION BY HASH` and `SUBPARTITION BY KEY` generally follow the same syntax rules as `PARTITION BY HASH` and `PARTITION BY KEY`, respectively. An exception to this is that `SUBPARTITION BY KEY` (unlike `PARTITION BY KEY`) does not currently support a default column, so the column used for this purpose must be specified, even if the table has an explicit primary key. This is a known issue which we are working to address; see Issues with subpartitions, for more information and an example.

It is also possible to define subpartitions explicitly using `SUBPARTITION` clauses to specify options for individual subpartitions. For example, a more verbose fashion of creating the same table `ts` as shown in the previous example would be:

```
CREATE TABLE ts (id INT, purchased DATE)
    PARTITION BY RANGE( YEAR(purchased) )
    SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
        PARTITION p0 VALUES LESS THAN (1990) (
            SUBPARTITION s0,
            SUBPARTITION s1
        ),
        PARTITION p1 VALUES LESS THAN (2000) (
            SUBPARTITION s2,
            SUBPARTITION s3
        ),
        PARTITION p2 VALUES LESS THAN MAXVALUE (
            SUBPARTITION s4,
            SUBPARTITION s5
        )
    );
```

Some syntactical items of note are listed here:

- Each partition must have the same number of subpartitions.
- If you explicitly define any subpartitions using `SUBPARTITION` on any partition of a partitioned table, you must define them all. In other words, the following statement fails:

```
CREATE TABLE ts (id INT, purchased DATE)
    PARTITION BY RANGE( YEAR(purchased) )
    SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
        PARTITION p0 VALUES LESS THAN (1990) (
            SUBPARTITION s0,
            SUBPARTITION s1
        ),
        PARTITION p1 VALUES LESS THAN (2000),
        PARTITION p2 VALUES LESS THAN MAXVALUE (
            SUBPARTITION s2,
            SUBPARTITION s3
```

```
        )
    );
```

This statement would still fail even if it included a `SUBPARTITIONS 2` clause.

- Each `SUBPARTITION` clause must include (at a minimum) a name for the subpartition. Otherwise, you may set any desired option for the subpartition or allow it to assume its default setting for that option.
- Subpartition names must be unique across the entire table. For example, the following `CREATE TABLE` statement is valid in MySQL 5.6:

```
CREATE TABLE ts (id INT, purchased DATE)
    PARTITION BY RANGE( YEAR(purchased) )
    SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
        PARTITION p0 VALUES LESS THAN (1990) (
            SUBPARTITION s0,
            SUBPARTITION s1
        ),
        PARTITION p1 VALUES LESS THAN (2000) (
            SUBPARTITION s2,
            SUBPARTITION s3
        ),
        PARTITION p2 VALUES LESS THAN MAXVALUE (
            SUBPARTITION s4,
            SUBPARTITION s5
        )
    );
```

Subpartitions can be used with especially large `MyISAM` tables to distribute data and indexes across many disks. Suppose that you have 6 disks mounted as `/disk0`, `/disk1`, `/disk2`, and so on. Now consider the following example:

```
CREATE TABLE ts (id INT, purchased DATE)
    ENGINE = MYISAM
    PARTITION BY RANGE( YEAR(purchased) )
    SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
        PARTITION p0 VALUES LESS THAN (1990) (
            SUBPARTITION s0
                DATA DIRECTORY = '/disk0/data'
                INDEX DIRECTORY = '/disk0/idx',
            SUBPARTITION s1
                DATA DIRECTORY = '/disk1/data'
                INDEX DIRECTORY = '/disk1/idx'
        ),
        PARTITION p1 VALUES LESS THAN (2000) (
            SUBPARTITION s2
                DATA DIRECTORY = '/disk2/data'
                INDEX DIRECTORY = '/disk2/idx',
            SUBPARTITION s3
                DATA DIRECTORY = '/disk3/data'
                INDEX DIRECTORY = '/disk3/idx'
        ),
        PARTITION p2 VALUES LESS THAN MAXVALUE (
            SUBPARTITION s4
                DATA DIRECTORY = '/disk4/data'
                INDEX DIRECTORY = '/disk4/idx',
            SUBPARTITION s5
                DATA DIRECTORY = '/disk5/data'
                INDEX DIRECTORY = '/disk5/idx'
        )
    );
```

In this case, a separate disk is used for the data and for the indexes of each `RANGE`. Many other variations are possible; another example might be:

```
CREATE TABLE ts (id INT, purchased DATE)
    ENGINE = MYISAM
    PARTITION BY RANGE(YEAR(purchased))
    SUBPARTITION BY HASH( TO_DAYS(purchased) ) (
```

```
        PARTITION p0 VALUES LESS THAN (1990) (
            SUBPARTITION s0a
                DATA DIRECTORY = '/disk0'
                INDEX DIRECTORY = '/disk1',
            SUBPARTITION s0b
                DATA DIRECTORY = '/disk2'
                INDEX DIRECTORY = '/disk3'
        ),
        PARTITION p1 VALUES LESS THAN (2000) (
            SUBPARTITION s1a
                DATA DIRECTORY = '/disk4/data'
                INDEX DIRECTORY = '/disk4/idx',
            SUBPARTITION s1b
                DATA DIRECTORY = '/disk5/data'
                INDEX DIRECTORY = '/disk5/idx'
        ),
        PARTITION p2 VALUES LESS THAN MAXVALUE (
            SUBPARTITION s2a,
            SUBPARTITION s2b
        )
    );
```

Here, the storage is as follows:

- Rows with `purchased` dates from before 1990 take up a vast amount of space, so are split up 4 ways, with a separate disk dedicated to the data and to the indexes for each of the two subpartitions (`s0a` and `s0b`) making up partition `p0`. In other words:
  - The data for subpartition `s0a` is stored on `/disk0`.
  - The indexes for subpartition `s0a` are stored on `/disk1`.
  - The data for subpartition `s0b` is stored on `/disk2`.
  - The indexes for subpartition `s0b` are stored on `/disk3`.
- Rows containing dates ranging from 1990 to 1999 (partition `p1`) do not require as much room as those from before 1990. These are split between 2 disks (`/disk4` and `/disk5`) rather than 4 disks as with the legacy records stored in `p0`:
  - Data and indexes belonging to `p1`'s first subpartition (`s1a`) are stored on `/disk4`—the data in `/disk4/data`, and the indexes in `/disk4/idx`.
  - Data and indexes belonging to `p1`'s second subpartition (`s1b`) are stored on `/disk5`—the data in `/disk5/data`, and the indexes in `/disk5/idx`.
- Rows reflecting dates from the year 2000 to the present (partition `p2`) do not take up as much space as required by either of the two previous ranges. Currently, it is sufficient to store all of these in the default location.

  In future, when the number of purchases for the decade beginning with the year 2000 grows to a point where the default location no longer provides sufficient space, the corresponding rows can be moved using an `ALTER TABLE ... REORGANIZE PARTITION` statement. See Section 19.3, “Partition Management”, for an explanation of how this can be done.

The `DATA DIRECTORY` and `INDEX DIRECTORY` options are not permitted in partition definitions when the `NO_DIR_IN_CREATE` server SQL mode is in effect. In MySQL 5.6, these options are also not permitted when defining subpartitions (Bug #42954).

19.2.7 How MySQL Partitioning Handles NULL

Partitioning in MySQL does nothing to disallow `NULL` as the value of a partitioning expression, whether it is a column value or the value of a user-supplied expression. Even though it is permitted to use `NULL` as the value of an expression that must otherwise yield an integer, it is important to keep in mind

that NULL is not a number. MySQL's partitioning implementation treats NULL as being less than any non-NULL value, just as ORDER BY does.

This means that treatment of NULL varies between partitioning of different types, and may produce behavior which you do not expect if you are not prepared for it. This being the case, we discuss in this section how each MySQL partitioning type handles NULL values when determining the partition in which a row should be stored, and provide examples for each.

Handling of NULL with RANGE partitioning. If you insert a row into a table partitioned by RANGE such that the column value used to determine the partition is NULL, the row is inserted into the lowest partition. Consider these two tables in a database named p, created as follows:

```
mysql> CREATE TABLE t1 (
    ->     c1 INT,
    ->     c2 VARCHAR(20)
    -> )
    -> PARTITION BY RANGE(c1) (
    ->     PARTITION p0 VALUES LESS THAN (0),
    ->     PARTITION p1 VALUES LESS THAN (10),
    ->     PARTITION p2 VALUES LESS THAN MAXVALUE
    -> );
Query OK, 0 rows affected (0.09 sec)

mysql> CREATE TABLE t2 (
    ->     c1 INT,
    ->     c2 VARCHAR(20)
    -> )
    -> PARTITION BY RANGE(c1) (
    ->     PARTITION p0 VALUES LESS THAN (-5),
    ->     PARTITION p1 VALUES LESS THAN (0),
    ->     PARTITION p2 VALUES LESS THAN (10),
    ->     PARTITION p3 VALUES LESS THAN MAXVALUE
    -> );
Query OK, 0 rows affected (0.09 sec)
```

You can see the partitions created by these two CREATE TABLE statements using the following query against the PARTITIONS table in the INFORMATION_SCHEMA database:

```
mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH, DATA_LENGTH
     >   FROM INFORMATION_SCHEMA.PARTITIONS
     >   WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME LIKE 't_';
+------------+----------------+------------+----------------+-------------+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+------------+----------------+------------+----------------+-------------+
| t1         | p0             |          0 |              0 |           0 |
| t1         | p1             |          0 |              0 |           0 |
| t1         | p2             |          0 |              0 |           0 |
| t2         | p0             |          0 |              0 |           0 |
| t2         | p1             |          0 |              0 |           0 |
| t2         | p2             |          0 |              0 |           0 |
| t2         | p3             |          0 |              0 |           0 |
+------------+----------------+------------+----------------+-------------+
7 rows in set (0.00 sec)
```

(For more information about this table, see Section 21.14, “The INFORMATION_SCHEMA PARTITIONS Table”.) Now let us populate each of these tables with a single row containing a NULL in the column used as the partitioning key, and verify that the rows were inserted using a pair of SELECT statements:

```
mysql> INSERT INTO t1 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO t2 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM t1;
+------+--------+
| id   | name   |
+------+--------+
```

```
| NULL | mothra |
+------+--------+
1 row in set (0.00 sec)

mysql> SELECT * FROM t2;
+------+--------+
| id   | name   |
+------+--------+
| NULL | mothra |
+------+--------+
1 row in set (0.00 sec)
```

You can see which partitions are used to store the inserted rows by rerunning the previous query against `INFORMATION_SCHEMA.PARTITIONS` and inspecting the output:

```
mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH, DATA_LENGTH
     >   FROM INFORMATION_SCHEMA.PARTITIONS
     >   WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME LIKE 't_';
+------------+----------------+------------+----------------+-------------+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+------------+----------------+------------+----------------+-------------+
| t1         | p0             |          1 |             20 |          20 |
| t1         | p1             |          0 |              0 |           0 |
| t1         | p2             |          0 |              0 |           0 |
| t2         | p0             |          1 |             20 |          20 |
| t2         | p1             |          0 |              0 |           0 |
| t2         | p2             |          0 |              0 |           0 |
| t2         | p3             |          0 |              0 |           0 |
+------------+----------------+------------+----------------+-------------+
7 rows in set (0.01 sec)
```

You can also demonstrate that these rows were stored in the lowest partition of each table by dropping these partitions, and then re-running the `SELECT` statements:

```
mysql> ALTER TABLE t1 DROP PARTITION p0;
Query OK, 0 rows affected (0.16 sec)

mysql> ALTER TABLE t2 DROP PARTITION p0;
Query OK, 0 rows affected (0.16 sec)

mysql> SELECT * FROM t1;
Empty set (0.00 sec)

mysql> SELECT * FROM t2;
Empty set (0.00 sec)
```

(For more information on `ALTER TABLE ... DROP PARTITION`, see Section 13.1.7, “ALTER TABLE Statement”.)

`NULL` is also treated in this way for partitioning expressions that use SQL functions. Suppose that we define a table using a `CREATE TABLE` statement such as this one:

```
CREATE TABLE tndate (
    id INT,
    dt DATE
)
PARTITION BY RANGE( YEAR(dt) ) (
    PARTITION p0 VALUES LESS THAN (1990),
    PARTITION p1 VALUES LESS THAN (2000),
    PARTITION p2 VALUES LESS THAN MAXVALUE
);
```

As with other MySQL functions, `YEAR(NULL)` returns `NULL`. A row with a `dt` column value of `NULL` is treated as though the partitioning expression evaluated to a value less than any other value, and so is inserted into partition `p0`.

Handling of NULL with LIST partitioning. A table that is partitioned by `LIST` admits `NULL` values if and only if one of its partitions is defined using that value-list that contains `NULL`. The converse of

this is that a table partitioned by `LIST` which does not explicitly use `NULL` in a value list rejects rows resulting in a `NULL` value for the partitioning expression, as shown in this example:

```
mysql> CREATE TABLE ts1 (
    ->     c1 INT,
    ->     c2 VARCHAR(20)
    -> )
    -> PARTITION BY LIST(c1) (
    ->     PARTITION p0 VALUES IN (0, 3, 6),
    ->     PARTITION p1 VALUES IN (1, 4, 7),
    ->     PARTITION p2 VALUES IN (2, 5, 8)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> INSERT INTO ts1 VALUES (9, 'mothra');
ERROR 1504 (HY000): Table has no partition for value 9

mysql> INSERT INTO ts1 VALUES (NULL, 'mothra');
ERROR 1504 (HY000): Table has no partition for value NULL
```

Only rows having a `c1` value between `0` and `8` inclusive can be inserted into `ts1`. `NULL` falls outside this range, just like the number `9`. We can create tables `ts2` and `ts3` having value lists containing `NULL`, as shown here:

```
mysql> CREATE TABLE ts2 (
    ->     c1 INT,
    ->     c2 VARCHAR(20)
    -> )
    -> PARTITION BY LIST(c1) (
    ->     PARTITION p0 VALUES IN (0, 3, 6),
    ->     PARTITION p1 VALUES IN (1, 4, 7),
    ->     PARTITION p2 VALUES IN (2, 5, 8),
    ->     PARTITION p3 VALUES IN (NULL)
    -> );
Query OK, 0 rows affected (0.01 sec)

mysql> CREATE TABLE ts3 (
    ->     c1 INT,
    ->     c2 VARCHAR(20)
    -> )
    -> PARTITION BY LIST(c1) (
    ->     PARTITION p0 VALUES IN (0, 3, 6),
    ->     PARTITION p1 VALUES IN (1, 4, 7, NULL),
    ->     PARTITION p2 VALUES IN (2, 5, 8)
    -> );
Query OK, 0 rows affected (0.01 sec)
```

When defining value lists for partitioning, you can (and should) treat `NULL` just as you would any other value. For example, both `VALUES IN (NULL)` and `VALUES IN (1, 4, 7, NULL)` are valid, as are `VALUES IN (1, NULL, 4, 7)`, `VALUES IN (NULL, 1, 4, 7)`, and so on. You can insert a row having `NULL` for column `c1` into each of the tables `ts2` and `ts3`:

```
mysql> INSERT INTO ts2 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO ts3 VALUES (NULL, 'mothra');
Query OK, 1 row affected (0.00 sec)
```

By issuing the appropriate query against `INFORMATION_SCHEMA.PARTITIONS`, you can determine which partitions were used to store the rows just inserted (we assume, as in the previous examples, that the partitioned tables were created in the `p` database):

```
mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH, DATA_LENGTH
     >   FROM INFORMATION_SCHEMA.PARTITIONS
     >   WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME LIKE 'ts_';
+------------+----------------+------------+----------------+-------------+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+------------+----------------+------------+----------------+-------------+
| ts2        | p0             |          0 |              0 |           0 |
```

```
| ts2        | p1             |           0 |               0 |           0 |
| ts2        | p2             |           0 |               0 |           0 |
| ts2        | p3             |           1 |              20 |          20 |
| ts3        | p0             |           0 |               0 |           0 |
| ts3        | p1             |           1 |              20 |          20 |
| ts3        | p2             |           0 |               0 |           0 |
+------------+----------------+-------------+-----------------+-------------+
7 rows in set (0.01 sec)
```

As shown earlier in this section, you can also verify which partitions were used for storing the rows by deleting these partitions and then performing a `SELECT`.

Handling of NULL with HASH and KEY partitioning. `NULL` is handled somewhat differently for tables partitioned by `HASH` or `KEY`. In these cases, any partition expression that yields a `NULL` value is treated as though its return value were zero. We can verify this behavior by examining the effects on the file system of creating a table partitioned by `HASH` and populating it with a record containing appropriate values. Suppose that you have a table `th` (also in the `p` database) created using the following statement:

```
mysql> CREATE TABLE th (
    ->     c1 INT,
    ->     c2 VARCHAR(20)
    -> )
    -> PARTITION BY HASH(c1)
    -> PARTITIONS 2;
Query OK, 0 rows affected (0.00 sec)
```

The partitions belonging to this table can be viewed using the query shown here:

```
mysql> SELECT TABLE_NAME,PARTITION_NAME,TABLE_ROWS,AVG_ROW_LENGTH,DATA_LENGTH
     >   FROM INFORMATION_SCHEMA.PARTITIONS
     >   WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME ='th';
+------------+----------------+------------+----------------+-------------+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+------------+----------------+------------+----------------+-------------+
| th         | p0             |          0 |              0 |           0 |
| th         | p1             |          0 |              0 |           0 |
+------------+----------------+------------+----------------+-------------+
2 rows in set (0.00 sec)
```

Notice that `TABLE_ROWS` for each partition is 0. Now insert two rows into `th` whose `c1` column values are `NULL` and 0, and verify that these rows were inserted, as shown here:

```
mysql> INSERT INTO th VALUES (NULL, 'mothra'), (0, 'gigan');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM th;
+------+---------+
| c1   | c2      |
+------+---------+
| NULL | mothra  |
+------+---------+
|    0 | gigan   |
+------+---------+
2 rows in set (0.01 sec)
```

Recall that for any integer *`N`*, the value of `NULL MOD N` is always `NULL`. For tables that are partitioned by `HASH` or `KEY`, this result is treated for determining the correct partition as `0`. Checking the `INFORMATION_SCHEMA.PARTITIONS` table once again, we can see that both rows were inserted into partition `p0`:

```
mysql> SELECT TABLE_NAME, PARTITION_NAME, TABLE_ROWS, AVG_ROW_LENGTH, DATA_LENGTH
     >   FROM INFORMATION_SCHEMA.PARTITIONS
     >   WHERE TABLE_SCHEMA = 'p' AND TABLE_NAME ='th';
+------------+----------------+------------+----------------+-------------+
| TABLE_NAME | PARTITION_NAME | TABLE_ROWS | AVG_ROW_LENGTH | DATA_LENGTH |
+------------+----------------+------------+----------------+-------------+
| th         | p0             |          2 |             20 |          20 |
```

```
| th         | p1             |          0 |              0 |           0 |
+------------+----------------+------------+----------------+-------------+
2 rows in set (0.00 sec)
```

If you repeat this example using `PARTITION BY KEY` in place of `PARTITION BY HASH` in the definition of the table, you can verify easily that `NULL` is also treated like 0 for this type of partitioning.

19.3 Partition Management

MySQL 5.6 provides a number of ways to modify partitioned tables. It is possible to add, drop, redefine, merge, or split existing partitions. All of these actions can be carried out using the partitioning extensions to the `ALTER TABLE` statement. There are also ways to obtain information about partitioned tables and partitions. We discuss these topics in the sections that follow.

- For information about partition management in tables partitioned by `RANGE` or `LIST`, see Section 19.3.1, “Management of RANGE and LIST Partitions”.
- For a discussion of managing `HASH` and `KEY` partitions, see Section 19.3.2, “Management of HASH and KEY Partitions”.
- See Section 19.3.5, “Obtaining Information About Partitions”, for a discussion of mechanisms provided in MySQL 5.6 for obtaining information about partitioned tables and partitions.
- For a discussion of performing maintenance operations on partitions, see Section 19.3.4, “Maintenance of Partitions”.

Note

In MySQL 5.6, all partitions of a partitioned table must have the same number of subpartitions, and it is not possible to change the subpartitioning once the table has been created.

To change a table's partitioning scheme, it is necessary only to use the `ALTER TABLE` statement with a *`partition_options`* clause. This clause has the same syntax as that as used with `CREATE TABLE` for creating a partitioned table, and always begins with the keywords `PARTITION BY`. Suppose that you have a table partitioned by range using the following `CREATE TABLE` statement:

```
CREATE TABLE trb3 (id INT, name VARCHAR(50), purchased DATE)
    PARTITION BY RANGE( YEAR(purchased) ) (
        PARTITION p0 VALUES LESS THAN (1990),
        PARTITION p1 VALUES LESS THAN (1995),
        PARTITION p2 VALUES LESS THAN (2000),
        PARTITION p3 VALUES LESS THAN (2005)
    );
```

To repartition this table so that it is partitioned by key into two partitions using the `id` column value as the basis for the key, you can use this statement:

```
ALTER TABLE trb3 PARTITION BY KEY(id) PARTITIONS 2;
```

This has the same effect on the structure of the table as dropping the table and re-creating it using `CREATE TABLE trb3 PARTITION BY KEY(id) PARTITIONS 2;`.

`ALTER TABLE ... ENGINE = ...` changes only the storage engine used by the table, and leaves the table's partitioning scheme intact. Use `ALTER TABLE ... REMOVE PARTITIONING` to remove a table's partitioning. See Section 13.1.7, “ALTER TABLE Statement”.

Important

Only a single `PARTITION BY`, `ADD PARTITION`, `DROP PARTITION`, `REORGANIZE PARTITION`, or `COALESCE PARTITION` clause can be used in a given `ALTER TABLE` statement. If you (for example) wish to drop a partition and

> reorganize a table's remaining partitions, you must do so in two separate `ALTER TABLE` statements (one using `DROP PARTITION` and then a second one using `REORGANIZE PARTITION`).

In MySQL 5.6, it is possible to delete all rows from one or more selected partitions using `ALTER TABLE ... TRUNCATE PARTITION`.

19.3.1 Management of RANGE and LIST Partitions

Adding and dropping of range and list partitions are handled in a similar fashion, so we discuss the management of both sorts of partitioning in this section. For information about working with tables that are partitioned by hash or key, see Section 19.3.2, “Management of HASH and KEY Partitions”.

Dropping a partition from a table that is partitioned by either `RANGE` or by `LIST` can be accomplished using the `ALTER TABLE` statement with the `DROP PARTITION` option. Suppose that you have created a table that is partitioned by range and then populated with 10 records using the following `CREATE TABLE` and `INSERT` statements:

```
mysql> CREATE TABLE tr (id INT, name VARCHAR(50), purchased DATE)
    ->     PARTITION BY RANGE( YEAR(purchased) ) (
    ->         PARTITION p0 VALUES LESS THAN (1990),
    ->         PARTITION p1 VALUES LESS THAN (1995),
    ->         PARTITION p2 VALUES LESS THAN (2000),
    ->         PARTITION p3 VALUES LESS THAN (2005),
    ->         PARTITION p4 VALUES LESS THAN (2010),
    ->         PARTITION p5 VALUES LESS THAN (2015)
    ->     );
Query OK, 0 rows affected (0.28 sec)

mysql> INSERT INTO tr VALUES
    ->     (1, 'desk organiser', '2003-10-15'),
    ->     (2, 'alarm clock', '1997-11-05'),
    ->     (3, 'chair', '2009-03-10'),
    ->     (4, 'bookcase', '1989-01-10'),
    ->     (5, 'exercise bike', '2014-05-09'),
    ->     (6, 'sofa', '1987-06-05'),
    ->     (7, 'espresso maker', '2011-11-22'),
    ->     (8, 'aquarium', '1992-08-04'),
    ->     (9, 'study desk', '2006-09-16'),
    ->     (10, 'lava lamp', '1998-12-25');
Query OK, 10 rows affected (0.05 sec)
Records: 10  Duplicates: 0  Warnings: 0
```

You can see which items should have been inserted into partition `p2` as shown here:

```
mysql> SELECT * FROM tr
    ->     WHERE purchased BETWEEN '1995-01-01' AND '1999-12-31';
+------+-------------+------------+
| id   | name        | purchased  |
+------+-------------+------------+
|    2 | alarm clock | 1997-11-05 |
|   10 | lava lamp   | 1998-12-25 |
+------+-------------+------------+
2 rows in set (0.00 sec)
```

You can also get this information using partition selection, as shown here:

```
mysql> SELECT * FROM tr PARTITION (p2);
+------+-------------+------------+
| id   | name        | purchased  |
+------+-------------+------------+
|    2 | alarm clock | 1997-11-05 |
|   10 | lava lamp   | 1998-12-25 |
+------+-------------+------------+
2 rows in set (0.00 sec)
```

See Section 19.5, “Partition Selection”, for more information.

To drop the partition named `p2`, execute the following command:

```
mysql> ALTER TABLE tr DROP PARTITION p2;
Query OK, 0 rows affected (0.03 sec)
```

Note

The `NDBCLUSTER` storage engine does not support `ALTER TABLE ... DROP PARTITION`. It does, however, support the other partitioning-related extensions to `ALTER TABLE` that are described in this chapter.

It is very important to remember that, *when you drop a partition, you also delete all the data that was stored in that partition*. You can see that this is the case by re-running the previous `SELECT` query:

```
mysql> SELECT * FROM tr WHERE purchased
    -> BETWEEN '1995-01-01' AND '1999-12-31';
Empty set (0.00 sec)
```

Because of this, you must have the `DROP` privilege for a table before you can execute `ALTER TABLE ... DROP PARTITION` on that table.

If you wish to drop all data from all partitions while preserving the table definition and its partitioning scheme, use the `TRUNCATE TABLE` statement. (See Section 13.1.33, “TRUNCATE TABLE Statement”.)

If you intend to change the partitioning of a table *without* losing data, use `ALTER TABLE ... REORGANIZE PARTITION` instead. See below or in Section 13.1.7, “ALTER TABLE Statement”, for information about `REORGANIZE PARTITION`.

If you now execute a `SHOW CREATE TABLE` statement, you can see how the partitioning makeup of the table has been changed:

```
mysql> SHOW CREATE TABLE tr\G
*************************** 1. row ***************************
       Table: tr
Create Table: CREATE TABLE `tr` (
  `id` int(11) DEFAULT NULL,
  `name` varchar(50) DEFAULT NULL,
  `purchased` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
/*!50100 PARTITION BY RANGE ( YEAR(purchased))
(PARTITION p0 VALUES LESS THAN (1990) ENGINE = InnoDB,
 PARTITION p1 VALUES LESS THAN (1995) ENGINE = InnoDB,
 PARTITION p3 VALUES LESS THAN (2005) ENGINE = InnoDB,
 PARTITION p4 VALUES LESS THAN (2010) ENGINE = InnoDB,
 PARTITION p5 VALUES LESS THAN (2015) ENGINE = InnoDB) */
1 row in set (0.00 sec)
```

When you insert new rows into the changed table with `purchased` column values between `'1995-01-01'` and `'2004-12-31'` inclusive, those rows are stored in partition `p3`. You can verify this as follows:

```
mysql> INSERT INTO tr VALUES (11, 'pencil holder', '1995-07-12');
Query OK, 1 row affected (0.00 sec)

mysql> SELECT * FROM tr WHERE purchased
    -> BETWEEN '1995-01-01' AND '2004-12-31';
+------+---------------+------------+
| id   | name          | purchased  |
+------+---------------+------------+
|    1 | desk organiser | 2003-10-15 |
|   11 | pencil holder  | 1995-07-12 |
+------+---------------+------------+
2 rows in set (0.00 sec)

mysql> ALTER TABLE tr DROP PARTITION p3;
Query OK, 0 rows affected (0.03 sec)
```

```
mysql> SELECT * FROM tr WHERE purchased
    -> BETWEEN '1995-01-01' AND '2004-12-31';
Empty set (0.00 sec)
```

The number of rows dropped from the table as a result of `ALTER TABLE ... DROP PARTITION` is not reported by the server as it would be by the equivalent `DELETE` query.

Dropping `LIST` partitions uses exactly the same `ALTER TABLE ... DROP PARTITION` syntax as used for dropping `RANGE` partitions. However, there is one important difference in the effect this has on your use of the table afterward: You can no longer insert into the table any rows having any of the values that were included in the value list defining the deleted partition. (See Section 19.2.2, “LIST Partitioning”, for an example.)

To add a new range or list partition to a previously partitioned table, use the `ALTER TABLE ... ADD PARTITION` statement. For tables which are partitioned by `RANGE`, this can be used to add a new range to the end of the list of existing partitions. Suppose that you have a partitioned table containing membership data for your organization, which is defined as follows:

```
CREATE TABLE members (
    id INT,
    fname VARCHAR(25),
    lname VARCHAR(25),
    dob DATE
)
PARTITION BY RANGE( YEAR(dob) ) (
    PARTITION p0 VALUES LESS THAN (1980),
    PARTITION p1 VALUES LESS THAN (1990),
    PARTITION p2 VALUES LESS THAN (2000)
);
```

Suppose further that the minimum age for members is 16. As the calendar approaches the end of 2015, you realize that you are soon likely to be admitting members who were born in 2000 (and later). You can modify the `members` table to accommodate new members born in the years 2000 to 2010 as shown here:

```
ALTER TABLE members ADD PARTITION (PARTITION p3 VALUES LESS THAN (2010));
```

With tables that are partitioned by range, you can use `ADD PARTITION` to add new partitions to the high end of the partitions list only. Trying to add a new partition in this manner between or before existing partitions results in an error as shown here:

```
mysql> ALTER TABLE members
     >     ADD PARTITION (
     >     PARTITION n VALUES LESS THAN (1970));
ERROR 1463 (HY000): VALUES LESS THAN value must be strictly »
   increasing for each partition
```

You can work around this problem by reorganizing the first partition into two new ones that split the range between them, like this:

```
ALTER TABLE members
    REORGANIZE PARTITION p0 INTO (
        PARTITION n0 VALUES LESS THAN (1970),
        PARTITION n1 VALUES LESS THAN (1980)
);
```

Using `SHOW CREATE TABLE` you can see that the `ALTER TABLE` statement has had the desired effect:

```
mysql> SHOW CREATE TABLE members\G
*************************** 1. row ***************************
       Table: members
Create Table: CREATE TABLE `members` (
  `id` int(11) DEFAULT NULL,
  `fname` varchar(25) DEFAULT NULL,
```

```
  `lname` varchar(25) DEFAULT NULL,
  `dob` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
/*!50100 PARTITION BY RANGE ( YEAR(dob))
(PARTITION n0 VALUES LESS THAN (1970) ENGINE = InnoDB,
 PARTITION n1 VALUES LESS THAN (1980) ENGINE = InnoDB,
 PARTITION p1 VALUES LESS THAN (1990) ENGINE = InnoDB,
 PARTITION p2 VALUES LESS THAN (2000) ENGINE = InnoDB,
 PARTITION p3 VALUES LESS THAN (2010) ENGINE = InnoDB) */
1 row in set (0.00 sec)
```

See also Section 13.1.7.1, “ALTER TABLE Partition Operations”.

You can also use `ALTER TABLE ... ADD PARTITION` to add new partitions to a table that is partitioned by `LIST`. Suppose a table `tt` is defined using the following `CREATE TABLE` statement:

```
CREATE TABLE tt (
    id INT,
    data INT
)
PARTITION BY LIST(data) (
    PARTITION p0 VALUES IN (5, 10, 15),
    PARTITION p1 VALUES IN (6, 12, 18)
);
```

You can add a new partition in which to store rows having the `data` column values `7`, `14`, and `21` as shown:

```
ALTER TABLE tt ADD PARTITION (PARTITION p2 VALUES IN (7, 14, 21));
```

Keep in mind that you *cannot* add a new `LIST` partition encompassing any values that are already included in the value list of an existing partition. If you attempt to do so, an error results:

```
mysql> ALTER TABLE tt ADD PARTITION
     >     (PARTITION np VALUES IN (4, 8, 12));
ERROR 1465 (HY000): Multiple definition of same constant »
                    in list partitioning
```

Because any rows with the `data` column value `12` have already been assigned to partition `p1`, you cannot create a new partition on table `tt` that includes `12` in its value list. To accomplish this, you could drop `p1`, and add `np` and then a new `p1` with a modified definition. However, as discussed earlier, this would result in the loss of all data stored in `p1`—and it is often the case that this is not what you really want to do. Another solution might appear to be to make a copy of the table with the new partitioning and to copy the data into it using `CREATE TABLE ... SELECT ...`, then drop the old table and rename the new one, but this could be very time-consuming when dealing with a large amounts of data. This also might not be feasible in situations where high availability is a requirement.

You can add multiple partitions in a single `ALTER TABLE ... ADD PARTITION` statement as shown here:

```
CREATE TABLE employees (
  id INT NOT NULL,
  fname VARCHAR(50) NOT NULL,
  lname VARCHAR(50) NOT NULL,
  hired DATE NOT NULL
)
PARTITION BY RANGE( YEAR(hired) ) (
  PARTITION p1 VALUES LESS THAN (1991),
  PARTITION p2 VALUES LESS THAN (1996),
  PARTITION p3 VALUES LESS THAN (2001),
  PARTITION p4 VALUES LESS THAN (2005)
);

ALTER TABLE employees ADD PARTITION (
    PARTITION p5 VALUES LESS THAN (2010),
    PARTITION p6 VALUES LESS THAN MAXVALUE
);
```

Fortunately, MySQL's partitioning implementation provides ways to redefine partitions without losing data. Let us look first at a couple of simple examples involving `RANGE` partitioning. Recall the `members` table which is now defined as shown here:

```
mysql> SHOW CREATE TABLE members\G
*************************** 1. row ***************************
       Table: members
Create Table: CREATE TABLE `members` (
  `id` int(11) DEFAULT NULL,
  `fname` varchar(25) DEFAULT NULL,
  `lname` varchar(25) DEFAULT NULL,
  `dob` date DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
/*!50100 PARTITION BY RANGE ( YEAR(dob))
(PARTITION n0 VALUES LESS THAN (1970) ENGINE = InnoDB,
 PARTITION n1 VALUES LESS THAN (1980) ENGINE = InnoDB,
 PARTITION p1 VALUES LESS THAN (1990) ENGINE = InnoDB,
 PARTITION p2 VALUES LESS THAN (2000) ENGINE = InnoDB,
 PARTITION p3 VALUES LESS THAN (2010) ENGINE = InnoDB) */
1 row in set (0.00 sec)
```

Suppose that you would like to move all rows representing members born before 1960 into a separate partition. As we have already seen, this cannot be done using `ALTER TABLE ... ADD PARTITION`. However, you can use another partition-related extension to `ALTER TABLE` to accomplish this:

```
ALTER TABLE members REORGANIZE PARTITION n0 INTO (
    PARTITION s0 VALUES LESS THAN (1960),
    PARTITION s1 VALUES LESS THAN (1970)
);
```

In effect, this command splits partition `p0` into two new partitions `s0` and `s1`. It also moves the data that was stored in `p0` into the new partitions according to the rules embodied in the two `PARTITION ... VALUES ...` clauses, so that `s0` contains only those records for which `YEAR(dob)` is less than 1960 and `s1` contains those rows in which `YEAR(dob)` is greater than or equal to 1960 but less than 1970.

A `REORGANIZE PARTITION` clause may also be used for merging adjacent partitions. You can reverse the effect of the previous statement on the `members` table as shown here:

```
ALTER TABLE members REORGANIZE PARTITION s0,s1 INTO (
    PARTITION p0 VALUES LESS THAN (1970)
);
```

No data is lost in splitting or merging partitions using `REORGANIZE PARTITION`. In executing the above statement, MySQL moves all of the records that were stored in partitions `s0` and `s1` into partition `p0`.

The general syntax for `REORGANIZE PARTITION` is shown here:

```
ALTER TABLE tbl_name
    REORGANIZE PARTITION partition_list
    INTO (partition_definitions);
```

Here, *`tbl_name`* is the name of the partitioned table, and *`partition_list`* is a comma-separated list of names of one or more existing partitions to be changed. *`partition_definitions`* is a comma-separated list of new partition definitions, which follow the same rules as for the *`partition_definitions`* list used in `CREATE TABLE`. You are not limited to merging several partitions into one, or to splitting one partition into many, when using `REORGANIZE PARTITION`. For example, you can reorganize all four partitions of the `members` table into two, like this:

```
ALTER TABLE members REORGANIZE PARTITION p0,p1,p2,p3 INTO (
    PARTITION m0 VALUES LESS THAN (1980),
    PARTITION m1 VALUES LESS THAN (2000)
);
```

You can also use `REORGANIZE PARTITION` with tables that are partitioned by `LIST`. Let us return to the problem of adding a new partition to the list-partitioned `tt` table and failing because the new partition had a value that was already present in the value-list of one of the existing partitions. We can

handle this by adding a partition that contains only nonconflicting values, and then reorganizing the new partition and the existing one so that the value which was stored in the existing one is now moved to the new one:

```
ALTER TABLE tt ADD PARTITION (PARTITION np VALUES IN (4, 8));
ALTER TABLE tt REORGANIZE PARTITION p1,np INTO (
    PARTITION p1 VALUES IN (6, 18),
    PARTITION np VALUES in (4, 8, 12)
);
```

Here are some key points to keep in mind when using `ALTER TABLE ... REORGANIZE PARTITION` to repartition tables that are partitioned by `RANGE` or `LIST`:

- The `PARTITION` options used to determine the new partitioning scheme are subject to the same rules as those used with a `CREATE TABLE` statement.

  A new `RANGE` partitioning scheme cannot have any overlapping ranges; a new `LIST` partitioning scheme cannot have any overlapping sets of values.

- The combination of partitions in the *`partition_definitions`* list should account for the same range or set of values overall as the combined partitions named in the *`partition_list`*.

  For example, partitions `p1` and `p2` together cover the years 1980 through 1999 in the `members` table used as an example in this section. Any reorganization of these two partitions should cover the same range of years overall.

- For tables partitioned by `RANGE`, you can reorganize only adjacent partitions; you cannot skip range partitions.

  For instance, you could not reorganize the example `members` table using a statement beginning with `ALTER TABLE members REORGANIZE PARTITION p0,p2 INTO ...` because `p0` covers the years prior to 1970 and `p2` the years from 1990 through 1999 inclusive, so these are not adjacent partitions. (You cannot skip partition `p1` in this case.)

- You cannot use `REORGANIZE PARTITION` to change the type of partitioning used by the table (for example, you cannot change `RANGE` partitions to `HASH` partitions or the reverse). You also cannot use this statement to change the partitioning expression or column. To accomplish either of these tasks without dropping and re-creating the table, you can use `ALTER TABLE ... PARTITION BY ...`, as shown here:

  ```
  ALTER TABLE members
      PARTITION BY HASH( YEAR(dob) )
      PARTITIONS 8;
  ```

19.3.2 Management of HASH and KEY Partitions

Tables which are partitioned by hash or by key are very similar to one another with regard to making changes in a partitioning setup, and both differ in a number of ways from tables which have been partitioned by range or list. For that reason, this section addresses the modification of tables partitioned by hash or by key only. For a discussion of adding and dropping of partitions of tables that are partitioned by range or list, see Section 19.3.1, “Management of RANGE and LIST Partitions”.

You cannot drop partitions from tables that are partitioned by `HASH` or `KEY` in the same way that you can from tables that are partitioned by `RANGE` or `LIST`. However, you can merge `HASH` or `KEY` partitions using the `ALTER TABLE ... COALESCE PARTITION` statement. Suppose that you have a table containing data about clients, which is divided into twelve partitions. The `clients` table is defined as shown here:

```
CREATE TABLE clients (
    id INT,
    fname VARCHAR(30),
    lname VARCHAR(30),
    signed DATE
```

```
)
PARTITION BY HASH( MONTH(signed) )
PARTITIONS 12;
```

To reduce the number of partitions from twelve to eight, execute the following `ALTER TABLE` command:

```
mysql> ALTER TABLE clients COALESCE PARTITION 4;
Query OK, 0 rows affected (0.02 sec)
```

`COALESCE` works equally well with tables that are partitioned by `HASH`, `KEY`, `LINEAR HASH`, or `LINEAR KEY`. Here is an example similar to the previous one, differing only in that the table is partitioned by `LINEAR KEY`:

```
mysql> CREATE TABLE clients_lk (
    ->     id INT,
    ->     fname VARCHAR(30),
    ->     lname VARCHAR(30),
    ->     signed DATE
    -> )
    -> PARTITION BY LINEAR KEY(signed)
    -> PARTITIONS 12;
Query OK, 0 rows affected (0.03 sec)

mysql> ALTER TABLE clients_lk COALESCE PARTITION 4;
Query OK, 0 rows affected (0.06 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

The number following `COALESCE PARTITION` is the number of partitions to merge into the remainder—in other words, it is the number of partitions to remove from the table.

If you attempt to remove more partitions than the table has, the result is an error like the one shown:

```
mysql> ALTER TABLE clients COALESCE PARTITION 18;
ERROR 1478 (HY000): Cannot remove all partitions, use DROP TABLE instead
```

To increase the number of partitions for the `clients` table from 12 to 18. use `ALTER TABLE ... ADD PARTITION` as shown here:

```
ALTER TABLE clients ADD PARTITION PARTITIONS 6;
```

19.3.3 Exchanging Partitions and Subpartitions with Tables

In MySQL 5.6, it is possible to exchange a table partition or subpartition with a table using `ALTER TABLE pt EXCHANGE PARTITION p WITH TABLE nt`, where *`pt`* is the partitioned table and *`p`* is the partition or subpartition of *`pt`* to be exchanged with unpartitioned table *`nt`*, provided that the following statements are true:

1. Table *`nt`* is not itself partitioned.
2. Table *`nt`* is not a temporary table.
3. The structures of tables *`pt`* and *`nt`* are otherwise identical.
4. Table `nt` contains no foreign key references, and no other table has any foreign keys that refer to `nt`.
5. There are no rows in *`nt`* that lie outside the boundaries of the partition definition for *`p`*.
6. For `InnoDB` tables, both tables use the same row format. To determine the row format of an `InnoDB` table, query `INFORMATION_SCHEMA.INNODB_SYS_TABLES`.

In addition to the `ALTER`, `INSERT`, and `CREATE` privileges usually required for `ALTER TABLE` statements, you must have the `DROP` privilege to perform `ALTER TABLE ... EXCHANGE PARTITION`.

You should also be aware of the following effects of `ALTER TABLE ... EXCHANGE PARTITION`:

- Executing `ALTER TABLE ... EXCHANGE PARTITION` does not invoke any triggers on either the partitioned table or the table to be exchanged.
- Any `AUTO_INCREMENT` columns in the exchanged table are reset.
- The `IGNORE` keyword has no effect when used with `ALTER TABLE ... EXCHANGE PARTITION`.

The complete syntax of the `ALTER TABLE ... EXCHANGE PARTITION` statement is shown here, where *`pt`* is the partitioned table, *`p`* is the partition or subpartition to be exchanged, and *`nt`* is the nonpartitioned table to be exchanged with *`p`*:

```
ALTER TABLE pt
    EXCHANGE PARTITION p
    WITH TABLE nt;
```

One and only one partition or subpartition may be exchanged with one and only one nonpartitioned table in a single `ALTER TABLE EXCHANGE PARTITION` statement. To exchange multiple partitions or subpartitions, use multiple `ALTER TABLE EXCHANGE PARTITION` statements. `EXCHANGE PARTITION` may not be combined with other `ALTER TABLE` options. The partitioning and (if applicable) subpartitioning used by the partitioned table may be of any type or types supported in MySQL 5.6.

Exchanging a Partition with a Nonpartitioned Table

Suppose that a partitioned table `e` has been created and populated using the following SQL statements:

```
CREATE TABLE e (
    id INT NOT NULL,
    fname VARCHAR(30),
    lname VARCHAR(30)
)
    PARTITION BY RANGE (id) (
        PARTITION p0 VALUES LESS THAN (50),
        PARTITION p1 VALUES LESS THAN (100),
        PARTITION p2 VALUES LESS THAN (150),
        PARTITION p3 VALUES LESS THAN (MAXVALUE)
);

INSERT INTO e VALUES
    (1669, "Jim", "Smith"),
    (337, "Mary", "Jones"),
    (16, "Frank", "White"),
    (2005, "Linda", "Black");
```

Now we create a nonpartitioned copy of `e` named `e2`. This can be done using the `mysql` client as shown here:

```
mysql> CREATE TABLE e2 LIKE e;
Query OK, 0 rows affected (1.34 sec)

mysql> ALTER TABLE e2 REMOVE PARTITIONING;
Query OK, 0 rows affected (0.90 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

You can see which partitions in table `e` contain rows by querying the `INFORMATION_SCHEMA.PARTITIONS` table, like this:

```
mysql> SELECT PARTITION_NAME, TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'e';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |          1 |
| p1             |          0 |
| p2             |          0 |
| p3             |          3 |
```

```
+---------------+------------+
4 rows in set (0.00 sec)
```

Note

For partitioned `InnoDB` tables, the row count given in the `TABLE_ROWS` column of the `INFORMATION_SCHEMA.PARTITIONS` table is only an estimated value used in SQL optimization, and is not always exact.

To exchange partition `p0` in table `e` with table `e2`, you can use the `ALTER TABLE` statement shown here:

```
mysql> ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2;
Query OK, 0 rows affected (0.28 sec)
```

More precisely, the statement just issued causes any rows found in the partition to be swapped with those found in the table. You can observe how this has happened by querying the `INFORMATION_SCHEMA.PARTITIONS` table, as before. The table row that was previously found in partition `p0` is no longer present:

```
mysql> SELECT PARTITION_NAME, TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'e';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |          0 |
| p1             |          0 |
| p2             |          0 |
| p3             |          3 |
+----------------+------------+
4 rows in set (0.00 sec)
```

If you query table `e2`, you can see that the “missing” row can now be found there:

```
mysql> SELECT * FROM e2;
+----+-------+-------+
| id | fname | lname |
+----+-------+-------+
| 16 | Frank | White |
+----+-------+-------+
1 row in set (0.00 sec)
```

The table to be exchanged with the partition does not necessarily have to be empty. To demonstrate this, we first insert a new row into table `e`, making sure that this row is stored in partition `p0` by choosing an `id` column value that is less than 50, and verifying this afterward by querying the `PARTITIONS` table:

```
mysql> INSERT INTO e VALUES (41, "Michael", "Green");
Query OK, 1 row affected (0.05 sec)

mysql> SELECT PARTITION_NAME, TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'e';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |          1 |
| p1             |          0 |
| p2             |          0 |
| p3             |          3 |
+----------------+------------+
4 rows in set (0.00 sec)
```

Now we once again exchange partition `p0` with table `e2` using the same `ALTER TABLE` statement as previously:

```
mysql> ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2;
```

```
Query OK, 0 rows affected (0.28 sec)
```

The output of the following queries shows that the table row that was stored in partition `p0` and the table row that was stored in table `e2`, prior to issuing the `ALTER TABLE` statement, have now switched places:

```
mysql> SELECT * FROM e;
+------+-------+-------+
| id   | fname | lname |
+------+-------+-------+
|   16 | Frank | White |
| 1669 | Jim   | Smith |
|  337 | Mary  | Jones |
| 2005 | Linda | Black |
+------+-------+-------+
4 rows in set (0.00 sec)

mysql> SELECT PARTITION_NAME, TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'e';
+----------------+------------+
| PARTITION_NAME | TABLE_ROWS |
+----------------+------------+
| p0             |          1 |
| p1             |          0 |
| p2             |          0 |
| p3             |          3 |
+----------------+------------+
4 rows in set (0.00 sec)

mysql> SELECT * FROM e2;
+----+---------+-------+
| id | fname   | lname |
+----+---------+-------+
| 41 | Michael | Green |
+----+---------+-------+
1 row in set (0.00 sec)
```

Non-Matching Rows

You should keep in mind that any rows found in the nonpartitioned table prior to issuing the `ALTER TABLE ... EXCHANGE PARTITION` statement must meet the conditions required for them to be stored in the target partition; otherwise, the statement fails. To see how this occurs, first insert a row into `e2` that is outside the boundaries of the partition definition for partition `p0` of table `e`. For example, insert a row with an `id` column value that is too large; then, try to exchange the table with the partition again:

```
mysql> INSERT INTO e2 VALUES (51, "Ellen", "McDonald");
Query OK, 1 row affected (0.08 sec)

mysql> ALTER TABLE e EXCHANGE PARTITION p0 WITH TABLE e2;
ERROR 1707 (HY000): Found row that does not match the partition
```

The `IGNORE` keyword is accepted, but has no effect when used with `EXCHANGE PARTITION`, as shown here:

```
mysql> ALTER IGNORE TABLE e EXCHANGE PARTITION p0 WITH TABLE e2;
ERROR 1707 (HY000): Found row that does not match the partition
```

Exchanging a Subpartition with a Nonpartitioned Table

You can also exchange a subpartition of a subpartitioned table (see Section 19.2.6, “Subpartitioning”) with a nonpartitioned table using an `ALTER TABLE ... EXCHANGE PARTITION` statement. In the following example, we first create a table `es` that is partitioned by `RANGE` and subpartitioned by `KEY`, populate this table as we did table `e`, and then create an empty, nonpartitioned copy `es2` of the table, as shown here:

```
mysql> CREATE TABLE es (
```

```
    ->     id INT NOT NULL,
    ->     fname VARCHAR(30),
    ->     lname VARCHAR(30)
    -> )
    ->     PARTITION BY RANGE (id)
    ->     SUBPARTITION BY KEY (lname)
    ->     SUBPARTITIONS 2 (
    ->         PARTITION p0 VALUES LESS THAN (50),
    ->         PARTITION p1 VALUES LESS THAN (100),
    ->         PARTITION p2 VALUES LESS THAN (150),
    ->         PARTITION p3 VALUES LESS THAN (MAXVALUE)
    ->     );
Query OK, 0 rows affected (2.76 sec)

mysql> INSERT INTO es VALUES
    ->     (1669, "Jim", "Smith"),
    ->     (337, "Mary", "Jones"),
    ->     (16, "Frank", "White"),
    ->     (2005, "Linda", "Black");
Query OK, 4 rows affected (0.04 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> CREATE TABLE es2 LIKE es;
Query OK, 0 rows affected (1.27 sec)

mysql> ALTER TABLE es2 REMOVE PARTITIONING;
Query OK, 0 rows affected (0.70 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

Although we did not explicitly name any of the subpartitions when creating table `es`, we can obtain generated names for these by including the `SUBPARTITION_NAME` of the `PARTITIONS` table from `INFORMATION_SCHEMA` when selecting from that table, as shown here:

```
mysql> SELECT PARTITION_NAME, SUBPARTITION_NAME, TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'es';
+----------------+-------------------+------------+
| PARTITION_NAME | SUBPARTITION_NAME | TABLE_ROWS |
+----------------+-------------------+------------+
| p0             | p0sp0             |          1 |
| p0             | p0sp1             |          0 |
| p1             | p1sp0             |          0 |
| p1             | p1sp1             |          0 |
| p2             | p2sp0             |          0 |
| p2             | p2sp1             |          0 |
| p3             | p3sp0             |          3 |
| p3             | p3sp1             |          0 |
+----------------+-------------------+------------+
8 rows in set (0.00 sec)
```

The following `ALTER TABLE` statement exchanges subpartition `p3sp0` table `es` with the nonpartitioned table `es2`:

```
mysql> ALTER TABLE es EXCHANGE PARTITION p3sp0 WITH TABLE es2;
Query OK, 0 rows affected (0.29 sec)
```

You can verify that the rows were exchanged by issuing the following queries:

```
mysql> SELECT PARTITION_NAME, SUBPARTITION_NAME, TABLE_ROWS
    ->     FROM INFORMATION_SCHEMA.PARTITIONS
    ->     WHERE TABLE_NAME = 'es';
+----------------+-------------------+------------+
| PARTITION_NAME | SUBPARTITION_NAME | TABLE_ROWS |
+----------------+-------------------+------------+
| p0             | p0sp0             |          1 |
| p0             | p0sp1             |          0 |
| p1             | p1sp0             |          0 |
| p1             | p1sp1             |          0 |
| p2             | p2sp0             |          0 |
| p2             | p2sp1             |          0 |
| p3             | p3sp0             |          0 |
```

```
| p3             | p3sp1             |          0 |
+----------------+-------------------+------------+
8 rows in set (0.00 sec)

mysql> SELECT * FROM es2;
+------+-------+-------+
| id   | fname | lname |
+------+-------+-------+
| 1669 | Jim   | Smith |
|  337 | Mary  | Jones |
| 2005 | Linda | Black |
+------+-------+-------+
3 rows in set (0.00 sec)
```

If a table is subpartitioned, you can exchange only a subpartition of the table—not an entire partition—with an unpartitioned table, as shown here:

```
mysql> ALTER TABLE es EXCHANGE PARTITION p3 WITH TABLE es2;
ERROR 1704 (HY000): Subpartitioned table, use subpartition instead of partition
```

The comparison of table structures used by MySQL is very strict. The number, order, names, and types of columns and indexes of the partitioned table and the nonpartitioned table must match exactly. In addition, both tables must use the same storage engine:

```
mysql> CREATE TABLE es3 LIKE e;
Query OK, 0 rows affected (1.31 sec)

mysql> ALTER TABLE es3 REMOVE PARTITIONING;
Query OK, 0 rows affected (0.53 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW CREATE TABLE es3\G
*************************** 1. row ***************************
       Table: es3
Create Table: CREATE TABLE `es3` (
  `id` int(11) NOT NULL,
  `fname` varchar(30) DEFAULT NULL,
  `lname` varchar(30) DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

mysql> ALTER TABLE es3 ENGINE = MyISAM;
Query OK, 0 rows affected (0.15 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE es EXCHANGE PARTITION p3sp0 WITH TABLE es3;
ERROR 1497 (HY000): The mix of handlers in the partitions is not allowed in this version of MySQL
```

19.3.4 Maintenance of Partitions

A number of table and partition maintenance tasks can be carried out using SQL statements intended for such purposes on partitioned tables in MySQL 5.6.

Table maintenance of partitioned tables can be accomplished using the statements `CHECK TABLE`, `OPTIMIZE TABLE`, `ANALYZE TABLE`, and `REPAIR TABLE`, which are supported for partitioned tables.

You can use a number of extensions to `ALTER TABLE` for performing operations of this type on one or more partitions directly, as described in the following list:

- **Rebuilding partitions.** Rebuilds the partition; this has the same effect as dropping all records stored in the partition, then reinserting them. This can be useful for purposes of defragmentation.

  Example:

  ```
  ALTER TABLE t1 REBUILD PARTITION p0, p1;
  ```

- **Optimizing partitions.** If you have deleted a large number of rows from a partition or if you have made many changes to a partitioned table with variable-length rows (that is, having `VARCHAR`, `BLOB`,

or TEXT columns), you can use ALTER TABLE ... OPTIMIZE PARTITION to reclaim any unused space and to defragment the partition data file.

Example:

```
ALTER TABLE t1 OPTIMIZE PARTITION p0, p1;
```

Using OPTIMIZE PARTITION on a given partition is equivalent to running CHECK PARTITION, ANALYZE PARTITION, and REPAIR PARTITION on that partition.

Some MySQL storage engines, including InnoDB, do not support per-partition optimization; in these cases, ALTER TABLE ... OPTIMIZE PARTITION rebuilds the entire table. In MySQL 5.6.9 and later, running this statement on such a table causes the entire table to rebuilt and analyzed, and an appropriate warning to be issued. (Bug #11751825, Bug #42822) Use ALTER TABLE ... REBUILD PARTITION and ALTER TABLE ... ANALYZE PARTITION instead, to avoid this issue.

- **Analyzing partitions.** This reads and stores the key distributions for partitions.

  Example:

  ```
  ALTER TABLE t1 ANALYZE PARTITION p3;
  ```

- **Repairing partitions.** This repairs corrupted partitions.

  Example:

  ```
  ALTER TABLE t1 REPAIR PARTITION p0,p1;
  ```

- **Checking partitions.** You can check partitions for errors in much the same way that you can use CHECK TABLE with nonpartitioned tables.

  Example:

  ```
  ALTER TABLE trb3 CHECK PARTITION p1;
  ```

  This command tells you if the data or indexes in partition p1 of table t1 are corrupted. If this is the case, use ALTER TABLE ... REPAIR PARTITION to repair the partition.

Each of the statements in the list just shown also supports the keyword ALL in place of the list of partition names. Using ALL causes the statement to act on all partitions in the table.

The use of mysqlcheck and myisamchk is not supported with partitioned tables.

In MySQL 5.6, you can also truncate partitions using ALTER TABLE ... TRUNCATE PARTITION. This statement can be used to delete all rows from one or more partitions in much the same way that TRUNCATE TABLE deletes all rows from a table.

ALTER TABLE ... TRUNCATE PARTITION ALL truncates all partitions in the table.

ANALYZE, CHECK, OPTIMIZE, REBUILD, REPAIR, and TRUNCATE operations are not supported for subpartitions.

19.3.5 Obtaining Information About Partitions

This section discusses obtaining information about existing partitions, which can be done in a number of ways. Methods of obtaining such information include the following:

- Using the SHOW CREATE TABLE statement to view the partitioning clauses used in creating a partitioned table.
- Using the SHOW TABLE STATUS statement to determine whether a table is partitioned.
- Querying the INFORMATION_SCHEMA.PARTITIONS table.

- Using the statement `EXPLAIN PARTITIONS SELECT` to see which partitions are used by a given `SELECT`.

As discussed elsewhere in this chapter, `SHOW CREATE TABLE` includes in its output the `PARTITION BY` clause used to create a partitioned table. For example:

```
mysql> SHOW CREATE TABLE trb3\G
*************************** 1. row ***************************
       Table: trb3
Create Table: CREATE TABLE `trb3` (
  `id` int(11) default NULL,
  `name` varchar(50) default NULL,
  `purchased` date default NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
PARTITION BY RANGE (YEAR(purchased)) (
  PARTITION p0 VALUES LESS THAN (1990) ENGINE = MyISAM,
  PARTITION p1 VALUES LESS THAN (1995) ENGINE = MyISAM,
  PARTITION p2 VALUES LESS THAN (2000) ENGINE = MyISAM,
  PARTITION p3 VALUES LESS THAN (2005) ENGINE = MyISAM
)
1 row in set (0.00 sec)
```

The output from `SHOW TABLE STATUS` for partitioned tables is the same as that for nonpartitioned tables, except that the `Create_options` column contains the string `partitioned`. The `Engine` column contains the name of the storage engine used by all partitions of the table. (See Section 13.7.5.37, “SHOW TABLE STATUS Statement”, for more information about this statement.)

You can also obtain information about partitions from `INFORMATION_SCHEMA`, which contains a `PARTITIONS` table. See Section 21.14, “The INFORMATION_SCHEMA PARTITIONS Table”.

It is possible to determine which partitions of a partitioned table are involved in a given `SELECT` query using `EXPLAIN PARTITIONS`. The `PARTITIONS` keyword adds a `partitions` column to the output of `EXPLAIN` listing the partitions from which records would be matched by the query.

Suppose that you have a table `trb1` created and populated as follows:

```
CREATE TABLE trb1 (id INT, name VARCHAR(50), purchased DATE)
    PARTITION BY RANGE(id)
    (
        PARTITION p0 VALUES LESS THAN (3),
        PARTITION p1 VALUES LESS THAN (7),
        PARTITION p2 VALUES LESS THAN (9),
        PARTITION p3 VALUES LESS THAN (11)
    );

INSERT INTO trb1 VALUES
    (1, 'desk organiser', '2003-10-15'),
    (2, 'CD player', '1993-11-05'),
    (3, 'TV set', '1996-03-10'),
    (4, 'bookcase', '1982-01-10'),
    (5, 'exercise bike', '2004-05-09'),
    (6, 'sofa', '1987-06-05'),
    (7, 'popcorn maker', '2001-11-22'),
    (8, 'aquarium', '1992-08-04'),
    (9, 'study desk', '1984-09-16'),
    (10, 'lava lamp', '1998-12-25');
```

You can see which partitions are used in a query such as `SELECT * FROM trb1;`, as shown here:

```
mysql> EXPLAIN PARTITIONS SELECT * FROM trb1\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: trb1
   partitions: p0,p1,p2,p3
         type: ALL
possible_keys: NULL
          key: NULL
```

```
      key_len: NULL
          ref: NULL
         rows: 10
        Extra: Using filesort
```

In this case, all four partitions are searched. However, when a limiting condition making use of the partitioning key is added to the query, you can see that only those partitions containing matching values are scanned, as shown here:

```
mysql> EXPLAIN PARTITIONS SELECT * FROM trb1 WHERE id < 5\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: trb1
   partitions: p0,p1
         type: ALL
possible_keys: NULL
          key: NULL
      key_len: NULL
          ref: NULL
         rows: 10
        Extra: Using where
```

`EXPLAIN PARTITIONS` provides information about keys used and possible keys, just as with the standard `EXPLAIN SELECT` statement:

```
mysql> ALTER TABLE trb1 ADD PRIMARY KEY (id);
Query OK, 10 rows affected (0.03 sec)
Records: 10  Duplicates: 0  Warnings: 0

mysql> EXPLAIN PARTITIONS SELECT * FROM trb1 WHERE id < 5\G
*************************** 1. row ***************************
           id: 1
  select_type: SIMPLE
        table: trb1
   partitions: p0,p1
         type: range
possible_keys: PRIMARY
          key: PRIMARY
      key_len: 4
          ref: NULL
         rows: 7
        Extra: Using where
```

You should take note of the following restrictions and limitations on `EXPLAIN PARTITIONS`:

- You cannot use the `PARTITIONS` and `EXTENDED` keywords together in the same `EXPLAIN ... SELECT` statement. Attempting to do so produces a syntax error.
- If `EXPLAIN PARTITIONS` is used to examine a query against a nonpartitioned table, no error is produced, but the value of the `partitions` column is always `NULL`.

The `rows` column of `EXPLAIN PARTITIONS` output displays the total number of rows in the table.

See also Section 13.8.2, “EXPLAIN Statement”.

19.4 Partition Pruning

This section discusses an optimization known as *partition pruning*. The core concept behind partition pruning is relatively simple, and can be described as “Do not scan partitions where there can be no matching values”. Suppose that you have a partitioned table `t1` defined by this statement:

```
CREATE TABLE t1 (
    fname VARCHAR(50) NOT NULL,
    lname VARCHAR(50) NOT NULL,
    region_code TINYINT UNSIGNED NOT NULL,
    dob DATE NOT NULL
```

```
)
PARTITION BY RANGE( region_code ) (
    PARTITION p0 VALUES LESS THAN (64),
    PARTITION p1 VALUES LESS THAN (128),
    PARTITION p2 VALUES LESS THAN (192),
    PARTITION p3 VALUES LESS THAN MAXVALUE
);
```

Consider the case where you wish to obtain results from a `SELECT` statement such as this one:

```
SELECT fname, lname, region_code, dob
    FROM t1
    WHERE region_code > 125 AND region_code < 130;
```

It is easy to see that none of the rows which ought to be returned are in either of the partitions `p0` or `p3`; that is, we need to search only in partitions `p1` and `p2` to find matching rows. By doing so, it is possible to expend much less time and effort in finding matching rows than would be required to scan all partitions in the table. This “cutting away” of unneeded partitions is known as *pruning*. When the optimizer can make use of partition pruning in performing this query, execution of the query can be an order of magnitude faster than the same query against a nonpartitioned table containing the same column definitions and data.

Note

When pruning is performed on a partitioned `MyISAM` table, all partitions are opened, whether or not they are examined, due to the design of the `MyISAM` storage engine. This means that you must have a sufficient number of file descriptors available to cover all partitions of the table. See MyISAM and partition file descriptor usage.

This limitation does not apply to partitioned tables using other MySQL storage engines such as `InnoDB`.

The optimizer can perform pruning whenever a `WHERE` condition can be reduced to either one of the following two cases:

- *`partition_column = constant`*
- *`partition_column`* `IN (`*`constant1, constant2, ..., constantN`*`)`

In the first case, the optimizer simply evaluates the partitioning expression for the value given, determines which partition contains that value, and scans only this partition. In many cases, the equal sign can be replaced with another arithmetic comparison, including `<`, `>`, `<=`, `>=`, and `<>`. Some queries using `BETWEEN` in the `WHERE` clause can also take advantage of partition pruning. See the examples later in this section.

In the second case, the optimizer evaluates the partitioning expression for each value in the list, creates a list of matching partitions, and then scans only the partitions in this partition list.

MySQL can apply partition pruning to `SELECT`, `DELETE`, and `UPDATE` statements. An `INSERT` statement also accesses only one partition per inserted row; this is true even for a table that is partitioned by `HASH` or `KEY` although this is not currently shown in the output of `EXPLAIN`.

Pruning can also be applied to short ranges, which the optimizer can convert into equivalent lists of values. For instance, in the previous example, the `WHERE` clause can be converted to `WHERE region_code IN (126, 127, 128, 129)`. Then the optimizer can determine that the first two values in the list are found in partition `p1`, the remaining two values in partition `p2`, and that the other partitions contain no relevant values and so do not need to be searched for matching rows.

In MySQL 5.6, the optimizer can also perform pruning for `WHERE` conditions that involve comparisons of the preceding types on multiple columns for tables that use `RANGE COLUMNS` or `LIST COLUMNS` partitioning.

This type of optimization can be applied whenever the partitioning expression consists of an equality or a range which can be reduced to a set of equalities, or when the partitioning expression represents an increasing or decreasing relationship. Pruning can also be applied for tables partitioned on a `DATE` or `DATETIME` column when the partitioning expression uses the `YEAR()` or `TO_DAYS()` function. In addition, in MySQL 5.6, pruning can be applied for such tables when the partitioning expression uses the `TO_SECONDS()` function.

Suppose that table `t2`, defined as shown here, is partitioned on a `DATE` column:

```
CREATE TABLE t2 (
    fname VARCHAR(50) NOT NULL,
    lname VARCHAR(50) NOT NULL,
    region_code TINYINT UNSIGNED NOT NULL,
    dob DATE NOT NULL
)
PARTITION BY RANGE( YEAR(dob) ) (
    PARTITION d0 VALUES LESS THAN (1970),
    PARTITION d1 VALUES LESS THAN (1975),
    PARTITION d2 VALUES LESS THAN (1980),
    PARTITION d3 VALUES LESS THAN (1985),
    PARTITION d4 VALUES LESS THAN (1990),
    PARTITION d5 VALUES LESS THAN (2000),
    PARTITION d6 VALUES LESS THAN (2005),
    PARTITION d7 VALUES LESS THAN MAXVALUE
);
```

The following statements using `t2` can make of use partition pruning:

```
SELECT * FROM t2 WHERE dob = '1982-06-23';

UPDATE t2 SET region_code = 8 WHERE dob BETWEEN '1991-02-15' AND '1997-04-25';

DELETE FROM t2 WHERE dob >= '1984-06-21' AND dob <= '1999-06-21'
```

In the case of the last statement, the optimizer can also act as follows:

1. *Find the partition containing the low end of the range*.

   `YEAR('1984-06-21')` yields the value `1984`, which is found in partition `d3`.

2. *Find the partition containing the high end of the range*.

   `YEAR('1999-06-21')` evaluates to `1999`, which is found in partition `d5`.

3. *Scan only these two partitions and any partitions that may lie between them*.

   In this case, this means that only partitions `d3`, `d4`, and `d5` are scanned. The remaining partitions may be safely ignored (and are ignored).

Important

Invalid `DATE` and `DATETIME` values referenced in the `WHERE` condition of a statement against a partitioned table are treated as `NULL`. This means that a query such as `SELECT * FROM partitioned_table WHERE date_column < '2008-12-00'` does not return any values (see Bug #40972).

So far, we have looked only at examples using `RANGE` partitioning, but pruning can be applied with other partitioning types as well.

Consider a table that is partitioned by `LIST`, where the partitioning expression is increasing or decreasing, such as the table `t3` shown here. (In this example, we assume for the sake of brevity that the `region_code` column is limited to values between 1 and 10 inclusive.)

```
CREATE TABLE t3 (
```

```
    fname VARCHAR(50) NOT NULL,
    lname VARCHAR(50) NOT NULL,
    region_code TINYINT UNSIGNED NOT NULL,
    dob DATE NOT NULL
)
PARTITION BY LIST(region_code) (
    PARTITION r0 VALUES IN (1, 3),
    PARTITION r1 VALUES IN (2, 5, 8),
    PARTITION r2 VALUES IN (4, 9),
    PARTITION r3 VALUES IN (6, 7, 10)
);
```

For a statement such as `SELECT * FROM t3 WHERE region_code BETWEEN 1 AND 3`, the optimizer determines in which partitions the values 1, 2, and 3 are found (`r0` and `r1`) and skips the remaining ones (`r2` and `r3`).

For tables that are partitioned by `HASH` or `[LINEAR] KEY`, partition pruning is also possible in cases in which the `WHERE` clause uses a simple `=` relation against a column used in the partitioning expression. Consider a table created like this:

```
CREATE TABLE t4 (
    fname VARCHAR(50) NOT NULL,
    lname VARCHAR(50) NOT NULL,
    region_code TINYINT UNSIGNED NOT NULL,
    dob DATE NOT NULL
)
PARTITION BY KEY(region_code)
PARTITIONS 8;
```

A statement that compares a column value with a constant can be pruned:

```
UPDATE t4 WHERE region_code = 7;
```

Pruning can also be employed for short ranges, because the optimizer can turn such conditions into `IN` relations. For example, using the same table `t4` as defined previously, queries such as these can be pruned:

```
SELECT * FROM t4 WHERE region_code > 2 AND region_code < 6;

SELECT * FROM t4 WHERE region_code BETWEEN 3 AND 5;
```

In both these cases, the `WHERE` clause is transformed by the optimizer into `WHERE region_code IN (3, 4, 5)`.

Important

This optimization is used only if the range size is smaller than the number of partitions. Consider this statement:

```
DELETE FROM t4 WHERE region_code BETWEEN 4 AND 12;
```

The range in the `WHERE` clause covers 9 values (4, 5, 6, 7, 8, 9, 10, 11, 12), but `t4` has only 8 partitions. This means that the `DELETE` cannot be pruned.

When a table is partitioned by `HASH` or `[LINEAR] KEY`, pruning can be used only on integer columns. For example, this statement cannot use pruning because `dob` is a `DATE` column:

```
SELECT * FROM t4 WHERE dob >= '2001-04-14' AND dob <= '2005-10-15';
```

However, if the table stores year values in an `INT` column, then a query having `WHERE year_col >= 2001 AND year_col <= 2005` can be pruned.

In MySQL 5.6.8 and later, partition pruning is disabled for all tables using a storage engine that provides automatic partitioning, such as the `NDB` storage engine used by NDB Cluster. (Bug #14672885) Beginning with MySQL 5.6.10, such tables can be pruned if they are explicitly partitioned. (Bug #14827952)

19.5 Partition Selection

MySQL 5.6 supports explicit selection of partitions and subpartitions that, when executing a statement, should be checked for rows matching a given `WHERE` condition. Partition selection is similar to partition pruning, in that only specific partitions are checked for matches, but differs in two key respects:

1. The partitions to be checked are specified by the issuer of the statement, unlike partition pruning, which is automatic.

2. Whereas partition pruning applies only to queries, explicit selection of partitions is supported for both queries and a number of DML statements.

SQL statements supporting explicit partition selection are listed here:

- `SELECT`
- `DELETE`
- `INSERT`
- `REPLACE`
- `UPDATE`
- `LOAD DATA`.
- `LOAD XML`.

The remainder of this section discusses explicit partition selection as it applies generally to the statements just listed, and provides some examples.

Explicit partition selection is implemented using a `PARTITION` option. For all supported statements, this option uses the syntax shown here:

```
      PARTITION (partition_names)

      partition_names:
          partition_name, ...
```

This option always follows the name of the table to which the partition or partitions belong. *`partition_names`* is a comma-separated list of partitions or subpartitions to be used. Each name in this list must be the name of an existing partition or subpartition of the specified table; if any of the partitions or subpartitions are not found, the statement fails with an error (`partition 'partition_name' doesn't exist`). Partitions and subpartitions named in *`partition_names`* may be listed in any order, and may overlap.

When the `PARTITION` option is used, only the partitions and subpartitions listed are checked for matching rows. This option can be used in a `SELECT` statement to determine which rows belong to a given partition. Consider a partitioned table named `employees`, created and populated using the statements shown here:

```
SET @@SQL_MODE = '';

CREATE TABLE employees  (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    fname VARCHAR(25) NOT NULL,
    lname VARCHAR(25) NOT NULL,
    store_id INT NOT NULL,
    department_id INT NOT NULL
)
    PARTITION BY RANGE(id)  (
        PARTITION p0 VALUES LESS THAN (5),
        PARTITION p1 VALUES LESS THAN (10),
        PARTITION p2 VALUES LESS THAN (15),
        PARTITION p3 VALUES LESS THAN MAXVALUE
```

```
);

INSERT INTO employees VALUES
    ('', 'Bob', 'Taylor', 3, 2), ('', 'Frank', 'Williams', 1, 2),
    ('', 'Ellen', 'Johnson', 3, 4), ('', 'Jim', 'Smith', 2, 4),
    ('', 'Mary', 'Jones', 1, 1), ('', 'Linda', 'Black', 2, 3),
    ('', 'Ed', 'Jones', 2, 1), ('', 'June', 'Wilson', 3, 1),
    ('', 'Andy', 'Smith', 1, 3), ('', 'Lou', 'Waters', 2, 4),
    ('', 'Jill', 'Stone', 1, 4), ('', 'Roger', 'White', 3, 2),
    ('', 'Howard', 'Andrews', 1, 2), ('', 'Fred', 'Goldberg', 3, 3),
    ('', 'Barbara', 'Brown', 2, 3), ('', 'Alice', 'Rogers', 2, 2),
    ('', 'Mark', 'Morgan', 3, 3), ('', 'Karen', 'Cole', 3, 2);
```

You can see which rows are stored in partition `p1` like this:

```
mysql> SELECT * FROM employees PARTITION (p1);
+----+-------+--------+----------+---------------+
| id | fname | lname  | store_id | department_id |
+----+-------+--------+----------+---------------+
|  5 | Mary  | Jones  |        1 |             1 |
|  6 | Linda | Black  |        2 |             3 |
|  7 | Ed    | Jones  |        2 |             1 |
|  8 | June  | Wilson |        3 |             1 |
|  9 | Andy  | Smith  |        1 |             3 |
+----+-------+--------+----------+---------------+
5 rows in set (0.00 sec)
```

The result is the same as obtained by the query `SELECT * FROM employees WHERE id BETWEEN 5 AND 9`.

To obtain rows from multiple partitions, supply their names as a comma-delimited list. For example, `SELECT * FROM employees PARTITION (p1, p2)` returns all rows from partitions `p1` and `p2` while excluding rows from the remaining partitions.

Any valid query against a partitioned table can be rewritten with a `PARTITION` option to restrict the result to one or more desired partitions. You can use `WHERE` conditions, `ORDER BY` and `LIMIT` options, and so on. You can also use aggregate functions with `HAVING` and `GROUP BY` options. Each of the following queries produces a valid result when run on the `employees` table as previously defined:

```
mysql> SELECT * FROM employees PARTITION (p0, p2)
    ->     WHERE lname LIKE 'S%';
+----+-------+-------+----------+---------------+
| id | fname | lname | store_id | department_id |
+----+-------+-------+----------+---------------+
|  4 | Jim   | Smith |        2 |             4 |
| 11 | Jill  | Stone |        1 |             4 |
+----+-------+-------+----------+---------------+
2 rows in set (0.00 sec)

mysql> SELECT id, CONCAT(fname, ' ', lname) AS name
    ->     FROM employees PARTITION (p0) ORDER BY lname;
+----+----------------+
| id | name           |
+----+----------------+
|  3 | Ellen Johnson  |
|  4 | Jim Smith      |
|  1 | Bob Taylor     |
|  2 | Frank Williams |
+----+----------------+
4 rows in set (0.06 sec)

mysql> SELECT store_id, COUNT(department_id) AS c
    ->     FROM employees PARTITION (p1,p2,p3)
    ->     GROUP BY store_id HAVING c > 4;
+---+----------+
| c | store_id |
+---+----------+
| 5 |        2 |
| 5 |        3 |
+---+----------+
```

```
2 rows in set (0.00 sec)
```

Statements using partition selection can be employed with tables using any of the partitioning types supported in MySQL 5.6. When a table is created using `[LINEAR] HASH` or `[LINEAR] KEY` partitioning and the names of the partitions are not specified, MySQL automatically names the partitions `p0`, `p1`, `p2`, ..., `pN-1`, where `N` is the number of partitions. For subpartitions not explicitly named, MySQL assigns automatically to the subpartitions in each partition `pX` the names `pXsp0`, `pXsp1`, `pXsp2`, ..., `pXspM-1`, where `M` is the number of subpartitions. When executing against this table a `SELECT` (or other SQL statement for which explicit partition selection is allowed), you can use these generated names in a `PARTITION` option, as shown here:

```
mysql> CREATE TABLE employees_sub  (
    ->     id INT NOT NULL AUTO_INCREMENT,
    ->     fname VARCHAR(25) NOT NULL,
    ->     lname VARCHAR(25) NOT NULL,
    ->     store_id INT NOT NULL,
    ->     department_id INT NOT NULL,
    ->     PRIMARY KEY pk (id, lname)
    -> )
    ->     PARTITION BY RANGE(id)
    ->     SUBPARTITION BY KEY (lname)
    ->     SUBPARTITIONS 2 (
    ->         PARTITION p0 VALUES LESS THAN (5),
    ->         PARTITION p1 VALUES LESS THAN (10),
    ->         PARTITION p2 VALUES LESS THAN (15),
    ->         PARTITION p3 VALUES LESS THAN MAXVALUE
    -> );
Query OK, 0 rows affected (1.14 sec)

mysql> INSERT INTO employees_sub   # re-use data in employees table
    ->     SELECT * FROM employees;
Query OK, 18 rows affected (0.09 sec)
Records: 18  Duplicates: 0  Warnings: 0

mysql> SELECT id, CONCAT(fname, ' ', lname) AS name
    ->     FROM employees_sub PARTITION (p2sp1);
+----+---------------+
| id | name          |
+----+---------------+
| 10 | Lou Waters    |
| 14 | Fred Goldberg |
+----+---------------+
2 rows in set (0.00 sec)
```

You may also use a `PARTITION` option in the `SELECT` portion of an `INSERT ... SELECT` statement, as shown here:

```
mysql> CREATE TABLE employees_copy LIKE employees;
Query OK, 0 rows affected (0.28 sec)

mysql> INSERT INTO employees_copy
    ->     SELECT * FROM employees PARTITION (p2);
Query OK, 5 rows affected (0.04 sec)
Records: 5  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM employees_copy;
+----+--------+----------+----------+---------------+
| id | fname  | lname    | store_id | department_id |
+----+--------+----------+----------+---------------+
| 10 | Lou    | Waters   |        2 |             4 |
| 11 | Jill   | Stone    |        1 |             4 |
| 12 | Roger  | White    |        3 |             2 |
| 13 | Howard | Andrews  |        1 |             2 |
| 14 | Fred   | Goldberg |        3 |             3 |
+----+--------+----------+----------+---------------+
5 rows in set (0.00 sec)
```

Partition selection can also be used with joins. Suppose we create and populate two tables using the statements shown here:

```
CREATE TABLE stores (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    city VARCHAR(30) NOT NULL
)
    PARTITION BY HASH(id)
    PARTITIONS 2;

INSERT INTO stores VALUES
    ('', 'Nambucca'), ('', 'Uranga'),
    ('', 'Bellingen'), ('', 'Grafton');

CREATE TABLE departments  (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(30) NOT NULL
)
    PARTITION BY KEY(id)
    PARTITIONS 2;

INSERT INTO departments VALUES
    ('', 'Sales'), ('', 'Customer Service'),
    ('', 'Delivery'), ('', 'Accounting');
```

You can explicitly select partitions (or subpartitions, or both) from any or all of the tables in a join. (The `PARTITION` option used to select partitions from a given table immediately follows the name of the table, before all other options, including any table alias.) For example, the following query gets the name, employee ID, department, and city of all employees who work in the Sales or Delivery department (partition `p1` of the `departments` table) at the stores in either of the cities of Nambucca and Bellingen (partition `p0` of the `stores` table):

```
mysql> SELECT
    ->     e.id AS 'Employee ID', CONCAT(e.fname, ' ', e.lname) AS Name,
    ->     s.city AS City, d.name AS department
    -> FROM employees AS e
    ->     JOIN stores PARTITION (p1) AS s ON e.store_id=s.id
    ->     JOIN departments PARTITION (p0) AS d ON e.department_id=d.id
    -> ORDER BY e.lname;
+-------------+---------------+-----------+------------+
| Employee ID | Name          | City      | department |
+-------------+---------------+-----------+------------+
|          14 | Fred Goldberg | Bellingen | Delivery   |
|           5 | Mary Jones    | Nambucca  | Sales      |
|          17 | Mark Morgan   | Bellingen | Delivery   |
|           9 | Andy Smith    | Nambucca  | Delivery   |
|           8 | June Wilson   | Bellingen | Sales      |
+-------------+---------------+-----------+------------+
5 rows in set (0.00 sec)
```

For general information about joins in MySQL, see Section 13.2.9.2, “JOIN Clause”.

When the `PARTITION` option is used with `DELETE` statements, only those partitions (and subpartitions, if any) listed with the option are checked for rows to be deleted. Any other partitions are ignored, as shown here:

```
mysql> SELECT * FROM employees WHERE fname LIKE 'j%';
+----+-------+--------+----------+---------------+
| id | fname | lname  | store_id | department_id |
+----+-------+--------+----------+---------------+
|  4 | Jim   | Smith  |        2 |             4 |
|  8 | June  | Wilson |        3 |             1 |
| 11 | Jill  | Stone  |        1 |             4 |
+----+-------+--------+----------+---------------+
3 rows in set (0.00 sec)

mysql> DELETE FROM employees PARTITION (p0, p1)
    ->     WHERE fname LIKE 'j%';
Query OK, 2 rows affected (0.09 sec)

mysql> SELECT * FROM employees WHERE fname LIKE 'j%';
+----+-------+-------+----------+---------------+
| id | fname | lname | store_id | department_id |
```

```
+----+-------+-------+----------+---------------+
| 11 | Jill  | Stone |        1 |             4 |
+----+-------+-------+----------+---------------+
1 row in set (0.00 sec)
```

Only the two rows in partitions `p0` and `p1` matching the `WHERE` condition were deleted. As you can see from the result when the `SELECT` is run a second time, there remains a row in the table matching the `WHERE` condition, but residing in a different partition (`p2`).

`UPDATE` statements using explicit partition selection behave in the same way; only rows in the partitions referenced by the `PARTITION` option are considered when determining the rows to be updated, as can be seen by executing the following statements:

```
mysql> UPDATE employees PARTITION (p0)
    ->     SET store_id = 2 WHERE fname = 'Jill';
Query OK, 0 rows affected (0.00 sec)
Rows matched: 0  Changed: 0  Warnings: 0

mysql> SELECT * FROM employees WHERE fname = 'Jill';
+----+-------+-------+----------+---------------+
| id | fname | lname | store_id | department_id |
+----+-------+-------+----------+---------------+
| 11 | Jill  | Stone |        1 |             4 |
+----+-------+-------+----------+---------------+
1 row in set (0.00 sec)

mysql> UPDATE employees PARTITION (p2)
    ->     SET store_id = 2 WHERE fname = 'Jill';
Query OK, 1 row affected (0.09 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM employees WHERE fname = 'Jill';
+----+-------+-------+----------+---------------+
| id | fname | lname | store_id | department_id |
+----+-------+-------+----------+---------------+
| 11 | Jill  | Stone |        2 |             4 |
+----+-------+-------+----------+---------------+
1 row in set (0.00 sec)
```

In the same way, when `PARTITION` is used with `DELETE`, only rows in the partition or partitions named in the partition list are checked for deletion.

For statements that insert rows, the behavior differs in that failure to find a suitable partition causes the statement to fail. This is true for both `INSERT` and `REPLACE` statements, as shown here:

```
mysql> INSERT INTO employees PARTITION (p2) VALUES (20, 'Jan', 'Jones', 1, 3);
ERROR 1729 (HY000): Found a row not matching the given partition set
mysql> INSERT INTO employees PARTITION (p3) VALUES (20, 'Jan', 'Jones', 1, 3);
Query OK, 1 row affected (0.07 sec)

mysql> REPLACE INTO employees PARTITION (p0) VALUES (20, 'Jan', 'Jones', 3, 2);
ERROR 1729 (HY000): Found a row not matching the given partition set

mysql> REPLACE INTO employees PARTITION (p3) VALUES (20, 'Jan', 'Jones', 3, 2);
Query OK, 2 rows affected (0.09 sec)
```

For statements that write multiple rows to a partitioned table that uses the `InnoDB` storage engine: If any row in the list following `VALUES` cannot be written to one of the partitions specified in the *`partition_names`* list, the entire statement fails and no rows are written. This is shown for `INSERT` statements in the following example, reusing the `employees` table created previously:

```
mysql> ALTER TABLE employees
    ->     REORGANIZE PARTITION p3 INTO (
    ->         PARTITION p3 VALUES LESS THAN (20),
    ->         PARTITION p4 VALUES LESS THAN (25),
    ->         PARTITION p5 VALUES LESS THAN MAXVALUE
    ->     );
Query OK, 6 rows affected (2.09 sec)
Records: 6  Duplicates: 0  Warnings: 0
```

```
mysql> SHOW CREATE TABLE employees\G
*************************** 1. row ***************************
       Table: employees
Create Table: CREATE TABLE `employees` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `fname` varchar(25) NOT NULL,
  `lname` varchar(25) NOT NULL,
  `store_id` int(11) NOT NULL,
  `department_id` int(11) NOT NULL,
  PRIMARY KEY (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=27 DEFAULT CHARSET=latin1
/*!50100 PARTITION BY RANGE (id)
(PARTITION p0 VALUES LESS THAN (5) ENGINE = InnoDB,
 PARTITION p1 VALUES LESS THAN (10) ENGINE = InnoDB,
 PARTITION p2 VALUES LESS THAN (15) ENGINE = InnoDB,
 PARTITION p3 VALUES LESS THAN (20) ENGINE = InnoDB,
 PARTITION p4 VALUES LESS THAN (25) ENGINE = InnoDB,
 PARTITION p5 VALUES LESS THAN MAXVALUE ENGINE = InnoDB) */
1 row in set (0.00 sec)

mysql> INSERT INTO employees PARTITION (p3, p4) VALUES
    ->     (24, 'Tim', 'Greene', 3, 1),  (26, 'Linda', 'Mills', 2, 1);
ERROR 1729 (HY000): Found a row not matching the given partition set

mysql> INSERT INTO employees PARTITION (p3, p4. p5) VALUES
    ->     (24, 'Tim', 'Greene', 3, 1),  (26, 'Linda', 'Mills', 2, 1);
Query OK, 2 rows affected (0.06 sec)
Records: 2  Duplicates: 0  Warnings: 0
```

The preceding is true for both `INSERT` statements and `REPLACE` statements that write multiple rows.

In MySQL 5.6.10 and later, partition selection is disabled for tables employing a storage engine that supplies automatic partitioning, such as `NDB`. (Bug #14827952)

19.6 Restrictions and Limitations on Partitioning

This section discusses current restrictions and limitations on MySQL partitioning support.

Prohibited constructs. The following constructs are not permitted in partitioning expressions:

- Stored procedures, stored functions, UDFs, or plugins.
- Declared variables or user variables.

For a list of SQL functions which are permitted in partitioning expressions, see Section 19.6.3, “Partitioning Limitations Relating to Functions”.

Arithmetic and logical operators. Use of the arithmetic operators `+`, `-`, and `*` is permitted in partitioning expressions. However, the result must be an integer value or `NULL` (except in the case of `[LINEAR] KEY` partitioning, as discussed elsewhere in this chapter; see Section 19.2, “Partitioning Types”, for more information).

The `DIV` operator is also supported, and the `/` operator is not permitted. (Bug #30188, Bug #33182)

The bit operators `|`, `&`, `^`, `<<`, `>>`, and `~` are not permitted in partitioning expressions.

HANDLER statements. In MySQL 5.6, the `HANDLER` statement is not supported with partitioned tables.

Server SQL mode. Tables employing user-defined partitioning do not preserve the SQL mode in effect at the time that they were created. As discussed in Section 5.1.10, “Server SQL Modes”, the results of many MySQL functions and operators may change according to the server SQL mode. Therefore, a change in the SQL mode at any time after the creation of partitioned tables may lead to major changes in the behavior of such tables, and could easily lead to corruption or loss of data. For

these reasons, *it is strongly recommended that you never change the server SQL mode after creating partitioned tables*.

Examples. The following examples illustrate some changes in behavior of partitioned tables due to a change in the server SQL mode:

1. **Error handling.** Suppose that you create a partitioned table whose partitioning expression is one such as `column DIV 0` or `column MOD 0`, as shown here:

```
mysql> CREATE TABLE tn (c1 INT)
    ->     PARTITION BY LIST(1 DIV c1) (
    ->       PARTITION p0 VALUES IN (NULL),
    ->       PARTITION p1 VALUES IN (1)
    -> );
Query OK, 0 rows affected (0.05 sec)
```

   The default behavior for MySQL is to return `NULL` for the result of a division by zero, without producing any errors:

```
mysql> SELECT @@sql_mode;
+------------+
| @@sql_mode |
+------------+
|            |
+------------+
1 row in set (0.00 sec)


mysql> INSERT INTO tn VALUES (NULL), (0), (1);
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
```

   However, changing the server SQL mode to treat division by zero as an error and to enforce strict error handling causes the same `INSERT` statement to fail, as shown here:

```
mysql> SET sql_mode='STRICT_ALL_TABLES,ERROR_FOR_DIVISION_BY_ZERO';
Query OK, 0 rows affected (0.00 sec)

mysql> INSERT INTO tn VALUES (NULL), (0), (1);
ERROR 1365 (22012): Division by 0
```

2. **Table accessibility.** Sometimes a change in the server SQL mode can make partitioned tables unusable. The following `CREATE TABLE` statement can be executed successfully only if the `NO_UNSIGNED_SUBTRACTION` mode is in effect:

```
mysql> SELECT @@sql_mode;
+------------+
| @@sql_mode |
+------------+
|            |
+------------+
1 row in set (0.00 sec)

mysql> CREATE TABLE tu (c1 BIGINT UNSIGNED)
    ->   PARTITION BY RANGE(c1 - 10) (
    ->     PARTITION p0 VALUES LESS THAN (-5),
    ->     PARTITION p1 VALUES LESS THAN (0),
    ->     PARTITION p2 VALUES LESS THAN (5),
    ->     PARTITION p3 VALUES LESS THAN (10),
    ->     PARTITION p4 VALUES LESS THAN (MAXVALUE)
    -> );
ERROR 1563 (HY000): Partition constant is out of partition function domain

mysql> SET sql_mode='NO_UNSIGNED_SUBTRACTION';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @@sql_mode;
+------------------------+
| @@sql_mode             |
```

```
+------------------------+
| NO_UNSIGNED_SUBTRACTION |
+------------------------+
1 row in set (0.00 sec)

mysql> CREATE TABLE tu (c1 BIGINT UNSIGNED)
    ->   PARTITION BY RANGE(c1 - 10) (
    ->     PARTITION p0 VALUES LESS THAN (-5),
    ->     PARTITION p1 VALUES LESS THAN (0),
    ->     PARTITION p2 VALUES LESS THAN (5),
    ->     PARTITION p3 VALUES LESS THAN (10),
    ->     PARTITION p4 VALUES LESS THAN (MAXVALUE)
    -> );
Query OK, 0 rows affected (0.05 sec)
```

If you remove the `NO_UNSIGNED_SUBTRACTION` server SQL mode after creating `tu`, you may no longer be able to access this table:

```
mysql> SET sql_mode='';
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM tu;
ERROR 1563 (HY000): Partition constant is out of partition function domain
mysql> INSERT INTO tu VALUES (20);
ERROR 1563 (HY000): Partition constant is out of partition function domain
```

Server SQL modes also impact replication of partitioned tables. Differing SQL modes on source and replica can lead to partitioning expressions being evaluated differently; this can cause the distribution of data among partitions to be different in the source's and replica's copies of a given table, and may even cause inserts into partitioned tables that succeed on the source to fail on the replica. For best results, you should always use the same server SQL mode on the source and on the replica.

Performance considerations. Some effects of partitioning operations on performance are given in the following list:

- **File system operations.** Partitioning and repartitioning operations (such as `ALTER TABLE` with `PARTITION BY ...`, `REORGANIZE PARTITION`, or `REMOVE PARTITIONING`) depend on file system operations for their implementation. This means that the speed of these operations is affected by such factors as file system type and characteristics, disk speed, swap space, file handling efficiency of the operating system, and MySQL server options and variables that relate to file handling. In particular, you should make sure that `large_files_support` is enabled and that `open_files_limit` is set properly. For partitioned tables using the `MyISAM` storage engine, increasing `myisam_max_sort_file_size` may improve performance; partitioning and repartitioning operations involving `InnoDB` tables may be made more efficient by enabling `innodb_file_per_table`.

  See also Maximum number of partitions.

- **MyISAM and partition file descriptor usage.** For a partitioned `MyISAM` table, MySQL uses 2 file descriptors for each partition, for each such table that is open. This means that you need many more file descriptors to perform operations on a partitioned `MyISAM` table than on a table which is identical to it except that the latter table is not partitioned, particularly when performing `ALTER TABLE` operations.

  Assume a `MyISAM` table `t` with 100 partitions, such as the table created by this SQL statement:

```
CREATE TABLE t (c1 VARCHAR(50))
PARTITION BY KEY (c1) PARTITIONS 100
ENGINE=MYISAM;
```

Note

For brevity, we use `KEY` partitioning for the table shown in this example, but file descriptor usage as described here applies to all partitioned `MyISAM` tables, regardless of the type of partitioning that is employed. Partitioned

tables using other storage engines such as `InnoDB` are not affected by this issue.

Now assume that you wish to repartition `t` so that it has 101 partitions, using the statement shown here:

```
ALTER TABLE t PARTITION BY KEY (c1) PARTITIONS 101;
```

To process this `ALTER TABLE` statement, MySQL uses 402 file descriptors—that is, two for each of the 100 original partitions, plus two for each of the 101 new partitions. This is because all partitions (old and new) must be opened concurrently during the reorganization of the table data. It is recommended that, if you expect to perform such operations, you should make sure that the `open_files_limit` system variable is not set too low to accommodate them.

- **ALGORITHM and ALTER TABLE.** Partitioned tables do not support `ALTER TABLE` statements with `ALGORITHM=DEFAULT`, `ALGORITHM=INPLACE`, or `ALGORITHM=COPY`. See Section 13.1.7.1, “ALTER TABLE Partition Operations”.

- **Table locks.** Generally, the process executing a partitioning operation on a table takes a write lock on the table. Reads from such tables are relatively unaffected; pending `INSERT` and `UPDATE` operations are performed as soon as the partitioning operation has completed. For `InnoDB`-specific information related to this limitation, see Partitioning Operations.

  `LOCK` is not supported for `ALTER TABLE` statements on partitioned tables.

- **Storage engine.** Partitioning operations, queries, and update operations generally tend to be faster with `MyISAM` tables than with `InnoDB` or `NDB` tables.

- **Indexes; partition pruning.** As with nonpartitioned tables, proper use of indexes can speed up queries on partitioned tables significantly. In addition, designing partitioned tables and queries on these tables to take advantage of *partition pruning* can improve performance dramatically. See Section 19.4, “Partition Pruning”, for more information.

  Index condition pushdown is not supported for partitioned tables. See Section 8.2.1.5, “Index Condition Pushdown Optimization”.

- **Performance with LOAD DATA.** In MySQL 5.6, `LOAD DATA` uses buffering to improve performance. You should be aware that the buffer uses 130 KB memory per partition to achieve this.

Maximum number of partitions.
Prior to MySQL 5.6.7, the maximum possible number of partitions for a given table not using the `NDB` storage engine was 1024. Beginning with MySQL 5.6.7, this limit is increased to 8192 partitions. Regardless of the MySQL Server version, this maximum includes subpartitions.

The maximum possible number of user-defined partitions for a table using the `NDB` storage engine is determined according to the version of the NDB Cluster software being used, the number of data nodes, and other factors. See NDB and user-defined partitioning, for more information.

If, when creating tables with a large number of partitions (but less than the maximum), you encounter an error message such as `Got error ... from storage engine: Out of resources when opening file`, you may be able to address the issue by increasing the value of the `open_files_limit` system variable. However, this is dependent on the operating system, and may not be possible or advisable on all platforms; see Section B.3.2.17, “File Not Found and Similar Errors”, for more information. In some cases, using large numbers (hundreds) of partitions may also not be advisable due to other concerns, so using more partitions does not automatically lead to better results.

See also File system operations.

Query cache not supported.
The query cache is not supported for partitioned tables. Beginning with MySQL 5.6.5, the query cache is automatically disabled for queries involving partitioned tables, and cannot be enabled for such queries. (Bug #53775)

Per-partition key caches.
In MySQL 5.6, key caches are supported for partitioned `MyISAM` tables, using the `CACHE INDEX` and `LOAD INDEX INTO CACHE` statements. Key caches may be defined for one, several, or all partitions, and indexes for one, several, or all partitions may be preloaded into key caches.

Foreign keys not supported for partitioned InnoDB tables.
Partitioned tables using the `InnoDB` storage engine do not support foreign keys. More specifically, this means that the following two statements are true:

1. No definition of an `InnoDB` table employing user-defined partitioning may contain foreign key references; no `InnoDB` table whose definition contains foreign key references may be partitioned.

2. No `InnoDB` table definition may contain a foreign key reference to a user-partitioned table; no `InnoDB` table with user-defined partitioning may contain columns referenced by foreign keys.

The scope of the restrictions just listed includes all tables that use the `InnoDB` storage engine. `CREATE TABLE` and `ALTER TABLE` statements that would result in tables violating these restrictions are not allowed.

ALTER TABLE ... ORDER BY. An `ALTER TABLE ... ORDER BY` *`column`* statement run against a partitioned table causes ordering of rows only within each partition.

Effects on REPLACE statements by modification of primary keys. It can be desirable in some cases (see Section 19.6.1, “Partitioning Keys, Primary Keys, and Unique Keys”) to modify a table's primary key. Be aware that, if your application uses `REPLACE` statements and you do this, the results of these statements can be drastically altered. See Section 13.2.8, “REPLACE Statement”, for more information and an example.

FULLTEXT indexes.
Partitioned tables do not support `FULLTEXT` indexes or searches, even for partitioned tables employing the `InnoDB` or `MyISAM` storage engine.

Spatial columns. Columns with spatial data types such as `POINT` or `GEOMETRY` cannot be used in partitioned tables.

Temporary tables.
Temporary tables cannot be partitioned. (Bug #17497)

Log tables. It is not possible to partition the log tables; an `ALTER TABLE ... PARTITION BY ...` statement on such a table fails with an error.

Data type of partitioning key.
A partitioning key must be either an integer column or an expression that resolves to an integer. Expressions employing `ENUM` columns cannot be used. The column or expression value may also be `NULL`. (See Section 19.2.7, “How MySQL Partitioning Handles NULL”.)

There are two exceptions to this restriction:

1. When partitioning by `[LINEAR] KEY`, it is possible to use columns of any valid MySQL data type other than `TEXT` or `BLOB` as partitioning keys, because MySQL's internal key-hashing functions produce the correct data type from these types. For example, the following two `CREATE TABLE` statements are valid:

```
CREATE TABLE tkc (c1 CHAR)
PARTITION BY KEY(c1)
PARTITIONS 4;

CREATE TABLE tke
    ( c1 ENUM('red', 'orange', 'yellow', 'green', 'blue', 'indigo', 'violet') )
PARTITION BY LINEAR KEY(c1)
PARTITIONS 6;
```

2. When partitioning by `RANGE COLUMNS` or `LIST COLUMNS`, it is possible to use string, `DATE`, and `DATETIME` columns. For example, each of the following `CREATE TABLE` statements is valid:

```
CREATE TABLE rc (c1 INT, c2 DATE)
PARTITION BY RANGE COLUMNS(c2) (
    PARTITION p0 VALUES LESS THAN('1990-01-01'),
    PARTITION p1 VALUES LESS THAN('1995-01-01'),
    PARTITION p2 VALUES LESS THAN('2000-01-01'),
    PARTITION p3 VALUES LESS THAN('2005-01-01'),
    PARTITION p4 VALUES LESS THAN(MAXVALUE)
);

CREATE TABLE lc (c1 INT, c2 CHAR(1))
PARTITION BY LIST COLUMNS(c2) (
    PARTITION p0 VALUES IN('a', 'd', 'g', 'j', 'm', 'p', 's', 'v', 'y'),
    PARTITION p1 VALUES IN('b', 'e', 'h', 'k', 'n', 'q', 't', 'w', 'z'),
    PARTITION p2 VALUES IN('c', 'f', 'i', 'l', 'o', 'r', 'u', 'x', NULL)
);
```

Neither of the preceding exceptions applies to `BLOB` or `TEXT` column types.

Subqueries.
A partitioning key may not be a subquery, even if that subquery resolves to an integer value or `NULL`.

Column index prefixes not supported for key partitioning. When creating a table that is partitioned by key, any columns in the partitioning key which use column prefixes are ignored by the table's partitioning function. Consider the following `CREATE TABLE` statement, which has three `VARCHAR` columns, and whose primary key uses all three columns and specifies prefixes for two of them:

```
CREATE TABLE t1 (
    a VARCHAR(10000),
    b VARCHAR(25),
    c VARCHAR(10),
    PRIMARY KEY (a(10), b, c(2))
) PARTITION BY KEY() PARTITIONS 2;
```

This statement is accepted, but the resulting table is actually created as if you had issued the following statement, using only the primary key column which does not include a prefix (column `b`) for the partitioning key:

```
CREATE TABLE t1 (
    a VARCHAR(10000),
    b VARCHAR(25),
    c VARCHAR(10),
    PRIMARY KEY (a(10), b, c(2))
) PARTITION BY KEY(b) PARTITIONS 2;
```

No warning is issued or any other indication provided that this has occurred, except in the event that all columns specified for the partitioning key use prefixes, in which case the statement fails with the error message shown here:

```
mysql> CREATE TABLE t2 (
    ->     a VARCHAR(10000),
    ->     b VARCHAR(25),
    ->     c VARCHAR(10),
    ->     PRIMARY KEY (a(10), b(5), c(2))
    -> ) PARTITION BY KEY() PARTITIONS 2;
ERROR 1503 (HY000): A PRIMARY KEY must include all columns in the
table's partitioning function
```

This also occurs when altering or upgrading such tables, and includes cases in which the columns used in the partitioning function are defined implicitly as those in the table's primary key by employing an empty `PARTITION BY KEY()` clause.

This is a known issue which is addressed in MySQL 8.0 by deprecating the permissive behavior; in MYSQL 8.0, if any columns using prefixes are included in a table's partitioning function, the server logs

an appropriate warning for each such column, or raises a descriptive error if necessary. (Allowing the use of columns with prefixes in partitioning keys is subject to removal altogether in a future version of MySQL.)

For general information about partitioning tables by key, see Section 19.2.5, “KEY Partitioning”.

Issues with subpartitions.
Subpartitions must use `HASH` or `KEY` partitioning. Only `RANGE` and `LIST` partitions may be subpartitioned; `HASH` and `KEY` partitions cannot be subpartitioned.

`SUBPARTITION BY KEY` requires that the subpartitioning column or columns be specified explicitly, unlike the case with `PARTITION BY KEY`, where it can be omitted (in which case the table's primary key column is used by default). Consider the table created by this statement:

```
CREATE TABLE ts (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(30)
);
```

You can create a table having the same columns, partitioned by `KEY`, using a statement such as this one:

```
CREATE TABLE ts (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(30)
)
PARTITION BY KEY()
PARTITIONS 4;
```

The previous statement is treated as though it had been written like this, with the table's primary key column used as the partitioning column:

```
CREATE TABLE ts (
    id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    name VARCHAR(30)
)
PARTITION BY KEY(id)
PARTITIONS 4;
```

However, the following statement that attempts to create a subpartitioned table using the default column as the subpartitioning column fails, and the column must be specified for the statement to succeed, as shown here:

```
mysql> CREATE TABLE ts (
    ->     id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->     name VARCHAR(30)
    -> )
    -> PARTITION BY RANGE(id)
    -> SUBPARTITION BY KEY()
    -> SUBPARTITIONS 4
    -> (
    ->     PARTITION p0 VALUES LESS THAN (100),
    ->     PARTITION p1 VALUES LESS THAN (MAXVALUE)
    -> );
ERROR 1064 (42000): You have an error in your SQL syntax; check the manual that
corresponds to your MySQL server version for the right syntax to use near ')

mysql> CREATE TABLE ts (
    ->     id INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    ->     name VARCHAR(30)
    -> )
    -> PARTITION BY RANGE(id)
    -> SUBPARTITION BY KEY(id)
    -> SUBPARTITIONS 4
    -> (
    ->     PARTITION p0 VALUES LESS THAN (100),
```

```
    ->     PARTITION p1 VALUES LESS THAN (MAXVALUE)
    -> );
Query OK, 0 rows affected (0.07 sec)
```

This is a known issue (see Bug #51470).

DELAYED option not supported. Use of `INSERT DELAYED` to insert rows into a partitioned table is not supported. Attempting to do so fails with an error.

DATA DIRECTORY and INDEX DIRECTORY options. `DATA DIRECTORY` and `INDEX DIRECTORY` are subject to the following restrictions when used with partitioned tables:

- Table-level `DATA DIRECTORY` and `INDEX DIRECTORY` options are ignored (see Bug #32091).
- On Windows, the `DATA DIRECTORY` and `INDEX DIRECTORY` options are not supported for individual partitions or subpartitions of `MyISAM` tables (Bug #30459). However, you can use `DATA DIRECTORY` for individual partitions or subpartitions of `InnoDB` tables.

Repairing and rebuilding partitioned tables. The statements `CHECK TABLE`, `OPTIMIZE TABLE`, `ANALYZE TABLE`, and `REPAIR TABLE` are supported for partitioned tables.

In addition, you can use `ALTER TABLE ... REBUILD PARTITION` to rebuild one or more partitions of a partitioned table; `ALTER TABLE ... REORGANIZE PARTITION` also causes partitions to be rebuilt. See Section 13.1.7, “ALTER TABLE Statement”, for more information about these two statements.

`mysqlcheck`, `myisamchk`, and `myisampack` are not supported with partitioned tables.

FOR EXPORT option (FLUSH TABLES). The `FLUSH TABLES` statement's `FOR EXPORT` option is not supported for partitioned `InnoDB` tables in MySQL 5.6.16 and earlier. (Bug #16943907)

19.6.1 Partitioning Keys, Primary Keys, and Unique Keys

This section discusses the relationship of partitioning keys with primary keys and unique keys. The rule governing this relationship can be expressed as follows: All columns used in the partitioning expression for a partitioned table must be part of every unique key that the table may have.

In other words, *every unique key on the table must use every column in the table's partitioning expression*. (This also includes the table's primary key, since it is by definition a unique key. This particular case is discussed later in this section.) For example, each of the following table creation statements is invalid:

```
CREATE TABLE t1 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    UNIQUE KEY (col1, col2)
)
PARTITION BY HASH(col3)
PARTITIONS 4;

CREATE TABLE t2 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    UNIQUE KEY (col1),
    UNIQUE KEY (col3)
)
PARTITION BY HASH(col1 + col3)
PARTITIONS 4;
```

In each case, the proposed table would have at least one unique key that does not include all columns used in the partitioning expression.

Each of the following statements is valid, and represents one way in which the corresponding invalid table creation statement could be made to work:

```
CREATE TABLE t1 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    UNIQUE KEY (col1, col2, col3)
)
PARTITION BY HASH(col3)
PARTITIONS 4;

CREATE TABLE t2 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    UNIQUE KEY (col1, col3)
)
PARTITION BY HASH(col1 + col3)
PARTITIONS 4;
```

This example shows the error produced in such cases:

```
mysql> CREATE TABLE t3 (
    ->     col1 INT NOT NULL,
    ->     col2 DATE NOT NULL,
    ->     col3 INT NOT NULL,
    ->     col4 INT NOT NULL,
    ->     UNIQUE KEY (col1, col2),
    ->     UNIQUE KEY (col3)
    -> )
    -> PARTITION BY HASH(col1 + col3)
    -> PARTITIONS 4;
ERROR 1491 (HY000): A PRIMARY KEY must include all columns in the table's partitioning function
```

The `CREATE TABLE` statement fails because both `col1` and `col3` are included in the proposed partitioning key, but neither of these columns is part of both of unique keys on the table. This shows one possible fix for the invalid table definition:

```
mysql> CREATE TABLE t3 (
    ->     col1 INT NOT NULL,
    ->     col2 DATE NOT NULL,
    ->     col3 INT NOT NULL,
    ->     col4 INT NOT NULL,
    ->     UNIQUE KEY (col1, col2, col3),
    ->     UNIQUE KEY (col3)
    -> )
    -> PARTITION BY HASH(col3)
    -> PARTITIONS 4;
Query OK, 0 rows affected (0.05 sec)
```

In this case, the proposed partitioning key `col3` is part of both unique keys, and the table creation statement succeeds.

The following table cannot be partitioned at all, because there is no way to include in a partitioning key any columns that belong to both unique keys:

```
CREATE TABLE t4 (
    col1 INT NOT NULL,
    col2 INT NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    UNIQUE KEY (col1, col3),
    UNIQUE KEY (col2, col4)
);
```

Since every primary key is by definition a unique key, this restriction also includes the table's primary key, if it has one. For example, the next two statements are invalid:

```
CREATE TABLE t5 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    PRIMARY KEY(col1, col2)
)
PARTITION BY HASH(col3)
PARTITIONS 4;

CREATE TABLE t6 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    PRIMARY KEY(col1, col3),
    UNIQUE KEY(col2)
)
PARTITION BY HASH( YEAR(col2) )
PARTITIONS 4;
```

In both cases, the primary key does not include all columns referenced in the partitioning expression. However, both of the next two statements are valid:

```
CREATE TABLE t7 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    PRIMARY KEY(col1, col2)
)
PARTITION BY HASH(col1 + YEAR(col2))
PARTITIONS 4;

CREATE TABLE t8 (
    col1 INT NOT NULL,
    col2 DATE NOT NULL,
    col3 INT NOT NULL,
    col4 INT NOT NULL,
    PRIMARY KEY(col1, col2, col4),
    UNIQUE KEY(col2, col1)
)
PARTITION BY HASH(col1 + YEAR(col2))
PARTITIONS 4;
```

If a table has no unique keys—this includes having no primary key—then this restriction does not apply, and you may use any column or columns in the partitioning expression as long as the column type is compatible with the partitioning type.

For the same reason, you cannot later add a unique key to a partitioned table unless the key includes all columns used by the table's partitioning expression. Consider the partitioned table created as shown here:

```
mysql> CREATE TABLE t_no_pk (c1 INT, c2 INT)
    ->     PARTITION BY RANGE(c1) (
    ->         PARTITION p0 VALUES LESS THAN (10),
    ->         PARTITION p1 VALUES LESS THAN (20),
    ->         PARTITION p2 VALUES LESS THAN (30),
    ->         PARTITION p3 VALUES LESS THAN (40)
    ->     );
Query OK, 0 rows affected (0.12 sec)
```

It is possible to add a primary key to `t_no_pk` using either of these `ALTER TABLE` statements:

```
#  possible PK
mysql> ALTER TABLE t_no_pk ADD PRIMARY KEY(c1);
Query OK, 0 rows affected (0.13 sec)
Records: 0  Duplicates: 0  Warnings: 0

# drop this PK
```

```
mysql> ALTER TABLE t_no_pk DROP PRIMARY KEY;
Query OK, 0 rows affected (0.10 sec)
Records: 0  Duplicates: 0  Warnings: 0

#  use another possible PK
mysql> ALTER TABLE t_no_pk ADD PRIMARY KEY(c1, c2);
Query OK, 0 rows affected (0.12 sec)
Records: 0  Duplicates: 0  Warnings: 0

# drop this PK
mysql> ALTER TABLE t_no_pk DROP PRIMARY KEY;
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

However, the next statement fails, because `c1` is part of the partitioning key, but is not part of the proposed primary key:

```
#  fails with error 1503
mysql> ALTER TABLE t_no_pk ADD PRIMARY KEY(c2);
ERROR 1503 (HY000): A PRIMARY KEY must include all columns in the table's partitioning function
```

Since `t_no_pk` has only `c1` in its partitioning expression, attempting to adding a unique key on `c2` alone fails. However, you can add a unique key that uses both `c1` and `c2`.

These rules also apply to existing nonpartitioned tables that you wish to partition using `ALTER TABLE ... PARTITION BY`. Consider a table `np_pk` created as shown here:

```
mysql> CREATE TABLE np_pk (
    ->     id INT NOT NULL AUTO_INCREMENT,
    ->     name VARCHAR(50),
    ->     added DATE,
    ->     PRIMARY KEY (id)
    -> );
Query OK, 0 rows affected (0.08 sec)
```

The following `ALTER TABLE` statement fails with an error, because the `added` column is not part of any unique key in the table:

```
mysql> ALTER TABLE np_pk
    ->     PARTITION BY HASH( TO_DAYS(added) )
    ->     PARTITIONS 4;
ERROR 1503 (HY000): A PRIMARY KEY must include all columns in the table's partitioning function
```

However, this statement using the `id` column for the partitioning column is valid, as shown here:

```
mysql> ALTER TABLE np_pk
    ->     PARTITION BY HASH(id)
    ->     PARTITIONS 4;
Query OK, 0 rows affected (0.11 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

In the case of `np_pk`, the only column that may be used as part of a partitioning expression is `id`; if you wish to partition this table using any other column or columns in the partitioning expression, you must first modify the table, either by adding the desired column or columns to the primary key, or by dropping the primary key altogether.

19.6.2 Partitioning Limitations Relating to Storage Engines

The following limitations apply to the use of storage engines with user-defined partitioning of tables.

MERGE storage engine. User-defined partitioning and the `MERGE` storage engine are not compatible. Tables using the `MERGE` storage engine cannot be partitioned. Partitioned tables cannot be merged.

FEDERATED storage engine. Partitioning of `FEDERATED` tables is not supported; it is not possible to create partitioned `FEDERATED` tables.

CSV storage engine. Partitioned tables using the `CSV` storage engine are not supported; it is not possible to create partitioned `CSV` tables.

InnoDB storage engine. `InnoDB` foreign keys and MySQL partitioning are not compatible. Partitioned `InnoDB` tables cannot have foreign key references, nor can they have columns referenced by foreign keys. `InnoDB` tables which have or which are referenced by foreign keys cannot be partitioned.

`InnoDB` does not support the use of multiple disks for subpartitions. (This is currently supported only by `MyISAM`.)

In addition, `ALTER TABLE ... OPTIMIZE PARTITION` does not work correctly with partitioned tables that use the `InnoDB` storage engine. Use `ALTER TABLE ... REBUILD PARTITION` and `ALTER TABLE ... ANALYZE PARTITION`, instead, for such tables. For more information, see Section 13.1.7.1, “ALTER TABLE Partition Operations”.

User-defined partitioning and the NDB storage engine (NDB Cluster). Partitioning by `KEY` (including `LINEAR KEY`) is the only type of partitioning supported for the `NDB` storage engine. It is not possible under normal circumstances in MySQL NDB Cluster 7.4 to create an NDB Cluster table using any partitioning type other than [`LINEAR`] `KEY`, and attempting to do so fails with an error.

Exception (not for production): It is possible to override this restriction by setting the `new` system variable on NDB Cluster SQL nodes to `ON`. If you choose to do this, you should be aware that tables using partitioning types other than `[LINEAR] KEY` are not supported in production. *In such cases, you can create and use tables with partitioning types other than `KEY` or `LINEAR KEY`, but you do this entirely at your own risk*.

The maximum number of partitions that can be defined for an `NDB` table depends on the number of data nodes and node groups in the cluster, the version of the NDB Cluster software in use, and other factors. See NDB and user-defined partitioning, for more information.

The maximum amount of fixed-size data that can be stored per partition in an `NDB` table is 16 GB.

`CREATE TABLE` and `ALTER TABLE` statements that would cause a user-partitioned `NDB` table not to meet either or both of the following two requirements are not permitted, and fail with an error:

1. The table must have an explicit primary key.

2. All columns listed in the table's partitioning expression must be part of the primary key.

Exception. If a user-partitioned `NDB` table is created using an empty column-list (that is, using `PARTITION BY KEY()` or `PARTITION BY LINEAR KEY()`), then no explicit primary key is required.

Upgrading partitioned tables. When performing an upgrade, tables which are partitioned by `KEY` and which use any storage engine other than `NDB` must be dumped and reloaded.

Same storage engine for all partitions. All partitions of a partitioned table must use the same storage engine and it must be the same storage engine used by the table as a whole. In addition, if one does not specify an engine on the table level, then one must do either of the following when creating or altering a partitioned table:

- Do *not* specify any engine for *any* partition or subpartition
- Specify the engine for *all* partitions or subpartitions

19.6.3 Partitioning Limitations Relating to Functions

This section discusses limitations in MySQL Partitioning relating specifically to functions used in partitioning expressions.

Only the MySQL functions shown in the following list are allowed in partitioning expressions:

- `ABS()`
- `CEILING()` (see CEILING() and FLOOR())
- `DATEDIFF()`
- `DAY()`
- `DAYOFMONTH()`
- `DAYOFWEEK()`
- `DAYOFYEAR()`
- `EXTRACT()` (see EXTRACT() function with WEEK specifier)
- `FLOOR()` (see CEILING() and FLOOR())
- `HOUR()`
- `MICROSECOND()`
- `MINUTE()`
- `MOD()`
- `MONTH()`
- `QUARTER()`
- `SECOND()`
- `TIME_TO_SEC()`
- `TO_DAYS()`
- `TO_SECONDS()`
- `UNIX_TIMESTAMP()` (with `TIMESTAMP` columns)
- `WEEKDAY()`
- `YEAR()`
- `YEARWEEK()`

In MySQL 5.6, range optimization can be used for the `TO_DAYS()`, `TO_SECONDS()`, and `YEAR()` functions. In addition, beginning with MySQL 5.6.3, `UNIX_TIMESTAMP()` is treated as monotonic in partitioning expressions. See Section 19.4, “Partition Pruning”, for more information.

CEILING() and FLOOR(). Each of these functions returns an integer only if it is passed an argument of an exact numeric type, such as one of the `INT` types or `DECIMAL`. This means, for example, that the following `CREATE TABLE` statement fails with an error, as shown here:

```
mysql> CREATE TABLE t (c FLOAT) PARTITION BY LIST( FLOOR(c) )(
    ->     PARTITION p0 VALUES IN (1,3,5),
    ->     PARTITION p1 VALUES IN (2,4,6)
    -> );
ERROR 1490 (HY000): The PARTITION function returns the wrong type
```

EXTRACT() function with WEEK specifier. The value returned by the `EXTRACT()` function, when used as `EXTRACT(WEEK FROM col)`, depends on the value of the `default_week_format` system variable. For this reason, beginning with MySQL 5.6.2, `EXTRACT()` is no longer permitted as a partitioning function when it specifies the unit as `WEEK`. (Bug #54483)

See Section 12.6.2, “Mathematical Functions”, for more information about the return types of these functions, as well as Section 11.1, “Numeric Data Types”.

19.6.4 Partitioning and Locking

In MySQL 5.6.5 and earlier, for storage engines such as `MyISAM` that actually execute table-level locks when executing DML or DDL statements, such a statement affecting a partitioned table imposed a lock on the table as a whole; that is, all partitions were locked until the statement was finished. MySQL 5.6.6 implements *partition lock pruning*, which eliminates unneeded locks in many cases. In MySQL 5.6.6 and later, most statements reading from or updating a partitioned `MyISAM` table cause only the effected partitions to be locked. For example, prior to MySQL 5.6.6, a `SELECT` from a partitioned `MyISAM` table caused a lock on the entire table; in MySQL 5.6.6 and later, only those partitions actually containing rows that satisfy the `SELECT` statement's `WHERE` condition are locked. This has the effect of increasing the speed and efficiency of concurrent operations on partitioned `MyISAM` tables. This improvement becomes particularly noticeable when working with `MyISAM` tables that have many (32 or more) partitions.

This change in behavior does not have any impact on statements affecting partitioned tables using storage engines such as `InnoDB`, that employ row-level locking and do not actually perform (or need to perform) the locks prior to partition pruning.

The next few paragraphs discuss the effects of partition lock pruning for various MySQL statements on tables using storage engines that employ table-level locks.

Effects on DML statements

`SELECT` statements (including those containing unions or joins) now lock only those partitions that actually need to be read. This also applies to `SELECT ... PARTITION`.

An `UPDATE` prunes locks only for tables on which no partitioning columns are updated.

`REPLACE` and `INSERT` now lock only those partitions having rows to be inserted or replaced. However, if an `AUTO_INCREMENT` value is generated for any partitioning column then all partitions are locked.

`INSERT ... ON DUPLICATE KEY UPDATE` is pruned as long as no partitioning column is updated.

`INSERT ... SELECT` now locks only those partitions in the source table that need to be read, although all partitions in the target table are locked.

Note

`INSERT DELAYED` is not supported for partitioned tables.

Locks imposed by `LOAD DATA` statements on partitioned tables cannot be pruned.

The presence of `BEFORE INSERT` or `BEFORE UPDATE` triggers using any partitioning column of a partitioned table means that locks on `INSERT` and `UPDATE` statements updating this table cannot be pruned, since the trigger can alter its values: A `BEFORE INSERT` trigger on any of the table's partitioning columns means that locks set by `INSERT` or `REPLACE` cannot be pruned, since the `BEFORE INSERT` trigger may change a row's partitioning columns before the row is inserted, forcing the row into a different partition than it would be otherwise. A `BEFORE UPDATE` trigger on a partitioning column means that locks imposed by `UPDATE` or `INSERT ... ON DUPLICATE KEY UPDATE` cannot be pruned.

Affected DDL statements

`CREATE VIEW` no longer causes any locks.

`ALTER TABLE ... EXCHANGE PARTITION` now prunes locks; only the exchanged table and the exchanged partition are locked.

`ALTER TABLE ... TRUNCATE PARTITION` now prunes locks; only the partitions to be emptied are locked.

`ALTER TABLE` statements still take metadata locks on the table level.

Other statements

`LOCK TABLES` cannot prune partition locks.

`CALL stored_procedure(expr)` supports lock pruning, but evaluating *`expr`* does not.

`DO` and `SET` statements do not support partitioning lock pruning.

Chapter 20 Stored Objects

Table of Contents

20.1 Defining Stored Programs 2940
20.2 Using Stored Routines 2941
20.2.1 Stored Routine Syntax 2942
20.2.2 Stored Routines and MySQL Privileges 2942
20.2.3 Stored Routine Metadata 2943
20.2.4 Stored Procedures, Functions, Triggers, and LAST_INSERT_ID() 2943
20.3 Using Triggers 2943
20.3.1 Trigger Syntax and Examples 2944
20.3.2 Trigger Metadata 2947
20.4 Using the Event Scheduler 2948
20.4.1 Event Scheduler Overview 2948
20.4.2 Event Scheduler Configuration 2949
20.4.3 Event Syntax 2951
20.4.4 Event Metadata 2951
20.4.5 Event Scheduler Status 2952
20.4.6 The Event Scheduler and MySQL Privileges 2953
20.5 Using Views 2955
20.5.1 View Syntax 2955
20.5.2 View Processing Algorithms 2956
20.5.3 Updatable and Insertable Views 2957
20.5.4 The View WITH CHECK OPTION Clause 2959
20.5.5 View Metadata 2959
20.6 Stored Object Access Control 2959
20.7 Stored Program Binary Logging 2963
20.8 Restrictions on Stored Programs 2968
20.9 Restrictions on Views 2972

This chapter discusses stored database objects that are defined in terms of SQL code that is stored on the server for later execution.

Stored objects include these object types:

- Stored procedure: An object created with `CREATE PROCEDURE` and invoked using the `CALL` statement. A procedure does not have a return value but can modify its parameters for later inspection by the caller. It can also generate result sets to be returned to the client program.
- Stored function: An object created with `CREATE FUNCTION` and used much like a built-in function. You invoke it in an expression and it returns a value during expression evaluation.
- Trigger: An object created with `CREATE TRIGGER` that is associated with a table. A trigger is activated when a particular event occurs for the table, such as an insert or update.
- Event: An object created with `CREATE EVENT` and invoked by the server according to schedule.
- View: An object created with `CREATE VIEW` that when referenced produces a result set. A view acts as a virtual table.

Terminology used in this document reflects the stored object hierarchy:

- Stored routines include stored procedures and functions.
- Stored programs include stored routines, triggers, and events.
- Stored objects include stored programs and views.

This chapter describes how to use stored objects. The following sections provide additional information about SQL syntax for statements related to these objects, and about object processing:

- For each object type, there are `CREATE`, `ALTER`, and `DROP` statements that control which objects exist and how they are defined. See Section 13.1, “Data Definition Statements”.
- The `CALL` statement is used to invoke stored procedures. See Section 13.2.1, “CALL Statement”.
- Stored program definitions include a body that may use compound statements, loops, conditionals, and declared variables. See Section 13.6, “Compound Statements”.
- Metadata changes to objects referred to by stored programs are detected and cause automatic reparsing of the affected statements when the program is next executed. For more information, see Section 8.10.4, “Caching of Prepared Statements and Stored Programs”.

20.1 Defining Stored Programs

Each stored program contains a body that consists of an SQL statement. This statement may be a compound statement made up of several statements separated by semicolon (`;`) characters. For example, the following stored procedure has a body made up of a `BEGIN ... END` block that contains a `SET` statement and a `REPEAT` loop that itself contains another `SET` statement:

```
CREATE PROCEDURE dorepeat(p1 INT)
BEGIN
  SET @x = 0;
  REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
END;
```

If you use the `mysql` client program to define a stored program containing semicolon characters, a problem arises. By default, `mysql` itself recognizes the semicolon as a statement delimiter, so you must redefine the delimiter temporarily to cause `mysql` to pass the entire stored program definition to the server.

To redefine the `mysql` delimiter, use the `delimiter` command. The following example shows how to do this for the `dorepeat()` procedure just shown. The delimiter is changed to `//` to enable the entire definition to be passed to the server as a single statement, and then restored to `;` before invoking the procedure. This enables the `;` delimiter used in the procedure body to be passed through to the server rather than being interpreted by `mysql` itself.

```
mysql> delimiter //

mysql> CREATE PROCEDURE dorepeat(p1 INT)
    -> BEGIN
    ->   SET @x = 0;
    ->   REPEAT SET @x = @x + 1; UNTIL @x > p1 END REPEAT;
    -> END
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql> delimiter ;

mysql> CALL dorepeat(1000);
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT @x;
+------+
| @x   |
+------+
| 1001 |
+------+
1 row in set (0.00 sec)
```

You can redefine the delimiter to a string other than `//`, and the delimiter can consist of a single character or multiple characters. You should avoid the use of the backslash (`\`) character because that is the escape character for MySQL.

The following is an example of a function that takes a parameter, performs an operation using an SQL function, and returns the result. In this case, it is unnecessary to use `delimiter` because the function definition contains no internal `;` statement delimiters:

```
mysql> CREATE FUNCTION hello (s CHAR(20))
mysql> RETURNS CHAR(50) DETERMINISTIC
    -> RETURN CONCAT('Hello, ',s,'!');
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT hello('world');
+----------------+
| hello('world') |
+----------------+
| Hello, world!  |
+----------------+
1 row in set (0.00 sec)
```

20.2 Using Stored Routines

MySQL supports stored routines (procedures and functions). A stored routine is a set of SQL statements that can be stored in the server. Once this has been done, clients don't need to keep reissuing the individual statements but can refer to the stored routine instead.

Stored routines require the `proc` table in the `mysql` database. This table is created during the MySQL installation procedure. If you are upgrading to MySQL 5.6 from an earlier version, be sure to update your grant tables to make sure that the `proc` table exists. See Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”.

Stored routines can be particularly useful in certain situations:

- When multiple client applications are written in different languages or work on different platforms, but need to perform the same database operations.
- When security is paramount. Banks, for example, use stored procedures and functions for all common operations. This provides a consistent and secure environment, and routines can ensure that each operation is properly logged. In such a setup, applications and users would have no access to the database tables directly, but can only execute specific stored routines.

Stored routines can provide improved performance because less information needs to be sent between the server and the client. The tradeoff is that this does increase the load on the database server because more of the work is done on the server side and less is done on the client (application) side. Consider this if many client machines (such as Web servers) are serviced by only one or a few database servers.

Stored routines also enable you to have libraries of functions in the database server. This is a feature shared by modern application languages that enable such design internally (for example, by using classes). Using these client application language features is beneficial for the programmer even outside the scope of database use.

MySQL follows the SQL:2003 syntax for stored routines, which is also used by IBM's DB2. All syntax described here is supported and any limitations and extensions are documented where appropriate.

Additional Resources

- You may find the Stored Procedures User Forum of use when working with stored procedures and functions.
- For answers to some commonly asked questions regarding stored routines in MySQL, see Section A.4, “MySQL 5.6 FAQ: Stored Procedures and Functions”.
- There are some restrictions on the use of stored routines. See Section 20.8, “Restrictions on Stored Programs”.

- Binary logging for stored routines takes place as described in Section 20.7, “Stored Program Binary Logging”.

20.2.1 Stored Routine Syntax

A stored routine is either a procedure or a function. Stored routines are created with the `CREATE PROCEDURE` and `CREATE FUNCTION` statements (see Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”). A procedure is invoked using a `CALL` statement (see Section 13.2.1, “CALL Statement”), and can only pass back values using output variables. A function can be called from inside a statement just like any other function (that is, by invoking the function's name), and can return a scalar value. The body of a stored routine can use compound statements (see Section 13.6, “Compound Statements”).

Stored routines can be dropped with the `DROP PROCEDURE` and `DROP FUNCTION` statements (see Section 13.1.26, “DROP PROCEDURE and DROP FUNCTION Statements”), and altered with the `ALTER PROCEDURE` and `ALTER FUNCTION` statements (see Section 13.1.5, “ALTER PROCEDURE Statement”).

A stored procedure or function is associated with a particular database. This has several implications:

- When the routine is invoked, an implicit `USE` *`db_name`* is performed (and undone when the routine terminates). `USE` statements within stored routines are not permitted.
- You can qualify routine names with the database name. This can be used to refer to a routine that is not in the current database. For example, to invoke a stored procedure `p` or function `f` that is associated with the `test` database, you can say `CALL test.p()` or `test.f()`.
- When a database is dropped, all stored routines associated with it are dropped as well.

Stored functions cannot be recursive.

Recursion in stored procedures is permitted but disabled by default. To enable recursion, set the `max_sp_recursion_depth` server system variable to a value greater than zero. Stored procedure recursion increases the demand on thread stack space. If you increase the value of `max_sp_recursion_depth`, it may be necessary to increase thread stack size by increasing the value of `thread_stack` at server startup. See Section 5.1.7, “Server System Variables”, for more information.

MySQL supports a very useful extension that enables the use of regular `SELECT` statements (that is, without using cursors or local variables) inside a stored procedure. The result set of such a query is simply sent directly to the client. Multiple `SELECT` statements generate multiple result sets, so the client must use a MySQL client library that supports multiple result sets. This means the client must use a client library from a version of MySQL at least as recent as 4.1. The client should also specify the `CLIENT_MULTI_RESULTS` option when it connects. For C programs, this can be done with the `mysql_real_connect()` C API function. See mysql_real_connect(), and C API Multiple Statement Execution Support.

20.2.2 Stored Routines and MySQL Privileges

The MySQL grant system takes stored routines into account as follows:

- The `CREATE ROUTINE` privilege is needed to create stored routines.
- The `ALTER ROUTINE` privilege is needed to alter or drop stored routines. This privilege is granted automatically to the creator of a routine if necessary, and dropped from the creator when the routine is dropped.
- The `EXECUTE` privilege is required to execute stored routines. However, this privilege is granted automatically to the creator of a routine if necessary (and dropped from the creator when the routine is dropped). Also, the default `SQL SECURITY` characteristic for a routine is `DEFINER`, which enables users who have access to the database with which the routine is associated to execute the routine.

- If the `automatic_sp_privileges` system variable is 0, the `EXECUTE` and `ALTER ROUTINE` privileges are not automatically granted to and dropped from the routine creator.
- The creator of a routine is the account used to execute the `CREATE` statement for it. This might not be the same as the account named as the `DEFINER` in the routine definition.

The server manipulates the `mysql.proc` table in response to statements that create, alter, or drop stored routines. It is not supported that the server notices manual manipulation of this table.

20.2.3 Stored Routine Metadata

To obtain metadata about stored routines:

- Query the `ROUTINES` table of the `INFORMATION_SCHEMA` database. See Section 21.19, "The INFORMATION_SCHEMA ROUTINES Table".
- Use the `SHOW CREATE PROCEDURE` and `SHOW CREATE FUNCTION` statements to see routine definitions. See Section 13.7.5.11, "SHOW CREATE PROCEDURE Statement".
- Use the `SHOW PROCEDURE STATUS` and `SHOW FUNCTION STATUS` statements to see routine characteristics. See Section 13.7.5.29, "SHOW PROCEDURE STATUS Statement".
- Use the `SHOW PROCEDURE CODE` and `SHOW FUNCTION CODE` statements to see a representation of the internal implementation of the routine. See Section 13.7.5.28, "SHOW PROCEDURE CODE Statement".

20.2.4 Stored Procedures, Functions, Triggers, and LAST_INSERT_ID()

Within the body of a stored routine (procedure or function) or a trigger, the value of `LAST_INSERT_ID()` changes the same way as for statements executed outside the body of these kinds of objects (see Section 12.16, "Information Functions"). The effect of a stored routine or trigger upon the value of `LAST_INSERT_ID()` that is seen by following statements depends on the kind of routine:

- If a stored procedure executes statements that change the value of `LAST_INSERT_ID()`, the changed value is seen by statements that follow the procedure call.
- For stored functions and triggers that change the value, the value is restored when the function or trigger ends, so following statements do not see a changed value.

20.3 Using Triggers

A trigger is a named database object that is associated with a table, and that activates when a particular event occurs for the table. Some uses for triggers are to perform checks of values to be inserted into a table or to perform calculations on values involved in an update.

A trigger is defined to activate when a statement inserts, updates, or deletes rows in the associated table. These row operations are trigger events. For example, rows can be inserted by `INSERT` or `LOAD DATA` statements, and an insert trigger activates for each inserted row. A trigger can be set to activate either before or after the trigger event. For example, you can have a trigger activate before each row that is inserted into a table or after each row that is updated.

Important

MySQL triggers activate only for changes made to tables by SQL statements. This includes changes to base tables that underlie updatable views. Triggers do not activate for changes to tables made by APIs that do not transmit SQL statements to the MySQL Server. This means that triggers are not activated by updates made using the `NDB` API.

> Triggers are not activated by changes in `INFORMATION_SCHEMA` or `performance_schema` tables. Those tables are actually views and triggers are not permitted on views.

The following sections describe the syntax for creating and dropping triggers, show some examples of how to use them, and indicate how to obtain trigger metadata.

Additional Resources

- You may find the Triggers User Forum of use when working with triggers.
- For answers to commonly asked questions regarding triggers in MySQL, see Section A.5, “MySQL 5.6 FAQ: Triggers”.
- There are some restrictions on the use of triggers; see Section 20.8, “Restrictions on Stored Programs”.
- Binary logging for triggers takes place as described in Section 20.7, “Stored Program Binary Logging”.

20.3.1 Trigger Syntax and Examples

To create a trigger or drop a trigger, use the `CREATE TRIGGER` or `DROP TRIGGER` statement, described in Section 13.1.19, “CREATE TRIGGER Statement”, and Section 13.1.30, “DROP TRIGGER Statement”.

Here is a simple example that associates a trigger with a table, to activate for `INSERT` operations. The trigger acts as an accumulator, summing the values inserted into one of the columns of the table.

```
mysql> CREATE TABLE account (acct_num INT, amount DECIMAL(10,2));
Query OK, 0 rows affected (0.03 sec)

mysql> CREATE TRIGGER ins_sum BEFORE INSERT ON account
       FOR EACH ROW SET @sum = @sum + NEW.amount;
Query OK, 0 rows affected (0.01 sec)
```

The `CREATE TRIGGER` statement creates a trigger named `ins_sum` that is associated with the `account` table. It also includes clauses that specify the trigger action time, the triggering event, and what to do when the trigger activates:

- The keyword `BEFORE` indicates the trigger action time. In this case, the trigger activates before each row inserted into the table. The other permitted keyword here is `AFTER`.
- The keyword `INSERT` indicates the trigger event; that is, the type of operation that activates the trigger. In the example, `INSERT` operations cause trigger activation. You can also create triggers for `DELETE` and `UPDATE` operations.
- The statement following `FOR EACH ROW` defines the trigger body; that is, the statement to execute each time the trigger activates, which occurs once for each row affected by the triggering event. In the example, the trigger body is a simple `SET` that accumulates into a user variable the values inserted into the `amount` column. The statement refers to the column as `NEW.amount` which means “the value of the `amount` column to be inserted into the new row.”

To use the trigger, set the accumulator variable to zero, execute an `INSERT` statement, and then see what value the variable has afterward:

```
mysql> SET @sum = 0;
mysql> INSERT INTO account VALUES(137,14.98),(141,1937.50),(97,-100.00);
mysql> SELECT @sum AS 'Total amount inserted';
+-----------------------+
| Total amount inserted |
+-----------------------+
|               1852.48 |
```

```
+----------------------+
```

In this case, the value of `@sum` after the `INSERT` statement has executed is `14.98 + 1937.50 - 100`, or `1852.48`.

To destroy the trigger, use a `DROP TRIGGER` statement. You must specify the schema name if the trigger is not in the default schema:

```
mysql> DROP TRIGGER test.ins_sum;
```

If you drop a table, any triggers for the table are also dropped.

Trigger names exist in the schema namespace, meaning that all triggers must have unique names within a schema. Triggers in different schemas can have the same name.

In addition to the requirement that trigger names be unique for a schema, there are other limitations on the types of triggers you can create. In particular, there cannot be multiple triggers for a given table that have the same trigger event and action time. For example, you cannot have two `BEFORE UPDATE` triggers for a table. To work around this, you can define a trigger that executes multiple statements by using the `BEGIN ... END` compound statement construct after `FOR EACH ROW`. (An example appears later in this section.)

Within the trigger body, the `OLD` and `NEW` keywords enable you to access columns in the rows affected by a trigger. `OLD` and `NEW` are MySQL extensions to triggers; they are not case-sensitive.

In an `INSERT` trigger, only `NEW.col_name` can be used; there is no old row. In a `DELETE` trigger, only `OLD.col_name` can be used; there is no new row. In an `UPDATE` trigger, you can use `OLD.col_name` to refer to the columns of a row before it is updated and `NEW.col_name` to refer to the columns of the row after it is updated.

A column named with `OLD` is read only. You can refer to it (if you have the `SELECT` privilege), but not modify it. You can refer to a column named with `NEW` if you have the `SELECT` privilege for it. In a `BEFORE` trigger, you can also change its value with `SET NEW.col_name = value` if you have the `UPDATE` privilege for it. This means you can use a trigger to modify the values to be inserted into a new row or used to update a row. (Such a `SET` statement has no effect in an `AFTER` trigger because the row change will have already occurred.)

In a `BEFORE` trigger, the `NEW` value for an `AUTO_INCREMENT` column is 0, not the sequence number that is generated automatically when the new row actually is inserted.

By using the `BEGIN ... END` construct, you can define a trigger that executes multiple statements. Within the `BEGIN` block, you also can use other syntax that is permitted within stored routines such as conditionals and loops. However, just as for stored routines, if you use the `mysql` program to define a trigger that executes multiple statements, it is necessary to redefine the `mysql` statement delimiter so that you can use the `;` statement delimiter within the trigger definition. The following example illustrates these points. It defines an `UPDATE` trigger that checks the new value to be used for updating each row, and modifies the value to be within the range from 0 to 100. This must be a `BEFORE` trigger because the value must be checked before it is used to update the row:

```
mysql> delimiter //
mysql> CREATE TRIGGER upd_check BEFORE UPDATE ON account
    -> FOR EACH ROW
    -> BEGIN
    ->     IF NEW.amount < 0 THEN
    ->         SET NEW.amount = 0;
    ->     ELSEIF NEW.amount > 100 THEN
    ->         SET NEW.amount = 100;
    ->     END IF;
    -> END;//
mysql> delimiter ;
```

It can be easier to define a stored procedure separately and then invoke it from the trigger using a simple `CALL` statement. This is also advantageous if you want to execute the same code from within several triggers.

There are limitations on what can appear in statements that a trigger executes when activated:

- The trigger cannot use the `CALL` statement to invoke stored procedures that return data to the client or that use dynamic SQL. (Stored procedures are permitted to return data to the trigger through `OUT` or `INOUT` parameters.)
- The trigger cannot use statements that explicitly or implicitly begin or end a transaction, such as `START TRANSACTION`, `COMMIT`, or `ROLLBACK`. (`ROLLBACK to SAVEPOINT` is permitted because it does not end a transaction.).

See also Section 20.8, “Restrictions on Stored Programs”.

MySQL handles errors during trigger execution as follows:

- If a `BEFORE` trigger fails, the operation on the corresponding row is not performed.
- A `BEFORE` trigger is activated by the *attempt* to insert or modify the row, regardless of whether the attempt subsequently succeeds.
- An `AFTER` trigger is executed only if any `BEFORE` triggers and the row operation execute successfully.
- An error during either a `BEFORE` or `AFTER` trigger results in failure of the entire statement that caused trigger invocation.
- For transactional tables, failure of a statement should cause rollback of all changes performed by the statement. Failure of a trigger causes the statement to fail, so trigger failure also causes rollback. For nontransactional tables, such rollback cannot be done, so although the statement fails, any changes performed prior to the point of the error remain in effect.

Triggers can contain direct references to tables by name, such as the trigger named `testref` shown in this example:

```
CREATE TABLE test1(a1 INT);
CREATE TABLE test2(a2 INT);
CREATE TABLE test3(a3 INT NOT NULL AUTO_INCREMENT PRIMARY KEY);
CREATE TABLE test4(
  a4 INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
  b4 INT DEFAULT 0
);

delimiter |

CREATE TRIGGER testref BEFORE INSERT ON test1
  FOR EACH ROW
  BEGIN
    INSERT INTO test2 SET a2 = NEW.a1;
    DELETE FROM test3 WHERE a3 = NEW.a1;
    UPDATE test4 SET b4 = b4 + 1 WHERE a4 = NEW.a1;
  END;
|

delimiter ;

INSERT INTO test3 (a3) VALUES
  (NULL), (NULL), (NULL), (NULL), (NULL),
  (NULL), (NULL), (NULL), (NULL), (NULL);

INSERT INTO test4 (a4) VALUES
  (0), (0), (0), (0), (0), (0), (0), (0), (0), (0);
```

Suppose that you insert the following values into table `test1` as shown here:

```
mysql> INSERT INTO test1 VALUES
       (1), (3), (1), (7), (1), (8), (4), (4);
Query OK, 8 rows affected (0.01 sec)
Records: 8  Duplicates: 0  Warnings: 0
```

As a result, the four tables contain the following data:

```
mysql> SELECT * FROM test1;
+------+
| a1   |
+------+
|    1 |
|    3 |
|    1 |
|    7 |
|    1 |
|    8 |
|    4 |
|    4 |
+------+
8 rows in set (0.00 sec)

mysql> SELECT * FROM test2;
+------+
| a2   |
+------+
|    1 |
|    3 |
|    1 |
|    7 |
|    1 |
|    8 |
|    4 |
|    4 |
+------+
8 rows in set (0.00 sec)

mysql> SELECT * FROM test3;
+----+
| a3 |
+----+
|  2 |
|  5 |
|  6 |
|  9 |
| 10 |
+----+
5 rows in set (0.00 sec)

mysql> SELECT * FROM test4;
+----+------+
| a4 | b4   |
+----+------+
|  1 |    3 |
|  2 |    0 |
|  3 |    1 |
|  4 |    2 |
|  5 |    0 |
|  6 |    0 |
|  7 |    1 |
|  8 |    1 |
|  9 |    0 |
| 10 |    0 |
+----+------+
10 rows in set (0.00 sec)
```

20.3.2 Trigger Metadata

To obtain metadata about triggers:

- Query the `TRIGGERS` table of the `INFORMATION_SCHEMA` database. See Section 21.27, “The INFORMATION_SCHEMA TRIGGERS Table”.
- Use the `SHOW CREATE TRIGGER` statement. See Section 13.7.5.13, “SHOW CREATE TRIGGER Statement”.

- Use the `SHOW TRIGGERS` statement. See Section 13.7.5.39, “SHOW TRIGGERS Statement”.

20.4 Using the Event Scheduler

The *MySQL Event Scheduler* manages the scheduling and execution of events, that is, tasks that run according to a schedule. The following discussion covers the Event Scheduler and is divided into the following sections:

- Section 20.4.1, “Event Scheduler Overview”, provides an introduction to and conceptual overview of MySQL Events.
- Section 20.4.3, “Event Syntax”, discusses the SQL statements for creating, altering, and dropping MySQL Events.
- Section 20.4.4, “Event Metadata”, shows how to obtain information about events and how this information is stored by the MySQL Server.
- Section 20.4.6, “The Event Scheduler and MySQL Privileges”, discusses the privileges required to work with events and the ramifications that events have with regard to privileges when executing.

Stored routines require the `event` table in the `mysql` database. This table is created during the MySQL 5.6 installation procedure. If you are upgrading to MySQL 5.6 from an earlier version, be sure to update your grant tables to make sure that the `event` table exists. See Section 2.11, “Upgrading MySQL”.

Additional Resources

- You may find the MySQL Event Scheduler User Forum of use when working with scheduled events.
- There are some restrictions on the use of events; see Section 20.8, “Restrictions on Stored Programs”.
- Binary logging for events takes place as described in Section 20.7, “Stored Program Binary Logging”.

20.4.1 Event Scheduler Overview

MySQL Events are tasks that run according to a schedule. Therefore, we sometimes refer to them as *scheduled* events. When you create an event, you are creating a named database object containing one or more SQL statements to be executed at one or more regular intervals, beginning and ending at a specific date and time. Conceptually, this is similar to the idea of the Unix `crontab` (also known as a “cron job”) or the Windows Task Scheduler.

Scheduled tasks of this type are also sometimes known as “temporal triggers”, implying that these are objects that are triggered by the passage of time. While this is essentially correct, we prefer to use the term *events* to avoid confusion with triggers of the type discussed in Section 20.3, “Using Triggers”. Events should more specifically not be confused with “temporary triggers”. Whereas a trigger is a database object whose statements are executed in response to a specific type of event that occurs on a given table, a (scheduled) event is an object whose statements are executed in response to the passage of a specified time interval.

While there is no provision in the SQL Standard for event scheduling, there are precedents in other database systems, and you may notice some similarities between these implementations and that found in the MySQL Server.

MySQL Events have the following major features and properties:

- In MySQL, an event is uniquely identified by its name and the schema to which it is assigned.
- An event performs a specific action according to a schedule. This action consists of an SQL statement, which can be a compound statement in a `BEGIN ... END` block if desired (see

Section 13.6, “Compound Statements”). An event's timing can be either *one-time* or *recurrent*. A one-time event executes one time only. A recurrent event repeats its action at a regular interval, and the schedule for a recurring event can be assigned a specific start day and time, end day and time, both, or neither. (By default, a recurring event's schedule begins as soon as it is created, and continues indefinitely, until it is disabled or dropped.)

If a repeating event does not terminate within its scheduling interval, the result may be multiple instances of the event executing simultaneously. If this is undesirable, you should institute a mechanism to prevent simultaneous instances. For example, you could use the `GET_LOCK()` function, or row or table locking.

- Users can create, modify, and drop scheduled events using SQL statements intended for these purposes. Syntactically invalid event creation and modification statements fail with an appropriate error message. *A user may include statements in an event's action which require privileges that the user does not actually have*. The event creation or modification statement succeeds but the event's action fails. See Section 20.4.6, “The Event Scheduler and MySQL Privileges” for details.

- Many of the properties of an event can be set or modified using SQL statements. These properties include the event's name, timing, persistence (that is, whether it is preserved following the expiration of its schedule), status (enabled or disabled), action to be performed, and the schema to which it is assigned. See Section 13.1.2, “ALTER EVENT Statement”.

  The default definer of an event is the user who created the event, unless the event has been altered, in which case the definer is the user who issued the last `ALTER EVENT` statement affecting that event. An event can be modified by any user having the `EVENT` privilege on the database for which the event is defined. See Section 20.4.6, “The Event Scheduler and MySQL Privileges”.

- An event's action statement may include most SQL statements permitted within stored routines. For restrictions, see Section 20.8, “Restrictions on Stored Programs”.

20.4.2 Event Scheduler Configuration

Events are executed by a special *event scheduler thread*; when we refer to the Event Scheduler, we actually refer to this thread. When running, the event scheduler thread and its current state can be seen by users having the `PROCESS` privilege in the output of `SHOW PROCESSLIST`, as shown in the discussion that follows.

The global `event_scheduler` system variable determines whether the Event Scheduler is enabled and running on the server. It has one of these 3 values, which affect event scheduling as described here:

- `OFF`: The Event Scheduler is stopped. The event scheduler thread does not run, is not shown in the output of `SHOW PROCESSLIST`, and no scheduled events are executed. `OFF` is the default value for `event_scheduler`.

  When the Event Scheduler is stopped (`event_scheduler` is `OFF`), it can be started by setting the value of `event_scheduler` to `ON`. (See next item.)

- `ON`: The Event Scheduler is started; the event scheduler thread runs and executes all scheduled events.

  When the Event Scheduler is `ON`, the event scheduler thread is listed in the output of `SHOW PROCESSLIST` as a daemon process, and its state is represented as shown here:

```
mysql> SHOW PROCESSLIST\G
*************************** 1. row ***************************
     Id: 1
   User: root
   Host: localhost
     db: NULL
Command: Query
   Time: 0
```

```
  State: NULL
   Info: show processlist
*************************** 2. row ***************************
     Id: 2
   User: event_scheduler
   Host: localhost
     db: NULL
Command: Daemon
   Time: 3
  State: Waiting for next activation
   Info: NULL
2 rows in set (0.00 sec)
```

Event scheduling can be stopped by setting the value of `event_scheduler` to `OFF`.

- `DISABLED`: This value renders the Event Scheduler nonoperational. When the Event Scheduler is `DISABLED`, the event scheduler thread does not run (and so does not appear in the output of `SHOW PROCESSLIST`). In addition, the Event Scheduler state cannot be changed at runtime.

If the Event Scheduler status has not been set to `DISABLED`, `event_scheduler` can be toggled between `ON` and `OFF` (using `SET`). It is also possible to use `0` for `OFF`, and `1` for `ON` when setting this variable. Thus, any of the following 4 statements can be used in the `mysql` client to turn on the Event Scheduler:

```
SET GLOBAL event_scheduler = ON;
SET @@GLOBAL.event_scheduler = ON;
SET GLOBAL event_scheduler = 1;
SET @@GLOBAL.event_scheduler = 1;
```

Similarly, any of these 4 statements can be used to turn off the Event Scheduler:

```
SET GLOBAL event_scheduler = OFF;
SET @@GLOBAL.event_scheduler = OFF;
SET GLOBAL event_scheduler = 0;
SET @@GLOBAL.event_scheduler = 0;
```

Although `ON` and `OFF` have numeric equivalents, the value displayed for `event_scheduler` by `SELECT` or `SHOW VARIABLES` is always one of `OFF`, `ON`, or `DISABLED`. *`DISABLED` has no numeric equivalent*. For this reason, `ON` and `OFF` are usually preferred over `1` and `0` when setting this variable.

Attempting to set `event_scheduler` without specifying it as a global variable causes an error:

```
mysql< SET @@event_scheduler = OFF;
ERROR 1229 (HY000): Variable 'event_scheduler' is a GLOBAL
variable and should be set with SET GLOBAL
```

Important

It is possible to set the Event Scheduler to `DISABLED` only at server startup. If `event_scheduler` is `ON` or `OFF`, you cannot set it to `DISABLED` at runtime. Also, if the Event Scheduler is set to `DISABLED` at startup, you cannot change the value of `event_scheduler` at runtime.

To disable the event scheduler, use one of the following two methods:

- As a command-line option when starting the server:

```
--event-scheduler=DISABLED
```

- In the server configuration file (`my.cnf`, or `my.ini` on Windows systems), include the line where it is read by the server (for example, in a `[mysqld]` section):

```
event_scheduler=DISABLED
```

To enable the Event Scheduler, restart the server without the `--event-scheduler=DISABLED` command-line option, or after removing or commenting out the line containing `event-`

`scheduler=DISABLED` in the server configuration file, as appropriate. Alternatively, you can use `ON` (or `1`) or `OFF` (or `0`) in place of the `DISABLED` value when starting the server.

Note

You can issue event-manipulation statements when `event_scheduler` is set to `DISABLED`. No warnings or errors are generated in such cases (provided that the statements are themselves valid). However, scheduled events cannot execute until this variable is set to `ON` (or `1`). Once this has been done, the event scheduler thread executes all events whose scheduling conditions are satisfied.

Starting the MySQL server with the `--skip-grant-tables` option causes `event_scheduler` to be set to `DISABLED`, overriding any other value set either on the command line or in the `my.cnf` or `my.ini` file (Bug #26807).

For SQL statements used to create, alter, and drop events, see Section 20.4.3, “Event Syntax”.

MySQL provides an `EVENTS` table in the `INFORMATION_SCHEMA` database. This table can be queried to obtain information about scheduled events which have been defined on the server. See Section 20.4.4, “Event Metadata”, and Section 21.8, “The INFORMATION_SCHEMA EVENTS Table”, for more information.

For information regarding event scheduling and the MySQL privilege system, see Section 20.4.6, “The Event Scheduler and MySQL Privileges”.

20.4.3 Event Syntax

MySQL provides several SQL statements for working with scheduled events:

- New events are defined using the `CREATE EVENT` statement. See Section 13.1.11, “CREATE EVENT Statement”.
- The definition of an existing event can be changed by means of the `ALTER EVENT` statement. See Section 13.1.2, “ALTER EVENT Statement”.
- When a scheduled event is no longer wanted or needed, it can be deleted from the server by its definer using the `DROP EVENT` statement. See Section 13.1.22, “DROP EVENT Statement”. Whether an event persists past the end of its schedule also depends on its `ON COMPLETION` clause, if it has one. See Section 13.1.11, “CREATE EVENT Statement”.

  An event can be dropped by any user having the `EVENT` privilege for the database on which the event is defined. See Section 20.4.6, “The Event Scheduler and MySQL Privileges”.

20.4.4 Event Metadata

To obtain metadata about events:

- Query the `event` table of the `mysql` database.
- Query the `EVENTS` table of the `INFORMATION_SCHEMA` database. See Section 21.8, “The INFORMATION_SCHEMA EVENTS Table”.
- Use the `SHOW CREATE EVENT` statement. See Section 13.7.5.9, “SHOW CREATE EVENT Statement”.
- Use the `SHOW EVENTS` statement. See Section 13.7.5.19, “SHOW EVENTS Statement”.

Event Scheduler Time Representation

Each session in MySQL has a session time zone (STZ). This is the session `time_zone` value that is initialized from the server's global `time_zone` value when the session begins but may be changed during the session.

The session time zone that is current when a `CREATE EVENT` or `ALTER EVENT` statement executes is used to interpret times specified in the event definition. This becomes the event time zone (ETZ); that is, the time zone that is used for event scheduling and is in effect within the event as it executes.

For representation of event information in the `mysql.event` table, the `execute_at`, `starts`, and `ends` times are converted to UTC and stored along with the event time zone. This enables event execution to proceed as defined regardless of any subsequent changes to the server time zone or daylight saving time effects. The `last_executed` time is also stored in UTC.

If you select information from `mysql.event`, the times just mentioned are retrieved as UTC values. These times can also be obtained by selecting from the `INFORMATION_SCHEMA.EVENTS` table or from `SHOW EVENTS`, but they are reported as ETZ values. Other times available from these sources indicate when an event was created or last altered; these are displayed as STZ values. The following table summarizes representation of event times.

Value	`mysql.event`	`INFORMATION_SCHEMA.EVENTS`	`SHOW EVENTS`
Execute at	UTC	ETZ	ETZ
Starts	UTC	ETZ	ETZ
Ends	UTC	ETZ	ETZ
Last executed	UTC	ETZ	n/a
Created	STZ	STZ	n/a
Last altered	STZ	STZ	n/a

20.4.5 Event Scheduler Status

The Event Scheduler writes information about event execution that terminates with an error or warning to the MySQL Server's error log. See Section 20.4.6, “The Event Scheduler and MySQL Privileges” for an example.

To obtain information about the state of the Event Scheduler for debugging and troubleshooting purposes, run `mysqladmin debug` (see Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”); after running this command, the server's error log contains output relating to the Event Scheduler, similar to what is shown here:

```
Events status:
LLA = Last Locked At  LUA = Last Unlocked At
WOC = Waiting On Condition  DL = Data Locked

Event scheduler status:
State      : INITIALIZED
Thread id  : 0
LLA        : n/a:0
LUA        : n/a:0
WOC        : NO
Workers    : 0
Executed   : 0
Data locked: NO

Event queue status:
Element count   : 0
Data locked     : NO
Attempting lock : NO
LLA             : init_queue:95
LUA             : init_queue:103
WOC             : NO
Next activation : never
```

In statements that occur as part of events executed by the Event Scheduler, diagnostics messages (not only errors, but also warnings) are written to the error log, and, on Windows, to the application event log. For frequently executed events, it is possible for this to result in many logged messages. For example, for `SELECT ... INTO var_list` statements, if the query returns no rows, a warning

with error code 1329 occurs (`No data`), and the variable values remain unchanged. If the query returns multiple rows, error 1172 occurs (`Result consisted of more than one row`). For either condition, you can avoid having the warnings be logged by declaring a condition handler; see Section 13.6.7.2, “DECLARE ... HANDLER Statement”. For statements that may retrieve multiple rows, another strategy is to use `LIMIT 1` to limit the result set to a single row.

20.4.6 The Event Scheduler and MySQL Privileges

To enable or disable the execution of scheduled events, it is necessary to set the value of the global `event_scheduler` system variable. This requires privileges sufficient to set global system variables. See Section 5.1.8.1, “System Variable Privileges”.

The `EVENT` privilege governs the creation, modification, and deletion of events. This privilege can be bestowed using `GRANT`. For example, this `GRANT` statement confers the `EVENT` privilege for the schema named `myschema` on the user `jon@ghidora`:

```
GRANT EVENT ON myschema.* TO jon@ghidora;
```

(We assume that this user account already exists, and that we wish for it to remain unchanged otherwise.)

To grant this same user the `EVENT` privilege on all schemas, use the following statement:

```
GRANT EVENT ON *.* TO jon@ghidora;
```

The `EVENT` privilege has global or schema-level scope. Therefore, trying to grant it on a single table results in an error as shown:

```
mysql> GRANT EVENT ON myschema.mytable TO jon@ghidora;
ERROR 1144 (42000): Illegal GRANT/REVOKE command; please
consult the manual to see which privileges can be used
```

It is important to understand that an event is executed with the privileges of its definer, and that it cannot perform any actions for which its definer does not have the requisite privileges. For example, suppose that `jon@ghidora` has the `EVENT` privilege for `myschema`. Suppose also that this user has the `SELECT` privilege for `myschema`, but no other privileges for this schema. It is possible for `jon@ghidora` to create a new event such as this one:

```
CREATE EVENT e_store_ts
    ON SCHEDULE
      EVERY 10 SECOND
    DO
      INSERT INTO myschema.mytable VALUES (UNIX_TIMESTAMP());
```

The user waits for a minute or so, and then performs a `SELECT * FROM mytable;` query, expecting to see several new rows in the table. Instead, the table is empty. Since the user does not have the `INSERT` privilege for the table in question, the event has no effect.

If you inspect the MySQL error log (*`hostname`*`.err`), you can see that the event is executing, but the action it is attempting to perform fails, as indicated by `RetCode=0`:

```
060209 22:39:44 [Note]     EVEX EXECUTING event newdb.e [EXPR:10]
060209 22:39:44 [Note]     EVEX EXECUTED event newdb.e  [EXPR:10]. RetCode=0
060209 22:39:54 [Note]     EVEX EXECUTING event newdb.e [EXPR:10]
060209 22:39:54 [Note]     EVEX EXECUTED event newdb.e  [EXPR:10]. RetCode=0
060209 22:40:04 [Note]     EVEX EXECUTING event newdb.e [EXPR:10]
060209 22:40:04 [Note]     EVEX EXECUTED event newdb.e  [EXPR:10]. RetCode=0
```

Since this user very likely does not have access to the error log, it is possible to verify whether the event's action statement is valid by executing it directly:

```
mysql> INSERT INTO myschema.mytable VALUES (UNIX_TIMESTAMP());
ERROR 1142 (42000): INSERT command denied to user
'jon'@'ghidora' for table 'mytable'
```

Inspection of the `INFORMATION_SCHEMA.EVENTS` table shows that `e_store_ts` exists and is enabled, but its `LAST_EXECUTED` column is `NULL`:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.EVENTS
     >     WHERE EVENT_NAME='e_store_ts'
     >     AND EVENT_SCHEMA='myschema'\G
*************************** 1. row ***************************
   EVENT_CATALOG: NULL
    EVENT_SCHEMA: myschema
      EVENT_NAME: e_store_ts
         DEFINER: jon@ghidora
      EVENT_BODY: SQL
EVENT_DEFINITION: INSERT INTO myschema.mytable VALUES (UNIX_TIMESTAMP())
      EVENT_TYPE: RECURRING
      EXECUTE_AT: NULL
  INTERVAL_VALUE: 5
  INTERVAL_FIELD: SECOND
        SQL_MODE: NULL
          STARTS: 0000-00-00 00:00:00
            ENDS: 0000-00-00 00:00:00
          STATUS: ENABLED
   ON_COMPLETION: NOT PRESERVE
         CREATED: 2006-02-09 22:36:06
    LAST_ALTERED: 2006-02-09 22:36:06
   LAST_EXECUTED: NULL
   EVENT_COMMENT:
1 row in set (0.00 sec)
```

To rescind the `EVENT` privilege, use the `REVOKE` statement. In this example, the `EVENT` privilege on the schema `myschema` is removed from the `jon@ghidora` user account:

```
REVOKE EVENT ON myschema.* FROM jon@ghidora;
```

Important

Revoking the `EVENT` privilege from a user does not delete or disable any events that may have been created by that user.

An event is not migrated or dropped as a result of renaming or dropping the user who created it.

Suppose that the user `jon@ghidora` has been granted the `EVENT` and `INSERT` privileges on the `myschema` schema. This user then creates the following event:

```
CREATE EVENT e_insert
    ON SCHEDULE
      EVERY 7 SECOND
    DO
      INSERT INTO myschema.mytable;
```

After this event has been created, `root` revokes the `EVENT` privilege for `jon@ghidora`. However, `e_insert` continues to execute, inserting a new row into `mytable` each seven seconds. The same would be true if `root` had issued either of these statements:

- `DROP USER jon@ghidora;`
- `RENAME USER jon@ghidora TO someotherguy@ghidora;`

You can verify that this is true by examining the `mysql.event` table (discussed later in this section) or the `INFORMATION_SCHEMA.EVENTS` table (see Section 21.8, “The INFORMATION_SCHEMA EVENTS Table”) before and after issuing a `DROP USER` or `RENAME USER` statement.

Event definitions are stored in the `mysql.event` table. To drop an event created by another user account, the MySQL `root` user (or another user with the necessary privileges) can delete rows from this table. For example, to remove the event `e_insert` shown previously, `root` can use the following statement:

```
DELETE FROM mysql.event
    WHERE db = 'myschema'
      AND name = 'e_insert';
```

It is very important to match the event name and database schema name when deleting rows from the `mysql.event` table. This is because different events of the same name can exist in different schemas.

Users' `EVENT` privileges are stored in the `Event_priv` columns of the `mysql.user` and `mysql.db` tables. In both cases, this column holds one of the values '`Y`' or '`N`'. '`N`' is the default. `mysql.user.Event_priv` is set to '`Y`' for a given user only if that user has the global `EVENT` privilege (that is, if the privilege was bestowed using `GRANT EVENT ON *.*`). For a schema-level `EVENT` privilege, `GRANT` creates a row in `mysql.db` and sets that row's `Db` column to the name of the schema, the `User` column to the name of the user, and the `Event_priv` column to '`Y`'. There should never be any need to manipulate these tables directly, since the `GRANT EVENT` and `REVOKE EVENT` statements perform the required operations on them.

Five status variables provide counts of event-related operations (but *not* of statements executed by events; see Section 20.8, “Restrictions on Stored Programs”). These are:

- `Com_create_event`: The number of `CREATE EVENT` statements executed since the last server restart.
- `Com_alter_event`: The number of `ALTER EVENT` statements executed since the last server restart.
- `Com_drop_event`: The number of `DROP EVENT` statements executed since the last server restart.
- `Com_show_create_event`: The number of `SHOW CREATE EVENT` statements executed since the last server restart.
- `Com_show_events`: The number of `SHOW EVENTS` statements executed since the last server restart.

You can view current values for all of these at one time by running the statement `SHOW STATUS LIKE '%event%';`.

20.5 Using Views

MySQL supports views, including updatable views. Views are stored queries that when invoked produce a result set. A view acts as a virtual table.

To use views if you have upgraded to MySQL 5.6 from an older release that did not support views, you should upgrade your grant tables so that they contain the view-related privileges. See Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”.

The following discussion describes the syntax for creating and dropping views, and shows some examples of how to use them.

Additional Resources

- You may find the Views User Forum of use when working with views.
- For answers to some commonly asked questions regarding views in MySQL, see Section A.6, “MySQL 5.6 FAQ: Views”.
- There are some restrictions on the use of views; see Section 20.9, “Restrictions on Views”.

20.5.1 View Syntax

The `CREATE VIEW` statement creates a new view (see Section 13.1.20, “CREATE VIEW Statement”). To alter the definition of a view or drop a view, use `ALTER VIEW` (see Section 13.1.9, “ALTER VIEW Statement”), or `DROP VIEW` (see Section 13.1.31, “DROP VIEW Statement”).

A view can be created from many kinds of `SELECT` statements. It can refer to base tables or other views. It can use joins, `UNION`, and subqueries. The `SELECT` need not even refer to any tables. The following example defines a view that selects two columns from another table, as well as an expression calculated from those columns:

```
mysql> CREATE TABLE t (qty INT, price INT);
mysql> INSERT INTO t VALUES(3, 50), (5, 60);
mysql> CREATE VIEW v AS SELECT qty, price, qty*price AS value FROM t;
mysql> SELECT * FROM v;
+------+-------+-------+
| qty  | price | value |
+------+-------+-------+
|    3 |    50 |   150 |
|    5 |    60 |   300 |
+------+-------+-------+
mysql> SELECT * FROM v WHERE qty = 5;
+------+-------+-------+
| qty  | price | value |
+------+-------+-------+
|    5 |    60 |   300 |
+------+-------+-------+
```

20.5.2 View Processing Algorithms

The optional `ALGORITHM` clause for `CREATE VIEW` or `ALTER VIEW` is a MySQL extension to standard SQL. It affects how MySQL processes the view. `ALGORITHM` takes three values: `MERGE`, `TEMPTABLE`, or `UNDEFINED`.

- For `MERGE`, the text of a statement that refers to the view and the view definition are merged such that parts of the view definition replace corresponding parts of the statement.
- For `TEMPTABLE`, the results from the view are retrieved into a temporary table, which then is used to execute the statement.
- For `UNDEFINED`, MySQL chooses which algorithm to use. It prefers `MERGE` over `TEMPTABLE` if possible, because `MERGE` is usually more efficient and because a view cannot be updatable if a temporary table is used.
- If no `ALGORITHM` clause is present, `UNDEFINED` is the default algorithm.

A reason to specify `TEMPTABLE` explicitly is that locks can be released on underlying tables after the temporary table has been created and before it is used to finish processing the statement. This might result in quicker lock release than the `MERGE` algorithm so that other clients that use the view are not blocked as long.

A view algorithm can be `UNDEFINED` for three reasons:

- No `ALGORITHM` clause is present in the `CREATE VIEW` statement.
- The `CREATE VIEW` statement has an explicit `ALGORITHM = UNDEFINED` clause.
- `ALGORITHM = MERGE` is specified for a view that can be processed only with a temporary table. In this case, MySQL generates a warning and sets the algorithm to `UNDEFINED`.

As mentioned earlier, `MERGE` is handled by merging corresponding parts of a view definition into the statement that refers to the view. The following examples briefly illustrate how the `MERGE` algorithm works. The examples assume that there is a view `v_merge` that has this definition:

```
CREATE ALGORITHM = MERGE VIEW v_merge (vc1, vc2) AS
SELECT c1, c2 FROM t WHERE c3 > 100;
```

Example 1: Suppose that we issue this statement:

```
SELECT * FROM v_merge;
```

MySQL handles the statement as follows:

- `v_merge` becomes `t`
- `*` becomes `vc1, vc2`, which corresponds to `c1, c2`
- The view `WHERE` clause is added

The resulting statement to be executed becomes:

```
SELECT c1, c2 FROM t WHERE c3 > 100;
```

Example 2: Suppose that we issue this statement:

```
SELECT * FROM v_merge WHERE vc1 < 100;
```

This statement is handled similarly to the previous one, except that `vc1 < 100` becomes `c1 < 100` and the view `WHERE` clause is added to the statement `WHERE` clause using an `AND` connective (and parentheses are added to make sure the parts of the clause are executed with correct precedence). The resulting statement to be executed becomes:

```
SELECT c1, c2 FROM t WHERE (c3 > 100) AND (c1 < 100);
```

Effectively, the statement to be executed has a `WHERE` clause of this form:

```
WHERE (select WHERE) AND (view WHERE)
```

If the `MERGE` algorithm cannot be used, a temporary table must be used instead. `MERGE` cannot be used if the view contains any of the following constructs:

- Aggregate functions (`SUM()`, `MIN()`, `MAX()`, `COUNT()`, and so forth)
- `DISTINCT`
- `GROUP BY`
- `HAVING`
- `LIMIT`
- `UNION` or `UNION ALL`
- Subquery in the select list
- Assignment to user variables
- Refers only to literal values (in this case, there is no underlying table)

20.5.3 Updatable and Insertable Views

Some views are updatable and references to them can be used to specify tables to be updated in data change statements. That is, you can use them in statements such as `UPDATE`, `DELETE`, or `INSERT` to update the contents of the underlying table.

For a view to be updatable, there must be a one-to-one relationship between the rows in the view and the rows in the underlying table. There are also certain other constructs that make a view nonupdatable. To be more specific, a view is not updatable if it contains any of the following:

- Aggregate functions (`SUM()`, `MIN()`, `MAX()`, `COUNT()`, and so forth)
- `DISTINCT`
- `GROUP BY`
- `HAVING`
- `UNION` or `UNION ALL`
- Subquery in the select list

- Certain joins (see additional join discussion later in this section)
- Reference to nonupdatable view in the `FROM` clause
- Subquery in the `WHERE` clause that refers to a table in the `FROM` clause
- Refers only to literal values (in this case, there is no underlying table to update)
- `ALGORITHM = TEMPTABLE` (use of a temporary table always makes a view nonupdatable)
- Multiple references to any column of a base table

It is sometimes possible for a multiple-table view to be updatable, assuming that it can be processed with the `MERGE` algorithm. For this to work, the view must use an inner join (not an outer join or a `UNION`). Also, only a single table in the view definition can be updated, so the `SET` clause must name only columns from one of the tables in the view. Views that use `UNION ALL` are not permitted even though they might be theoretically updatable.

With respect to insertability (being updatable with `INSERT` statements), an updatable view is insertable if it also satisfies these additional requirements for the view columns:

- There must be no duplicate view column names.
- The view must contain all columns in the base table that do not have a default value.
- The view columns must be simple column references. They must not be expressions, such as these:

```
3.14159
col1 + 3
UPPER(col2)
col3 / col4
(subquery)
```

MySQL sets a flag, called the view updatability flag, at `CREATE VIEW` time. The flag is set to `YES` (true) if `UPDATE` and `DELETE` (and similar operations) are legal for the view. Otherwise, the flag is set to `NO` (false). The `IS_UPDATABLE` column in the `INFORMATION_SCHEMA.VIEWS` table displays the status of this flag. It means that the server always knows whether a view is updatable.

If a view is not updatable, statements such `UPDATE`, `DELETE`, and `INSERT` are illegal and are rejected. (Even if a view is updatable, it might not be possible to insert into it, as described elsewhere in this section.)

The updatability of views may be affected by the value of the `updatable_views_with_limit` system variable. See Section 5.1.7, “Server System Variables”.

Earlier discussion in this section pointed out that a view is not insertable if not all columns are simple column references (for example, if it contains columns that are expressions or composite expressions). Although such a view is not insertable, it can be updatable if you update only columns that are not expressions. Consider this view:

```
CREATE VIEW v AS SELECT col1, 1 AS col2 FROM t;
```

This view is not insertable because `col2` is an expression. But it is updatable if the update does not try to update `col2`. This update is permissible:

```
UPDATE v SET col1 = 0;
```

This update is not permissible because it attempts to update an expression column:

```
UPDATE v SET col2 = 0;
```

For a multiple-table updatable view, `INSERT` can work if it inserts into a single table. `DELETE` is not supported.

`INSERT DELAYED` is not supported for views.

If a table contains an AUTO_INCREMENT column, inserting into an insertable view on the table that does not include the AUTO_INCREMENT column does not change the value of LAST_INSERT_ID(), because the side effects of inserting default values into columns not part of the view should not be visible.

20.5.4 The View WITH CHECK OPTION Clause

The WITH CHECK OPTION clause can be given for an updatable view to prevent inserts to rows for which the WHERE clause in the *select_statement* is not true. It also prevents updates to rows for which the WHERE clause is true but the update would cause it to be not true (in other words, it prevents visible rows from being updated to nonvisible rows).

In a WITH CHECK OPTION clause for an updatable view, the LOCAL and CASCADED keywords determine the scope of check testing when the view is defined in terms of another view. When neither keyword is given, the default is CASCADED. The LOCAL keyword restricts the CHECK OPTION only to the view being defined. CASCADED causes the checks for underlying views to be evaluated as well.

Consider the definitions for the following table and set of views:

```
CREATE TABLE t1 (a INT);
CREATE VIEW v1 AS SELECT * FROM t1 WHERE a < 2
WITH CHECK OPTION;
CREATE VIEW v2 AS SELECT * FROM v1 WHERE a > 0
WITH LOCAL CHECK OPTION;
CREATE VIEW v3 AS SELECT * FROM v1 WHERE a > 0
WITH CASCADED CHECK OPTION;
```

Here the v2 and v3 views are defined in terms of another view, v1. v2 has a LOCAL check option, so inserts are tested only against the v2 check. v3 has a CASCADED check option, so inserts are tested not only against its own check, but against those of underlying views. The following statements illustrate these differences:

```
mysql> INSERT INTO v2 VALUES (2);
Query OK, 1 row affected (0.00 sec)
mysql> INSERT INTO v3 VALUES (2);
ERROR 1369 (HY000): CHECK OPTION failed 'test.v3'
```

20.5.5 View Metadata

To obtain metadata about views:

- Query the VIEWS table of the INFORMATION_SCHEMA database. See Section 21.29, “The INFORMATION_SCHEMA VIEWS Table”.
- Use the SHOW CREATE VIEW statement. See Section 13.7.5.14, “SHOW CREATE VIEW Statement”.

20.6 Stored Object Access Control

Stored programs (procedures, functions, triggers, and events) and views are defined prior to use and, when referenced, execute within a security context that determines their privileges. The privileges applicable to execution of a stored object are controlled by its DEFINER attribute and SQL SECURITY characteristic.

- The DEFINER Attribute
- The SQL SECURITY Characteristic
- Examples
- Orphan Stored Objects
- Risk-Minimization Guidelines

The DEFINER Attribute

A stored object definition can include a `DEFINER` attribute that names a MySQL account. If a definition omits the `DEFINER` attribute, the default object definer is the user who creates it.

The following rules determine which accounts you can specify as the `DEFINER` attribute for a stored object:

- If you have the `SUPER` privilege, you can specify any account as the `DEFINER` attribute. If the account does not exist, a warning is generated.
- Otherwise, the only permitted account is your own, specified either literally or as `CURRENT_USER` or `CURRENT_USER()`. You cannot set the definer to any other account.

Creating a stored object with a nonexistent `DEFINER` account creates an orphan object, which may have negative consequences; see Orphan Stored Objects.

The SQL SECURITY Characteristic

For stored routines (procedures and functions) and views, the object definition can include an `SQL SECURITY` characteristic with a value of `DEFINER` or `INVOKER` to specify whether the object executes in definer or invoker context. If the definition omits the `SQL SECURITY` characteristic, the default is definer context.

Triggers and events have no `SQL SECURITY` characteristic and always execute in definer context. The server invokes these objects automatically as necessary, so there is no invoking user.

Definer and invoker security contexts differ as follows:

- A stored object that executes in definer security context executes with the privileges of the account named by its `DEFINER` attribute. These privileges may be entirely different from those of the invoking user. The invoker must have appropriate privileges to reference the object (for example, `EXECUTE` to call a stored procedure or `SELECT` to select from a view), but during object execution, the invoker's privileges are ignored and only the `DEFINER` account privileges matter. If the `DEFINER` account has few privileges, the object is correspondingly limited in the operations it can perform. If the `DEFINER` account is highly privileged (such as an administrative account), the object can perform powerful operations *no matter who invokes it.*
- A stored routine or view that executes in invoker security context can perform only operations for which the invoker has privileges. The `DEFINER` attribute has no effect on object execution.

Examples

Consider the following stored procedure, which is declared with `SQL SECURITY DEFINER` to execute in definer security context:

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE p1()
SQL SECURITY DEFINER
BEGIN
  UPDATE t1 SET counter = counter + 1;
END;
```

Any user who has the `EXECUTE` privilege for `p1` can invoke it with a `CALL` statement. However, when `p1` executes, it does so in definer security context and thus executes with the privileges of `'admin'@'localhost'`, the account named as its `DEFINER` attribute. This account must have the `EXECUTE` privilege for `p1` as well as the `UPDATE` privilege for the table `t1` referenced within the object body. Otherwise, the procedure fails.

Now consider this stored procedure, which is identical to `p1` except that its `SQL SECURITY` characteristic is `INVOKER`:

```
CREATE DEFINER = 'admin'@'localhost' PROCEDURE p2()
```

```
SQL SECURITY INVOKER
BEGIN
  UPDATE t1 SET counter = counter + 1;
END;
```

Unlike `p1`, `p2` executes in invoker security context and thus with the privileges of the invoking user regardless of the `DEFINER` attribute value. `p2` fails if the invoker lacks the `EXECUTE` privilege for `p2` or the `UPDATE` privilege for the table `t1`.

Orphan Stored Objects

An orphan stored object is one for which its `DEFINER` attribute names a nonexistent account:

- An orphan stored object can be created by specifying a nonexistent `DEFINER` account at object-creation time.
- An existing stored object can become orphaned through execution of a `DROP USER` statement that drops the object `DEFINER` account, or a `RENAME USER` statement that renames the object `DEFINER` account.

An orphan stored object may be problematic in these ways:

- Because the `DEFINER` account does not exist, the object may not work as expected if it executes in definer security context:
  - For a stored routine, an error occurs at routine execution time if the `SQL SECURITY` value is `DEFINER` but the definer account does not exist.
  - For a trigger, it is not a good idea for trigger activation to occur until the account actually does exist. Otherwise, the behavior with respect to privilege checking is undefined.
  - For an event, an error occurs at event execution time if the account does not exist.
  - For a view, an error occurs when the view is referenced if the `SQL SECURITY` value is `DEFINER` but the definer account does not exist.
- The object may present a security risk if the nonexistent `DEFINER` account is subsequently re-created for a purpose unrelated to the object. In this case, the account “adopts” the object and, with the appropriate privileges, is able to execute it even if that is not intended.

To obtain information about the accounts used as stored object definers in a MySQL installation, query the `INFORMATION_SCHEMA`.

This query identifies which `INFORMATION_SCHEMA` tables describe objects that have a `DEFINER` attribute:

```
mysql> SELECT TABLE_SCHEMA, TABLE_NAME FROM INFORMATION_SCHEMA.COLUMNS
       WHERE COLUMN_NAME = 'DEFINER';
+--------------------+------------+
| TABLE_SCHEMA       | TABLE_NAME |
+--------------------+------------+
| information_schema | EVENTS     |
| information_schema | ROUTINES   |
| information_schema | TRIGGERS   |
| information_schema | VIEWS      |
+--------------------+------------+
```

The result tells you which tables to query to discover which stored object `DEFINER` values exist and which objects have a particular `DEFINER` value:

- To identify which `DEFINER` values exist in each table, use these queries:

```
SELECT DISTINCT DEFINER FROM INFORMATION_SCHEMA.EVENTS;
SELECT DISTINCT DEFINER FROM INFORMATION_SCHEMA.ROUTINES;
SELECT DISTINCT DEFINER FROM INFORMATION_SCHEMA.TRIGGERS;
```

```
SELECT DISTINCT DEFINER FROM INFORMATION_SCHEMA.VIEWS;
```

The query results are significant for any account displayed as follows:

- If the account exists, dropping or renaming it causes stored objects to become orphaned. If you plan to drop or rename the account, consider first dropping its associated stored objects or redefining them to have a different definer.
- If the account does not exist, creating it causes it to adopt currently orphaned stored objects. If you plan to create the account, consider whether the orphaned objects should be associated with it. If not, redefine them to have a different definer.

To redefine an object with a different definer, you can use `ALTER EVENT` or `ALTER VIEW` to directly modify the `DEFINER` account of events and views. For stored procedures and functions and for triggers, you must drop the object and re-create it with a different `DEFINER` account

- To identify which objects have a given `DEFINER` account, use these queries, substituting the account of interest for *`user_name@host_name`*:

```
SELECT EVENT_SCHEMA, EVENT_NAME FROM INFORMATION_SCHEMA.EVENTS
WHERE DEFINER = 'user_name@host_name';
SELECT ROUTINE_SCHEMA, ROUTINE_NAME, ROUTINE_TYPE
FROM INFORMATION_SCHEMA.ROUTINES
WHERE DEFINER = 'user_name@host_name';
SELECT TRIGGER_SCHEMA, TRIGGER_NAME FROM INFORMATION_SCHEMA.TRIGGERS
WHERE DEFINER = 'user_name@host_name';
SELECT TABLE_SCHEMA, TABLE_NAME FROM INFORMATION_SCHEMA.VIEWS
WHERE DEFINER = 'user_name@host_name';
```

For the `ROUTINES` table, the query includes the `ROUTINE_TYPE` column so that output rows distinguish whether the `DEFINER` is for a stored procedure or stored function.

If the account you are searching for does not exist, any objects displayed by those queries are orphan objects.

Risk-Minimization Guidelines

To minimize the risk potential for stored object creation and use, follow these guidelines:

- Do not create orphan stored objects; that is, objects for which the `DEFINER` attribute names a nonexistent account. Do not cause stored objects to become orphaned by dropping or renaming an account named by the `DEFINER` attribute of any existing object.
- For a stored routine or view, use `SQL SECURITY INVOKER` in the object definition when possible so that it can be used only by users with permissions appropriate for the operations performed by the object.
- If you create definer-context stored objects while using an account that has the `SUPER` privilege, specify an explicit `DEFINER` attribute that names an account possessing only the privileges required for the operations performed by the object. Specify a highly privileged `DEFINER` account only when absolutely necessary.
- Administrators can prevent users from creating stored objects that specify highly privileged `DEFINER` accounts by not granting them the `SUPER` privilege.
- Definer-context objects should be written keeping in mind that they may be able to access data for which the invoking user has no privileges. In some cases, you can prevent references to these objects by not granting unauthorized users particular privileges:
  - A stored routine cannot be referenced by a user who does not have the `EXECUTE` privilege for it.
  - A view cannot be referenced by a user who does not have the appropriate privilege for it (`SELECT` to select from it, `INSERT` to insert into it, and so forth).

However, no such control exists for triggers and events because they always execute in definer context. The server invokes these objects automatically as necessary, and users do not reference them directly:

- A trigger is activated by access to the table with which it is associated, even ordinary table accesses by users with no special privileges.
- An event is executed by the server on a scheduled basis.

In both cases, if the `DEFINER` account is highly privileged, the object may be able to perform sensitive or dangerous operations. This remains true if the privileges needed to create the object are revoked from the account of the user who created it. Administrators should be especially careful about granting users object-creation privileges.

20.7 Stored Program Binary Logging

The binary log contains information about SQL statements that modify database contents. This information is stored in the form of “events” that describe the modifications. (Binary log events differ from scheduled event stored objects.) The binary log has two important purposes:

- For replication, the binary log is used on source replication servers as a record of the statements to be sent to replica servers. The source sends the events contained in its binary log to its replicas, which execute those events to make the same data changes that were made on the source. See Section 17.2, “Replication Implementation”.
- Certain data recovery operations require use of the binary log. After a backup file has been restored, the events in the binary log that were recorded after the backup was made are re-executed. These events bring databases up to date from the point of the backup. See Section 7.3.2, “Using Backups for Recovery”.

However, if logging occurs at the statement level, there are certain binary logging issues with respect to stored programs (stored procedures and functions, triggers, and events):

- In some cases, a statement might affect different sets of rows on source and replica.
- Replicated statements executed on a replica are processed by the replica SQL thread, which has full privileges. It is possible for a procedure to follow different execution paths on source and replica servers, so a user can write a routine containing a dangerous statement that executes only on the replica where it is processed by a thread that has full privileges.
- If a stored program that modifies data is nondeterministic, it is not repeatable. This can result in different data on source and replica, or cause restored data to differ from the original data.

This section describes how MySQL handles binary logging for stored programs. It states the current conditions that the implementation places on the use of stored programs, and what you can do to avoid logging problems. It also provides additional information about the reasons for these conditions.

In general, the issues described here result when binary logging occurs at the SQL statement level (statement-based binary logging). If you use row-based binary logging, the log contains changes made to individual rows as a result of executing SQL statements. When routines or triggers execute, row changes are logged, not the statements that make the changes. For stored procedures, this means that the `CALL` statement is not logged. For stored functions, row changes made within the function are logged, not the function invocation. For triggers, row changes made by the trigger are logged. On the replica side, only the row changes are seen, not the stored program invocation.

Mixed format binary logging (`binlog_format=MIXED`) uses statement-based binary logging, except for cases where only row-based binary logging is guaranteed to lead to proper results. With mixed format, when a stored function, stored procedure, trigger, event, or prepared statement contains anything that is not safe for statement-based binary logging, the entire statement is marked as unsafe and logged in row format. The statements used to create and drop procedures, functions, triggers,

and events are always safe, and are logged in statement format. For more information about row-based, mixed, and statement-based logging, and how safe and unsafe statements are determined, see Section 17.1.2, “Replication Formats”.

Unless noted otherwise, the remarks here assume that binary logging is enabled on the server (see Section 5.4.4, “The Binary Log”.) If the binary log is not enabled, replication is not possible, nor is the binary log available for data recovery.

The conditions on the use of stored functions in MySQL can be summarized as follows. These conditions do not apply to stored procedures or Event Scheduler events and they do not apply unless binary logging is enabled.

- To create or alter a stored function, you must have the `SUPER` privilege, in addition to the `CREATE ROUTINE` or `ALTER ROUTINE` privilege that is normally required. (Depending on the `DEFINER` value in the function definition, `SUPER` might be required regardless of whether binary logging is enabled. See Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”.)

- When you create a stored function, you must declare either that it is deterministic or that it does not modify data. Otherwise, it may be unsafe for data recovery or replication.

  By default, for a `CREATE FUNCTION` statement to be accepted, at least one of `DETERMINISTIC`, `NO SQL`, or `READS SQL DATA` must be specified explicitly. Otherwise an error occurs:

  ```
  ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO SQL,
  or READS SQL DATA in its declaration and binary logging is enabled
  (you *might* want to use the less safe log_bin_trust_function_creators
  variable)
  ```

  This function is deterministic (and does not modify data), so it is safe:

  ```
  CREATE FUNCTION f1(i INT)
  RETURNS INT
  DETERMINISTIC
  READS SQL DATA
  BEGIN
    RETURN i;
  END;
  ```

  This function uses `UUID()`, which is not deterministic, so the function also is not deterministic and is not safe:

  ```
  CREATE FUNCTION f2()
  RETURNS CHAR(36) CHARACTER SET utf8
  BEGIN
    RETURN UUID();
  END;
  ```

  This function modifies data, so it may not be safe:

  ```
  CREATE FUNCTION f3(p_id INT)
  RETURNS INT
  BEGIN
    UPDATE t SET modtime = NOW() WHERE id = p_id;
    RETURN ROW_COUNT();
  END;
  ```

  Assessment of the nature of a function is based on the “honesty” of the creator. MySQL does not check that a function declared `DETERMINISTIC` is free of statements that produce nondeterministic results.

- When you attempt to execute a stored function, if `binlog_format=STATEMENT` is set, the `DETERMINISTIC` keyword must be specified in the function definition. If this is not the case, an error is generated and the function does not run, unless `log_bin_trust_function_creators=1` is specified to override this check (see below). For recursive function calls, the `DETERMINISTIC` keyword is required on the outermost call only. If row-based or mixed binary logging is in use, the

statement is accepted and replicated even if the function was defined without the DETERMINISTIC keyword.

- Because MySQL does not check if a function really is deterministic at creation time, the invocation of a stored function with the DETERMINISTIC keyword might carry out an action that is unsafe for statement-based logging, or invoke a function or procedure containing unsafe statements. If this occurs when binlog_format=STATEMENT is set, a warning message is issued. If row-based or mixed binary logging is in use, no warning is issued, and the statement is replicated in row-based format.

- To relax the preceding conditions on function creation (that you must have the SUPER privilege and that a function must be declared deterministic or to not modify data), set the global log_bin_trust_function_creators system variable to 1. By default, this variable has a value of 0, but you can change it like this:

```
mysql> SET GLOBAL log_bin_trust_function_creators = 1;
```

  You can also set this variable at server startup.

  If binary logging is not enabled, log_bin_trust_function_creators does not apply. SUPER is not required for function creation unless, as described previously, the DEFINER value in the function definition requires it.

- For information about built-in functions that may be unsafe for replication (and thus cause stored functions that use them to be unsafe as well), see Section 17.4.1, “Replication Features and Issues”.

Triggers are similar to stored functions, so the preceding remarks regarding functions also apply to triggers with the following exception: CREATE TRIGGER does not have an optional DETERMINISTIC characteristic, so triggers are assumed to be always deterministic. However, this assumption might be invalid in some cases. For example, the UUID() function is nondeterministic (and does not replicate). Be careful about using such functions in triggers.

Triggers can update tables, so error messages similar to those for stored functions occur with CREATE TRIGGER if you do not have the required privileges. On the replica side, the replica uses the trigger DEFINER attribute to determine which user is considered to be the creator of the trigger.

The rest of this section provides additional detail about the logging implementation and its implications. You need not read it unless you are interested in the background on the rationale for the current logging-related conditions on stored routine use. This discussion applies only for statement-based logging, and not for row-based logging, with the exception of the first item: CREATE and DROP statements are logged as statements regardless of the logging mode.

- The server writes CREATE EVENT, CREATE PROCEDURE, CREATE FUNCTION, ALTER EVENT, ALTER PROCEDURE, ALTER FUNCTION, DROP EVENT, DROP PROCEDURE, and DROP FUNCTION statements to the binary log.

- A stored function invocation is logged as a SELECT statement if the function changes data and occurs within a statement that would not otherwise be logged. This prevents nonreplication of data changes that result from use of stored functions in nonlogged statements. For example, SELECT statements are not written to the binary log, but a SELECT might invoke a stored function that makes changes. To handle this, a SELECT *func_name*() statement is written to the binary log when the given function makes a change. Suppose that the following statements are executed on the source server:

```
CREATE FUNCTION f1(a INT) RETURNS INT
BEGIN
  IF (a < 3) THEN
    INSERT INTO t2 VALUES (a);
  END IF;
  RETURN 0;
END;
```

```
CREATE TABLE t1 (a INT);
INSERT INTO t1 VALUES (1),(2),(3);

SELECT f1(a) FROM t1;
```

When the `SELECT` statement executes, the function `f1()` is invoked three times. Two of those invocations insert a row, and MySQL logs a `SELECT` statement for each of them. That is, MySQL writes the following statements to the binary log:

```
SELECT f1(1);
SELECT f1(2);
```

The server also logs a `SELECT` statement for a stored function invocation when the function invokes a stored procedure that causes an error. In this case, the server writes the `SELECT` statement to the log along with the expected error code. On the replica, if the same error occurs, that is the expected result and replication continues. Otherwise, replication stops.

- Logging stored function invocations rather than the statements executed by a function has a security implication for replication, which arises from two factors:
  - It is possible for a function to follow different execution paths on source and replica servers.
  - Statements executed on a replica are processed by the replica SQL thread which has full privileges.

  The implication is that although a user must have the `CREATE ROUTINE` privilege to create a function, the user can write a function containing a dangerous statement that executes only on the replica where it is processed by a thread that has full privileges. For example, if the source and replica servers have server ID values of 1 and 2, respectively, a user on the source server could create and invoke an unsafe function `unsafe_func()` as follows:

```
mysql> delimiter //
mysql> CREATE FUNCTION unsafe_func () RETURNS INT
    -> BEGIN
    ->   IF @@server_id=2 THEN dangerous_statement; END IF;
    ->   RETURN 1;
    -> END;
    -> //
mysql> delimiter ;
mysql> INSERT INTO t VALUES(unsafe_func());
```

  The `CREATE FUNCTION` and `INSERT` statements are written to the binary log, so the replica executes them. Because the replica SQL thread has full privileges, it executes the dangerous statement. Thus, the function invocation has different effects on the source and replica and is not replication-safe.

  To guard against this danger for servers that have binary logging enabled, stored function creators must have the `SUPER` privilege, in addition to the usual `CREATE ROUTINE` privilege that is required. Similarly, to use `ALTER FUNCTION`, you must have the `SUPER` privilege in addition to the `ALTER ROUTINE` privilege. Without the `SUPER` privilege, an error occurs:

```
ERROR 1419 (HY000): You do not have the SUPER privilege and
binary logging is enabled (you *might* want to use the less safe
log_bin_trust_function_creators variable)
```

  If you do not want to require function creators to have the `SUPER` privilege (for example, if all users with the `CREATE ROUTINE` privilege on your system are experienced application developers), set the global `log_bin_trust_function_creators` system variable to 1. You can also set this variable at server startup. If binary logging is not enabled, `log_bin_trust_function_creators` does not apply. `SUPER` is not required for function creation unless, as described previously, the `DEFINER` value in the function definition requires it.

- If a function that performs updates is nondeterministic, it is not repeatable. This can have two undesirable effects:

- It makes a replica different from the source.
- Restored data is different from the original data.

To deal with these problems, MySQL enforces the following requirement: On a source server, creation and alteration of a function is refused unless you declare the function to be deterministic or to not modify data. Two sets of function characteristics apply here:

- The `DETERMINISTIC` and `NOT DETERMINISTIC` characteristics indicate whether a function always produces the same result for given inputs. The default is `NOT DETERMINISTIC` if neither characteristic is given. To declare that a function is deterministic, you must specify `DETERMINISTIC` explicitly.
- The `CONTAINS SQL`, `NO SQL`, `READS SQL DATA`, and `MODIFIES SQL DATA` characteristics provide information about whether the function reads or writes data. Either `NO SQL` or `READS SQL DATA` indicates that a function does not change data, but you must specify one of these explicitly because the default is `CONTAINS SQL` if no characteristic is given.

By default, for a `CREATE FUNCTION` statement to be accepted, at least one of `DETERMINISTIC`, `NO SQL`, or `READS SQL DATA` must be specified explicitly. Otherwise an error occurs:

```
ERROR 1418 (HY000): This function has none of DETERMINISTIC, NO SQL,
or READS SQL DATA in its declaration and binary logging is enabled
(you *might* want to use the less safe log_bin_trust_function_creators
variable)
```

If you set `log_bin_trust_function_creators` to 1, the requirement that functions be deterministic or not modify data is dropped.

- Stored procedure calls are logged at the statement level rather than at the `CALL` level. That is, the server does not log the `CALL` statement, it logs those statements within the procedure that actually execute. As a result, the same changes that occur on the source server are observed on replicas. This prevents problems that could result from a procedure having different execution paths on different machines.

  In general, statements executed within a stored procedure are written to the binary log using the same rules that would apply were the statements to be executed in standalone fashion. Some special care is taken when logging procedure statements because statement execution within procedures is not quite the same as in nonprocedure context:

  - A statement to be logged might contain references to local procedure variables. These variables do not exist outside of stored procedure context, so a statement that refers to such a variable cannot be logged literally. Instead, each reference to a local variable is replaced by this construct for logging purposes:

    ```
    NAME_CONST(var_name, var_value)
    ```

    `var_name` is the local variable name, and *`var_value`* is a constant indicating the value that the variable has at the time the statement is logged. `NAME_CONST()` has a value of *`var_value`*, and a “name” of *`var_name`*. Thus, if you invoke this function directly, you get a result like this:

    ```
    mysql> SELECT NAME_CONST('myname', 14);
    +--------+
    | myname |
    +--------+
    |     14 |
    ```

```
+--------+
```

`NAME_CONST()` enables a logged standalone statement to be executed on a replica with the same effect as the original statement that was executed on the source within a stored procedure.

The use of `NAME_CONST()` can result in a problem for `CREATE TABLE ... SELECT` statements when the source column expressions refer to local variables. Converting these references to `NAME_CONST()` expressions can result in column names that are different on the source and replica servers, or names that are too long to be legal column identifiers. A workaround is to supply aliases for columns that refer to local variables. Consider this statement when `myvar` has a value of 1:

```
CREATE TABLE t1 SELECT myvar;
```

That is rewritten as follows:

```
CREATE TABLE t1 SELECT NAME_CONST(myvar, 1);
```

To ensure that the source and replica tables have the same column names, write the statement like this:

```
CREATE TABLE t1 SELECT myvar AS myvar;
```

The rewritten statement becomes:

```
CREATE TABLE t1 SELECT NAME_CONST(myvar, 1) AS myvar;
```

- A statement to be logged might contain references to user-defined variables. To handle this, MySQL writes a `SET` statement to the binary log to make sure that the variable exists on the replica with the same value as on the source. For example, if a statement refers to a variable `@my_var`, that statement is preceded in the binary log by the following statement, where *`value`* is the value of `@my_var` on the source:

  ```
  SET @my_var = value;
  ```

- Procedure calls can occur within a committed or rolled-back transaction. Transactional context is accounted for so that the transactional aspects of procedure execution are replicated correctly. That is, the server logs those statements within the procedure that actually execute and modify data, and also logs `BEGIN`, `COMMIT`, and `ROLLBACK` statements as necessary. For example, if a procedure updates only transactional tables and is executed within a transaction that is rolled back, those updates are not logged. If the procedure occurs within a committed transaction, `BEGIN` and `COMMIT` statements are logged with the updates. For a procedure that executes within a rolled-back transaction, its statements are logged using the same rules that would apply if the statements were executed in standalone fashion:

  - Updates to transactional tables are not logged.
  - Updates to nontransactional tables are logged because rollback does not cancel them.
  - Updates to a mix of transactional and nontransactional tables are logged surrounded by `BEGIN` and `ROLLBACK` so that replicas make the same changes and rollbacks as on the source.

- A stored procedure call is *not* written to the binary log at the statement level if the procedure is invoked from within a stored function. In that case, the only thing logged is the statement that invokes the function (if it occurs within a statement that is logged) or a `DO` statement (if it occurs within a statement that is not logged). For this reason, care should be exercised in the use of stored functions that invoke a procedure, even if the procedure is otherwise safe in itself.

20.8 Restrictions on Stored Programs

- SQL Statements Not Permitted in Stored Routines

- Restrictions for Stored Functions
- Restrictions for Triggers
- Name Conflicts within Stored Routines
- Replication Considerations
- Debugging Considerations
- Unsupported Syntax from the SQL:2003 Standard
- Stored Routine Concurrency Considerations
- Event Scheduler Restrictions
- Stored Programs in NDB Cluster

These restrictions apply to the features described in Chapter 20, *Stored Objects*.

Some of the restrictions noted here apply to all stored routines; that is, both to stored procedures and stored functions. There are also some restrictions specific to stored functions but not to stored procedures.

The restrictions for stored functions also apply to triggers. There are also some restrictions specific to triggers.

The restrictions for stored procedures also apply to the `DO` clause of Event Scheduler event definitions. There are also some restrictions specific to events.

SQL Statements Not Permitted in Stored Routines

Stored routines cannot contain arbitrary SQL statements. The following statements are not permitted:

- The locking statements `LOCK TABLES` and `UNLOCK TABLES`.
- `ALTER VIEW`.
- `LOAD DATA`.
- SQL prepared statements (`PREPARE`, `EXECUTE`, `DEALLOCATE PREPARE`) can be used in stored procedures, but not stored functions or triggers. Thus, stored functions and triggers cannot use dynamic SQL (where you construct statements as strings and then execute them).
- Generally, statements not permitted in SQL prepared statements are also not permitted in stored programs. For a list of statements supported as prepared statements, see Section 13.5, “Prepared Statements”. Exceptions are `SIGNAL`, `RESIGNAL`, and `GET DIAGNOSTICS`, which are not permissible as prepared statements but are permitted in stored programs.
- Because local variables are in scope only during stored program execution, references to them are not permitted in prepared statements created within a stored program. Prepared statement scope is the current session, not the stored program, so the statement could be executed after the program ends, at which point the variables would no longer be in scope. For example, `SELECT ... INTO local_var` cannot be used as a prepared statement. This restriction also applies to stored procedure and function parameters. See Section 13.5.1, “PREPARE Statement”.
- Inserts cannot be delayed. `INSERT DELAYED` syntax is accepted, but the statement is handled as a normal `INSERT`.
- Within all stored programs (stored procedures and functions, triggers, and events), the parser treats `BEGIN [WORK]` as the beginning of a `BEGIN ... END` block. To begin a transaction in this context, use `START TRANSACTION` instead.

Restrictions for Stored Functions

The following additional statements or operations are not permitted within stored functions. They are permitted within stored procedures, except stored procedures that are invoked from within a stored function or trigger. For example, if you use `FLUSH` in a stored procedure, that stored procedure cannot be called from a stored function or trigger.

- Statements that perform explicit or implicit commit or rollback. Support for these statements is not required by the SQL standard, which states that each DBMS vendor may decide whether to permit them.
- Statements that return a result set. This includes `SELECT` statements that do not have an `INTO *var_list*` clause and other statements such as `SHOW`, `EXPLAIN`, and `CHECK TABLE`. A function can process a result set either with `SELECT ... INTO *var_list*` or by using a cursor and `FETCH` statements. See Section 13.2.9.1, “SELECT ... INTO Statement”, and Section 13.6.6, “Cursors”.
- `FLUSH` statements.
- Stored functions cannot be used recursively.
- A stored function or trigger cannot modify a table that is already being used (for reading or writing) by the statement that invoked the function or trigger.
- If you refer to a temporary table multiple times in a stored function under different aliases, a `Can't reopen table: '*tbl_name*'` error occurs, even if the references occur in different statements within the function.
- `HANDLER ... READ` statements that invoke stored functions can cause replication errors and are disallowed.

Restrictions for Triggers

For triggers, the following additional restrictions apply:

- Triggers are not activated by foreign key actions.
- When using row-based replication, triggers on the replica are not activated by statements originating on the source. The triggers on the replica are activated when using statement-based replication. For more information, see Section 17.4.1.33, “Replication and Triggers”.
- The `RETURN` statement is not permitted in triggers, which cannot return a value. To exit a trigger immediately, use the `LEAVE` statement.
- Triggers are not permitted on tables in the `mysql` database. Nor are they permitted on `INFORMATION_SCHEMA` or `performance_schema` tables. Those tables are actually views and triggers are not permitted on views.
- The trigger cache does not detect when metadata of the underlying objects has changed. If a trigger uses a table and the table has changed since the trigger was loaded into the cache, the trigger operates using the outdated metadata.

Name Conflicts within Stored Routines

The same identifier might be used for a routine parameter, a local variable, and a table column. Also, the same local variable name can be used in nested blocks. For example:

```
CREATE PROCEDURE p (i INT)
BEGIN
  DECLARE i INT DEFAULT 0;
  SELECT i FROM t;
  BEGIN
```

```
    DECLARE i INT DEFAULT 1;
    SELECT i FROM t;
  END;
END;
```

In such cases, the identifier is ambiguous and the following precedence rules apply:

- A local variable takes precedence over a routine parameter or table column.
- A routine parameter takes precedence over a table column.
- A local variable in an inner block takes precedence over a local variable in an outer block.

The behavior that variables take precedence over table columns is nonstandard.

Replication Considerations

Use of stored routines can cause replication problems. This issue is discussed further in Section 20.7, “Stored Program Binary Logging”.

The `--replicate-wild-do-table=db_name.tbl_name` option applies to tables, views, and triggers. It does not apply to stored procedures and functions, or events. To filter statements operating on the latter objects, use one or more of the `--replicate-*-db` options.

Debugging Considerations

There are no stored routine debugging facilities.

Unsupported Syntax from the SQL:2003 Standard

The MySQL stored routine syntax is based on the SQL:2003 standard. The following items from that standard are not currently supported:

- `UNDO` handlers
- `FOR` loops

Stored Routine Concurrency Considerations

To prevent problems of interaction between sessions, when a client issues a statement, the server uses a snapshot of routines and triggers available for execution of the statement. That is, the server calculates a list of procedures, functions, and triggers that may be used during execution of the statement, loads them, and then proceeds to execute the statement. While the statement executes, it does not see changes to routines performed by other sessions.

For maximum concurrency, stored functions should minimize their side-effects; in particular, updating a table within a stored function can reduce concurrent operations on that table. A stored function acquires table locks before executing, to avoid inconsistency in the binary log due to mismatch of the order in which statements execute and when they appear in the log. When statement-based binary logging is used, statements that invoke a function are recorded rather than the statements executed within the function. Consequently, stored functions that update the same underlying tables do not execute in parallel. In contrast, stored procedures do not acquire table-level locks. All statements executed within stored procedures are written to the binary log, even for statement-based binary logging. See Section 20.7, “Stored Program Binary Logging”.

Event Scheduler Restrictions

The following limitations are specific to the Event Scheduler:

- Event names are handled in case-insensitive fashion. For example, you cannot have two events in the same database with the names `anEvent` and `AnEvent`.

- An event may not be created, altered, or dropped from within a stored program, if the event name is specified by means of a variable. An event also may not create, alter, or drop stored routines or triggers.
- DDL statements on events are prohibited while a `LOCK TABLES` statement is in effect.
- Event timings using the intervals `YEAR`, `QUARTER`, `MONTH`, and `YEAR_MONTH` are resolved in months; those using any other interval are resolved in seconds. There is no way to cause events scheduled to occur at the same second to execute in a given order. In addition—due to rounding, the nature of threaded applications, and the fact that a nonzero length of time is required to create events and to signal their execution—events may be delayed by as much as 1 or 2 seconds. However, the time shown in the `INFORMATION_SCHEMA.EVENTS` table's `LAST_EXECUTED` column or the `mysql.event` table's `last_executed` column is always accurate to within one second of the actual event execution time. (See also Bug #16522.)
- Each execution of the statements contained in the body of an event takes place in a new connection; thus, these statements have no effect in a given user session on the server's statement counts such as `Com_select` and `Com_insert` that are displayed by `SHOW STATUS`. However, such counts *are* updated in the global scope. (Bug #16422)
- Events do not support times later than the end of the Unix Epoch; this is approximately the beginning of the year 2038. Such dates are specifically not permitted by the Event Scheduler. (Bug #16396)
- References to stored functions, user-defined functions, and tables in the `ON SCHEDULE` clauses of `CREATE EVENT` and `ALTER EVENT` statements are not supported. These sorts of references are not permitted. (See Bug #22830 for more information.)

Stored Programs in NDB Cluster

While stored procedures, stored functions, triggers, and scheduled events are all supported by tables using the `NDB` storage engine, you must keep in mind that these do *not* propagate automatically between MySQL Servers acting as Cluster SQL nodes. This is because of the following:

- Stored routine definitions are kept in tables in the `mysql` system database using the `MyISAM` storage engine, and so do not participate in clustering.
- The `.TRN` and `.TRG` files containing trigger definitions are not read by the `NDB` storage engine, and are not copied between Cluster nodes.

Any stored routine or trigger that interacts with NDB Cluster tables must be re-created by running the appropriate `CREATE PROCEDURE`, `CREATE FUNCTION`, or `CREATE TRIGGER` statements on each MySQL Server that participates in the cluster where you wish to use the stored routine or trigger. Similarly, any changes to existing stored routines or triggers must be carried out explicitly on all Cluster SQL nodes, using the appropriate `ALTER` or `DROP` statements on each MySQL Server accessing the cluster.

Warning

Do *not* attempt to work around the issue described in the first item mentioned previously by converting any `mysql` database tables to use the `NDB` storage engine. *Altering the system tables in the `mysql` database is not supported* and is very likely to produce undesirable results.

20.9 Restrictions on Views

The maximum number of tables that can be referenced in the definition of a view is 61.

View processing is not optimized:

- It is not possible to create an index on a view.

- Indexes can be used for views processed using the merge algorithm. However, a view that is processed with the temptable algorithm is unable to take advantage of indexes on its underlying tables (although indexes can be used during generation of the temporary tables).

Subqueries cannot be used in the `FROM` clause of a view.

There is a general principle that you cannot modify a table and select from the same table in a subquery. See Section 13.2.10.12, “Restrictions on Subqueries”.

The same principle also applies if you select from a view that selects from the table, if the view selects from the table in a subquery and the view is evaluated using the merge algorithm. Example:

```
CREATE VIEW v1 AS
SELECT * FROM t2 WHERE EXISTS (SELECT 1 FROM t1 WHERE t1.a = t2.a);

UPDATE t1, v2 SET t1.a = 1 WHERE t1.b = v2.b;
```

If the view is evaluated using a temporary table, you *can* select from the table in the view subquery and still modify that table in the outer query. In this case the view is stored in a temporary table and thus you are not really selecting from the table in a subquery and modifying it “at the same time.” (This is another reason you might wish to force MySQL to use the temptable algorithm by specifying `ALGORITHM = TEMPTABLE` in the view definition.)

You can use `DROP TABLE` or `ALTER TABLE` to drop or alter a table that is used in a view definition. No warning results from the `DROP` or `ALTER` operation, even though this invalidates the view. Instead, an error occurs later, when the view is used. `CHECK TABLE` can be used to check for views that have been invalidated by `DROP` or `ALTER` operations.

Before MySQL 5.6.5, a view definition is “frozen” by certain statements. If a statement prepared by `PREPARE` refers to a view, the view definition seen each time the statement is executed later is the definition of the view at the time it was prepared. This is true even if the view definition is changed after the statement is prepared and before it is executed. In the following example, the result returned by the `EXECUTE` statement is a random number, not the current date and time:

```
CREATE VIEW v AS SELECT RAND();
PREPARE s FROM 'SELECT * FROM v';
ALTER VIEW v AS SELECT NOW();
EXECUTE s;
```

With regard to view updatability, the overall goal for views is that if any view is theoretically updatable, it should be updatable in practice. MySQL as quickly as possible. Many theoretically updatable views can be updated now, but limitations still exist. For details, see Section 20.5.3, “Updatable and Insertable Views”.

There exists a shortcoming with the current implementation of views. If a user is granted the basic privileges necessary to create a view (the `CREATE VIEW` and `SELECT` privileges), that user cannot call `SHOW CREATE VIEW` on that object unless the user is also granted the `SHOW VIEW` privilege.

That shortcoming can lead to problems backing up a database with `mysqldump`, which may fail due to insufficient privileges. This problem is described in Bug #22062.

The workaround to the problem is for the administrator to manually grant the `SHOW VIEW` privilege to users who are granted `CREATE VIEW`, since MySQL doesn't grant it implicitly when views are created.

Views do not have indexes, so index hints do not apply. Use of index hints when selecting from a view is not permitted.

`SHOW CREATE VIEW` displays view definitions using an `AS` *`alias_name`* clause for each column. If a column is created from an expression, the default alias is the expression text, which can be quite long. Aliases for column names in `CREATE VIEW` statements are checked against the maximum column length of 64 characters (not the maximum alias length of 256 characters). As a result, views created from the output of `SHOW CREATE VIEW` fail if any column alias exceeds 64 characters. This can cause problems in the following circumstances for views with too-long aliases:

- View definitions fail to replicate to newer replicas that enforce the column-length restriction.
- Dump files created with `mysqldump` cannot be loaded into servers that enforce the column-length restriction.

A workaround for either problem is to modify each problematic view definition to use aliases that provide shorter column names. Then the view replicates properly, and can be dumped and reloaded without causing an error. To modify the definition, drop and create the view again with `DROP VIEW` and `CREATE VIEW`, or replace the definition with `CREATE OR REPLACE VIEW`.

For problems that occur when reloading view definitions in dump files, another workaround is to edit the dump file to modify its `CREATE VIEW` statements. However, this does not change the original view definitions, which may cause problems for subsequent dump operations.

Chapter 21 INFORMATION_SCHEMA Tables

Table of Contents

21.1 Introduction 2976
21.2 The INFORMATION_SCHEMA CHARACTER_SETS Table 2979
21.3 The INFORMATION_SCHEMA COLLATIONS Table 2979
21.4 The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table ... 2980
21.5 The INFORMATION_SCHEMA COLUMNS Table 2980
21.6 The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table 2983
21.7 The INFORMATION_SCHEMA ENGINES Table 2983
21.8 The INFORMATION_SCHEMA EVENTS Table 2984
21.9 The INFORMATION_SCHEMA GLOBAL_STATUS and SESSION_STATUS Tables 2988
21.10 The INFORMATION_SCHEMA GLOBAL_VARIABLES and SESSION_VARIABLES Tables 2988
21.11 The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table 2988
21.12 The INFORMATION_SCHEMA OPTIMIZER_TRACE Table 2990
21.13 The INFORMATION_SCHEMA PARAMETERS Table 2990
21.14 The INFORMATION_SCHEMA PARTITIONS Table 2991
21.15 The INFORMATION_SCHEMA PLUGINS Table 2995
21.16 The INFORMATION_SCHEMA PROCESSLIST Table 2996
21.17 The INFORMATION_SCHEMA PROFILING Table 2997
21.18 The INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS Table 2998
21.19 The INFORMATION_SCHEMA ROUTINES Table 2999
21.20 The INFORMATION_SCHEMA SCHEMATA Table 3001
21.21 The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table 3002
21.22 The INFORMATION_SCHEMA STATISTICS Table 3003
21.23 The INFORMATION_SCHEMA TABLES Table 3004
21.24 The INFORMATION_SCHEMA TABLESPACES Table 3008
21.25 The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table 3008
21.26 The INFORMATION_SCHEMA TABLE_PRIVILEGES Table 3009
21.27 The INFORMATION_SCHEMA TRIGGERS Table 3009
21.28 The INFORMATION_SCHEMA USER_PRIVILEGES Table 3011
21.29 The INFORMATION_SCHEMA VIEWS Table 3012
21.30 INFORMATION_SCHEMA InnoDB Tables 3013
- 21.30.1 The INFORMATION_SCHEMA INNODB_BUFFER_PAGE Table 3013
- 21.30.2 The INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU Table 3016
- 21.30.3 The INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS Table 3019
- 21.30.4 The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables 3022
- 21.30.5 The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables 3023
- 21.30.6 The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables 3025
- 21.30.7 The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table 3026
- 21.30.8 The INFORMATION_SCHEMA INNODB_FT_CONFIG Table 3027
- 21.30.9 The INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD Table 3028
- 21.30.10 The INFORMATION_SCHEMA INNODB_FT_DELETED Table 3029
- 21.30.11 The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table 3030
- 21.30.12 The INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE Table 3031
- 21.30.13 The INFORMATION_SCHEMA INNODB_LOCKS Table 3033
- 21.30.14 The INFORMATION_SCHEMA INNODB_LOCK_WAITS Table 3034
- 21.30.15 The INFORMATION_SCHEMA INNODB_METRICS Table 3035
- 21.30.16 The INFORMATION_SCHEMA INNODB_SYS_COLUMNS Table 3037
- 21.30.17 The INFORMATION_SCHEMA INNODB_SYS_DATAFILES Table 3038
- 21.30.18 The INFORMATION_SCHEMA INNODB_SYS_FIELDS Table 3038
- 21.30.19 The INFORMATION_SCHEMA INNODB_SYS_FOREIGN Table 3039
- 21.30.20 The INFORMATION_SCHEMA INNODB_SYS_FOREIGN_COLS Table 3040

21.30.21 The INFORMATION_SCHEMA INNODB_SYS_INDEXES Table 3040
21.30.22 The INFORMATION_SCHEMA INNODB_SYS_TABLES Table 3042
21.30.23 The INFORMATION_SCHEMA INNODB_SYS_TABLESPACES Table 3043
21.30.24 The INFORMATION_SCHEMA INNODB_SYS_TABLESTATS View 3044
21.30.25 The INFORMATION_SCHEMA INNODB_TRX Table 3045
21.31 INFORMATION_SCHEMA NDB Cluster Tables 3048
21.31.1 The INFORMATION_SCHEMA FILES Table 3048
21.31.2 The INFORMATION_SCHEMA ndb_transid_mysql_connection_map Table 3053
21.32 INFORMATION_SCHEMA Thread Pool Tables 3054
21.32.1 The INFORMATION_SCHEMA TP_THREAD_GROUP_STATE Table 3055
21.32.2 The INFORMATION_SCHEMA TP_THREAD_GROUP_STATS Table 3056
21.32.3 The INFORMATION_SCHEMA TP_THREAD_STATE Table 3058
21.33 INFORMATION_SCHEMA Connection-Control Tables 3059
21.33.1 The INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS Table 3059
21.34 INFORMATION_SCHEMA MySQL Enterprise Firewall Tables 3059
21.34.1 The INFORMATION_SCHEMA MYSQL_FIREWALL_USERS Table 3060
21.34.2 The INFORMATION_SCHEMA MYSQL_FIREWALL_WHITELIST Table 3060
21.35 Extensions to SHOW Statements 3060

`INFORMATION_SCHEMA` provides access to database *metadata*, information about the MySQL server such as the name of a database or table, the data type of a column, or access privileges. Other terms that are sometimes used for this information are *data dictionary* and *system catalog*.

21.1 Introduction

`INFORMATION_SCHEMA` provides access to database *metadata*, information about the MySQL server such as the name of a database or table, the data type of a column, or access privileges. Other terms that are sometimes used for this information are *data dictionary* and *system catalog*.

- INFORMATION_SCHEMA Usage Notes
- Character Set Considerations
- INFORMATION_SCHEMA as Alternative to SHOW Statements
- INFORMATION_SCHEMA and Privileges
- Performance Considerations
- Standards Considerations
- Conventions in the INFORMATION_SCHEMA Reference Sections
- Related Information

INFORMATION_SCHEMA Usage Notes

`INFORMATION_SCHEMA` is a database within each MySQL instance, the place that stores information about all the other databases that the MySQL server maintains. The `INFORMATION_SCHEMA` database contains several read-only tables. They are actually views, not base tables, so there are no files associated with them, and you cannot set triggers on them. Also, there is no database directory with that name.

Although you can select `INFORMATION_SCHEMA` as the default database with a `USE` statement, you can only read the contents of tables, not perform `INSERT`, `UPDATE`, or `DELETE` operations on them.

Here is an example of a statement that retrieves information from `INFORMATION_SCHEMA`:

```
mysql> SELECT table_name, table_type, engine
       FROM information_schema.tables
       WHERE table_schema = 'db5'
       ORDER BY table_name;
+------------+------------+--------+
| table_name | table_type | engine |
+------------+------------+--------+
| fk         | BASE TABLE | InnoDB |
| fk2        | BASE TABLE | InnoDB |
| goto       | BASE TABLE | MyISAM |
| into       | BASE TABLE | MyISAM |
| k          | BASE TABLE | MyISAM |
| kurs       | BASE TABLE | MyISAM |
| loop       | BASE TABLE | MyISAM |
| pk         | BASE TABLE | InnoDB |
| t          | BASE TABLE | MyISAM |
| t2         | BASE TABLE | MyISAM |
| t3         | BASE TABLE | MyISAM |
| t7         | BASE TABLE | MyISAM |
| tables     | BASE TABLE | MyISAM |
| v          | VIEW       | NULL   |
| v2         | VIEW       | NULL   |
| v3         | VIEW       | NULL   |
| v56        | VIEW       | NULL   |
+------------+------------+--------+
17 rows in set (0.01 sec)
```

Explanation: The statement requests a list of all the tables in database `db5`, showing just three pieces of information: the name of the table, its type, and its storage engine.

Character Set Considerations

The definition for character columns (for example, `TABLES.TABLE_NAME`) is generally `VARCHAR(N) CHARACTER SET utf8` where *N* is at least 64. MySQL uses the default collation for this character set (`utf8_general_ci`) for all searches, sorts, comparisons, and other string operations on such columns.

Because some MySQL objects are represented as files, searches in `INFORMATION_SCHEMA` string columns can be affected by file system case sensitivity. For more information, see Section 10.8.7, “Using Collation in INFORMATION_SCHEMA Searches”.

INFORMATION_SCHEMA as Alternative to SHOW Statements

The `SELECT ... FROM INFORMATION_SCHEMA` statement is intended as a more consistent way to provide access to the information provided by the various `SHOW` statements that MySQL supports (`SHOW DATABASES`, `SHOW TABLES`, and so forth). Using `SELECT` has these advantages, compared to `SHOW`:

- It conforms to Codd's rules, because all access is done on tables.
- You can use the familiar syntax of the `SELECT` statement, and only need to learn some table and column names.
- The implementor need not worry about adding keywords.
- You can filter, sort, concatenate, and transform the results from `INFORMATION_SCHEMA` queries into whatever format your application needs, such as a data structure or a text representation to parse.
- This technique is more interoperable with other database systems. For example, Oracle Database users are familiar with querying tables in the Oracle data dictionary.

Because `SHOW` is familiar and widely used, the `SHOW` statements remain as an alternative. In fact, along with the implementation of `INFORMATION_SCHEMA`, there are enhancements to `SHOW` as described in Section 21.35, “Extensions to SHOW Statements”.

INFORMATION_SCHEMA and Privileges

For most INFORMATION_SCHEMA tables, each MySQL user has the right to access them, but can see only the rows in the tables that correspond to objects for which the user has the proper access privileges. In some cases (for example, the ROUTINE_DEFINITION column in the INFORMATION_SCHEMA ROUTINES table), users who have insufficient privileges see NULL. Some tables have different privilege requirements; for these, the requirements are mentioned in the applicable table descriptions. For example, InnoDB tables (tables with names that begin with INNODB_) require the PROCESS privilege.

The same privileges apply to selecting information from INFORMATION_SCHEMA and viewing the same information through SHOW statements. In either case, you must have some privilege on an object to see information about it.

Performance Considerations

INFORMATION_SCHEMA queries that search for information from more than one database might take a long time and impact performance. To check the efficiency of a query, you can use EXPLAIN. For information about using EXPLAIN output to tune INFORMATION_SCHEMA queries, see Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”.

Standards Considerations

The implementation for the INFORMATION_SCHEMA table structures in MySQL follows the ANSI/ISO SQL:2003 standard Part 11 *Schemata*. Our intent is approximate compliance with SQL:2003 core feature F021 *Basic information schema*.

Users of SQL Server 2000 (which also follows the standard) may notice a strong similarity. However, MySQL has omitted many columns that are not relevant for our implementation, and added columns that are MySQL-specific. One such added column is the ENGINE column in the INFORMATION_SCHEMA TABLES table.

Although other DBMSs use a variety of names, like syscat or system, the standard name is INFORMATION_SCHEMA.

To avoid using any name that is reserved in the standard or in DB2, SQL Server, or Oracle, we changed the names of some columns marked “MySQL extension”. (For example, we changed COLLATION to TABLE_COLLATION in the TABLES table.) See the list of reserved words near the end of this article: https://web.archive.org/web/20070428032454/http://www.dbazine.com/db2/db2-disarticles/gulutzan5.

Conventions in the INFORMATION_SCHEMA Reference Sections

The following sections describe each of the tables and columns in INFORMATION_SCHEMA. For each column, there are three pieces of information:

- “INFORMATION_SCHEMA Name” indicates the name for the column in the INFORMATION_SCHEMA table. This corresponds to the standard SQL name unless the “Remarks” field says “MySQL extension.”
- “SHOW Name” indicates the equivalent field name in the closest SHOW statement, if there is one.
- “Remarks” provides additional information where applicable. If this field is NULL, it means that the value of the column is always NULL. If this field says “MySQL extension,” the column is a MySQL extension to standard SQL.

Many sections indicate what SHOW statement is equivalent to a SELECT that retrieves information from INFORMATION_SCHEMA. For SHOW statements that display information for the default database if you omit a FROM *db_name* clause, you can often select information for the default database by adding an AND TABLE_SCHEMA = SCHEMA() condition to the WHERE clause of a query that retrieves information from an INFORMATION_SCHEMA table.

Related Information

These sections discuss additional INFORMATION_SCHEMA-related topics:

- information about INFORMATION_SCHEMA tables specific to the InnoDB storage engine: Section 21.30, “INFORMATION_SCHEMA InnoDB Tables”
- information about INFORMATION_SCHEMA tables specific to the NDB storage engine (NDB Cluster): Section 21.31, “INFORMATION_SCHEMA NDB Cluster Tables”
- information about INFORMATION_SCHEMA tables specific to the thread pool plugin: Section 21.32, “INFORMATION_SCHEMA Thread Pool Tables”
- information about INFORMATION_SCHEMA tables specific to the CONNECTION_CONTROL plugin: Section 21.33, “INFORMATION_SCHEMA Connection-Control Tables”
- Answers to questions that are often asked concerning the INFORMATION_SCHEMA database: Section A.7, “MySQL 5.6 FAQ: INFORMATION_SCHEMA”
- INFORMATION_SCHEMA queries and the optimizer: Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”
- The effect of collation on INFORMATION_SCHEMA comparisons: Section 10.8.7, “Using Collation in INFORMATION_SCHEMA Searches”
- As of MySQL 5.6.49, you must have the PROCESS privilege to query this table.

21.2 The INFORMATION_SCHEMA CHARACTER_SETS Table

The CHARACTER_SETS table provides information about available character sets.

The CHARACTER_SETS table has these columns:

- CHARACTER_SET_NAME

  The character set name.

- DEFAULT_COLLATE_NAME

  The default collation for the character set.

- DESCRIPTION

  A description of the character set.

- MAXLEN

  The maximum number of bytes required to store one character.

Notes

Character set information is also available from the SHOW CHARACTER SET statement. See Section 13.7.5.4, “SHOW CHARACTER SET Statement”. The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.CHARACTER_SETS
  [WHERE CHARACTER_SET_NAME LIKE 'wild']

SHOW CHARACTER SET
  [LIKE 'wild']
```

21.3 The INFORMATION_SCHEMA COLLATIONS Table

The COLLATIONS table provides information about collations for each character set.

The COLLATIONS table has these columns:

- COLLATION_NAME

  The collation name.

- CHARACTER_SET_NAME

  The name of the character set with which the collation is associated.

- ID

  The collation ID.

- IS_DEFAULT

  Whether the collation is the default for its character set.

- IS_COMPILED

  Whether the character set is compiled into the server.

- SORTLEN

  This is related to the amount of memory required to sort strings expressed in the character set.

Notes

Collation information is also available from the SHOW COLLATION statement. See Section 13.7.5.5, “SHOW COLLATION Statement”. The following statements are equivalent:

```
SELECT COLLATION_NAME FROM INFORMATION_SCHEMA.COLLATIONS
  [WHERE COLLATION_NAME LIKE 'wild']

SHOW COLLATION
  [LIKE 'wild']
```

21.4 The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table

The COLLATION_CHARACTER_SET_APPLICABILITY table indicates what character set is applicable for what collation.

The COLLATION_CHARACTER_SET_APPLICABILITY table has these columns:

- COLLATION_NAME

  The collation name.

- CHARACTER_SET_NAME

  The name of the character set with which the collation is associated.

Notes

The COLLATION_CHARACTER_SET_APPLICABILITY columns are equivalent to the first two columns displayed by the SHOW COLLATION statement.

21.5 The INFORMATION_SCHEMA COLUMNS Table

The COLUMNS table provides information about columns in tables.

The COLUMNS table has these columns:

- `TABLE_CATALOG`

  The name of the catalog to which the table containing the column belongs. This value is always `def`.

- `TABLE_SCHEMA`

  The name of the schema (database) to which the table containing the column belongs.

- `TABLE_NAME`

  The name of the table containing the column.

- `COLUMN_NAME`

  The name of the column.

- `ORDINAL_POSITION`

  The position of the column within the table. `ORDINAL_POSITION` is necessary because you might want to say `ORDER BY ORDINAL_POSITION`. Unlike `SHOW COLUMNS`, `SELECT` from the `COLUMNS` table does not have automatic ordering.

- `COLUMN_DEFAULT`

  The default value for the column. This is `NULL` if the column has an explicit default of `NULL`, or if the column definition includes no `DEFAULT` clause.

- `IS_NULLABLE`

  The column nullability. The value is `YES` if `NULL` values can be stored in the column, `NO` if not.

- `DATA_TYPE`

  The column data type.

  The `DATA_TYPE` value is the type name only with no other information. The `COLUMN_TYPE` value contains the type name and possibly other information such as the precision or length.

- `CHARACTER_MAXIMUM_LENGTH`

  For string columns, the maximum length in characters.

- `CHARACTER_OCTET_LENGTH`

  For string columns, the maximum length in bytes.

- `NUMERIC_PRECISION`

  For numeric columns, the numeric precision.

- `NUMERIC_SCALE`

  For numeric columns, the numeric scale.

- `DATETIME_PRECISION`

  For temporal columns, the fractional seconds precision.

- `CHARACTER_SET_NAME`

  For character string columns, the character set name.

- `COLLATION_NAME`

  For character string columns, the collation name.

- `COLUMN_TYPE`

  The column data type.

  The `DATA_TYPE` value is the type name only with no other information. The `COLUMN_TYPE` value contains the type name and possibly other information such as the precision or length.

- `COLUMN_KEY`

  Whether the column is indexed:

  - If `COLUMN_KEY` is empty, the column either is not indexed or is indexed only as a secondary column in a multiple-column, nonunique index.
  - If `COLUMN_KEY` is `PRI`, the column is a `PRIMARY KEY` or is one of the columns in a multiple-column `PRIMARY KEY`.
  - If `COLUMN_KEY` is `UNI`, the column is the first column of a `UNIQUE` index. (A `UNIQUE` index permits multiple `NULL` values, but you can tell whether the column permits `NULL` by checking the `Null` column.)
  - If `COLUMN_KEY` is `MUL`, the column is the first column of a nonunique index in which multiple occurrences of a given value are permitted within the column.

  If more than one of the `COLUMN_KEY` values applies to a given column of a table, `COLUMN_KEY` displays the one with the highest priority, in the order `PRI`, `UNI`, `MUL`.

  A `UNIQUE` index may be displayed as `PRI` if it cannot contain `NULL` values and there is no `PRIMARY KEY` in the table. A `UNIQUE` index may display as `MUL` if several columns form a composite `UNIQUE` index; although the combination of the columns is unique, each column can still hold multiple occurrences of a given value.

- `EXTRA`

  Any additional information that is available about a given column. The value is nonempty in these cases: `auto_increment` for columns that have the `AUTO_INCREMENT` attribute; `on update CURRENT_TIMESTAMP` for `TIMESTAMP` or `DATETIME` columns that have the `ON UPDATE CURRENT_TIMESTAMP` attribute.

- `PRIVILEGES`

  The privileges you have for the column.

- `COLUMN_COMMENT`

  Any comment included in the column definition.

Notes

- In `SHOW COLUMNS`, the `Type` display includes values from several different `COLUMNS` columns.
- `CHARACTER_OCTET_LENGTH` should be the same as `CHARACTER_MAXIMUM_LENGTH`, except for multibyte character sets.
- `CHARACTER_SET_NAME` can be derived from `COLLATION_NAME`. For example, if you say `SHOW FULL COLUMNS FROM t`, and you see in the `COLLATION_NAME` column a value of `latin1_swedish_ci`, the character set is what is before the first underscore: `latin1`.

Column information is also available from the `SHOW COLUMNS` statement. See Section 13.7.5.6, “SHOW COLUMNS Statement”. The following statements are nearly equivalent:

```
SELECT COLUMN_NAME, DATA_TYPE, IS_NULLABLE, COLUMN_DEFAULT
  FROM INFORMATION_SCHEMA.COLUMNS
```

```
  WHERE table_name = 'tbl_name'
  [AND table_schema = 'db_name']
  [AND column_name LIKE 'wild']

SHOW COLUMNS
  FROM tbl_name
  [FROM db_name]
  [LIKE 'wild']
```

21.6 The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table

The `COLUMN_PRIVILEGES` table provides information about column privileges. It takes its values from the `mysql.columns_priv` system table.

The `COLUMN_PRIVILEGES` table has these columns:

- `GRANTEE`

  The name of the account to which the privilege is granted, in `'user_name'@'host_name'` format.

- `TABLE_CATALOG`

  The name of the catalog to which the table containing the column belongs. This value is always `def`.

- `TABLE_SCHEMA`

  The name of the schema (database) to which the table containing the column belongs.

- `TABLE_NAME`

  The name of the table containing the column.

- `COLUMN_NAME`

  The name of the column.

- `PRIVILEGE_TYPE`

  The privilege granted. The value can be any privilege that can be granted at the column level; see Section 13.7.1.4, “GRANT Statement”. Each row lists a single privilege, so there is one row per column privilege held by the grantee.

  In the output from `SHOW FULL COLUMNS`, the privileges are all in one column and in lowercase, for example, `select,insert,update,references`. In `COLUMN_PRIVILEGES`, there is one privilege per row, in uppercase.

- `IS_GRANTABLE`

  `YES` if the user has the `GRANT OPTION` privilege, `NO` otherwise. The output does not list `GRANT OPTION` as a separate row with `PRIVILEGE_TYPE='GRANT OPTION'`.

Notes

- `COLUMN_PRIVILEGES` is a nonstandard `INFORMATION_SCHEMA` table.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.COLUMN_PRIVILEGES

SHOW GRANTS ...
```

21.7 The INFORMATION_SCHEMA ENGINES Table

The `ENGINES` table provides information about storage engines. This is particularly useful for checking whether a storage engine is supported, or to see what the default engine is.

The ENGINES table has these columns:

- ENGINE

  The name of the storage engine.

- SUPPORT

  The server's level of support for the storage engine, as shown in the following table.

Value	Meaning
YES	The engine is supported and is active
DEFAULT	Like YES, plus this is the default engine
NO	The engine is not supported
DISABLED	The engine is supported but has been disabled

  A value of NO means that the server was compiled without support for the engine, so it cannot be enabled at runtime.

  A value of DISABLED occurs either because the server was started with an option that disables the engine, or because not all options required to enable it were given. In the latter case, the error log should contain a reason indicating why the option is disabled. See Section 5.4.2, “The Error Log”.

  You might also see DISABLED for a storage engine if the server was compiled to support it, but was started with a `--skip-engine_name` option. For the NDB storage engine, DISABLED means the server was compiled with support for NDB Cluster, but was not started with the `--ndbcluster` option.

  All MySQL servers support MyISAM tables. It is not possible to disable MyISAM.

- COMMENT

  A brief description of the storage engine.

- TRANSACTIONS

  Whether the storage engine supports transactions.

- XA

  Whether the storage engine supports XA transactions.

- SAVEPOINTS

  Whether the storage engine supports savepoints.

Notes

- ENGINES is a nonstandard INFORMATION_SCHEMA table.

Storage engine information is also available from the SHOW ENGINES statement. See Section 13.7.5.17, “SHOW ENGINES Statement”. The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.ENGINES

SHOW ENGINES
```

21.8 The INFORMATION_SCHEMA EVENTS Table

The EVENTS table provides information about Event Manager events, which are discussed in Section 20.4, “Using the Event Scheduler”.

The EVENTS table has these columns:

- EVENT_CATALOG

  The name of the catalog to which the event belongs. This value is always def.

- EVENT_SCHEMA

  The name of the schema (database) to which the event belongs.

- EVENT_NAME

  The name of the event.

- DEFINER

  The account named in the DEFINER clause (often the user who created the event), in *'user_name'@'host_name'* format.

- TIME_ZONE

  The event time zone, which is the time zone used for scheduling the event and that is in effect within the event as it executes. The default value is SYSTEM.

- EVENT_BODY

  The language used for the statements in the event's DO clause. The value is always SQL.

- EVENT_DEFINITION

  The text of the SQL statement making up the event's DO clause; in other words, the statement executed by this event.

- EVENT_TYPE

  The event repetition type, either ONE TIME (transient) or RECURRING (repeating).

- EXECUTE_AT

  For a one-time event, this is the DATETIME value specified in the AT clause of the CREATE EVENT statement used to create the event, or of the last ALTER EVENT statement that modified the event. The value shown in this column reflects the addition or subtraction of any INTERVAL value included in the event's AT clause. For example, if an event is created using ON SCHEDULE AT CURRENT_TIMESTAMP + '1:6' DAY_HOUR, and the event was created at 2018-02-09 14:05:30, the value shown in this column would be '2018-02-10 20:05:30'. If the event's timing is determined by an EVERY clause instead of an AT clause (that is, if the event is recurring), the value of this column is NULL.

- INTERVAL_VALUE

  For a recurring event, the number of intervals to wait between event executions. For a transient event, the value is always NULL.

- INTERVAL_FIELD

  The time units used for the interval which a recurring event waits before repeating. For a transient event, the value is always NULL.

- SQL_MODE

  The SQL mode in effect when the event was created or altered, and under which the event executes. For the permitted values, see Section 5.1.10, “Server SQL Modes”.

- STARTS

The start date and time for a recurring event. This is displayed as a `DATETIME` value, and is `NULL` if no start date and time are defined for the event. For a transient event, this column is always `NULL`. For a recurring event whose definition includes a `STARTS` clause, this column contains the corresponding `DATETIME` value. As with the `EXECUTE_AT` column, this value resolves any expressions used. If there is no `STARTS` clause affecting the timing of the event, this column is `NULL`

- `ENDS`

  For a recurring event whose definition includes a `ENDS` clause, this column contains the corresponding `DATETIME` value. As with the `EXECUTE_AT` column, this value resolves any expressions used. If there is no `ENDS` clause affecting the timing of the event, this column is `NULL`.

- `STATUS`

  The event status. One of `ENABLED`, `DISABLED`, or `SLAVESIDE_DISABLED`. `SLAVESIDE_DISABLED` indicates that the creation of the event occurred on another MySQL server acting as a replication source and replicated to the current MySQL server which is acting as a replica, but the event is not presently being executed on the replica. For more information, see Section 17.4.1.16, “Replication of Invoked Features”. information.

- `ON_COMPLETION`

  One of the two values `PRESERVE` or `NOT PRESERVE`.

- `CREATED`

  The date and time when the event was created. This is a `TIMESTAMP` value.

- `LAST_ALTERED`

  The date and time when the event was last modified. This is a `TIMESTAMP` value. If the event has not been modified since its creation, this value is the same as the `CREATED` value.

- `LAST_EXECUTED`

  The date and time when the event last executed. This is a `DATETIME` value. If the event has never executed, this column is `NULL`.

  `LAST_EXECUTED` indicates when the event started. As a result, the `ENDS` column is never less than `LAST_EXECUTED`.

- `EVENT_COMMENT`

  The text of the comment, if the event has one. If not, this value is empty.

- `ORIGINATOR`

  The server ID of the MySQL server on which the event was created; used in replication. This value may be updated by `ALTER EVENT` to the server ID of the server on which that statement occurs, if executed on a replication source. The default value is 0.

- `CHARACTER_SET_CLIENT`

  The session value of the `character_set_client` system variable when the event was created.

- `COLLATION_CONNECTION`

  The session value of the `collation_connection` system variable when the event was created.

- `DATABASE_COLLATION`

  The collation of the database with which the event is associated.

Notes

- `EVENTS` is a nonstandard `INFORMATION_SCHEMA` table.
- Times in the `EVENTS` table are displayed using the event time zone, the current session time zone, or UTC, as described in Section 20.4.4, “Event Metadata”.
- For more information about `SLAVESIDE_DISABLED` and the `ORIGINATOR` column, see Section 17.4.1.16, “Replication of Invoked Features”.

Example

Suppose that the user `'jon'@'ghidora'` creates an event named `e_daily`, and then modifies it a few minutes later using an `ALTER EVENT` statement, as shown here:

```
DELIMITER |

CREATE EVENT e_daily
    ON SCHEDULE
      EVERY 1 DAY
    COMMENT 'Saves total number of sessions then clears the table each day'
    DO
      BEGIN
        INSERT INTO site_activity.totals (time, total)
          SELECT CURRENT_TIMESTAMP, COUNT(*)
            FROM site_activity.sessions;
        DELETE FROM site_activity.sessions;
      END |

DELIMITER ;

ALTER EVENT e_daily
    ENABLE;
```

(Note that comments can span multiple lines.)

This user can then run the following `SELECT` statement, and obtain the output shown:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.EVENTS
       WHERE EVENT_NAME = 'e_daily'
       AND EVENT_SCHEMA = 'myschema'\G
*************************** 1. row ***************************
       EVENT_CATALOG: def
        EVENT_SCHEMA: myschema
          EVENT_NAME: e_daily
             DEFINER: jon@ghidora
           TIME_ZONE: SYSTEM
          EVENT_BODY: SQL
    EVENT_DEFINITION: BEGIN
        INSERT INTO site_activity.totals (time, total)
          SELECT CURRENT_TIMESTAMP, COUNT(*)
            FROM site_activity.sessions;
        DELETE FROM site_activity.sessions;
      END
          EVENT_TYPE: RECURRING
          EXECUTE_AT: NULL
      INTERVAL_VALUE: 1
      INTERVAL_FIELD: DAY
            SQL_MODE: NO_ENGINE_SUBSTITUTION
              STARTS: 2018-08-08 11:06:34
                ENDS: NULL
              STATUS: ENABLED
       ON_COMPLETION: NOT PRESERVE
             CREATED: 2018-08-08 11:06:34
        LAST_ALTERED: 2018-08-08 11:06:34
       LAST_EXECUTED: 2018-08-08 16:06:34
       EVENT_COMMENT: Saves total number of sessions then clears the
                      table each day
          ORIGINATOR: 1
CHARACTER_SET_CLIENT: utf8
```

```
COLLATION_CONNECTION: utf8_general_ci
  DATABASE_COLLATION: latin1_swedish_ci
```

Event information is also available from the `SHOW EVENTS` statement. See Section 13.7.5.19, “SHOW EVENTS Statement”. The following statements are equivalent:

```
SELECT
    EVENT_SCHEMA, EVENT_NAME, DEFINER, TIME_ZONE, EVENT_TYPE, EXECUTE_AT,
    INTERVAL_VALUE, INTERVAL_FIELD, STARTS, ENDS, STATUS, ORIGINATOR,
    CHARACTER_SET_CLIENT, COLLATION_CONNECTION, DATABASE_COLLATION
  FROM INFORMATION_SCHEMA.EVENTS
  WHERE table_schema = 'db_name'
  [AND column_name LIKE 'wild']

SHOW EVENTS
  [FROM db_name]
  [LIKE 'wild']
```

21.9 The INFORMATION_SCHEMA GLOBAL_STATUS and SESSION_STATUS Tables

The `GLOBAL_STATUS` and `SESSION_STATUS` tables provide information about server status variables. Their contents correspond to the information produced by the `SHOW GLOBAL STATUS` and `SHOW SESSION STATUS` statements (see Section 13.7.5.36, “SHOW STATUS Statement”).

Notes

- The `VARIABLE_VALUE` column for each of these tables is defined as `VARCHAR(1024)`.

21.10 The INFORMATION_SCHEMA GLOBAL_VARIABLES and SESSION_VARIABLES Tables

The `GLOBAL_VARIABLES` and `SESSION_VARIABLES` tables provide information about server status variables. Their contents correspond to the information produced by the `SHOW GLOBAL VARIABLES` and `SHOW SESSION VARIABLES` statements (see Section 13.7.5.40, “SHOW VARIABLES Statement”).

Notes

- The `VARIABLE_VALUE` column for each of these tables is defined as `VARCHAR(1024)`. For variables with very long values that are not completely displayed, use `SELECT` as a workaround. For example:

  ```
  SELECT @@GLOBAL.innodb_data_file_path;
  ```

21.11 The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table

The `KEY_COLUMN_USAGE` table describes which key columns have constraints.

The `KEY_COLUMN_USAGE` table has these columns:

- `CONSTRAINT_CATALOG`

  The name of the catalog to which the constraint belongs. This value is always `def`.

- `CONSTRAINT_SCHEMA`

  The name of the schema (database) to which the constraint belongs.

- `CONSTRAINT_NAME`

  The name of the constraint.

- `TABLE_CATALOG`

  The name of the catalog to which the table belongs. This value is always `def`.

- `TABLE_SCHEMA`

  The name of the schema (database) to which the table belongs.

- `TABLE_NAME`

  The name of the table that has the constraint.

- `COLUMN_NAME`

  The name of the column that has the constraint.

  If the constraint is a foreign key, then this is the column of the foreign key, not the column that the foreign key references.

- `ORDINAL_POSITION`

  The column's position within the constraint, not the column's position within the table. Column positions are numbered beginning with 1.

- `POSITION_IN_UNIQUE_CONSTRAINT`

  `NULL` for unique and primary-key constraints. For foreign-key constraints, this column is the ordinal position in key of the table that is being referenced.

- `REFERENCED_TABLE_SCHEMA`

  The name of the schema (database) referenced by the constraint.

- `REFERENCED_TABLE_NAME`

  The name of the table referenced by the constraint.

- `REFERENCED_COLUMN_NAME`

  The name of the column referenced by the constraint.

Suppose that there are two tables name `t1` and `t3` that have the following definitions:

```
CREATE TABLE t1
(
    s1 INT,
    s2 INT,
    s3 INT,
    PRIMARY KEY(s3)
) ENGINE=InnoDB;

CREATE TABLE t3
(
    s1 INT,
    s2 INT,
    s3 INT,
    KEY(s1),
    CONSTRAINT CO FOREIGN KEY (s2) REFERENCES t1(s3)
) ENGINE=InnoDB;
```

For those two tables, the `KEY_COLUMN_USAGE` table has two rows:

- One row with `CONSTRAINT_NAME` = `'PRIMARY'`, `TABLE_NAME` = `'t1'`, `COLUMN_NAME` = `'s3'`, `ORDINAL_POSITION` = `1`, `POSITION_IN_UNIQUE_CONSTRAINT` = `NULL`.
- One row with `CONSTRAINT_NAME` = `'CO'`, `TABLE_NAME` = `'t3'`, `COLUMN_NAME` = `'s2'`, `ORDINAL_POSITION` = `1`, `POSITION_IN_UNIQUE_CONSTRAINT` = `1`.

21.12 The INFORMATION_SCHEMA OPTIMIZER_TRACE Table

The `OPTIMIZER_TRACE` table provides information produced by the optimizer tracing capability for traced statements. To enable tracking, use the `optimizer_trace` system variable. For details, see MySQL Internals: Tracing the Optimizer.

The `OPTIMIZER_TRACE` table has these columns:

- `QUERY`

  The text of the traced statement.

- `TRACE`

  The trace, in `JSON` format.

- `MISSING_BYTES_BEYOND_MAX_MEM_SIZE`

  Each remembered trace is a string that is extended as optimization progresses and appends data to it. The `optimizer_trace_max_mem_size` variable sets a limit on the total amount of memory used by all currently remembered traces. If this limit is reached, the current trace is not extended (and thus is incomplete), and the `MISSING_BYTES_BEYOND_MAX_MEM_SIZE` column shows the number of bytes missing from the trace.

- `INSUFFICIENT_PRIVILEGES`

  If a traced query uses views or stored routines that have `SQL SECURITY` with a value of `DEFINER`, it may be that a user other than the definer is denied from seeing the trace of the query. In that case, the trace is shown as empty and `INSUFFICIENT_PRIVILEGES` has a value of 1. Otherwise, the value is 0.

21.13 The INFORMATION_SCHEMA PARAMETERS Table

The `PARAMETERS` table provides information about parameters for stored routines (stored procedures and stored functions), and about return values for stored functions. The `PARAMETERS` table does not include built-in SQL functions or user-defined functions (UDFs). Parameter information is similar to the contents of the `param_list` column in the `mysql.proc` table.

The `PARAMETERS` table has these columns:

- `SPECIFIC_CATALOG`

  The name of the catalog to which the routine containing the parameter belongs. This value is always `def`.

- `SPECIFIC_SCHEMA`

  The name of the schema (database) to which the routine containing the parameter belongs.

- `SPECIFIC_NAME`

  The name of the routine containing the parameter.

- `ORDINAL_POSITION`

  For successive parameters of a stored procedure or function, the `ORDINAL_POSITION` values are 1, 2, 3, and so forth. For a stored function, there is also a row that applies to the function return value (as described by the `RETURNS` clause). The return value is not a true parameter, so the row that describes it has these unique characteristics:

  - The `ORDINAL_POSITION` value is 0.

  - The `PARAMETER_NAME` and `PARAMETER_MODE` values are `NULL` because the return value has no name and the mode does not apply.

- `PARAMETER_MODE`

  The mode of the parameter. This value is one of `IN`, `OUT`, or `INOUT`. For a stored function return value, this value is `NULL`.

- `PARAMETER_NAME`

  The name of the parameter. For a stored function return value, this value is `NULL`.

- `DATA_TYPE`

  The parameter data type.

  The `DATA_TYPE` value is the type name only with no other information. The `DTD_IDENTIFIER` value contains the type name and possibly other information such as the precision or length.

- `CHARACTER_MAXIMUM_LENGTH`

  For string parameters, the maximum length in characters.

- `CHARACTER_OCTET_LENGTH`

  For string parameters, the maximum length in bytes.

- `NUMERIC_PRECISION`

  For numeric parameters, the numeric precision.

- `NUMERIC_SCALE`

  For numeric parameters, the numeric scale.

- `DATETIME_PRECISION`

  For temporal parameters, the fractional seconds precision.

- `CHARACTER_SET_NAME`

  For character string parameters, the character set name.

- `COLLATION_NAME`

  For character string parameters, the collation name.

- `DTD_IDENTIFIER`

  The parameter data type.

  The `DATA_TYPE` value is the type name only with no other information. The `DTD_IDENTIFIER` value contains the type name and possibly other information such as the precision or length.

- `ROUTINE_TYPE`

  `PROCEDURE` for stored procedures, `FUNCTION` for stored functions.

21.14 The INFORMATION_SCHEMA PARTITIONS Table

The `PARTITIONS` table provides information about table partitions. Each row in this table corresponds to an individual partition or subpartition of a partitioned table. For more information about partitioning tables, see Chapter 19, *Partitioning*.

The PARTITIONS table has these columns:

- TABLE_CATALOG

  The name of the catalog to which the table belongs. This value is always def.

- TABLE_SCHEMA

  The name of the database to which the table belongs.

- TABLE_NAME

  The name of the table containing the partition.

- PARTITION_NAME

  The name of the partition.

- SUBPARTITION_NAME

  If the PARTITIONS table row represents a subpartition, the name of subpartition; otherwise NULL.

- PARTITION_ORDINAL_POSITION

  All partitions are indexed in the same order as they are defined, with 1 being the number assigned to the first partition. The indexing can change as partitions are added, dropped, and reorganized; the number shown is this column reflects the current order, taking into account any indexing changes.

- SUBPARTITION_ORDINAL_POSITION

  Subpartitions within a given partition are also indexed and reindexed in the same manner as partitions are indexed within a table.

- PARTITION_METHOD

  One of the values RANGE, LIST, HASH, LINEAR HASH, KEY, or LINEAR KEY; that is, one of the available partitioning types as discussed in Section 19.2, “Partitioning Types”.

- SUBPARTITION_METHOD

  One of the values HASH, LINEAR HASH, KEY, or LINEAR KEY; that is, one of the available subpartitioning types as discussed in Section 19.2.6, “Subpartitioning”.

- PARTITION_EXPRESSION

  The expression for the partitioning function used in the CREATE TABLE or ALTER TABLE statement that created the table's current partitioning scheme.

  For example, consider a partitioned table created in the test database using this statement:

```
CREATE TABLE tp (
    c1 INT,
    c2 INT,
    c3 VARCHAR(25)
)
PARTITION BY HASH(c1 + c2)
PARTITIONS 4;
```

  The PARTITION_EXPRESSION column in a PARTITIONS table row for a partition from this table displays c1 + c2, as shown here:

```
mysql> SELECT DISTINCT PARTITION_EXPRESSION
       FROM INFORMATION_SCHEMA.PARTITIONS
       WHERE TABLE_NAME='tp' AND TABLE_SCHEMA='test';
+----------------------+
```

```
| PARTITION_EXPRESSION |
+----------------------+
| c1 + c2              |
+----------------------+
```

- `SUBPARTITION_EXPRESSION`

  This works in the same fashion for the subpartitioning expression that defines the subpartitioning for a table as `PARTITION_EXPRESSION` does for the partitioning expression used to define a table's partitioning.

  If the table has no subpartitions, this column is `NULL`.

- `PARTITION_DESCRIPTION`

  This column is used for RANGE and LIST partitions. For a `RANGE` partition, it contains the value set in the partition's `VALUES LESS THAN` clause, which can be either an integer or `MAXVALUE`. For a `LIST` partition, this column contains the values defined in the partition's `VALUES IN` clause, which is a list of comma-separated integer values.

  For partitions whose `PARTITION_METHOD` is other than `RANGE` or `LIST`, this column is always `NULL`.

- `TABLE_ROWS`

  The number of table rows in the partition.

  For partitioned `InnoDB` tables, the row count given in the `TABLE_ROWS` column is only an estimated value used in SQL optimization, and may not always be exact.

  For `NDB` tables, you can also obtain this information using the `ndb_desc` utility.

- `AVG_ROW_LENGTH`

  The average length of the rows stored in this partition or subpartition, in bytes. This is the same as `DATA_LENGTH` divided by `TABLE_ROWS`.

  For `NDB` tables, you can also obtain this information using the `ndb_desc` utility.

- `DATA_LENGTH`

  The total length of all rows stored in this partition or subpartition, in bytes; that is, the total number of bytes stored in the partition or subpartition.

  For `NDB` tables, you can also obtain this information using the `ndb_desc` utility.

- `MAX_DATA_LENGTH`

  The maximum number of bytes that can be stored in this partition or subpartition.

  For `NDB` tables, you can also obtain this information using the `ndb_desc` utility.

- `INDEX_LENGTH`

  The length of the index file for this partition or subpartition, in bytes.

  For partitions of `NDB` tables, whether the tables use implicit or explicit partitioning, the `INDEX_LENGTH` column value is always 0. However, you can obtain equivalent information using the `ndb_desc` utility.

- `DATA_FREE`

  The number of bytes allocated to the partition or subpartition but not used.

  For `NDB` tables, you can also obtain this information using the `ndb_desc` utility.

- `CREATE_TIME`

  The time that the partition or subpartition was created.

  Prior to MySQL 5.6.25, for partitioned `InnoDB` tables, this column was always `NULL`. The correct creation time is shown in MySQL 5.6.25 and later. (Bug #17299181, Bug #69990)

- `UPDATE_TIME`

  The time that the partition or subpartition was last modified.

  For partitioned `InnoDB` tables, the value is always `NULL`.

- `CHECK_TIME`

  The last time that the table to which this partition or subpartition belongs was checked.

  For partitioned `InnoDB` tables, this column is always `NULL`.

- `CHECKSUM`

  The checksum value, if any; otherwise `NULL`.

- `PARTITION_COMMENT`

  The text of the comment, if the partition has one. If not, this value is empty.

  The maximum length for a partition comment is defined as 1024 characters, and the display width of the `PARTITION_COMMENT` column is also 1024, characters to match this limit.

- `NODEGROUP`

  This is the nodegroup to which the partition belongs. This is relevant only to NDB Cluster tables; otherwise, the value is always `0`.

- `TABLESPACE_NAME`

  The name of the tablespace to which the partition belongs. The value is always `DEFAULT`, unless the table uses the `NDB` storage engine (see the *Notes* at the end of this section).

Notes

- `PARTITIONS` is a nonstandard `INFORMATION_SCHEMA` table.
- A table using any storage engine other than `NDB` and which is not partitioned has one row in the `PARTITIONS` table. However, the values of the `PARTITION_NAME`, `SUBPARTITION_NAME`, `PARTITION_ORDINAL_POSITION`, `SUBPARTITION_ORDINAL_POSITION`, `PARTITION_METHOD`, `SUBPARTITION_METHOD`, `PARTITION_EXPRESSION`, `SUBPARTITION_EXPRESSION`, and `PARTITION_DESCRIPTION` columns are all `NULL`. Also, the `PARTITION_COMMENT` column in this case is blank.
- An `NDB` table which is not explicitly partitioned has one row in the `PARTITIONS` table for each data node in the NDB cluster. For each such row:
  - The `SUBPARTITION_NAME`, `SUBPARTITION_ORDINAL_POSITION`, `SUBPARTITION_METHOD`, `PARTITION_EXPRESSION`, `SUBPARTITION_EXPRESSION`, `CREATE_TIME`, `UPDATE_TIME`, `CHECK_TIME`, `CHECKSUM`, and `TABLESPACE_NAME` columns are all `NULL`.
  - The `PARTITION_METHOD` is always `KEY`.
  - The `NODEGROUP` column is `default`.
  - The `PARTITION_EXPRESSION` and `PARTITION_COMMENT` columns are empty.

21.15 The INFORMATION_SCHEMA PLUGINS Table

The `PLUGINS` table provides information about server plugins.

The `PLUGINS` table has these columns:

- `PLUGIN_NAME`

  The name used to refer to the plugin in statements such as `INSTALL PLUGIN` and `UNINSTALL PLUGIN`.

- `PLUGIN_VERSION`

  The version from the plugin's general type descriptor.

- `PLUGIN_STATUS`

  The plugin status, one of `ACTIVE`, `INACTIVE`, `DISABLED`, or `DELETED`.

- `PLUGIN_TYPE`

  The type of plugin, such as `STORAGE ENGINE`, `INFORMATION_SCHEMA`, or `AUTHENTICATION`.

- `PLUGIN_TYPE_VERSION`

  The version from the plugin's type-specific descriptor.

- `PLUGIN_LIBRARY`

  The name of the plugin shared library file. This is the name used to refer to the plugin file in statements such as `INSTALL PLUGIN` and `UNINSTALL PLUGIN`. This file is located in the directory named by the `plugin_dir` system variable. If the library name is `NULL`, the plugin is compiled in and cannot be uninstalled with `UNINSTALL PLUGIN`.

- `PLUGIN_LIBRARY_VERSION`

  The plugin API interface version.

- `PLUGIN_AUTHOR`

  The plugin author.

- `PLUGIN_DESCRIPTION`

  A short description of the plugin.

- `PLUGIN_LICENSE`

  How the plugin is licensed (for example, `GPL`).

- `LOAD_OPTION`

  How the plugin was loaded. The value is `OFF`, `ON`, `FORCE`, or `FORCE_PLUS_PERMANENT`. See Section 5.5.1, “Installing and Uninstalling Plugins”.

Notes

- `PLUGINS` is a nonstandard `INFORMATION_SCHEMA` table.
- For plugins installed with `INSTALL PLUGIN`, the `PLUGIN_NAME` and `PLUGIN_LIBRARY` values are also registered in the `mysql.plugin` table.
- For information about plugin data structures that form the basis of the information in the `PLUGINS` table, see The MySQL Plugin API.

Plugin information is also available from the `SHOW PLUGINS` statement. See Section 13.7.5.26, “SHOW PLUGINS Statement”. These statements are equivalent:

```
SELECT
  PLUGIN_NAME, PLUGIN_STATUS, PLUGIN_TYPE,
  PLUGIN_LIBRARY, PLUGIN_LICENSE
FROM INFORMATION_SCHEMA.PLUGINS;

SHOW PLUGINS;
```

21.16 The INFORMATION_SCHEMA PROCESSLIST Table

The MySQL process list indicates the operations currently being performed by the set of threads executing within the server. The `PROCESSLIST` table is one source of process information. For a comparison of this table with other sources, see Sources of Process Information.

The `PROCESSLIST` table has these columns:

- `ID`

  The connection identifier. This is the same value displayed in the `Id` column of the `SHOW PROCESSLIST` statement, displayed in the `PROCESSLIST_ID` column of the Performance Schema `threads` table, and returned by the `CONNECTION_ID()` function within the thread.

- `USER`

  The MySQL user who issued the statement. A value of `system user` refers to a nonclient thread spawned by the server to handle tasks internally, for example, a delayed-row handler thread or an I/O or SQL thread used on replica hosts. For `system user`, there is no host specified in the `Host` column. `unauthenticated user` refers to a thread that has become associated with a client connection but for which authentication of the client user has not yet occurred. `event_scheduler` refers to the thread that monitors scheduled events (see Section 20.4, “Using the Event Scheduler”).

- `HOST`

  The host name of the client issuing the statement (except for `system user`, for which there is no host). The host name for TCP/IP connections is reported in *`host_name:client_port`* format to make it easier to determine which client is doing what.

- `DB`

  The default database for the thread, or `NULL` if none has been selected.

- `COMMAND`

  The type of command the thread is executing on behalf of the client, or `Sleep` if the session is idle. For descriptions of thread commands, see Section 8.14, “Examining Server Thread (Process) Information”. The value of this column corresponds to the `COM_`*`xxx`* commands of the client/server protocol and `Com_`*`xxx`* status variables. See Section 5.1.9, “Server Status Variables”.

- `TIME`

  The time in seconds that the thread has been in its current state. For a replica SQL thread, the value is the number of seconds between the timestamp of the last replicated event and the real time of the replica host. See Section 17.2.1, “Replication Threads”.

- `STATE`

  An action, event, or state that indicates what the thread is doing. For descriptions of `STATE` values, see Section 8.14, “Examining Server Thread (Process) Information”.

  Most states correspond to very quick operations. If a thread stays in a given state for many seconds, there might be a problem that needs to be investigated.

- `INFO`

  The statement the thread is executing, or `NULL` if it is executing no statement. The statement might be the one sent to the server, or an innermost statement if the statement executes other statements. For example, if a `CALL` statement executes a stored procedure that is executing a `SELECT` statement, the `INFO` value shows the `SELECT` statement.

Notes

- `PROCESSLIST` is a nonstandard `INFORMATION_SCHEMA` table.
- Like the output from the `SHOW PROCESSLIST` statement, the `PROCESSLIST` table provides information about all threads, even those belonging to other users, if you have the `PROCESS` privilege. Otherwise (without the `PROCESS` privilege), nonanonymous users have access to information about their own threads but not threads for other users, and anonymous users have no access to thread information.
- If an SQL statement refers to the `PROCESSLIST` table, MySQL populates the entire table once, when statement execution begins, so there is read consistency during the statement. There is no read consistency for a multi-statement transaction.

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST

SHOW FULL PROCESSLIST
```

21.17 The INFORMATION_SCHEMA PROFILING Table

The `PROFILING` table provides statement profiling information. Its contents correspond to the information produced by the `SHOW PROFILE` and `SHOW PROFILES` statements (see Section 13.7.5.31, “SHOW PROFILE Statement”). The table is empty unless the `profiling` session variable is set to 1.

The `PROFILING` table has these columns:

- `QUERY_ID`

  A numeric statement identifier.

- `SEQ`

  A sequence number indicating the display order for rows with the same `QUERY_ID` value.

- `STATE`

  The profiling state to which the row measurements apply.

- `DURATION`

  How long statement execution remained in the given state, in seconds.

- `CPU_USER`, `CPU_SYSTEM`

  User and system CPU use, in seconds.

- `CONTEXT_VOLUNTARY`, `CONTEXT_INVOLUNTARY`

  How many voluntary and involuntary context switches occurred.

- `BLOCK_OPS_IN`, `BLOCK_OPS_OUT`

  The number of block input and output operations.

- `MESSAGES_SENT`, `MESSAGES_RECEIVED`

  The number of communication messages sent and received.

- `PAGE_FAULTS_MAJOR`, `PAGE_FAULTS_MINOR`

  The number of major and minor page faults.

- `SWAPS`

  How many swaps occurred.

- `SOURCE_FUNCTION`, `SOURCE_FILE`, and `SOURCE_LINE`

  Information indicating where in the source code the profiled state executes.

Notes

- `PROFILING` is a nonstandard `INFORMATION_SCHEMA` table.

Profiling information is also available from the `SHOW PROFILE` and `SHOW PROFILES` statements. See Section 13.7.5.31, “SHOW PROFILE Statement”. For example, the following queries are equivalent:

```
SHOW PROFILE FOR QUERY 2;

SELECT STATE, FORMAT(DURATION, 6) AS DURATION
FROM INFORMATION_SCHEMA.PROFILING
WHERE QUERY_ID = 2 ORDER BY SEQ;
```

21.18 The INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS Table

The `REFERENTIAL_CONSTRAINTS` table provides information about foreign keys.

The `REFERENTIAL_CONSTRAINTS` table has these columns:

- `CONSTRAINT_CATALOG`

  The name of the catalog to which the constraint belongs. This value is always `def`.

- `CONSTRAINT_SCHEMA`

  The name of the schema (database) to which the constraint belongs.

- `CONSTRAINT_NAME`

  The name of the constraint.

- `UNIQUE_CONSTRAINT_CATALOG`

  The name of the catalog containing the unique constraint that the constraint references. This value is always `def`.

- `UNIQUE_CONSTRAINT_SCHEMA`

  The name of the schema (database) containing the unique constraint that the constraint references.

- `UNIQUE_CONSTRAINT_NAME`

  The name of the unique constraint that the constraint references.

- `MATCH_OPTION`

  The value of the constraint `MATCH` attribute. The only valid value at this time is `NONE`.

- `UPDATE_RULE`

  The value of the constraint `ON UPDATE` attribute. The possible values are `CASCADE`, `SET NULL`, `SET DEFAULT`, `RESTRICT`, `NO ACTION`.

- `DELETE_RULE`

  The value of the constraint `ON DELETE` attribute. The possible values are `CASCADE`, `SET NULL`, `SET DEFAULT`, `RESTRICT`, `NO ACTION`.

- `TABLE_NAME`

  The name of the table. This value is the same as in the `TABLE_CONSTRAINTS` table.

- `REFERENCED_TABLE_NAME`

  The name of the table referenced by the constraint.

21.19 The INFORMATION_SCHEMA ROUTINES Table

The `ROUTINES` table provides information about stored routines (stored procedures and stored functions). The `ROUTINES` table does not include built-in SQL functions or user-defined functions (UDFs).

The column named "`mysql.proc` Name" indicates the `mysql.proc` table column that corresponds to the `INFORMATION_SCHEMA ROUTINES` table column, if any.

The `ROUTINES` table has these columns:

- `SPECIFIC_NAME`

  The name of the routine.

- `ROUTINE_CATALOG`

  The name of the catalog to which the routine belongs. This value is always `def`.

- `ROUTINE_SCHEMA`

  The name of the schema (database) to which the routine belongs.

- `ROUTINE_NAME`

  The name of the routine.

- `ROUTINE_TYPE`

  `PROCEDURE` for stored procedures, `FUNCTION` for stored functions.

- `DATA_TYPE`

  If the routine is a stored function, the return value data type. If the routine is a stored procedure, this value is empty.

  The `DATA_TYPE` value is the type name only with no other information. The `DTD_IDENTIFIER` value contains the type name and possibly other information such as the precision or length.

- `CHARACTER_MAXIMUM_LENGTH`

  For stored function string return values, the maximum length in characters. If the routine is a stored procedure, this value is `NULL`.

- `CHARACTER_OCTET_LENGTH`

For stored function string return values, the maximum length in bytes. If the routine is a stored procedure, this value is `NULL`.

- `NUMERIC_PRECISION`

  For stored function numeric return values, the numeric precision. If the routine is a stored procedure, this value is `NULL`.

- `NUMERIC_SCALE`

  For stored function numeric return values, the numeric scale. If the routine is a stored procedure, this value is `NULL`.

- `DATETIME_PRECISION`

  For stored function temporal return values, the fractional seconds precision. If the routine is a stored procedure, this value is `NULL`.

- `CHARACTER_SET_NAME`

  For stored function character string return values, the character set name. If the routine is a stored procedure, this value is `NULL`.

- `COLLATION_NAME`

  For stored function character string return values, the collation name. If the routine is a stored procedure, this value is `NULL`.

- `DTD_IDENTIFIER`

  If the routine is a stored function, the return value data type. If the routine is a stored procedure, this value is empty.

  The `DATA_TYPE` value is the type name only with no other information. The `DTD_IDENTIFIER` value contains the type name and possibly other information such as the precision or length.

- `ROUTINE_BODY`

  The language used for the routine definition. This value is always `SQL`.

- `ROUTINE_DEFINITION`

  The text of the SQL statement executed by the routine.

- `EXTERNAL_NAME`

  This value is always `NULL`.

- `EXTERNAL_LANGUAGE`

  The language of the stored routine. MySQL calculates `EXTERNAL_LANGUAGE` thus:

  - If `mysql.proc.language='SQL'`, `EXTERNAL_LANGUAGE` is `NULL`
  - Otherwise, `EXTERNAL_LANGUAGE` is what is in `mysql.proc.language`. However, we do not have external languages yet, so it is always `NULL`.

- `PARAMETER_STYLE`

  This value is always `SQL`.

- `IS_DETERMINISTIC`

YES or NO, depending on whether the routine is defined with the DETERMINISTIC characteristic.

- SQL_DATA_ACCESS

  The data access characteristic for the routine. The value is one of CONTAINS SQL, NO SQL, READS SQL DATA, or MODIFIES SQL DATA.

- SQL_PATH

  This value is always NULL.

- SECURITY_TYPE

  The routine SQL SECURITY characteristic. The value is one of DEFINER or INVOKER.

- CREATED

  The date and time when the routine was created. This is a TIMESTAMP value.

- LAST_ALTERED

  The date and time when the routine was last modified. This is a TIMESTAMP value. If the routine has not been modified since its creation, this value is the same as the CREATED value.

- SQL_MODE

  The SQL mode in effect when the routine was created or altered, and under which the routine executes. For the permitted values, see Section 5.1.10, “Server SQL Modes”.

- ROUTINE_COMMENT

  The text of the comment, if the routine has one. If not, this value is empty.

- DEFINER

  The account named in the DEFINER clause (often the user who created the routine), in *'user_name'@'host_name'* format.

- CHARACTER_SET_CLIENT

  The session value of the character_set_client system variable when the routine was created.

- COLLATION_CONNECTION

  The session value of the collation_connection system variable when the routine was created.

- DATABASE_COLLATION

  The collation of the database with which the routine is associated.

Notes

- To see information about a routine, you must be the user named in the routine DEFINER clause or have SELECT access to the mysql.proc table. If you do not have privileges for the routine itself, the value displayed for the ROUTINE_DEFINITION column is NULL.
- Information about stored function return values is also available in the PARAMETERS table. The return value row for a stored function can be identified as the row that has an ORDINAL_POSITION value of 0.

21.20 The INFORMATION_SCHEMA SCHEMATA Table

A schema is a database, so the SCHEMATA table provides information about databases.

The SCHEMATA table has these columns:

- CATALOG_NAME

  The name of the catalog to which the schema belongs. This value is always def.

- SCHEMA_NAME

  The name of the schema.

- DEFAULT_CHARACTER_SET_NAME

  The schema default character set.

- DEFAULT_COLLATION_NAME

  The schema default collation.

- SQL_PATH

  This value is always NULL.

Schema names are also available from the SHOW DATABASES statement. See Section 13.7.5.15, "SHOW DATABASES Statement". The following statements are equivalent:

```
SELECT SCHEMA_NAME AS `Database`
  FROM INFORMATION_SCHEMA.SCHEMATA
  [WHERE SCHEMA_NAME LIKE 'wild']

SHOW DATABASES
  [LIKE 'wild']
```

You see only those databases for which you have some kind of privilege, unless you have the global SHOW DATABASES privilege.

Caution

Because a global privilege is considered a privilege for all databases, *any* global privilege enables a user to see all database names with SHOW DATABASES or by examining the INFORMATION_SCHEMA SCHEMATA table.

21.21 The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table

The SCHEMA_PRIVILEGES table provides information about schema (database) privileges. It takes its values from the mysql.db system table.

The SCHEMA_PRIVILEGES table has these columns:

- GRANTEE

  The name of the account to which the privilege is granted, in *'user_name'@'host_name'* format.

- TABLE_CATALOG

  The name of the catalog to which the schema belongs. This value is always def.

- TABLE_SCHEMA

  The name of the schema.

- PRIVILEGE_TYPE

The privilege granted. The value can be any privilege that can be granted at the schema level; see Section 13.7.1.4, “GRANT Statement”. Each row lists a single privilege, so there is one row per schema privilege held by the grantee.

- `IS_GRANTABLE`

  `YES` if the user has the `GRANT OPTION` privilege, `NO` otherwise. The output does not list `GRANT OPTION` as a separate row with `PRIVILEGE_TYPE='GRANT OPTION'`.

Notes

- `SCHEMA_PRIVILEGES` is a nonstandard `INFORMATION_SCHEMA` table.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.SCHEMA_PRIVILEGES

SHOW GRANTS ...
```

21.22 The INFORMATION_SCHEMA STATISTICS Table

The `STATISTICS` table provides information about table indexes.

The `STATISTICS` table has these columns:

- `TABLE_CATALOG`

  The name of the catalog to which the table containing the index belongs. This value is always `def`.

- `TABLE_SCHEMA`

  The name of the schema (database) to which the table containing the index belongs.

- `TABLE_NAME`

  The name of the table containing the index.

- `NON_UNIQUE`

  0 if the index cannot contain duplicates, 1 if it can.

- `INDEX_SCHEMA`

  The name of the schema (database) to which the index belongs.

- `INDEX_NAME`

  The name of the index. If the index is the primary key, the name is always `PRIMARY`.

- `SEQ_IN_INDEX`

  The column sequence number in the index, starting with 1.

- `COLUMN_NAME`

  The column name. See also the description for the `EXPRESSION` column.

- `COLLATION`

  How the column is sorted in the index. This can have values `A` (ascending), `D` (descending), or `NULL` (not sorted).

- `CARDINALITY`

An estimate of the number of unique values in the index. To update this number, run `ANALYZE TABLE` or (for `MyISAM` tables) `myisamchk -a`.

`CARDINALITY` is counted based on statistics stored as integers, so the value is not necessarily exact even for small tables. The higher the cardinality, the greater the chance that MySQL uses the index when doing joins.

- `SUB_PART`

  The index prefix. That is, the number of indexed characters if the column is only partly indexed, `NULL` if the entire column is indexed.

  Note

  Prefix *limits* are measured in bytes. However, prefix *lengths* for index specifications in `CREATE TABLE`, `ALTER TABLE`, and `CREATE INDEX` statements are interpreted as number of characters for nonbinary string types (`CHAR`, `VARCHAR`, `TEXT`) and number of bytes for binary string types (`BINARY`, `VARBINARY`, `BLOB`). Take this into account when specifying a prefix length for a nonbinary string column that uses a multibyte character set.

  For additional information about index prefixes, see Section 8.3.4, “Column Indexes”, and Section 13.1.13, “CREATE INDEX Statement”.

- `PACKED`

  Indicates how the key is packed. `NULL` if it is not.

- `NULLABLE`

  Contains `YES` if the column may contain `NULL` values and `''` if not.

- `INDEX_TYPE`

  The index method used (`BTREE`, `FULLTEXT`, `HASH`, `RTREE`).

- `COMMENT`

  Information about the index not described in its own column, such as `disabled` if the index is disabled.

- `INDEX_COMMENT`

  Any comment provided for the index with a `COMMENT` attribute when the index was created.

Notes

- There is no standard `INFORMATION_SCHEMA` table for indexes. The MySQL column list is similar to what SQL Server 2000 returns for `sp_statistics`, except that `QUALIFIER` and `OWNER` are replaced with `CATALOG` and `SCHEMA`, respectively.

Information about table indexes is also available from the `SHOW INDEX` statement. See Section 13.7.5.23, “SHOW INDEX Statement”. The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.STATISTICS
  WHERE table_name = 'tbl_name'
  AND table_schema = 'db_name'

SHOW INDEX
  FROM tbl_name
  FROM db_name
```

21.23 The INFORMATION_SCHEMA TABLES Table

The `TABLES` table provides information about tables in databases.

The `TABLES` table has these columns:

- `TABLE_CATALOG`

  The name of the catalog to which the table belongs. This value is always `def`.

- `TABLE_SCHEMA`

  The name of the schema (database) to which the table belongs.

- `TABLE_NAME`

  The name of the table.

- `TABLE_TYPE`

  `BASE TABLE` for a table, `VIEW` for a view, or `SYSTEM VIEW` for an `INFORMATION_SCHEMA` table.

  The `TABLES` table does not list `TEMPORARY` tables.

- `ENGINE`

  The storage engine for the table. See Chapter 14, *The InnoDB Storage Engine*, and Chapter 15, *Alternative Storage Engines*.

  For partitioned tables, `ENGINE` shows the name of the storage engine used by all partitions.

- `VERSION`

  The version number of the table's `.frm` file.

- `ROW_FORMAT`

  The row-storage format (`Fixed`, `Dynamic`, `Compressed`, `Redundant`, `Compact`). For `MyISAM` tables, `Dynamic` corresponds to what `myisamchk -dvv` reports as `Packed`. `InnoDB` table format is either `Redundant` or `Compact` when using the `Antelope` file format, or `Compressed` or `Dynamic` when using the `Barracuda` file format.

- `TABLE_ROWS`

  The number of rows. Some storage engines, such as `MyISAM`, store the exact count. For other storage engines, such as `InnoDB`, this value is an approximation, and may vary from the actual value by as much as 40% to 50%. In such cases, use `SELECT COUNT(*)` to obtain an accurate count.

  `TABLE_ROWS` is `NULL` for `INFORMATION_SCHEMA` tables.

  For `InnoDB` tables, the row count is only a rough estimate used in SQL optimization. (This is also true if the `InnoDB` table is partitioned.)

- `AVG_ROW_LENGTH`

  The average row length.

  Refer to the notes at the end of this section for related information.

- `DATA_LENGTH`

  For `MyISAM`, `DATA_LENGTH` is the length of the data file, in bytes.

  For `InnoDB`, `DATA_LENGTH` is the approximate amount of space allocated for the clustered index, in bytes. Specifically, it is the clustered index size, in pages, multiplied by the `InnoDB` page size.

Refer to the notes at the end of this section for information regarding other storage engines.

- `MAX_DATA_LENGTH`

  For `MyISAM`, `MAX_DATA_LENGTH` is maximum length of the data file. This is the total number of bytes of data that can be stored in the table, given the data pointer size used.

  Unused for `InnoDB`.

  Refer to the notes at the end of this section for information regarding other storage engines.

- `INDEX_LENGTH`

  For `MyISAM`, `INDEX_LENGTH` is the length of the index file, in bytes.

  For `InnoDB`, `INDEX_LENGTH` is the approximate amount of space allocated for non-clustered indexes, in bytes. Specifically, it is the sum of non-clustered index sizes, in pages, multiplied by the `InnoDB` page size.

  Refer to the notes at the end of this section for information regarding other storage engines.

- `DATA_FREE`

  The number of allocated but unused bytes.

  `InnoDB` tables report the free space of the tablespace to which the table belongs. For a table located in the shared tablespace, this is the free space of the shared tablespace. If you are using multiple tablespaces and the table has its own tablespace, the free space is for only that table. Free space means the number of bytes in completely free extents minus a safety margin. Even if free space displays as 0, it may be possible to insert rows as long as new extents need not be allocated.

  For NDB Cluster, `DATA_FREE` shows the space allocated on disk for, but not used by, a Disk Data table or fragment on disk. (In-memory data resource usage is reported by the `DATA_LENGTH` column.)

  For partitioned tables, this value is only an estimate and may not be absolutely correct. A more accurate method of obtaining this information in such cases is to query the `INFORMATION_SCHEMA PARTITIONS` table, as shown in this example:

```
SELECT SUM(DATA_FREE)
    FROM  INFORMATION_SCHEMA.PARTITIONS
    WHERE TABLE_SCHEMA = 'mydb'
    AND   TABLE_NAME   = 'mytable';
```

  For more information, see Section 21.14, “The INFORMATION_SCHEMA PARTITIONS Table”.

- `AUTO_INCREMENT`

  The next `AUTO_INCREMENT` value.

- `CREATE_TIME`

  When the table was created.

  Prior to MySQL 5.6.25, for partitioned `InnoDB` tables, the `CREATE_TIME` column shows `NULL`. This column shows the correct table creation time for such tables in MySQL 5.6.25 and later. (Bug #17299181, Bug #69990)

- `UPDATE_TIME`

  When the data file was last updated. For some storage engines, this value is `NULL`. For example, `InnoDB` stores multiple tables in its system tablespace and the data file timestamp does not apply.

Even with file-per-table mode with each `InnoDB` table in a separate `.ibd` file, change buffering can delay the write to the data file, so the file modification time is different from the time of the last insert, update, or delete. For `MyISAM`, the data file timestamp is used; however, on Windows the timestamp is not updated by updates, so the value is inaccurate.

For partitioned `InnoDB` tables, `UPDATE_TIME` is always `NULL`.

- `CHECK_TIME`

  When the table was last checked. Not all storage engines update this time, in which case, the value is always `NULL`.

  For partitioned `InnoDB` tables, `CHECK_TIME` is always `NULL`.

- `TABLE_COLLATION`

  The table default collation. The output does not explicitly list the table default character set, but the collation name begins with the character set name.

- `CHECKSUM`

  The live checksum value, if any.

- `CREATE_OPTIONS`

  Extra options used with `CREATE TABLE`.

  `CREATE_OPTIONS` shows `partitioned` for partitioned tables.

  When creating a table with strict mode disabled, the storage engine's default row format is used if the specified row format is not supported. The actual row format of the table is reported in the `ROW_FORMAT` column. `CREATE_OPTIONS` shows the row format that was specified in the `CREATE TABLE` statement.

  When altering the storage engine of a table, table options that are not applicable to the new storage engine are retained in the table definition to enable reverting the table with its previously defined options to the original storage engine, if necessary. The `CREATE_OPTIONS` column may show retained options.

- `TABLE_COMMENT`

  The comment used when creating the table (or information as to why MySQL could not access the table information).

Notes

- For `NDB` tables, the output of this statement shows appropriate values for the `AVG_ROW_LENGTH` and `DATA_LENGTH` columns, with the exception that `BLOB` columns are not taken into account.
- For `NDB` tables, `DATA_LENGTH` includes data stored in main memory only; the `MAX_DATA_LENGTH` and `DATA_FREE` columns apply to Disk Data.
- For NDB Cluster Disk Data tables, `MAX_DATA_LENGTH` shows the space allocated for the disk part of a Disk Data table or fragment. (In-memory data resource usage is reported by the `DATA_LENGTH` column.)
- For `MEMORY` tables, the `DATA_LENGTH`, `MAX_DATA_LENGTH`, and `INDEX_LENGTH` values approximate the actual amount of allocated memory. The allocation algorithm reserves memory in large amounts to reduce the number of allocation operations.
- For views, all `TABLES` columns are `NULL` except that `TABLE_NAME` indicates the view name and `TABLE_COMMENT` says `VIEW`.

Table information is also available from the `SHOW TABLE STATUS` and `SHOW TABLES` statements. See Section 13.7.5.37, “SHOW TABLE STATUS Statement”, and Section 13.7.5.38, “SHOW TABLES Statement”. The following statements are equivalent:

```
SELECT
    TABLE_NAME, ENGINE, VERSION, ROW_FORMAT, TABLE_ROWS, AVG_ROW_LENGTH,
    DATA_LENGTH, MAX_DATA_LENGTH, INDEX_LENGTH, DATA_FREE, AUTO_INCREMENT,
    CREATE_TIME, UPDATE_TIME, CHECK_TIME, TABLE_COLLATION, CHECKSUM,
    CREATE_OPTIONS, TABLE_COMMENT
  FROM INFORMATION_SCHEMA.TABLES
  WHERE table_schema = 'db_name'
  [AND table_name LIKE 'wild']

SHOW TABLE STATUS
  FROM db_name
  [LIKE 'wild']
```

The following statements are equivalent:

```
SELECT
  TABLE_NAME, TABLE_TYPE
  FROM INFORMATION_SCHEMA.TABLES
  WHERE table_schema = 'db_name'
  [AND table_name LIKE 'wild']

SHOW FULL TABLES
  FROM db_name
  [LIKE 'wild']
```

21.24 The INFORMATION_SCHEMA TABLESPACES Table

This table is unused. Other `INFORMATION_SCHEMA` tables may provide related information:

- For `NDB`, the `INFORMATION_SCHEMA FILES` table provides tablespace-related information.
- For `InnoDB`, the `INFORMATION_SCHEMA INNODB_SYS_TABLESPACES` and `INNODB_SYS_DATAFILES` tables provide tablespace metadata.

21.25 The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table

The `TABLE_CONSTRAINTS` table describes which tables have constraints.

The `TABLE_CONSTRAINTS` table has these columns:

- `CONSTRAINT_CATALOG`

  The name of the catalog to which the constraint belongs. This value is always `def`.

- `CONSTRAINT_SCHEMA`

  The name of the schema (database) to which the constraint belongs.

- `TABLE_SCHEMA`

  The name of the schema (database) to which the table belongs.

- `TABLE_NAME`

  The name of the table.

- The `CONSTRAINT_TYPE`

  The type of constraint. The value can be `UNIQUE`, `PRIMARY KEY`, `FOREIGN KEY`, or `CHECK`. This is a `CHAR` (not `ENUM`) column. The `CHECK` value is not available until MySQL supports `CHECK`.

  The `UNIQUE` and `PRIMARY KEY` information is about the same as what you get from the `Key_name` column in the output from `SHOW INDEX` when the `Non_unique` column is `0`.

21.26 The INFORMATION_SCHEMA TABLE_PRIVILEGES Table

The `TABLE_PRIVILEGES` table provides information about table privileges. It takes its values from the `mysql.tables_priv` system table.

The `TABLE_PRIVILEGES` table has these columns:

- `GRANTEE`

  The name of the account to which the privilege is granted, in *`'user_name'@'host_name'`* format.

- `TABLE_CATALOG`

  The name of the catalog to which the table belongs. This value is always `def`.

- `TABLE_SCHEMA`

  The name of the schema (database) to which the table belongs.

- `TABLE_NAME`

  The name of the table.

- `PRIVILEGE_TYPE`

  The privilege granted. The value can be any privilege that can be granted at the table level; see Section 13.7.1.4, “GRANT Statement”. Each row lists a single privilege, so there is one row per table privilege held by the grantee.

- `IS_GRANTABLE`

  `YES` if the user has the `GRANT OPTION` privilege, `NO` otherwise. The output does not list `GRANT OPTION` as a separate row with `PRIVILEGE_TYPE='GRANT OPTION'`.

Notes

- `TABLE_PRIVILEGES` is a nonstandard `INFORMATION_SCHEMA` table.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES

SHOW GRANTS ...
```

21.27 The INFORMATION_SCHEMA TRIGGERS Table

The `TRIGGERS` table provides information about triggers. To see information about a table's triggers, you must have the `TRIGGER` privilege for the table.

The `TRIGGERS` table has these columns:

- `TRIGGER_CATALOG`

  The name of the catalog to which the trigger belongs. This value is always `def`.

- `TRIGGER_SCHEMA`

  The name of the schema (database) to which the trigger belongs.

- `TRIGGER_NAME`

  The name of the trigger.

- `EVENT_MANIPULATION`

The trigger event. This is the type of operation on the associated table for which the trigger activates. The value is `INSERT` (a row was inserted), `DELETE` (a row was deleted), or `UPDATE` (a row was modified).

- `EVENT_OBJECT_CATALOG`, `EVENT_OBJECT_SCHEMA`, and `EVENT_OBJECT_TABLE`

  As noted in Section 20.3, “Using Triggers”, every trigger is associated with exactly one table. These columns indicate the catalog and schema (database) in which this table occurs, and the table name, respectively. The `EVENT_OBJECT_CATALOG` value is always `def`.

- `ACTION_ORDER`

  The ordinal position of the trigger's action within the list of all similar triggers on the same table. This value is always `0` because it is not possible to have more than one trigger with the same `EVENT_MANIPULATION` and `ACTION_TIMING` on the same table.

- `ACTION_CONDITION`

  This value is always `NULL`.

- `ACTION_STATEMENT`

  The trigger body; that is, the statement executed when the trigger activates. This text uses UTF-8 encoding.

- `ACTION_ORIENTATION`

  This value is always `ROW`.

- `ACTION_TIMING`

  Whether the trigger activates before or after the triggering event. The value is `BEFORE` or `AFTER`.

- `ACTION_REFERENCE_OLD_TABLE`

  This value is always `NULL`.

- `ACTION_REFERENCE_NEW_TABLE`

  This value is always `NULL`.

- `ACTION_REFERENCE_OLD_ROW` and `ACTION_REFERENCE_NEW_ROW`

  The old and new column identifiers, respectively. The `ACTION_REFERENCE_OLD_ROW` value is always `OLD` and the `ACTION_REFERENCE_NEW_ROW` value is always `NEW`.

- `CREATED`

  This value is always `NULL`.

- `SQL_MODE`

  The SQL mode in effect when the trigger was created, and under which the trigger executes. For the permitted values, see Section 5.1.10, “Server SQL Modes”.

- `DEFINER`

  The account named in the `DEFINER` clause (often the user who created the trigger), in `'user_name'@'host_name'` format.

- `CHARACTER_SET_CLIENT`

  The session value of the `character_set_client` system variable when the trigger was created.

- `COLLATION_CONNECTION`

  The session value of the `collation_connection` system variable when the trigger was created.

- `DATABASE_COLLATION`

  The collation of the database with which the trigger is associated.

Example

The following example uses the `ins_sum` trigger defined in Section 20.3, “Using Triggers”:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.TRIGGERS
       WHERE TRIGGER_SCHEMA='test' AND TRIGGER_NAME='ins_sum'\G
*************************** 1. row ***************************
           TRIGGER_CATALOG: def
            TRIGGER_SCHEMA: test
              TRIGGER_NAME: ins_sum
        EVENT_MANIPULATION: INSERT
      EVENT_OBJECT_CATALOG: def
       EVENT_OBJECT_SCHEMA: test
        EVENT_OBJECT_TABLE: account
              ACTION_ORDER: 0
          ACTION_CONDITION: NULL
          ACTION_STATEMENT: SET @sum = @sum + NEW.amount
        ACTION_ORIENTATION: ROW
             ACTION_TIMING: BEFORE
ACTION_REFERENCE_OLD_TABLE: NULL
ACTION_REFERENCE_NEW_TABLE: NULL
  ACTION_REFERENCE_OLD_ROW: OLD
  ACTION_REFERENCE_NEW_ROW: NEW
                  CREATED: NULL
                SQL_MODE: NO_ENGINE_SUBSTITUTION
                 DEFINER: me@localhost
      CHARACTER_SET_CLIENT: utf8
      COLLATION_CONNECTION: utf8_general_ci
        DATABASE_COLLATION: latin1_swedish_ci
```

Trigger information is also available from the `SHOW TRIGGERS` statement. See Section 13.7.5.39, “SHOW TRIGGERS Statement”.

21.28 The INFORMATION_SCHEMA USER_PRIVILEGES Table

The `USER_PRIVILEGES` table provides information about global privileges. It takes its values from the `mysql.user` system table.

The `USER_PRIVILEGES` table has these columns:

- `GRANTEE`

  The name of the account to which the privilege is granted, in *`'user_name'@'host_name'`* format.

- `TABLE_CATALOG`

  The name of the catalog. This value is always `def`.

- `PRIVILEGE_TYPE`

  The privilege granted. The value can be any privilege that can be granted at the global level; see Section 13.7.1.4, “GRANT Statement”. Each row lists a single privilege, so there is one row per global privilege held by the grantee.

- `IS_GRANTABLE`

  `YES` if the user has the `GRANT OPTION` privilege, `NO` otherwise. The output does not list `GRANT OPTION` as a separate row with `PRIVILEGE_TYPE='GRANT OPTION'`.

Notes

- `USER_PRIVILEGES` is a nonstandard `INFORMATION_SCHEMA` table.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.USER_PRIVILEGES

SHOW GRANTS ...
```

21.29 The INFORMATION_SCHEMA VIEWS Table

The `VIEWS` table provides information about views in databases. You must have the `SHOW VIEW` privilege to access this table.

The `VIEWS` table has these columns:

- `TABLE_CATALOG`

  The name of the catalog to which the view belongs. This value is always `def`.

- `TABLE_SCHEMA`

  The name of the schema (database) to which the view belongs.

- `TABLE_NAME`

  The name of the view.

- `VIEW_DEFINITION`

  The `SELECT` statement that provides the definition of the view. This column has most of what you see in the `Create Table` column that `SHOW CREATE VIEW` produces. Skip the words before `SELECT` and skip the words `WITH CHECK OPTION`. Suppose that the original statement was:

  ```
  CREATE VIEW v AS
    SELECT s2,s1 FROM t
    WHERE s1 > 5
    ORDER BY s1
    WITH CHECK OPTION;
  ```

  Then the view definition looks like this:

  ```
  SELECT s2,s1 FROM t WHERE s1 > 5 ORDER BY s1
  ```

- `CHECK_OPTION`

  The value of the `CHECK_OPTION` attribute. The value is one of `NONE`, `CASCADE`, or `LOCAL`.

- `IS_UPDATABLE`

  MySQL sets a flag, called the view updatability flag, at `CREATE VIEW` time. The flag is set to `YES` (true) if `UPDATE` and `DELETE` (and similar operations) are legal for the view. Otherwise, the flag is set to `NO` (false). The `IS_UPDATABLE` column in the `VIEWS` table displays the status of this flag. It means that the server always knows whether a view is updatable.

  If a view is not updatable, statements such `UPDATE`, `DELETE`, and `INSERT` are illegal and are rejected. (Even if a view is updatable, it might not be possible to insert into it; for details, refer to Section 20.5.3, “Updatable and Insertable Views”.)

- `DEFINER`

  The account of the user who created the view, in *`'user_name'@'host_name'`* format.

- SECURITY_TYPE

  The view SQL SECURITY characteristic. The value is one of DEFINER or INVOKER.

- CHARACTER_SET_CLIENT

  The session value of the character_set_client system variable when the view was created.

- COLLATION_CONNECTION

  The session value of the collation_connection system variable when the view was created.

Notes

MySQL permits different sql_mode settings to tell the server the type of SQL syntax to support. For example, you might use the ANSI SQL mode to ensure MySQL correctly interprets the standard SQL concatenation operator, the double bar (||), in your queries. If you then create a view that concatenates items, you might worry that changing the sql_mode setting to a value different from ANSI could cause the view to become invalid. But this is not the case. No matter how you write out a view definition, MySQL always stores it the same way, in a canonical form. Here is an example that shows how the server changes a double bar concatenation operator to a CONCAT() function:

```
mysql> SET sql_mode = 'ANSI';
Query OK, 0 rows affected (0.00 sec)

mysql> CREATE VIEW test.v AS SELECT 'a' || 'b' as col1;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT VIEW_DEFINITION FROM INFORMATION_SCHEMA.VIEWS
       WHERE TABLE_SCHEMA = 'test' AND TABLE_NAME = 'v';
+--------------------------------+
| VIEW_DEFINITION                |
+--------------------------------+
| select concat('a','b') AS `col1` |
+--------------------------------+
1 row in set (0.00 sec)
```

The advantage of storing a view definition in canonical form is that changes made later to the value of sql_mode do not affect the results from the view. However, an additional consequence is that comments prior to SELECT are stripped from the definition by the server.

21.30 INFORMATION_SCHEMA InnoDB Tables

This section provides table definitions for InnoDB INFORMATION_SCHEMA tables. For related information and examples, see Section 14.15, “InnoDB INFORMATION_SCHEMA Tables”.

InnoDB INFORMATION_SCHEMA tables can be used to monitor ongoing InnoDB activity, to detect inefficiencies before they turn into issues, or to troubleshoot performance and capacity issues. As your database becomes bigger and busier, running up against the limits of your hardware capacity, you monitor and tune these aspects to keep the database running smoothly.

21.30.1 The INFORMATION_SCHEMA INNODB_BUFFER_PAGE Table

The INNODB_BUFFER_PAGE table provides information about each page in the InnoDB buffer pool.

For related usage information and examples, see Section 14.15.5, “InnoDB INFORMATION_SCHEMA Buffer Pool Tables”.

Warning

Querying the INNODB_BUFFER_PAGE table can affect performance. Do not query this table on a production system unless you are aware of the performance impact and have determined it to be acceptable. To avoid

impacting performance on a production system, reproduce the issue you want to investigate and query buffer pool statistics on a test instance.

The `INNODB_BUFFER_PAGE` table has these columns:

- `POOL_ID`

  The buffer pool ID. This is an identifier to distinguish between multiple buffer pool instances.

- `BLOCK_ID`

  The buffer pool block ID.

- `SPACE`

  The tablespace ID; the same value as `INNODB_SYS_TABLES.SPACE`.

- `PAGE_NUMBER`

  The page number.

- `PAGE_TYPE`

  The page type. The following table shows the permitted values.

  Table 21.1 INNODB_BUFFER_PAGE.PAGE_TYPE Values

Page Type	Description
`ALLOCATED`	Freshly allocated page
`BLOB`	Uncompressed BLOB page
`COMPRESSED_BLOB2`	Subsequent comp BLOB page
`COMPRESSED_BLOB`	First compressed BLOB page
`EXTENT_DESCRIPTOR`	Extent descriptor page
`FILE_SPACE_HEADER`	File space header
`IBUF_BITMAP`	Insert buffer bitmap
`IBUF_FREE_LIST`	Insert buffer free list
`IBUF_INDEX`	Insert buffer index
`INDEX`	B-tree node
`INODE`	Index node
`SYSTEM`	System page
`TRX_SYSTEM`	Transaction system data
`UNDO_LOG`	Undo log page
`UNKNOWN`	Unknown

- `FLUSH_TYPE`

  The flush type.

- `FIX_COUNT`

  The number of threads using this block within the buffer pool. When zero, the block is eligible to be evicted.

- `IS_HASHED`

Whether a hash index has been built on this page.

- `NEWEST_MODIFICATION`

  The Log Sequence Number of the youngest modification.

- `OLDEST_MODIFICATION`

  The Log Sequence Number of the oldest modification.

- `ACCESS_TIME`

  An abstract number used to judge the first access time of the page.

- `TABLE_NAME`

  The name of the table the page belongs to. This column is applicable only to pages with a `PAGE_TYPE` value of `INDEX`.

- `INDEX_NAME`

  The name of the index the page belongs to. This can be the name of a clustered index or a secondary index. This column is applicable only to pages with a `PAGE_TYPE` value of `INDEX`.

- `NUMBER_RECORDS`

  The number of records within the page.

- `DATA_SIZE`

  The sum of the sizes of the records. This column is applicable only to pages with a `PAGE_TYPE` value of `INDEX`.

- `COMPRESSED_SIZE`

  The compressed page size. `NULL` for pages that are not compressed.

- `PAGE_STATE`

  The page state. The following table shows the permitted values.

  Table 21.2 INNODB_BUFFER_PAGE.PAGE_STATE Values

Page State	Description
`FILE_PAGE`	A buffered file page
`MEMORY`	Contains a main memory object
`NOT_USED`	In the free list
`NULL`	Clean compressed pages, compressed pages in the flush list, pages used as buffer pool watch sentinels
`READY_FOR_USE`	A free page
`REMOVE_HASH`	Hash index should be removed before placing in the free list

- `IO_FIX`

  Whether any I/O is pending for this page: `IO_NONE` = no pending I/O, `IO_READ` = read pending, `IO_WRITE` = write pending.

- `IS_OLD`

  Whether the block is in the sublist of old blocks in the LRU list.

- `FREE_PAGE_CLOCK`

  The value of the `freed_page_clock` counter when the block was the last placed at the head of the LRU list. The `freed_page_clock` counter tracks the number of blocks removed from the end of the LRU list.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE LIMIT 1\G
*************************** 1. row ***************************
            POOL_ID: 0
          BLOCK_ID: 0
             SPACE: 97
       PAGE_NUMBER: 2473
         PAGE_TYPE: INDEX
        FLUSH_TYPE: 1
         FIX_COUNT: 0
         IS_HASHED: YES
NEWEST_MODIFICATION: 733855581
OLDEST_MODIFICATION: 0
        ACCESS_TIME: 3378385672
         TABLE_NAME: `employees`.`salaries`
         INDEX_NAME: PRIMARY
     NUMBER_RECORDS: 468
          DATA_SIZE: 14976
    COMPRESSED_SIZE: 0
         PAGE_STATE: FILE_PAGE
             IO_FIX: IO_NONE
             IS_OLD: YES
    FREE_PAGE_CLOCK: 66
```

Notes

- This table is useful primarily for expert-level performance monitoring, or when developing performance-related extensions for MySQL.
- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- When tables, table rows, partitions, or indexes are deleted, associated pages remain in the buffer pool until space is required for other data. The `INNODB_BUFFER_PAGE` table reports information about these pages until they are evicted from the buffer pool. For more information about how the `InnoDB` manages buffer pool data, see Section 14.5.1, “Buffer Pool”.

21.30.2 The INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU Table

The `INNODB_BUFFER_PAGE_LRU` table provides information about the pages in the `InnoDB` buffer pool; in particular, how they are ordered in the LRU list that determines which pages to evict from the buffer pool when it becomes full.

The `INNODB_BUFFER_PAGE_LRU` table has the same columns as the `INNODB_BUFFER_PAGE` table, except that the `INNODB_BUFFER_PAGE_LRU` table has `LRU_POSITION` and `COMPRESSED` columns instead of `BLOCK_ID` and `PAGE_STATE` columns.

For related usage information and examples, see Section 14.15.5, “InnoDB INFORMATION_SCHEMA Buffer Pool Tables”.

Warning

Querying the `INNODB_BUFFER_PAGE_LRU` table can affect performance. Do not query this table on a production system unless you are aware of the performance impact and have determined it to be acceptable. To avoid

impacting performance on a production system, reproduce the issue you want to investigate and query buffer pool statistics on a test instance.

The `INNODB_BUFFER_PAGE_LRU` table has these columns:

- `POOL_ID`

  The buffer pool ID. This is an identifier to distinguish between multiple buffer pool instances.

- `LRU_POSITION`

  The position of the page in the LRU list.

- `SPACE`

  The tablespace ID; the same value as `INNODB_SYS_TABLES.SPACE`.

- `PAGE_NUMBER`

  The page number.

- `PAGE_TYPE`

  The page type. The following table shows the permitted values.

  Table 21.3 INNODB_BUFFER_PAGE_LRU.PAGE_TYPE Values

Page Type	Description
`ALLOCATED`	Freshly allocated page
`BLOB`	Uncompressed BLOB page
`COMPRESSED_BLOB2`	Subsequent comp BLOB page
`COMPRESSED_BLOB`	First compressed BLOB page
`EXTENT_DESCRIPTOR`	Extent descriptor page
`FILE_SPACE_HEADER`	File space header
`IBUF_BITMAP`	Insert buffer bitmap
`IBUF_FREE_LIST`	Insert buffer free list
`IBUF_INDEX`	Insert buffer index
`INDEX`	B-tree node
`INODE`	Index node
`SYSTEM`	System page
`TRX_SYSTEM`	Transaction system data
`UNDO_LOG`	Undo log page
`UNKNOWN`	Unknown

- `FLUSH_TYPE`

  The flush type.

- `FIX_COUNT`

  The number of threads using this block within the buffer pool. When zero, the block is eligible to be evicted.

- `IS_HASHED`

Whether a hash index has been built on this page.

- `NEWEST_MODIFICATION`

  The Log Sequence Number of the youngest modification.

- `OLDEST_MODIFICATION`

  The Log Sequence Number of the oldest modification.

- `ACCESS_TIME`

  An abstract number used to judge the first access time of the page.

- `TABLE_NAME`

  The name of the table the page belongs to. This column is applicable only to pages with a `PAGE_TYPE` value of `INDEX`.

- `INDEX_NAME`

  The name of the index the page belongs to. This can be the name of a clustered index or a secondary index. This column is applicable only to pages with a `PAGE_TYPE` value of `INDEX`.

- `NUMBER_RECORDS`

  The number of records within the page.

- `DATA_SIZE`

  The sum of the sizes of the records. This column is applicable only to pages with a `PAGE_TYPE` value of `INDEX`.

- `COMPRESSED_SIZE`

  The compressed page size. `NULL` for pages that are not compressed.

- `COMPRESSED`

  Whether the page is compressed.

- `IO_FIX`

  Whether any I/O is pending for this page: `IO_NONE` = no pending I/O, `IO_READ` = read pending, `IO_WRITE` = write pending.

- `IS_OLD`

  Whether the block is in the sublist of old blocks in the LRU list.

- `FREE_PAGE_CLOCK`

  The value of the `freed_page_clock` counter when the block was the last placed at the head of the LRU list. The `freed_page_clock` counter tracks the number of blocks removed from the end of the LRU list.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE_LRU LIMIT 1\G
*************************** 1. row ***************************
            POOL_ID: 0
      LRU_POSITION: 0
             SPACE: 97
       PAGE_NUMBER: 1984
```

```
          PAGE_TYPE: INDEX
         FLUSH_TYPE: 1
          FIX_COUNT: 0
          IS_HASHED: YES
NEWEST_MODIFICATION: 719490396
OLDEST_MODIFICATION: 0
        ACCESS_TIME: 3378383796
         TABLE_NAME: `employees`.`salaries`
         INDEX_NAME: PRIMARY
     NUMBER_RECORDS: 468
          DATA_SIZE: 14976
    COMPRESSED_SIZE: 0
         COMPRESSED: NO
             IO_FIX: IO_NONE
             IS_OLD: YES
    FREE_PAGE_CLOCK: 0
```

Notes

- This table is useful primarily for expert-level performance monitoring, or when developing performance-related extensions for MySQL.
- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- Querying this table can require MySQL to allocate a large block of contiguous memory, more than 64 bytes times the number of active pages in the buffer pool. This allocation could potentially cause an out-of-memory error, especially for systems with multi-gigabyte buffer pools.
- Querying this table requires MySQL to lock the data structure representing the buffer pool while traversing the LRU list, which can reduce concurrency, especially for systems with multi-gigabyte buffer pools.
- When tables, table rows, partitions, or indexes are deleted, associated pages remain in the buffer pool until space is required for other data. The `INNODB_BUFFER_PAGE_LRU` table reports information about these pages until they are evicted from the buffer pool. For more information about how the `InnoDB` manages buffer pool data, see Section 14.5.1, “Buffer Pool”.

21.30.3 The INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS Table

The `INNODB_BUFFER_POOL_STATS` table provides much of the same buffer pool information provided in `SHOW ENGINE INNODB STATUS` output. Much of the same information may also be obtained using `InnoDB` buffer pool server status variables.

The idea of making pages in the buffer pool “young” or “not young” refers to transferring them between the sublists at the head and tail of the buffer pool data structure. Pages made “young” take longer to age out of the buffer pool, while pages made “not young” are moved much closer to the point of eviction.

For related usage information and examples, see Section 14.15.5, “InnoDB INFORMATION_SCHEMA Buffer Pool Tables”.

The `INNODB_BUFFER_POOL_STATS` table has these columns:

- `POOL_ID`

  The buffer pool ID. This is an identifier to distinguish between multiple buffer pool instances.

- `POOL_SIZE`

  The `InnoDB` buffer pool size in pages.

- `FREE_BUFFERS`

  The number of free pages in the `InnoDB` buffer pool.

- `DATABASE_PAGES`

  The number of pages in the `InnoDB` buffer pool containing data. This number includes both dirty and clean pages.

- `OLD_DATABASE_PAGES`

  The number of pages in the `old` buffer pool sublist.

- `MODIFIED_DATABASE_PAGES`

  The number of modified (dirty) database pages.

- `PENDING_DECOMPRESS`

  The number of pages pending decompression.

- `PENDING_READS`

  The number of pending reads.

- `PENDING_FLUSH_LRU`

  The number of pages pending flush in the LRU.

- `PENDING_FLUSH_LIST`

  The number of pages pending flush in the flush list.

- `PAGES_MADE_YOUNG`

  The number of pages made young.

- `PAGES_NOT_MADE_YOUNG`

  The number of pages not made young.

- `PAGES_MADE_YOUNG_RATE`

  The number of pages made young per second (pages made young since the last printout / time elapsed).

- `PAGES_MADE_NOT_YOUNG_RATE`

  The number of pages not made per second (pages not made young since the last printout / time elapsed).

- `NUMBER_PAGES_READ`

  The number of pages read.

- `NUMBER_PAGES_CREATED`

  The number of pages created.

- `NUMBER_PAGES_WRITTEN`

  The number of pages written.

- `PAGES_READ_RATE`

The number of pages read per second (pages read since the last printout / time elapsed).

- `PAGES_CREATE_RATE`

  The number of pages created per second (pages created since the last printout / time elapsed).

- `PAGES_WRITTEN_RATE`

  The number of pages written per second (pages written since the last printout / time elapsed).

- `NUMBER_PAGES_GET`

  The number of logical read requests.

- `HIT_RATE`

  The buffer pool hit rate.

- `YOUNG_MAKE_PER_THOUSAND_GETS`

  The number of pages made young per thousand gets.

- `NOT_YOUNG_MAKE_PER_THOUSAND_GETS`

  The number of pages not made young per thousand gets.

- `NUMBER_PAGES_READ_AHEAD`

  The number of pages read ahead.

- `NUMBER_READ_AHEAD_EVICTED`

  The number of pages read into the `InnoDB` buffer pool by the read-ahead background thread that were subsequently evicted without having been accessed by queries.

- `READ_AHEAD_RATE`

  The read-ahead rate per second (pages read ahead since the last printout / time elapsed).

- `READ_AHEAD_EVICTED_RATE`

  The number of read-ahead pages evicted without access per second (read-ahead pages not accessed since the last printout / time elapsed).

- `LRU_IO_TOTAL`

  Total LRU I/O.

- `LRU_IO_CURRENT`

  LRU I/O for the current interval.

- `UNCOMPRESS_TOTAL`

  The total number of pages decompressed.

- `UNCOMPRESS_CURRENT`

  The number of pages decompressed in the current interval.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_BUFFER_POOL_STATS\G
*************************** 1. row ***************************
```

```
                        POOL_ID: 0
                      POOL_SIZE: 8192
                     FREE_BUFFERS: 1
                   DATABASE_PAGES: 8085
               OLD_DATABASE_PAGES: 2964
          MODIFIED_DATABASE_PAGES: 0
               PENDING_DECOMPRESS: 0
                    PENDING_READS: 0
                PENDING_FLUSH_LRU: 0
               PENDING_FLUSH_LIST: 0
                 PAGES_MADE_YOUNG: 22821
             PAGES_NOT_MADE_YOUNG: 3544303
            PAGES_MADE_YOUNG_RATE: 357.62602199870594
        PAGES_MADE_NOT_YOUNG_RATE: 0
                NUMBER_PAGES_READ: 2389
             NUMBER_PAGES_CREATED: 12385
             NUMBER_PAGES_WRITTEN: 13111
                  PAGES_READ_RATE: 0
                PAGES_CREATE_RATE: 0
               PAGES_WRITTEN_RATE: 0
                 NUMBER_PAGES_GET: 33322210
                         HIT_RATE: 1000
    YOUNG_MAKE_PER_THOUSAND_GETS: 18
NOT_YOUNG_MAKE_PER_THOUSAND_GETS: 0
         NUMBER_PAGES_READ_AHEAD: 2024
       NUMBER_READ_AHEAD_EVICTED: 0
                 READ_AHEAD_RATE: 0
         READ_AHEAD_EVICTED_RATE: 0
                    LRU_IO_TOTAL: 0
                  LRU_IO_CURRENT: 0
                UNCOMPRESS_TOTAL: 0
               UNCOMPRESS_CURRENT: 0
```

Notes

- This table is useful primarily for expert-level performance monitoring, or when developing performance-related extensions for MySQL.
- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

21.30.4 The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables

The `INNODB_CMP` and `INNODB_CMP_RESET` tables provide status information on operations related to compressed `InnoDB` tables.

The `INNODB_CMP` and `INNODB_CMP_RESET` tables have these columns:

- `PAGE_SIZE`

  The compressed page size in bytes.

- `COMPRESS_OPS`

  The number of times a B-tree page of size `PAGE_SIZE` has been compressed. Pages are compressed whenever an empty page is created or the space for the uncompressed modification log runs out.

- `COMPRESS_OPS_OK`

  The number of times a B-tree page of size `PAGE_SIZE` has been successfully compressed. This count should never exceed `COMPRESS_OPS`.

- `COMPRESS_TIME`

The total time in seconds used for attempts to compress B-tree pages of size `PAGE_SIZE`.

- `UNCOMPRESS_OPS`

  The number of times a B-tree page of size `PAGE_SIZE` has been uncompressed. B-tree pages are uncompressed whenever compression fails or at first access when the uncompressed page does not exist in the buffer pool.

- `UNCOMPRESS_TIME`

  The total time in seconds used for uncompressing B-tree pages of the size `PAGE_SIZE`.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_CMP\G
*************************** 1. row ***************************
      page_size: 1024
   compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
 uncompress_ops: 0
uncompress_time: 0
*************************** 2. row ***************************
      page_size: 2048
   compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
 uncompress_ops: 0
uncompress_time: 0
*************************** 3. row ***************************
      page_size: 4096
   compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
 uncompress_ops: 0
uncompress_time: 0
*************************** 4. row ***************************
      page_size: 8192
   compress_ops: 86955
compress_ops_ok: 81182
  compress_time: 27
 uncompress_ops: 26828
uncompress_time: 5
*************************** 5. row ***************************
      page_size: 16384
   compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
 uncompress_ops: 0
uncompress_time: 0
```

Notes

- Use these tables to measure the effectiveness of `InnoDB` table compression in your database.
- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- For usage information, see Section 14.9.4, “Monitoring InnoDB Table Compression at Runtime” and Section 14.15.1.3, “Using the Compression Information Schema Tables”. For general information about `InnoDB` table compression, see Section 14.9, “InnoDB Table Compression”.

21.30.5 The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables

The `INNODB_CMPMEM` and `INNODB_CMPMEM_RESET` tables provide status information on compressed pages within the `InnoDB` buffer pool.

The `INNODB_CMPMEM` and `INNODB_CMPMEM_RESET` tables have these columns:

- `PAGE_SIZE`

  The block size in bytes. Each record of this table describes blocks of this size.

- `BUFFER_POOL_INSTANCE`

  A unique identifier for the buffer pool instance.

- `PAGES_USED`

  The number of blocks of size `PAGE_SIZE` that are currently in use.

- `PAGES_FREE`

  The number of blocks of size `PAGE_SIZE` that are currently available for allocation. This column shows the external fragmentation in the memory pool. Ideally, these numbers should be at most 1.

- `RELOCATION_OPS`

  The number of times a block of size `PAGE_SIZE` has been relocated. The buddy system can relocate the allocated “buddy neighbor” of a freed block when it tries to form a bigger freed block. Reading from the `INNODB_CMPMEM_RESET` table resets this count.

- `RELOCATION_TIME`

  The total time in microseconds used for relocating blocks of size `PAGE_SIZE`. Reading from the table `INNODB_CMPMEM_RESET` resets this count.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_CMPMEM\G
*************************** 1. row ***************************
           page_size: 1024
buffer_pool_instance: 0
          pages_used: 0
          pages_free: 0
      relocation_ops: 0
     relocation_time: 0
*************************** 2. row ***************************
           page_size: 2048
buffer_pool_instance: 0
          pages_used: 0
          pages_free: 0
      relocation_ops: 0
     relocation_time: 0
*************************** 3. row ***************************
           page_size: 4096
buffer_pool_instance: 0
          pages_used: 0
          pages_free: 0
      relocation_ops: 0
     relocation_time: 0
*************************** 4. row ***************************
           page_size: 8192
buffer_pool_instance: 0
          pages_used: 7673
          pages_free: 15
      relocation_ops: 4638
     relocation_time: 0
*************************** 5. row ***************************
           page_size: 16384
buffer_pool_instance: 0
          pages_used: 0
```

```
          pages_free: 0
      relocation_ops: 0
     relocation_time: 0
```

Notes

- Use these tables to measure the effectiveness of `InnoDB` table compression in your database.
- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- For usage information, see Section 14.9.4, “Monitoring InnoDB Table Compression at Runtime” and Section 14.15.1.3, “Using the Compression Information Schema Tables”. For general information about `InnoDB` table compression, see Section 14.9, “InnoDB Table Compression”.

21.30.6 The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables

The `INNODB_CMP_PER_INDEX` and `INNODB_CMP_PER_INDEX_RESET` tables provide status information on operations related to compressed `InnoDB` tables and indexes, with separate statistics for each combination of database, table, and index, to help you evaluate the performance and usefulness of compression for specific tables.

For a compressed `InnoDB` table, both the table data and all the secondary indexes are compressed. In this context, the table data is treated as just another index, one that happens to contain all the columns: the clustered index.

The `INNODB_CMP_PER_INDEX` and `INNODB_CMP_PER_INDEX_RESET` tables have these columns:

- `DATABASE_NAME`

  The schema (database) containing the applicable table.

- `TABLE_NAME`

  The table to monitor for compression statistics.

- `INDEX_NAME`

  The index to monitor for compression statistics.

- `COMPRESS_OPS`

  The number of compression operations attempted. Pages are compressed whenever an empty page is created or the space for the uncompressed modification log runs out.

- `COMPRESS_OPS_OK`

  The number of successful compression operations. Subtract from the `COMPRESS_OPS` value to get the number of compression failures. Divide by the `COMPRESS_OPS` value to get the percentage of compression failures.

- `COMPRESS_TIME`

  The total time in seconds used for compressing data in this index.

- `UNCOMPRESS_OPS`

  The number of uncompression operations performed. Compressed `InnoDB` pages are uncompressed whenever compression fails, or the first time a compressed page is accessed in the buffer pool and the uncompressed page does not exist.

- UNCOMPRESS_TIME

  The total time in seconds used for uncompressing data in this index.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX\G
*************************** 1. row ***************************
  database_name: employees
     table_name: salaries
     index_name: PRIMARY
   compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
 uncompress_ops: 23451
uncompress_time: 4
*************************** 2. row ***************************
  database_name: employees
     table_name: salaries
     index_name: emp_no
   compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
 uncompress_ops: 1597
uncompress_time: 0
```

Notes

- Use these tables to measure the effectiveness of InnoDB table compression for specific tables, indexes, or both.
- You must have the PROCESS privilege to query these tables.
- Use the INFORMATION_SCHEMA COLUMNS table or the SHOW COLUMNS statement to view additional information about the columns of these tables, including data types and default values.
- Because collecting separate measurements for every index imposes substantial performance overhead, INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET statistics are not gathered by default. You must enable the innodb_cmp_per_index_enabled system variable before performing the operations on compressed tables that you want to monitor.
- For usage information, see Section 14.9.4, “Monitoring InnoDB Table Compression at Runtime” and Section 14.15.1.3, “Using the Compression Information Schema Tables”. For general information about InnoDB table compression, see Section 14.9, “InnoDB Table Compression”.

21.30.7 The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table

The INNODB_FT_BEING_DELETED table provides a snapshot of the INNODB_FT_DELETED table; it is used only during an OPTIMIZE TABLE maintenance operation. When OPTIMIZE TABLE is run, the INNODB_FT_BEING_DELETED table is emptied, and DOC_ID values are removed from the INNODB_FT_DELETED table. Because the contents of INNODB_FT_BEING_DELETED typically have a short lifetime, this table has limited utility for monitoring or debugging. For information about running OPTIMIZE TABLE on tables with FULLTEXT indexes, see Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”.

This table is empty initially. Before querying it, set the value of the innodb_ft_aux_table system variable to the name (including the database name) of the table that contains the FULLTEXT index; for example test/articles. The output appears similar to the example provided for the INNODB_FT_DELETED table.

For related usage information and examples, see Section 14.15.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”.

The INNODB_FT_BEING_DELETED table has these columns:

- `DOC_ID`

  The document ID of the row that is in the process of being deleted. This value might reflect the value of an ID column that you defined for the underlying table, or it can be a sequence value generated by `InnoDB` when the table contains no suitable column. This value is used when you do text searches, to skip rows in the `INNODB_FT_INDEX_TABLE` table before data for deleted rows is physically removed from the `FULLTEXT` index by an `OPTIMIZE TABLE` statement. For more information, see Optimizing InnoDB Full-Text Indexes.

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- For more information about `InnoDB FULLTEXT` search, see Section 14.6.2.3, “InnoDB FULLTEXT Indexes”, and Section 12.10, “Full-Text Search Functions”.

21.30.8 The INFORMATION_SCHEMA INNODB_FT_CONFIG Table

The `INNODB_FT_CONFIG` table provides metadata about the `FULLTEXT` index and associated processing for an `InnoDB` table.

This table is empty initially. Before querying it, set the value of the `innodb_ft_aux_table` system variable to the name (including the database name) of the table that contains the `FULLTEXT` index; for example `test/articles`.

For related usage information and examples, see Section 14.15.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”.

The `INNODB_FT_CONFIG` table has these columns:

- `KEY`

  The name designating an item of metadata for an `InnoDB` table containing a `FULLTEXT` index.

  The values for this column might change, depending on the needs for performance tuning and debugging for `InnoDB` full-text processing. The key names and their meanings include:

  - `optimize_checkpoint_limit`: The number of seconds after which an `OPTIMIZE TABLE` run stops.
  - `synced_doc_id`: The next `DOC_ID` to be issued.
  - `stopword_table_name`: The *`database/table`* name for a user-defined stopword table. The `VALUE` column is empty if there is no user-defined stopword table.
  - `use_stopword`: Indicates whether a stopword table is used, which is defined when the `FULLTEXT` index is created.

- `VALUE`

  The value associated with the corresponding `KEY` column, reflecting some limit or current value for an aspect of a `FULLTEXT` index for an `InnoDB` table.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_CONFIG;
+---------------------------+-------------------+
| KEY                       | VALUE             |
+---------------------------+-------------------+
```

```
| optimize_checkpoint_limit | 180               |
| synced_doc_id             | 0                 |
| stopword_table_name       | test/my_stopwords |
| use_stopword              | 1                 |
+---------------------------+-------------------+
```

Notes

- This table is intended only for internal configuration. It is not intended for statistical information purposes.
- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- For more information about `InnoDB FULLTEXT` search, see Section 14.6.2.3, “InnoDB FULLTEXT Indexes”, and Section 12.10, “Full-Text Search Functions”.

21.30.9 The INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD Table

The `INNODB_FT_DEFAULT_STOPWORD` table holds a list of stopwords that are used by default when creating a `FULLTEXT` index on `InnoDB` tables. For information about the default `InnoDB` stopword list and how to define your own stopword lists, see Section 12.10.4, “Full-Text Stopwords”.

For related usage information and examples, see Section 14.15.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”.

The `INNODB_FT_DEFAULT_STOPWORD` table has these columns:

- `value`

  A word that is used by default as a stopword for `FULLTEXT` indexes on `InnoDB` tables. This is not used if you override the default stopword processing with either the `innodb_ft_server_stopword_table` or the `innodb_ft_user_stopword_table` system variable.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_DEFAULT_STOPWORD;
+-------+
| value |
+-------+
| a     |
| about |
| an    |
| are   |
| as    |
| at    |
| be    |
| by    |
| com   |
| de    |
| en    |
| for   |
| from  |
| how   |
| i     |
| in    |
| is    |
| it    |
| la    |
| of    |
| on    |
```

```
| or    |
| that  |
| the   |
| this  |
| to    |
| was   |
| what  |
| when  |
| where |
| who   |
| will  |
| with  |
| und   |
| the   |
| www   |
+-------+
36 rows in set (0.00 sec)
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- For more information about `InnoDB FULLTEXT` search, see Section 14.6.2.3, “InnoDB FULLTEXT Indexes”, and Section 12.10, “Full-Text Search Functions”.

21.30.10 The INFORMATION_SCHEMA INNODB_FT_DELETED Table

The `INNODB_FT_DELETED` table stores rows that are deleted from the `FULLTEXT` index for an `InnoDB` table. To avoid expensive index reorganization during DML operations for an `InnoDB FULLTEXT` index, the information about newly deleted words is stored separately, filtered out of search results when you do a text search, and removed from the main search index only when you issue an `OPTIMIZE TABLE` statement for the `InnoDB` table. For more information, see Optimizing InnoDB Full-Text Indexes.

This table is empty initially. Before querying it, set the value of the `innodb_ft_aux_table` system variable to the name (including the database name) of the table that contains the `FULLTEXT` index; for example `test/articles`.

For related usage information and examples, see Section 14.15.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”.

The `INNODB_FT_DELETED` table has these columns:

- `DOC_ID`

  The document ID of the newly deleted row. This value might reflect the value of an ID column that you defined for the underlying table, or it can be a sequence value generated by `InnoDB` when the table contains no suitable column. This value is used when you do text searches, to skip rows in the `INNODB_FT_INDEX_TABLE` table before data for deleted rows is physically removed from the `FULLTEXT` index by an `OPTIMIZE TABLE` statement. For more information, see Optimizing InnoDB Full-Text Indexes.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_DELETED;
+--------+
| DOC_ID |
+--------+
|      6 |
|      7 |
|      8 |
+--------+
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- For more information about `InnoDB FULLTEXT` search, see Section 14.6.2.3, “InnoDB FULLTEXT Indexes”, and Section 12.10, “Full-Text Search Functions”.

21.30.11 The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table

The `INNODB_FT_INDEX_CACHE` table provides token information about newly inserted rows in a `FULLTEXT` index. To avoid expensive index reorganization during DML operations, the information about newly indexed words is stored separately, and combined with the main search index only when `OPTIMIZE TABLE` is run, when the server is shut down, or when the cache size exceeds a limit defined by the `innodb_ft_cache_size` or `innodb_ft_total_cache_size` system variable.

This table is empty initially. Before querying it, set the value of the `innodb_ft_aux_table` system variable to the name (including the database name) of the table that contains the `FULLTEXT` index; for example `test/articles`.

For related usage information and examples, see Section 14.15.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”.

The `INNODB_FT_INDEX_CACHE` table has these columns:

- `WORD`

  A word extracted from the text of a newly inserted row.

- `FIRST_DOC_ID`

  The first document ID in which this word appears in the `FULLTEXT` index.

- `LAST_DOC_ID`

  The last document ID in which this word appears in the `FULLTEXT` index.

- `DOC_COUNT`

  The number of rows in which this word appears in the `FULLTEXT` index. The same word can occur several times within the cache table, once for each combination of `DOC_ID` and `POSITION` values.

- `DOC_ID`

  The document ID of the newly inserted row. This value might reflect the value of an ID column that you defined for the underlying table, or it can be a sequence value generated by `InnoDB` when the table contains no suitable column.

- `POSITION`

  The position of this particular instance of the word within the relevant document identified by the `DOC_ID` value. The value does not represent an absolute position; it is an offset added to the `POSITION` of the previous instance of that word.

Notes

- This table is empty initially. Before querying it, set the value of the `innodb_ft_aux_table` system variable to the name (including the database name) of the table that contains the `FULLTEXT` index; for example `test/articles`. The following example demonstrates how to use the `innodb_ft_aux_table` system variable to show information about a `FULLTEXT` index for a specified table.

```
mysql> USE test;

mysql> CREATE TABLE articles (
         id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
         title VARCHAR(200),
         body TEXT,
         FULLTEXT (title,body)
       ) ENGINE=InnoDB;

mysql> INSERT INTO articles (title,body) VALUES
       ('MySQL Tutorial','DBMS stands for DataBase ...'),
       ('How To Use MySQL Well','After you went through a ...'),
       ('Optimizing MySQL','In this tutorial we show ...'),
       ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
       ('MySQL vs. YourSQL','In the following database comparison ...'),
       ('MySQL Security','When configured properly, MySQL ...');

mysql> SET GLOBAL innodb_ft_aux_table = 'test/articles';

mysql> SELECT WORD, DOC_COUNT, DOC_ID, POSITION
       FROM INFORMATION_SCHEMA.INNODB_FT_INDEX_CACHE LIMIT 5;
+------------+-----------+--------+----------+
| WORD       | DOC_COUNT | DOC_ID | POSITION |
+------------+-----------+--------+----------+
| 1001       |         1 |      4 |        0 |
| after      |         1 |      2 |       22 |
| comparison |         1 |      5 |       44 |
| configured |         1 |      6 |       20 |
| database   |         2 |      1 |       31 |
+------------+-----------+--------+----------+
```

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- For more information about `InnoDB FULLTEXT` search, see Section 14.6.2.3, “InnoDB FULLTEXT Indexes”, and Section 12.10, “Full-Text Search Functions”.

21.30.12 The INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE Table

The `INNODB_FT_INDEX_TABLE` table provides information about the inverted index used to process text searches against the `FULLTEXT` index of an `InnoDB` table.

This table is empty initially. Before querying it, set the value of the `innodb_ft_aux_table` system variable to the name (including the database name) of the table that contains the `FULLTEXT` index; for example `test/articles`.

For related usage information and examples, see Section 14.15.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”.

The `INNODB_FT_INDEX_TABLE` table has these columns:

- `WORD`

  A word extracted from the text of the columns that are part of a `FULLTEXT`.

- `FIRST_DOC_ID`

  The first document ID in which this word appears in the `FULLTEXT` index.

- `LAST_DOC_ID`

  The last document ID in which this word appears in the `FULLTEXT` index.

- `DOC_COUNT`

The number of rows in which this word appears in the FULLTEXT index. The same word can occur several times within the cache table, once for each combination of DOC_ID and POSITION values.

- DOC_ID

  The document ID of the row containing the word. This value might reflect the value of an ID column that you defined for the underlying table, or it can be a sequence value generated by InnoDB when the table contains no suitable column.

- POSITION

  The position of this particular instance of the word within the relevant document identified by the DOC_ID value.

Notes

- This table is empty initially. Before querying it, set the value of the innodb_ft_aux_table system variable to the name (including the database name) of the table that contains the FULLTEXT index; for example test/articles. The following example demonstrates how to use the innodb_ft_aux_table system variable to show information about a FULLTEXT index for a specified table. Before information for newly inserted rows appears in INNODB_FT_INDEX_TABLE, the FULLTEXT index cache must be flushed to disk. This is accomplished by running an OPTIMIZE TABLE operation on the indexed table with the innodb_optimize_fulltext_only system variable enabled. (The example disables that variable again at the end because it is intended to be enabled only temporarily.)

```
mysql> USE test;

mysql> CREATE TABLE articles (
         id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
         title VARCHAR(200),
         body TEXT,
         FULLTEXT (title,body)
       ) ENGINE=InnoDB;

mysql> INSERT INTO articles (title,body) VALUES
       ('MySQL Tutorial','DBMS stands for DataBase ...'),
       ('How To Use MySQL Well','After you went through a ...'),
       ('Optimizing MySQL','In this tutorial we show ...'),
       ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
       ('MySQL vs. YourSQL','In the following database comparison ...'),
       ('MySQL Security','When configured properly, MySQL ...');

mysql> SET GLOBAL innodb_optimize_fulltext_only=ON;

mysql> OPTIMIZE TABLE articles;
+---------------+----------+----------+----------+
| Table         | Op       | Msg_type | Msg_text |
+---------------+----------+----------+----------+
| test.articles | optimize | status   | OK       |
+---------------+----------+----------+----------+

mysql> SET GLOBAL innodb_ft_aux_table = 'test/articles';

mysql> SELECT WORD, DOC_COUNT, DOC_ID, POSITION
       FROM INFORMATION_SCHEMA.INNODB_FT_INDEX_TABLE LIMIT 5;
+------------+-----------+--------+----------+
| WORD       | DOC_COUNT | DOC_ID | POSITION |
+------------+-----------+--------+----------+
| 1001       |         1 |      4 |        0 |
| after      |         1 |      2 |       22 |
| comparison |         1 |      5 |       44 |
| configured |         1 |      6 |       20 |
| database   |         2 |      1 |       31 |
+------------+-----------+--------+----------+

mysql> SET GLOBAL innodb_optimize_fulltext_only=OFF;
```

- You must have the PROCESS privilege to query this table.
- Use the INFORMATION_SCHEMA COLUMNS table or the SHOW COLUMNS statement to view additional information about the columns of this table, including data types and default values.
- For more information about InnoDB FULLTEXT search, see Section 14.6.2.3, “InnoDB FULLTEXT Indexes”, and Section 12.10, “Full-Text Search Functions”.

21.30.13 The INFORMATION_SCHEMA INNODB_LOCKS Table

The INNODB_LOCKS table provides information about each lock that an InnoDB transaction has requested but not yet acquired, and each lock that a transaction holds that is blocking another transaction.

The INNODB_LOCKS table has these columns:

- LOCK_ID

  A unique lock ID number, internal to InnoDB. Treat it as an opaque string. Although LOCK_ID currently contains TRX_ID, the format of the data in LOCK_ID is subject to change at any time. Do not write applications that parse the LOCK_ID value.

- LOCK_TRX_ID

  The ID of the transaction holding the lock. To obtain details about the transaction, join this column with the TRX_ID column of the INNODB_TRX table.

- LOCK_MODE

  How the lock is requested. Permitted lock mode descriptors are S, X, IS, IX, GAP, AUTO_INC, and UNKNOWN. Lock mode descriptors may be used in combination to identify particular lock modes. For information about InnoDB lock modes, see Section 14.7.1, “InnoDB Locking”.

- LOCK_TYPE

  The type of lock. Permitted values are RECORD for a row-level lock, TABLE for a table-level lock.

- LOCK_TABLE

  The name of the table that has been locked or contains locked records.

- LOCK_INDEX

  The name of the index, if LOCK_TYPE is RECORD; otherwise NULL.

- LOCK_SPACE

  The tablespace ID of the locked record, if LOCK_TYPE is RECORD; otherwise NULL.

- LOCK_PAGE

  The page number of the locked record, if LOCK_TYPE is RECORD; otherwise NULL.

- LOCK_REC

  The heap number of the locked record within the page, if LOCK_TYPE is RECORD; otherwise NULL.

- LOCK_DATA

  The data associated with the lock, if any. A value is shown if the LOCK_TYPE is RECORD, otherwise the value is NULL. Primary key values of the locked record are shown for a lock placed on the primary key index. Secondary index values of the locked record are shown for a lock placed on a unique secondary index. Secondary index values are shown with primary key values appended if the

secondary index is not unique. If there is no primary key, `LOCK_DATA` shows either the key values of a selected unique index or the unique `InnoDB` internal row ID number, according to the rules governing `InnoDB` clustered index use (see Section 14.6.2.1, “Clustered and Secondary Indexes”). `LOCK_DATA` reports “supremum pseudo-record” for a lock taken on a supremum pseudo-record. If the page containing the locked record is not in the buffer pool because it was written to disk while the lock was held, `InnoDB` does not fetch the page from disk. Instead, `LOCK_DATA` reports `NULL`.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_LOCKS\G
*************************** 1. row ***************************
    lock_id: 3723:72:3:2
lock_trx_id: 3723
  lock_mode: X
  lock_type: RECORD
 lock_table: `mysql`.`t`
 lock_index: PRIMARY
 lock_space: 72
  lock_page: 3
   lock_rec: 2
  lock_data: 1, 9
*************************** 2. row ***************************
    lock_id: 3722:72:3:2
lock_trx_id: 3722
  lock_mode: S
  lock_type: RECORD
 lock_table: `mysql`.`t`
 lock_index: PRIMARY
 lock_space: 72
  lock_page: 3
   lock_rec: 2
  lock_data: 1, 9
```

Notes

- Use this table to help diagnose performance problems that occur during times of heavy concurrent load. Its contents are updated as described in Section 14.15.2.3, “Persistence and Consistency of InnoDB Transaction and Locking Information”.
- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- For usage information, see Section 14.15.2.1, “Using InnoDB Transaction and Locking Information”.

21.30.14 The INFORMATION_SCHEMA INNODB_LOCK_WAITS Table

The `INNODB_LOCK_WAITS` table contains one or more rows for each blocked `InnoDB` transaction, indicating the lock it has requested and any locks that are blocking that request.

The `INNODB_LOCK_WAITS` table has these columns:

- `REQUESTING_TRX_ID`

  The ID of the requesting (blocked) transaction.

- `REQUESTED_LOCK_ID`

  The ID of the lock for which a transaction is waiting. To obtain details about the lock, join this column with the `LOCK_ID` column of the `INNODB_LOCKS` table.

- `BLOCKING_TRX_ID`

  The ID of the blocking transaction.

- `BLOCKING_LOCK_ID`

  The ID of a lock held by a transaction blocking another transaction from proceeding. To obtain details about the lock, join this column with the `LOCK_ID` column of the `INNODB_LOCKS` table.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_LOCK_WAITS\G
*************************** 1. row ***************************
requesting_trx_id: 3396
requested_lock_id: 3396:91:3:2
  blocking_trx_id: 3395
 blocking_lock_id: 3395:91:3:2
```

Notes

- Use this table to help diagnose performance problems that occur during times of heavy concurrent load. Its contents are updated as described in Section 14.15.2.3, “Persistence and Consistency of InnoDB Transaction and Locking Information”.
- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- For usage information, see Section 14.15.2.1, “Using InnoDB Transaction and Locking Information”.

21.30.15 The INFORMATION_SCHEMA INNODB_METRICS Table

The `INNODB_METRICS` table provides a wide variety of `InnoDB` performance information, complementing the specific focus areas of the Performance Schema tables for `InnoDB`. With simple queries, you can check the overall health of the system. With more detailed queries, you can diagnose issues such as performance bottlenecks, resource shortages, and application issues.

Each monitor represents a point within the `InnoDB` source code that is instrumented to gather counter information. Each counter can be started, stopped, and reset. You can also perform these actions for a group of counters using their common module name.

By default, relatively little data is collected. To start, stop, and reset counters, set one of the system variables `innodb_monitor_enable`, `innodb_monitor_disable`, `innodb_monitor_reset`, or `innodb_monitor_reset_all`, using the name of the counter, the name of the module, a wildcard match for such a name using the “%” character, or the special keyword `all`.

For usage information, see Section 14.15.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

The `INNODB_METRICS` table has these columns:

- `NAME`

  A unique name for the counter.

- `SUBSYSTEM`

  The aspect of `InnoDB` that the metric applies to.

- `COUNT`

  The value since the counter was enabled.

- `MAX_COUNT`

  The maximum value since the counter was enabled.

- `MIN_COUNT`

The minimum value since the counter was enabled.

- `AVG_COUNT`

  The average value since the counter was enabled.

- `COUNT_RESET`

  The counter value since it was last reset. (The `_RESET` columns act like the lap counter on a stopwatch: you can measure the activity during some time interval, while the cumulative figures are still available in `COUNT`, `MAX_COUNT`, and so on.)

- `MAX_COUNT_RESET`

  The maximum counter value since it was last reset.

- `MIN_COUNT_RESET`

  The minimum counter value since it was last reset.

- `AVG_COUNT_RESET`

  The average counter value since it was last reset.

- `TIME_ENABLED`

  The timestamp of the last start.

- `TIME_DISABLED`

  The timestamp of the last stop.

- `TIME_ELAPSED`

  The elapsed time in seconds since the counter started.

- `TIME_RESET`

  The timestamp of the last reset.

- `STATUS`

  Whether the counter is still running (`enabled`) or stopped (`disabled`).

- `TYPE`

  Whether the item is a cumulative counter, or measures the current value of some resource.

- `COMMENT`

  The counter description.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='dml_inserts'\G
*************************** 1. row ***************************
           NAME: dml_inserts
      SUBSYSTEM: dml
          COUNT: 3
      MAX_COUNT: 3
      MIN_COUNT: NULL
      AVG_COUNT: 0.046153846153846156
    COUNT_RESET: 3
MAX_COUNT_RESET: 3
MIN_COUNT_RESET: NULL
```

```
AVG_COUNT_RESET: NULL
   TIME_ENABLED: 2014-12-04 14:18:28
  TIME_DISABLED: NULL
   TIME_ELAPSED: 65
     TIME_RESET: NULL
         STATUS: enabled
           TYPE: status_counter
        COMMENT: Number of rows inserted
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

21.30.16 The INFORMATION_SCHEMA INNODB_SYS_COLUMNS Table

The `INNODB_SYS_COLUMNS` table provides metadata about `InnoDB` table columns, equivalent to the information from the `SYS_COLUMNS` table in the `InnoDB` data dictionary.

For related usage information and examples, see Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”.

The `INNODB_SYS_COLUMNS` table has these columns:

- `TABLE_ID`

  An identifier representing the table associated with the column; the same value as `INNODB_SYS_TABLES.TABLE_ID`.

- `NAME`

  The name of the column. These names can be uppercase or lowercase depending on the `lower_case_table_names` setting. There are no special system-reserved names for columns.

- `POS`

  The ordinal position of the column within the table, starting from 0 and incrementing sequentially. When a column is dropped, the remaining columns are reordered so that the sequence has no gaps.

- `MTYPE`

  Stands for “main type”. A numeric identifier for the column type. 1 = `VARCHAR`, 2 = `CHAR`, 3 = `FIXBINARY`, 4 = `BINARY`, 5 = `BLOB`, 6 = `INT`, 7 = `SYS_CHILD`, 8 = `SYS`, 9 = `FLOAT`, 10 = `DOUBLE`, 11 = `DECIMAL`, 12 = `VARMYSQL`, 13 = `MYSQL`.

- `PRTYPE`

  The `InnoDB` “precise type”, a binary value with bits representing MySQL data type, character set code, and nullability.

- `LEN`

  The column length, for example 4 for `INT` and 8 for `BIGINT`. For character columns in multibyte character sets, this length value is the maximum length in bytes needed to represent a definition such as `VARCHAR(N)`; that is, it might be `2*N`, `3*N`, and so on depending on the character encoding.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_COLUMNS where TABLE_ID = 71\G
*************************** 1. row ***************************
TABLE_ID: 71
    NAME: col1
     POS: 0
```

```
   MTYPE: 6
  PRTYPE: 1027
     LEN: 4
*************************** 2. row ***************************
TABLE_ID: 71
    NAME: col2
     POS: 1
   MTYPE: 2
  PRTYPE: 524542
     LEN: 10
*************************** 3. row ***************************
TABLE_ID: 71
    NAME: col3
     POS: 2
   MTYPE: 1
  PRTYPE: 524303
     LEN: 10
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

21.30.17 The INFORMATION_SCHEMA INNODB_SYS_DATAFILES Table

The `INNODB_SYS_DATAFILES` table provides data file path information for `InnoDB` tablespaces, equivalent to the information in the `SYS_DATAFILES` table in the `InnoDB` data dictionary.

For related usage information and examples, see Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”.

The `INNODB_SYS_DATAFILES` table has these columns:

- `SPACE`

  The tablespace ID.

- `PATH`

  The tablespace data file path. If a file-per-table tablespace is created in a location outside the MySQL data directory, the path value is a fully qualified directory path. Otherwise, the path is relative to the data directory.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_DATAFILES WHERE SPACE = 57\G
*************************** 1. row ***************************
SPACE: 57
 PATH: ./test/t1.ibd
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

21.30.18 The INFORMATION_SCHEMA INNODB_SYS_FIELDS Table

The `INNODB_SYS_FIELDS` table provides metadata about the key columns (fields) of `InnoDB` indexes, equivalent to the information from the `SYS_FIELDS` table in the `InnoDB` data dictionary.

For related usage information and examples, see Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”.

The INNODB_SYS_FIELDS table has these columns:

- INDEX_ID

  An identifier for the index associated with this key field; the same value as INNODB_SYS_INDEXES.INDEX_ID.

- NAME

  The name of the original column from the table; the same value as INNODB_SYS_COLUMNS.NAME.

- POS

  The ordinal position of the key field within the index, starting from 0 and incrementing sequentially. When a column is dropped, the remaining columns are reordered so that the sequence has no gaps.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_FIELDS WHERE INDEX_ID = 117\G
*************************** 1. row ***************************
INDEX_ID: 117
    NAME: col1
     POS: 0
```

Notes

- You must have the PROCESS privilege to query this table.
- Use the INFORMATION_SCHEMA COLUMNS table or the SHOW COLUMNS statement to view additional information about the columns of this table, including data types and default values.

21.30.19 The INFORMATION_SCHEMA INNODB_SYS_FOREIGN Table

The INNODB_SYS_FOREIGN table provides metadata about InnoDB foreign keys, equivalent to the information from the SYS_FOREIGN table in the InnoDB data dictionary.

For related usage information and examples, see Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”.

The INNODB_SYS_FOREIGN table has these columns:

- ID

  The name (not a numeric value) of the foreign key index, preceded by the schema (database) name (for example, test/products_fk).

- FOR_NAME

  The name of the child table in this foreign key relationship.

- REF_NAME

  The name of the parent table in this foreign key relationship.

- N_COLS

  The number of columns in the foreign key index.

- TYPE

  A collection of bit flags with information about the foreign key column, ORed together. 0 = ON DELETE/UPDATE RESTRICT, 1 = ON DELETE CASCADE, 2 = ON DELETE SET NULL, 4 = ON UPDATE CASCADE, 8 = ON UPDATE SET NULL, 16 = ON DELETE NO ACTION, 32 = ON UPDATE NO ACTION.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_FOREIGN\G
*************************** 1. row ***************************
      ID: test/fk1
FOR_NAME: test/child
REF_NAME: test/parent
  N_COLS: 1
    TYPE: 1
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

21.30.20 The INFORMATION_SCHEMA INNODB_SYS_FOREIGN_COLS Table

The `INNODB_SYS_FOREIGN_COLS` table provides status information about the columns of `InnoDB` foreign keys, equivalent to the information from the `SYS_FOREIGN_COLS` table in the `InnoDB` data dictionary.

For related usage information and examples, see Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”.

The `INNODB_SYS_FOREIGN_COLS` table has these columns:

- `ID`

  The foreign key index associated with this index key field, using the same value as `INNODB_SYS_FOREIGN.ID`.

- `FOR_COL_NAME`

  The name of the associated column in the child table.

- `REF_COL_NAME`

  The name of the associated column in the parent table.

- `POS`

  The ordinal position of this key field within the foreign key index, starting from 0.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_FOREIGN_COLS WHERE ID = 'test/fk1'\G
*************************** 1. row ***************************
          ID: test/fk1
FOR_COL_NAME: parent_id
REF_COL_NAME: id
         POS: 0
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

21.30.21 The INFORMATION_SCHEMA INNODB_SYS_INDEXES Table

The `INNODB_SYS_INDEXES` table provides metadata about `InnoDB` indexes, equivalent to the information in the internal `SYS_INDEXES` table in the `InnoDB` data dictionary.

For related usage information and examples, see Section 14.15.3, "InnoDB INFORMATION_SCHEMA System Tables".

The `INNODB_SYS_INDEXES` table has these columns:

- `INDEX_ID`

  An identifier for the index. Index identifiers are unique across all the databases in an instance.

- `NAME`

  The name of the index. Most indexes created implicitly by `InnoDB` have consistent names but the index names are not necessarily unique. Examples: `PRIMARY` for a primary key index, `GEN_CLUST_INDEX` for the index representing a primary key when one is not specified, and `ID_IND`, `FOR_IND`, and `REF_IND` for foreign key constraints.

- `TABLE_ID`

  An identifier representing the table associated with the index; the same value as `INNODB_SYS_TABLES.TABLE_ID`.

- `TYPE`

  A numeric value derived from bit-level information that identifies the index type. 0 = nonunique secondary index; 1 = automatically generated clustered index (`GEN_CLUST_INDEX`); 2 = unique nonclustered index; 3 = clustered index; 32 = full-text index

- `N_FIELDS`

  The number of columns in the index key. For `GEN_CLUST_INDEX` indexes, this value is 0 because the index is created using an artificial value rather than a real table column.

- `PAGE_NO`

  The root page number of the index B-tree. For full-text indexes, the `PAGE_NO` column is unused and set to -1 (`FIL_NULL`) because the full-text index is laid out in several B-trees (auxiliary tables).

- `SPACE`

  An identifier for the tablespace where the index resides. 0 means the `InnoDB` system tablespace. Any other number represents a table created with a separate `.ibd` file in file-per-table mode. This identifier stays the same after a `TRUNCATE TABLE` statement. Because all indexes for a table reside in the same tablespace as the table, this value is not necessarily unique.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_INDEXES WHERE TABLE_ID = 74\G
*************************** 1. row ***************************
INDEX_ID: 116
    NAME: GEN_CLUST_INDEX
TABLE_ID: 74
    TYPE: 1
N_FIELDS: 0
 PAGE_NO: 3
   SPACE: 60
*************************** 2. row ***************************
INDEX_ID: 117
    NAME: i1
TABLE_ID: 74
    TYPE: 0
N_FIELDS: 1
 PAGE_NO: 4
```

```
   SPACE: 60
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

21.30.22 The INFORMATION_SCHEMA INNODB_SYS_TABLES Table

The `INNODB_SYS_TABLES` table provides metadata about `InnoDB` tables, equivalent to the information from the `SYS_TABLES` table in the `InnoDB` data dictionary.

For related usage information and examples, see Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”.

The `INNODB_SYS_TABLES` table has these columns:

- `TABLE_ID`

  An identifier for the `InnoDB` table. This value is unique across all databases in the instance.

- `NAME`

  The name of the table, preceded by the schema (database) name where appropriate (for example, `test/t1`). Names of databases and user tables are in the same case as they were originally defined, possibly influenced by the `lower_case_table_names` setting.

- `FLAG`

  A numeric value that represents bit-level information about table format and storage characteristics.

- `N_COLS`

  The number of columns in the table. The number reported includes three hidden columns that are created by `InnoDB` (`DB_ROW_ID`, `DB_TRX_ID`, and `DB_ROLL_PTR`).

- `SPACE`

  An identifier for the tablespace where the table resides. 0 means the `InnoDB` system tablespace. Any other number represents a table created in file-per-table mode with a separate `.ibd` file. This identifier stays the same after a `TRUNCATE TABLE` statement. Other than the zero value, this identifier is unique for tables across all the databases in the instance.

- `FILE_FORMAT`

  The table's file format (`Antelope` or `Barracuda`).

- `ROW_FORMAT`

  The table's row format (`Compact`, `Redundant`, `Dynamic`, or `Compressed`).

- `ZIP_PAGE_SIZE`

  The zip page size. Applies only to tables with a row format of `Compressed`.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_TABLES WHERE TABLE_ID = 74\G
*************************** 1. row ***************************
     TABLE_ID: 74
         NAME: test/t1
         FLAG: 1
       N_COLS: 6
```

```
       SPACE: 60
  FILE_FORMAT: Antelope
   ROW_FORMAT: Compact
ZIP_PAGE_SIZE: 0
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

21.30.23 The INFORMATION_SCHEMA INNODB_SYS_TABLESPACES Table

The `INNODB_SYS_TABLESPACES` table provides metadata about `InnoDB` tablespaces, equivalent to the information in the `SYS_TABLESPACES` table in the `InnoDB` data dictionary.

For related usage information and examples, see Section 14.15.3, "InnoDB INFORMATION_SCHEMA System Tables".

The `INNODB_SYS_TABLESPACES` table has these columns:

- `SPACE`

  The tablespace ID.

- `NAME`

  The schema (database) and table name.

- `FLAG`

  A numeric value that represents bit-level information about tablespace format and storage characteristics.

- `FILE_FORMAT`

  The tablespace file format (for example, Antelope or Barracuda). The data in this field is interpreted from the tablespace flags information that resides in the .ibd file. For more information about `InnoDB` file formats, see Section 14.10, "InnoDB File-Format Management".

- `ROW_FORMAT`

  The tablespace row format (`Compact or Redundant`, `Dynamic`, or `Compressed`). The data in this column is interpreted from the tablespace flags information that resides in the `.ibd` file.

- `PAGE_SIZE`

  The tablespace page size. The data in this column is interpreted from the tablespace flags information that resides in the `.ibd` file.

- `ZIP_PAGE_SIZE`

  The tablespace zip page size. The data in this column is interpreted from the tablespace flags information that resides in the `.ibd` file.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_TABLESPACES WHERE SPACE = 57\G
*************************** 1. row ***************************
       SPACE: 57
        NAME: test/t1
        FLAG: 0
  FILE_FORMAT: Antelope
   ROW_FORMAT: Compact or Redundant
```

```
    PAGE_SIZE: 16384
ZIP_PAGE_SIZE: 0
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- Because tablespace flags are always zero for all Antelope file formats (unlike table flags), there is no way to determine from this flag integer if the tablespace row format is Redundant or Compact. As a result, the possible values for the `ROW_FORMAT` field are “Compact or Redundant”, “Compressed”, or “Dynamic.”

21.30.24 The INFORMATION_SCHEMA INNODB_SYS_TABLESTATS View

The `INNODB_SYS_TABLESTATS` table provides a view of low-level status information about `InnoDB` tables. This data is used by the MySQL optimizer to calculate which index to use when querying an `InnoDB` table. This information is derived from in-memory data structures rather than data stored on disk. There is no corresponding internal `InnoDB` system table.

`InnoDB` tables are represented in this view if they have been opened since the last server restart and have not aged out of the table cache. Tables for which persistent stats are available are always represented in this view.

Table statistics are updated only for `DELETE` or `UPDATE` operations that modify indexed columns. Statistics are not updated by operations that modify only nonindexed columns.

`ANALYZE TABLE` clears table statistics and sets the `STATS_INITIALIZED` column to `Uninitialized`. Statistics are collected again the next time the table is accessed.

For related usage information and examples, see Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”.

The `INNODB_SYS_TABLESTATS` table has these columns:

- `TABLE_ID`

  An identifier representing the table for which statistics are available; the same value as `INNODB_SYS_TABLES.TABLE_ID`.

- `NAME`

  The name of the table; the same value as `INNODB_SYS_TABLES.NAME`.

- `STATS_INITIALIZED`

  The value is `Initialized` if the statistics are already collected, `Uninitialized` if not.

- `NUM_ROWS`

  The current estimated number of rows in the table. Updated after each DML operation. The value could be imprecise if uncommitted transactions are inserting into or deleting from the table.

- `CLUST_INDEX_SIZE`

  The number of pages on disk that store the clustered index, which holds the `InnoDB` table data in primary key order. This value might be null if no statistics are collected yet for the table.

- `OTHER_INDEX_SIZE`

  The number of pages on disk that store all secondary indexes for the table. This value might be null if no statistics are collected yet for the table.

- `MODIFIED_COUNTER`

  The number of rows modified by DML operations, such as `INSERT`, `UPDATE`, `DELETE`, and also cascade operations from foreign keys. This column is reset each time table statistics are recalculated

- `AUTOINC`

  The next number to be issued for any auto-increment-based operation. The rate at which the `AUTOINC` value changes depends on how many times auto-increment numbers have been requested and how many numbers are granted per request.

- `REF_COUNT`

  When this counter reaches zero, the table metadata can be evicted from the table cache.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SYS_TABLESTATS where TABLE_ID = 71\G
*************************** 1. row ***************************
         TABLE_ID: 71
             NAME: test/t1
STATS_INITIALIZED: Initialized
         NUM_ROWS: 1
 CLUST_INDEX_SIZE: 1
 OTHER_INDEX_SIZE: 0
 MODIFIED_COUNTER: 1
          AUTOINC: 0
        REF_COUNT: 1
```

Notes

- This table is useful primarily for expert-level performance monitoring, or when developing performance-related extensions for MySQL.
- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

21.30.25 The INFORMATION_SCHEMA INNODB_TRX Table

The `INNODB_TRX` table provides information about every transaction currently executing inside `InnoDB`, including whether the transaction is waiting for a lock, when the transaction started, and the SQL statement the transaction is executing, if any.

For usage information, see Section 14.15.2.1, “Using InnoDB Transaction and Locking Information”.

The `INNODB_TRX` table has these columns:

- `TRX_ID`

  A unique transaction ID number, internal to `InnoDB`. (Starting in MySQL 5.6, these IDs are not created for transactions that are read only and nonlocking. For details, see Section 8.5.3, “Optimizing InnoDB Read-Only Transactions”.)

- `TRX_WEIGHT`

  The weight of a transaction, reflecting (but not necessarily the exact count of) the number of rows altered and the number of rows locked by the transaction. To resolve a deadlock, `InnoDB` selects the transaction with the smallest weight as the “victim” to roll back. Transactions that have changed nontransactional tables are considered heavier than others, regardless of the number of altered and locked rows.

- `TRX_STATE`

The transaction execution state. Permitted values are `RUNNING`, `LOCK WAIT`, `ROLLING BACK`, and `COMMITTING`.

- `TRX_STARTED`

  The transaction start time.

- `TRX_REQUESTED_LOCK_ID`

  The ID of the lock the transaction is currently waiting for, if `TRX_STATE` is `LOCK WAIT`; otherwise `NULL`. To obtain details about the lock, join this column with the `LOCK_ID` column of the `INNODB_LOCKS` table.

- `TRX_WAIT_STARTED`

  The time when the transaction started waiting on the lock, if `TRX_STATE` is `LOCK WAIT`; otherwise `NULL`.

- `TRX_MYSQL_THREAD_ID`

  The MySQL thread ID. To obtain details about the thread, join this column with the `ID` column of the `INFORMATION_SCHEMA PROCESSLIST` table, but see Section 14.15.2.3, “Persistence and Consistency of InnoDB Transaction and Locking Information”.

- `TRX_QUERY`

  The SQL statement that is being executed by the transaction.

- `TRX_OPERATION_STATE`

  The transaction's current operation, if any; otherwise `NULL`.

- `TRX_TABLES_IN_USE`

  The number of `InnoDB` tables used while processing the current SQL statement of this transaction.

- `TRX_TABLES_LOCKED`

  The number of `InnoDB` tables that the current SQL statement has row locks on. (Because these are row locks, not table locks, the tables can usually still be read from and written to by multiple transactions, despite some rows being locked.)

- `TRX_LOCK_STRUCTS`

  The number of locks reserved by the transaction.

- `TRX_LOCK_MEMORY_BYTES`

  The total size taken up by the lock structures of this transaction in memory.

- `TRX_ROWS_LOCKED`

  The approximate number or rows locked by this transaction. The value might include delete-marked rows that are physically present but not visible to the transaction.

- `TRX_ROWS_MODIFIED`

  The number of modified and inserted rows in this transaction.

- `TRX_CONCURRENCY_TICKETS`

  A value indicating how much work the current transaction can do before being swapped out, as specified by the `innodb_concurrency_tickets` system variable.

- TRX_ISOLATION_LEVEL

  The isolation level of the current transaction.

- TRX_UNIQUE_CHECKS

  Whether unique checks are turned on or off for the current transaction. For example, they might be turned off during a bulk data load.

- TRX_FOREIGN_KEY_CHECKS

  Whether foreign key checks are turned on or off for the current transaction. For example, they might be turned off during a bulk data load.

- TRX_LAST_FOREIGN_KEY_ERROR

  The detailed error message for the last foreign key error, if any; otherwise NULL.

- TRX_ADAPTIVE_HASH_LATCHED

  Whether the adaptive hash index is locked by the current transaction. (Only a single transaction at a time can modify the adaptive hash index.)

- TRX_ADAPTIVE_HASH_TIMEOUT

  Whether to relinquish the search latch immediately for the adaptive hash index, or reserve it across calls from MySQL. When there is no adaptive hash index contention, this value remains zero and statements reserve the latch until they finish. During times of contention, it counts down to zero, and statements release the latch immediately after each row lookup.

- TRX_IS_READ_ONLY

  A value of 1 indicates the transaction is read only.

- TRX_AUTOCOMMIT_NON_LOCKING

  A value of 1 indicates the transaction is a SELECT statement that does not use the FOR UPDATE or LOCK IN SHARED MODE clauses, and is executing with autocommit enabled so that the transaction contains only this one statement. When this column and TRX_IS_READ_ONLY are both 1, InnoDB optimizes the transaction to reduce the overhead associated with transactions that change table data.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TRX\G
*************************** 1. row ***************************
                    trx_id: 3298
                 trx_state: RUNNING
               trx_started: 2014-11-19 13:54:39
     trx_requested_lock_id: NULL
          trx_wait_started: NULL
                trx_weight: 316436
       trx_mysql_thread_id: 2
                 trx_query: DELETE FROM employees.salaries WHERE salary > 65000
       trx_operation_state: updating or deleting
         trx_tables_in_use: 1
         trx_tables_locked: 1
          trx_lock_structs: 1621
     trx_lock_memory_bytes: 243240
           trx_rows_locked: 759343
         trx_rows_modified: 314815
   trx_concurrency_tickets: 0
       trx_isolation_level: REPEATABLE READ
         trx_unique_checks: 1
    trx_foreign_key_checks: 1
trx_last_foreign_key_error: NULL
```

```
 trx_adaptive_hash_latched: 0
 trx_adaptive_hash_timeout: 10000
          trx_is_read_only: 0
trx_autocommit_non_locking: 0
```

Notes

- Use this table to help diagnose performance problems that occur during times of heavy concurrent load. Its contents are updated as described in Section 14.15.2.3, “Persistence and Consistency of InnoDB Transaction and Locking Information”.
- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

21.31 INFORMATION_SCHEMA NDB Cluster Tables

The following sections provide information about `INFORMATION_SCHEMA` tables which are specific to NDB Cluster. (The `FILES` table is available in standard MySQL 5.6 but is not used there.) The `ndb_transid_mysql_connection_map` table is implemented as an `INFORMATION_SCHEMA` plugin available only in NDB Cluster binaries or source, and does not exist in MySQL Server 5.6.

Additional statistical and other data about NDB Cluster transactions, operations, threads, blocks, and other aspects of performance can be obtained from the tables in the `ndbinfo` database. For information about these tables, see Section 18.5.14, “ndbinfo: The NDB Cluster Information Database”.

21.31.1 The INFORMATION_SCHEMA FILES Table

The `FILES` table provides information about the files in which MySQL `NDB` Disk Data tables are stored.

The `FILES` table has these columns:

- `FILE_ID`

  A file identifier. `FILE_ID` column values are auto-generated.

- `FILE_NAME`

  The name of an `UNDO` log file created by `CREATE LOGFILE GROUP` or `ALTER LOGFILE GROUP`, or of a data file created by `CREATE TABLESPACE` or `ALTER TABLESPACE`.

- `FILE_TYPE`

  One of the values `UNDO LOG`, `DATAFILE`, or `TABLESPACE`.

- `TABLESPACE_NAME`

  The name of the tablespace with which the file is associated.

- `TABLE_CATALOG`

  This value is always empty.

- `TABLE_SCHEMA`

  This value is always `NULL`.

- `TABLE_NAME`

  The name of the Disk Data table with which the file is associated, if any.

- `LOGFILE_GROUP_NAME`

The name of the log file group to which the log file or data file belongs.

- `LOGFILE_GROUP_NUMBER`

  For an `UNDO` log file, the auto-generated ID number of the log file group to which the log file belongs.

- `ENGINE`

  For an NDB Cluster Disk Data log file or data file, this value always `NDB` or `NDBCLUSTER`.

- `FULLTEXT_KEYS`

  For an NDB Cluster Disk Data log file or data file, this value is always empty.

- `DELETED_ROWS`

  This value is always `NULL`.

- `UPDATE_COUNT`

  This value is always `NULL`.

- `FREE_EXTENTS`

  The number of extents which have not yet been used by the file.

- `TOTAL_EXTENTS`

  The total number of extents allocated to the file.

- `EXTENT_SIZE`

  The size of an extent for the file in bytes.

- `INITIAL_SIZE`

  The size of the file in bytes. This is the same value that was used in the `INITIAL_SIZE` clause of the `CREATE LOGFILE GROUP`, `ALTER LOGFILE GROUP`, `CREATE TABLESPACE`, or `ALTER TABLESPACE` statement used to create the file.

- `MAXIMUM_SIZE`

  For NDB Cluster Disk Data files, this value is always the same as the `INITIAL_SIZE` value.

- `AUTOEXTEND_SIZE`

  For NDB Cluster Disk Data files, this value is always empty.

- `CREATION_TIME`

  The date and time when the file was created.

- `LAST_UPDATE_TIME`

  The date and time when the file was last modified.

- `LAST_ACCESS_TIME`

  The date and time when the file was last accessed by the server.

- `RECOVER_TIME`

  For NDB Cluster Disk Data files, this value is always `0`.

- `TRANSACTION_COUNTER`

  For NDB Cluster Disk Data files, this value is always `0`.

- `VERSION`

  For NDB Cluster Disk Data files, this value is always `NULL`.

- `ROW_FORMAT`

  For NDB Cluster Disk Data files, this value is always `NULL`.

- `TABLE_ROWS`

  For NDB Cluster Disk Data files, this value is always `NULL`.

- `AVG_ROW_LENGTH`

  For NDB Cluster Disk Data files, this value is always `NULL`.

- `DATA_LENGTH`

  For NDB Cluster Disk Data files, this value is always `NULL`.

- `MAX_DATA_LENGTH`

  For NDB Cluster Disk Data files, this value is always `NULL`.

- `INDEX_LENGTH`

  For NDB Cluster Disk Data files, this value is always `NULL`.

- `DATA_FREE`

  For NDB Cluster Disk Data files, this value is always `NULL`.

- `CREATE_TIME`

  For NDB Cluster Disk Data files, this value is always `NULL`.

- `UPDATE_TIME`

  For NDB Cluster Disk Data files, this value is always `NULL`.

- `CHECK_TIME`

  For NDB Cluster Disk Data files, this value is always `NULL`.

- `CHECKSUM`

  For NDB Cluster Disk Data files, this value is always `NULL`.

- `STATUS`

  For NDB Cluster Disk Data files, this value is always `NORMAL`.

- `EXTRA`

  For NDB Cluster Disk Data files, the `EXTRA` column shows which data node the file belongs to (each data node having its own copy), as well as the size of its undo buffer. Suppose that you use this statement on an NDB Cluster with four data nodes:

  ```
  CREATE LOGFILE GROUP mygroup
      ADD UNDOFILE 'new_undo.dat'
      INITIAL_SIZE 2G
  ```

```
    ENGINE NDB;
```

After running the `CREATE LOGFILE GROUP` statement successfully, you should see a result similar to the one shown here for this query against the `FILES` table:

```
mysql> SELECT LOGFILE_GROUP_NAME, FILE_TYPE, EXTRA
         FROM INFORMATION_SCHEMA.FILES
         WHERE FILE_NAME = 'new_undo.dat';

+--------------------+-----------+----------------------------------------+
| LOGFILE_GROUP_NAME | FILE_TYPE | EXTRA                                  |
+--------------------+-----------+----------------------------------------+
| mygroup            | UNDO LOG  | CLUSTER_NODE=5;UNDO_BUFFER_SIZE=8388608 |
| mygroup            | UNDO LOG  | CLUSTER_NODE=6;UNDO_BUFFER_SIZE=8388608 |
| mygroup            | UNDO LOG  | CLUSTER_NODE=7;UNDO_BUFFER_SIZE=8388608 |
| mygroup            | UNDO LOG  | CLUSTER_NODE=8;UNDO_BUFFER_SIZE=8388608 |
+--------------------+-----------+----------------------------------------+
```

Notes

- The `FILES` table is a nonstandard `INFORMATION_SCHEMA` table.

NDB Notes

- This table provides information about Disk Data *files* only; you cannot use it for determining disk space allocation or availability for individual `NDB` tables. However, it is possible to see how much space is allocated for each `NDB` table having data stored on disk—as well as how much remains available for storage of data on disk for that table—using `ndb_desc`. For more information, see Section 18.4.10, “`ndb_desc` — Describe NDB Tables”.

- The `CREATION_TIME`, `LAST_UPDATE_TIME`, and `LAST_ACCESSED` values are as reported by the operating system, and are not supplied by the `NDB` storage engine. Where no value is provided by the operating system, these columns display `0000-00-00 00:00:00`.

- The difference between the `TOTAL EXTENTS` and `FREE_EXTENTS` columns is the number of extents currently in use by the file:

```
SELECT TOTAL_EXTENTS - FREE_EXTENTS AS extents_used
    FROM INFORMATION_SCHEMA.FILES
    WHERE FILE_NAME = 'myfile.dat';
```

To approximate the amount of disk space in use by the file, multiply that difference by the value of the `EXTENT_SIZE` column, which gives the size of an extent for the file in bytes:

```
SELECT (TOTAL_EXTENTS - FREE_EXTENTS) * EXTENT_SIZE AS bytes_used
    FROM INFORMATION_SCHEMA.FILES
    WHERE FILE_NAME = 'myfile.dat';
```

Similarly, you can estimate the amount of space that remains available in a given file by multiplying `FREE_EXTENTS` by `EXTENT_SIZE`:

```
SELECT FREE_EXTENTS * EXTENT_SIZE AS bytes_free
    FROM INFORMATION_SCHEMA.FILES
    WHERE FILE_NAME = 'myfile.dat';
```

Important

The byte values produced by the preceding queries are approximations only, and their precision is inversely proportional to the value of `EXTENT_SIZE`. That is, the larger `EXTENT_SIZE` becomes, the less accurate the approximations are.

It is also important to remember that once an extent is used, it cannot be freed again without dropping the data file of which it is a part. This means that deletes from a Disk Data table do *not* release disk space.

The extent size can be set in a CREATE TABLESPACE statement. For more information, see Section 13.1.18, “CREATE TABLESPACE Statement”.

- An additional row is present in the FILES table following the creation of a logfile group. This row has NULL for the value of the FILE_NAME column. For this row, the value of the FILE_ID column is always 0, that of the FILE_TYPE column is always UNDO LOG, and that of the STATUS column is always NORMAL. The value of the ENGINE column is always NDBCLUSTER.

  The FREE_EXTENTS column in this row shows the total number of free extents available to all undo files belonging to a given log file group whose name and number are shown in the LOGFILE_GROUP_NAME and LOGFILE_GROUP_NUMBER columns, respectively.

  Suppose there are no existing log file groups on your NDB Cluster, and you create one using the following statement:

```
mysql> CREATE LOGFILE GROUP lg1
         ADD UNDOFILE 'undofile.dat'
         INITIAL_SIZE = 16M
         UNDO_BUFFER_SIZE = 1M
         ENGINE = NDB;
```

  You can now see this NULL row when you query the FILES table:

```
mysql> SELECT DISTINCT
         FILE_NAME AS File,
         FREE_EXTENTS AS Free,
         TOTAL_EXTENTS AS Total,
         EXTENT_SIZE AS Size,
         INITIAL_SIZE AS Initial
         FROM INFORMATION_SCHEMA.FILES;
+--------------+---------+---------+------+----------+
| File         | Free    | Total   | Size | Initial  |
+--------------+---------+---------+------+----------+
| undofile.dat |    NULL | 4194304 |    4 | 16777216 |
| NULL         | 4184068 |    NULL |    4 |     NULL |
+--------------+---------+---------+------+----------+
```

  The total number of free extents available for undo logging is always somewhat less than the sum of the TOTAL_EXTENTS column values for all undo files in the log file group due to overhead required for maintaining the undo files. This can be seen by adding a second undo file to the log file group, then repeating the previous query against the FILES table:

```
mysql> ALTER LOGFILE GROUP lg1
         ADD UNDOFILE 'undofile02.dat'
         INITIAL_SIZE = 4M
         ENGINE = NDB;

mysql> SELECT DISTINCT
         FILE_NAME AS File,
         FREE_EXTENTS AS Free,
         TOTAL_EXTENTS AS Total,
         EXTENT_SIZE AS Size,
         INITIAL_SIZE AS Initial
         FROM INFORMATION_SCHEMA.FILES;
+---------------+---------+---------+------+----------+
| File          | Free    | Total   | Size | Initial  |
+---------------+---------+---------+------+----------+
| undofile.dat  |    NULL | 4194304 |    4 | 16777216 |
| undofile02.dat |    NULL | 1048576 |    4 |  4194304 |
| NULL          | 5223944 |    NULL |    4 |     NULL |
+---------------+---------+---------+------+----------+
```

  The amount of free space in bytes which is available for undo logging by Disk Data tables using this log file group can be approximated by multiplying the number of free extents by the initial size:

```
mysql> SELECT
         FREE_EXTENTS AS 'Free Extents',
```

```
          FREE_EXTENTS * EXTENT_SIZE AS 'Free Bytes'
          FROM INFORMATION_SCHEMA.FILES
          WHERE LOGFILE_GROUP_NAME = 'lg1'
          AND FILE_NAME IS NULL;
+--------------+------------+
| Free Extents | Free Bytes |
+--------------+------------+
|      5223944 |   20895776 |
+--------------+------------+
```

If you create an NDB Cluster Disk Data table and then insert some rows into it, you can see approximately how much space remains for undo logging afterward, for example:

```
mysql> CREATE TABLESPACE ts1
          ADD DATAFILE 'data1.dat'
          USE LOGFILE GROUP lg1
          INITIAL_SIZE 512M
          ENGINE = NDB;

mysql> CREATE TABLE dd (
          c1 INT NOT NULL PRIMARY KEY,
          c2 INT,
          c3 DATE
          )
          TABLESPACE ts1 STORAGE DISK
          ENGINE = NDB;

mysql> INSERT INTO dd VALUES
          (NULL, 1234567890, '2007-02-02'),
          (NULL, 1126789005, '2007-02-03'),
          (NULL, 1357924680, '2007-02-04'),
          (NULL, 1642097531, '2007-02-05');

mysql> SELECT
          FREE_EXTENTS AS 'Free Extents',
          FREE_EXTENTS * EXTENT_SIZE AS 'Free Bytes'
          FROM INFORMATION_SCHEMA.FILES
          WHERE LOGFILE_GROUP_NAME = 'lg1'
          AND FILE_NAME IS NULL;
+--------------+------------+
| Free Extents | Free Bytes |
+--------------+------------+
|      5207565 |   20830260 |
+--------------+------------+
```

- An additional row is present in the `FILES` table for any NDB Cluster tablespace, whether or not any data files are associated with the tablespace. This row has `NULL` for the value of the `FILE_NAME` column. For this row, the value of the `FILE_ID` column is always `0`, that of the `FILE_TYPE` column is always `TABLESPACE`, and that of the `STATUS` column is always `NORMAL`. The value of the `ENGINE` column is always `NDBCLUSTER`.
- For additional information, and examples of creating and dropping NDB Cluster Disk Data objects, see Section 18.5.10, “NDB Cluster Disk Data Tables”.

21.31.2 The INFORMATION_SCHEMA ndb_transid_mysql_connection_map Table

The `ndb_transid_mysql_connection_map` table provides a mapping between `NDB` transactions, `NDB` transaction coordinators, and MySQL Servers attached to an NDB Cluster as API nodes. This information is used when populating the `server_operations` and `server_transactions` tables of the `ndbinfo` NDB Cluster information database.

The `ndb_transid_mysql_connection_map` table has these columns:

- `mysql_connection_id`

  The MySQL server connection ID.

- `node_id`

  The transaction coordinator node ID.

- `ndb_transid`

  The `NDB` transaction ID.

Notes

The `mysql_connection_id` value is the same as the connection or session ID shown in the output of `SHOW PROCESSLIST`.

There are no `SHOW` statements associated with this table.

This is a nonstandard table, specific to NDB Cluster. It is implemented as an `INFORMATION_SCHEMA` plugin; you can verify that it is supported by checking the output of `SHOW PLUGINS`. If `ndb_transid_mysql_connection_map` support is enabled, the output from this statement includes a plugin having this name, of type `INFORMATION SCHEMA`, and having status `ACTIVE`, as shown here (using emphasized text):

```
mysql> SHOW PLUGINS;
+----------------------------------+--------+--------------------+---------+---------+
| Name                             | Status | Type               | Library | License |
+----------------------------------+--------+--------------------+---------+---------+
| binlog                           | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| mysql_native_password            | ACTIVE | AUTHENTICATION     | NULL    | GPL     |
| mysql_old_password               | ACTIVE | AUTHENTICATION     | NULL    | GPL     |
| CSV                              | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| MEMORY                           | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| MRG_MYISAM                       | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| MyISAM                           | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| PERFORMANCE_SCHEMA               | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| BLACKHOLE                        | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| ARCHIVE                          | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| ndbcluster                       | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| ndbinfo                          | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| ndb_transid_mysql_connection_map | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| InnoDB                           | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| INNODB_TRX                       | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_LOCKS                     | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_LOCK_WAITS                | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMP                       | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMP_RESET                 | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMPMEM                    | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMPMEM_RESET              | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| partition                        | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
+----------------------------------+--------+--------------------+---------+---------+
22 rows in set (0.00 sec)
```

The plugin is enabled by default. You can disable it (or force the server not to run unless the plugin starts) by starting the server with the `--ndb-transid-mysql-connection-map` option. If the plugin is disabled, the status is shown by `SHOW PLUGINS` as `DISABLED`. The plugin cannot be enabled or disabled at runtime.

Although the names of this table and its columns are displayed using lowercase, you can use uppercase or lowercase when referring to them in SQL statements.

For this table to be created, the MySQL Server must be a binary supplied with the NDB Cluster distribution, or one built from the NDB Cluster sources with `NDB` storage engine support enabled. It is not available in the standard MySQL 5.6 Server.

21.32 INFORMATION_SCHEMA Thread Pool Tables

The following sections describe the `INFORMATION_SCHEMA` tables associated with the thread pool plugin (see Section 5.5.3, “MySQL Enterprise Thread Pool”). They provide information about thread pool operation:

- `TP_THREAD_GROUP_STATE`: Information about thread pool thread group states
- `TP_THREAD_GROUP_STATS`: Thread group statistics
- `TP_THREAD_STATE`: Information about thread pool thread states

Rows in these tables represent snapshots in time. In the case of `TP_THREAD_STATE`, all rows for a thread group comprise a snapshot in time. Thus, the MySQL server holds the mutex of the thread group while producing the snapshot. But it does not hold mutexes on all thread groups at the same time, to prevent a statement against `TP_THREAD_STATE` from blocking the entire MySQL server.

The thread pool `INFORMATION_SCHEMA` tables are implemented by individual plugins and the decision whether to load one can be made independently of the others (see Section 5.5.3.2, “Thread Pool Installation”). However, the content of all the tables depends on the thread pool plugin being enabled. If a table plugin is enabled but the thread pool plugin is not, the table becomes visible and can be accessed but is empty.

21.32.1 The INFORMATION_SCHEMA TP_THREAD_GROUP_STATE Table

The `TP_THREAD_GROUP_STATE` table has one row per thread group in the thread pool. Each row provides information about the current state of a group.

The `TP_THREAD_GROUP_STATE` table has these columns:

- `TP_GROUP_ID`

  The thread group ID. This is a unique key within the table.

- `CONSUMER_THREADS`

  The number of consumer threads. There is at most one thread ready to start executing if the active threads become stalled or blocked.

- `RESERVE_THREADS`

  The number of threads in the reserved state. This means that they are not started until there is a need to wake a new thread and there is no consumer thread. This is where most threads end up when the thread group has created more threads than needed for normal operation. Often a thread group needs additional threads for a short while and then does not need them again for a while. In this case, they go into the reserved state and remain until needed again. They take up some extra memory resources, but no extra computing resources.

- `CONNECT_THREAD_COUNT`

  The number of threads that are processing or waiting to process connection initialization and authentication. There can be a maximum of four connection threads per thread group; these threads expire after a period of inactivity.

  This column was added in MySQL 5.6.36.

- `CONNECTION_COUNT`

  The number of connections using this thread group.

- `QUEUED_QUERIES`

  The number of statements waiting in the high-priority queue.

- `QUEUED_TRANSACTIONS`

The number of statements waiting in the low-priority queue. These are the initial statements for transactions that have not started, so they also represent queued transactions.

- `STALL_LIMIT`

  The value of the `thread_pool_stall_limit` system variable for the thread group. This is the same value for all thread groups.

- `PRIO_KICKUP_TIMER`

  The value of the `thread_pool_prio_kickup_timer` system variable for the thread group. This is the same value for all thread groups.

- `ALGORITHM`

  The value of the `thread_pool_algorithm` system variable for the thread group. This is the same value for all thread groups.

- `THREAD_COUNT`

  The number of threads started in the thread pool as part of this thread group.

- `ACTIVE_THREAD_COUNT`

  The number of threads active in executing statements.

- `STALLED_THREAD_COUNT`

  The number of stalled statements in the thread group. A stalled statement could be executing, but from a thread pool perspective it is stalled and making no progress. A long-running statement quickly ends up in this category.

- `WAITING_THREAD_NUMBER`

  If there is a thread handling the polling of statements in the thread group, this specifies the thread number within this thread group. It is possible that this thread could be executing a statement.

- `OLDEST_QUEUED`

  How long in milliseconds the oldest queued statement has been waiting for execution.

- `MAX_THREAD_IDS_IN_GROUP`

  The maximum thread ID of the threads in the group. This is the same as `MAX(TP_THREAD_NUMBER)` for the threads when selected from the `TP_THREAD_STATE` table. That is, these two queries are equivalent:

```
SELECT TP_GROUP_ID, MAX_THREAD_IDS_IN_GROUP
FROM TP_THREAD_GROUP_STATE;

SELECT TP_GROUP_ID, MAX(TP_THREAD_NUMBER)
FROM TP_THREAD_STATE GROUP BY TP_GROUP_ID;
```

21.32.2 The INFORMATION_SCHEMA TP_THREAD_GROUP_STATS Table

The `TP_THREAD_GROUP_STATS` table reports statistics per thread group. There is one row per group.

The `TP_THREAD_GROUP_STATS` table has these columns:

- `TP_GROUP_ID`

  The thread group ID. This is a unique key within the table.

- `CONNECTIONS_STARTED`

The number of connections started.

- `CONNECTIONS_CLOSED`

  The number of connections closed.

- `QUERIES_EXECUTED`

  The number of statements executed. This number is incremented when a statement starts executing, not when it finishes.

- `QUERIES_QUEUED`

  The number of statements received that were queued for execution. This does not count statements that the thread group was able to begin executing immediately without queuing, which can happen under the conditions described in Section 5.5.3.3, “Thread Pool Operation”.

- `THREADS_STARTED`

  The number of threads started.

- `PRIO_KICKUPS`

  The number of statements that have been moved from low-priority queue to high-priority queue based on the value of the `thread_pool_prio_kickup_timer` system variable. If this number increases quickly, consider increasing the value of that variable. A quickly increasing counter means that the priority system is not keeping transactions from starting too early. For `InnoDB`, this most likely means deteriorating performance due to too many concurrent transactions..

- `STALLED_QUERIES_EXECUTED`

  The number of statements that have become defined as stalled due to executing for longer than the value of the `thread_pool_stall_limit` system variable.

- `BECOME_CONSUMER_THREAD`

  The number of times thread have been assigned the consumer thread role.

- `BECOME_RESERVE_THREAD`

  The number of times threads have been assigned the reserve thread role.

- `BECOME_WAITING_THREAD`

  The number of times threads have been assigned the waiter thread role. When statements are queued, this happens very often, even in normal operation, so rapid increases in this value are normal in the case of a highly loaded system where statements are queued up.

- `WAKE_THREAD_STALL_CHECKER`

  The number of times the stall check thread decided to wake or create a thread to possibly handle some statements or take care of the waiter thread role.

- `SLEEP_WAITS`

  The number of `THD_WAIT_SLEEP` waits. These occur when threads go to sleep; for example, by calling the `SLEEP()` function.

- `DISK_IO_WAITS`

  The number of `THD_WAIT_DISKIO` waits. These occur when threads perform disk I/O that is likely to not hit the file system cache. Such waits occur when the buffer pool reads and writes data to disk, not for normal reads from and writes to files.

- `ROW_LOCK_WAITS`

  The number of `THD_WAIT_ROW_LOCK` waits for release of a row lock by another transaction.

- `GLOBAL_LOCK_WAITS`

  The number of `THD_WAIT_GLOBAL_LOCK` waits for a global lock to be released.

- `META_DATA_LOCK_WAITS`

  The number of `THD_WAIT_META_DATA_LOCK` waits for a metadata lock to be released.

- `TABLE_LOCK_WAITS`

  The number of `THD_WAIT_TABLE_LOCK` waits for a table to be unlocked that the statement needs to access.

- `USER_LOCK_WAITS`

  The number of `THD_WAIT_USER_LOCK` waits for a special lock constructed by the user thread.

- `BINLOG_WAITS`

  The number of `THD_WAIT_BINLOG_WAITS` waits for the binary log to become free.

- `GROUP_COMMIT_WAITS`

  The number of `THD_WAIT_GROUP_COMMIT` waits. These occur when a group commit must wait for the other parties to complete their part of a transaction.

- `FSYNC_WAITS`

  The number of `THD_WAIT_SYNC` waits for a file sync operation.

21.32.3 The INFORMATION_SCHEMA TP_THREAD_STATE Table

The `TP_THREAD_STATE` table has one row per thread created by the thread pool to handle connections.

The `TP_THREAD_STATE` table has these columns:

- `TP_GROUP_ID`

  The thread group ID.

- `TP_THREAD_NUMBER`

  The ID of the thread within its thread group. `TP_GROUP_ID` and `TP_THREAD_NUMBER` together provide a unique key within the table.

- `PROCESS_COUNT`

  The 10ms interval in which the statement that uses this thread is currently executing. 0 means no statement is executing, 1 means it is in the first 10ms, and so forth.

- `WAIT_TYPE`

  The type of wait for the thread. `NULL` means the thread is not blocked. Otherwise, the thread is blocked by a call to `thd_wait_begin()` and the value specifies the type of wait. The *`xxx`*`_WAIT` columns of the `TP_THREAD_GROUP_STATS` table accumulate counts for each wait type.

  The `WAIT_TYPE` value is a string that describes the type of wait, as shown in the following table.

Table 21.4 TP_THREAD_STATE Table WAIT_TYPE Values

Wait Type	Meaning
THD_WAIT_SLEEP	Waiting for sleep
THD_WAIT_DISKIO	Waiting for Disk IO
THD_WAIT_ROW_LOCK	Waiting for row lock
THD_WAIT_GLOBAL_LOCK	Waiting for global lock
THD_WAIT_META_DATA_LOCK	Waiting for metadata lock
THD_WAIT_TABLE_LOCK	Waiting for table lock
THD_WAIT_USER_LOCK	Waiting for user lock
THD_WAIT_BINLOG	Waiting for binlog
THD_WAIT_GROUP_COMMIT	Waiting for group commit
THD_WAIT_SYNC	Waiting for fsync

21.33 INFORMATION_SCHEMA Connection-Control Tables

The following sections describe the INFORMATION_SCHEMA tables associated with the CONNECTION_CONTROL plugin.

21.33.1 The INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS Table

This table provides information about the current number of consecutive failed connection attempts per account (user/host combination). The table was added in MySQL 5.6.35.

CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS has these columns:

- USERHOST

  The user/host combination indicating an account that has failed connection attempts, in *'user_name'@'host_name'* format.

- FAILED_ATTEMPTS

  The current number of consecutive failed connection attempts for the USERHOST value. This counts all failed attempts, regardless of whether they were delayed. The number of attempts for which the server added a delay to its response is the difference between the FAILED_ATTEMPTS value and the connection_control_failed_connections_threshold system variable value.

Notes

- The CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS plugin must be activated for this table to be available, and the CONNECTION_CONTROL plugin must be activated or the table contents are always empty. See Section 6.4.2, “The Connection-Control Plugins”.
- The table contains rows only for accounts that have had one or more consecutive failed connection attempts without a subsequent successful attempt. When an account connects successfully, its failed-connection count is reset to zero and the server removes any row corresponding to the account.
- Assigning a value to the connection_control_failed_connections_threshold system variable at runtime resets all accumulated failed-connection counters to zero, which causes the table to become empty.

21.34 INFORMATION_SCHEMA MySQL Enterprise Firewall Tables

The following sections describe the INFORMATION_SCHEMA tables associated with MySQL Enterprise Firewall (see Section 6.4.5, “MySQL Enterprise Firewall”). They provide views into the firewall in-memory data cache. These tables are available only if the appropriate firewall plugins are enabled.

21.34.1 The INFORMATION_SCHEMA MYSQL_FIREWALL_USERS Table

The MYSQL_FIREWALL_USERS table provides a view into the in-memory data cache for MySQL Enterprise Firewall. It lists names and operational modes of registered firewall account profiles. It is used in conjunction with the mysql.firewall_users system table that provides persistent storage of firewall data; see MySQL Enterprise Firewall Tables.

The MYSQL_FIREWALL_USERS table has these columns:

- USERHOST

  The account profile name. Each account name has the format *user_name@host_name*.

- MODE

  The current operational mode for the profile. Permitted mode values are OFF, DETECTING (as of MySQL 5.6.26), PROTECTING, RECORDING, and RESET. For details about their meanings, see Firewall Operational Concepts.

21.34.2 The INFORMATION_SCHEMA MYSQL_FIREWALL_WHITELIST Table

The MYSQL_FIREWALL_WHITELIST table provides a view into the in-memory data cache for MySQL Enterprise Firewall. It lists allowlist rules of registered firewall account profiles. It is used in conjunction with the mysql.firewall_whitelist system table that provides persistent storage of firewall data; see MySQL Enterprise Firewall Tables.

The MYSQL_FIREWALL_WHITELIST table has these columns:

- USERHOST

  The account profile name. Each account name has the format *user_name@host_name*.

- RULE

  A normalized statement indicating an acceptable statement pattern for the profile. A profile allowlist is the union of its rules.

21.35 Extensions to SHOW Statements

Some extensions to SHOW statements accompany the implementation of INFORMATION_SCHEMA:

- SHOW can be used to get information about the structure of INFORMATION_SCHEMA itself.
- Several SHOW statements accept a WHERE clause that provides more flexibility in specifying which rows to display.

INFORMATION_SCHEMA is an information database, so its name is included in the output from SHOW DATABASES. Similarly, SHOW TABLES can be used with INFORMATION_SCHEMA to obtain a list of its tables:

```
mysql> SHOW TABLES FROM INFORMATION_SCHEMA;
+---------------------------------------+
| Tables_in_INFORMATION_SCHEMA          |
+---------------------------------------+
| CHARACTER_SETS                        |
| COLLATIONS                            |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS                               |
```

```
| COLUMN_PRIVILEGES                     |
| ENGINES                               |
| EVENTS                                |
| FILES                                 |
| GLOBAL_STATUS                         |
| GLOBAL_VARIABLES                      |
| KEY_COLUMN_USAGE                      |
| PARTITIONS                            |
| PLUGINS                               |
| PROCESSLIST                           |
| REFERENTIAL_CONSTRAINTS               |
| ROUTINES                              |
| SCHEMATA                              |
| SCHEMA_PRIVILEGES                     |
| SESSION_STATUS                        |
| SESSION_VARIABLES                     |
| STATISTICS                            |
| TABLES                                |
| TABLE_CONSTRAINTS                     |
| TABLE_PRIVILEGES                      |
| TRIGGERS                              |
| USER_PRIVILEGES                       |
| VIEWS                                 |
+---------------------------------------+
```

`SHOW COLUMNS` and `DESCRIBE` can display information about the columns in individual `INFORMATION_SCHEMA` tables.

`SHOW` statements that accept a `LIKE` clause to limit the rows displayed also permit a `WHERE` clause that specifies more general conditions that selected rows must satisfy:

```
SHOW CHARACTER SET
SHOW COLLATION
SHOW COLUMNS
SHOW DATABASES
SHOW FUNCTION STATUS
SHOW INDEX
SHOW OPEN TABLES
SHOW PROCEDURE STATUS
SHOW STATUS
SHOW TABLE STATUS
SHOW TABLES
SHOW TRIGGERS
SHOW VARIABLES
```

The `WHERE` clause, if present, is evaluated against the column names displayed by the `SHOW` statement. For example, the `SHOW CHARACTER SET` statement produces these output columns:

```
mysql> SHOW CHARACTER SET;
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |      2 |
| dec8     | DEC West European           | dec8_swedish_ci     |      1 |
| cp850    | DOS West European           | cp850_general_ci    |      1 |
| hp8      | HP West European            | hp8_english_ci      |      1 |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |      1 |
| latin1   | cp1252 West European        | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |      1 |
...
```

To use a `WHERE` clause with `SHOW CHARACTER SET`, you would refer to those column names. As an example, the following statement displays information about character sets for which the default collation contains the string `'japanese'`:

```
mysql> SHOW CHARACTER SET WHERE `Default collation` LIKE '%japanese%';
+---------+---------------------------+---------------------+--------+
| Charset | Description               | Default collation   | Maxlen |
+---------+---------------------------+---------------------+--------+
| ujis    | EUC-JP Japanese           | ujis_japanese_ci    |      3 |
```

```
| sjis    | Shift-JIS Japanese        | sjis_japanese_ci    |      2 |
| cp932   | SJIS for Windows Japanese | cp932_japanese_ci   |      2 |
| eucjpms | UJIS for Windows Japanese | eucjpms_japanese_ci |      3 |
+---------+---------------------------+---------------------+--------+
```

This statement displays the multibyte character sets:

```
mysql> SHOW CHARACTER SET WHERE Maxlen > 1;
+---------+---------------------------+---------------------+--------+
| Charset | Description               | Default collation   | Maxlen |
+---------+---------------------------+---------------------+--------+
| big5    | Big5 Traditional Chinese  | big5_chinese_ci     |      2 |
| ujis    | EUC-JP Japanese           | ujis_japanese_ci    |      3 |
| sjis    | Shift-JIS Japanese        | sjis_japanese_ci    |      2 |
| euckr   | EUC-KR Korean             | euckr_korean_ci     |      2 |
| gb2312  | GB2312 Simplified Chinese | gb2312_chinese_ci   |      2 |
| gbk     | GBK Simplified Chinese    | gbk_chinese_ci      |      2 |
| utf8    | UTF-8 Unicode             | utf8_general_ci     |      3 |
| ucs2    | UCS-2 Unicode             | ucs2_general_ci     |      2 |
| cp932   | SJIS for Windows Japanese | cp932_japanese_ci   |      2 |
| eucjpms | UJIS for Windows Japanese | eucjpms_japanese_ci |      3 |
+---------+---------------------------+---------------------+--------+
```

Chapter 22 MySQL Performance Schema

Table of Contents

22.1 Performance Schema Quick Start 3065
22.2 Performance Schema Build Configuration 3070
22.3 Performance Schema Startup Configuration 3071
22.4 Performance Schema Runtime Configuration 3073
22.4.1 Performance Schema Event Timing 3074
22.4.2 Performance Schema Event Filtering 3077
22.4.3 Event Pre-Filtering 3079
22.4.4 Pre-Filtering by Instrument 3079
22.4.5 Pre-Filtering by Object 3081
22.4.6 Pre-Filtering by Thread 3082
22.4.7 Pre-Filtering by Consumer 3083
22.4.8 Example Consumer Configurations 3086
22.4.9 Naming Instruments or Consumers for Filtering Operations 3090
22.4.10 Determining What Is Instrumented 3091
22.5 Performance Schema Queries 3091
22.6 Performance Schema Instrument Naming Conventions 3091
22.7 Performance Schema Status Monitoring 3094
22.8 Performance Schema Atom and Molecule Events 3097
22.9 Performance Schema Tables for Current and Historical Events 3097
22.10 Performance Schema Statement Digests 3099
22.11 Performance Schema General Table Characteristics 3102
22.12 Performance Schema Table Descriptions 3103
22.12.1 Performance Schema Table Index 3104
22.12.2 Performance Schema Setup Tables 3105
22.12.3 Performance Schema Instance Tables 3110
22.12.4 Performance Schema Wait Event Tables 3114
22.12.5 Performance Schema Stage Event Tables 3119
22.12.6 Performance Schema Statement Event Tables 3123
22.12.7 Performance Schema Connection Tables 3131
22.12.8 Performance Schema Connection Attribute Tables 3134
22.12.9 Performance Schema Summary Tables 3137
22.12.10 Performance Schema Miscellaneous Tables 3149
22.13 Performance Schema Option and Variable Reference 3156
22.14 Performance Schema Command Options 3158
22.15 Performance Schema System Variables 3159
22.16 Performance Schema Status Variables 3169
22.17 Performance Schema and Plugins 3171
22.18 Using the Performance Schema to Diagnose Problems 3172
22.18.1 Query Profiling Using Performance Schema 3173
22.19 Restrictions on Performance Schema 3174

The MySQL Performance Schema is a feature for monitoring MySQL Server execution at a low level. The Performance Schema has these characteristics:

- The Performance Schema provides a way to inspect internal execution of the server at runtime. It is implemented using the `PERFORMANCE_SCHEMA` storage engine and the `performance_schema` database. The Performance Schema focuses primarily on performance data. This differs from `INFORMATION_SCHEMA`, which serves for inspection of metadata.

- The Performance Schema monitors server events. An “event” is anything the server does that takes time and has been instrumented so that timing information can be collected. In general, an event could be a function call, a wait for the operating system, a stage of an SQL statement execution such

as parsing or sorting, or an entire statement or group of statements. Event collection provides access to information about synchronization calls (such as for mutexes) file and table I/O, table locks, and so forth for the server and for several storage engines.

- Performance Schema events are distinct from events written to the server's binary log (which describe data modifications) and Event Scheduler events (which are a type of stored program).
- Performance Schema events are specific to a given instance of the MySQL Server. Performance Schema tables are considered local to the server, and changes to them are not replicated or written to the binary log.
- Current events are available, as well as event histories and summaries. This enables you to determine how many times instrumented activities were performed and how much time they took. Event information is available to show the activities of specific threads, or activity associated with particular objects such as a mutex or file.
- The `PERFORMANCE_SCHEMA` storage engine collects event data using “instrumentation points” in server source code.
- Collected events are stored in tables in the `performance_schema` database. These tables can be queried using `SELECT` statements like other tables.
- Performance Schema configuration can be modified dynamically by updating tables in the `performance_schema` database through SQL statements. Configuration changes affect data collection immediately.
- Tables in the Performance Schema are in-memory tables that use no persistent on-disk storage. The contents are repopulated beginning at server startup and discarded at server shutdown.
- Monitoring is available on all platforms supported by MySQL.

  Some limitations might apply: The types of timers might vary per platform. Instruments that apply to storage engines might not be implemented for all storage engines. Instrumentation of each third-party engine is the responsibility of the engine maintainer. See also Section 22.19, “Restrictions on Performance Schema”.
- Data collection is implemented by modifying the server source code to add instrumentation. There are no separate threads associated with the Performance Schema, unlike other features such as replication or the Event Scheduler.

The Performance Schema is intended to provide access to useful information about server execution while having minimal impact on server performance. The implementation follows these design goals:

- Activating the Performance Schema causes no changes in server behavior. For example, it does not cause thread scheduling to change, and it does not cause query execution plans (as shown by `EXPLAIN`) to change.
- No memory allocation is done beyond that which occurs during server startup. By using early allocation of structures with a fixed size, it is never necessary to resize or reallocate them, which is critical for achieving good runtime performance.
- Server monitoring occurs continuously and unobtrusively with very little overhead. Activating the Performance Schema does not make the server unusable.
- The parser is unchanged. There are no new keywords or statements.
- Execution of server code proceeds normally even if the Performance Schema fails internally.
- When there is a choice between performing processing during event collection initially or during event retrieval later, priority is given to making collection faster. This is because collection is ongoing whereas retrieval is on demand and might never happen at all.

- It is easy to add new instrumentation points.
- Instrumentation is versioned. If the instrumentation implementation changes, previously instrumented code should continue to work. This benefits developers of third-party plugins because it is not necessary to upgrade each plugin to stay synchronized with the latest Performance Schema changes.

Note

The MySQL `sys` schema is a set of objects that provides convenient access to data collected by the Performance Schema. In MySQL 5.7, the `sys` schema is installed by default. For MySQL 5.6, you can obtain it from the schema development website at https://github.com/mysql/mysql-sys. For usage instructions, see MySQL sys Schema.

22.1 Performance Schema Quick Start

This section briefly introduces the Performance Schema with examples that show how to use it. For additional examples, see Section 22.18, “Using the Performance Schema to Diagnose Problems”.

For the Performance Schema to be available, support for it must have been configured when MySQL was built. You can verify whether this is the case by checking the server's help output. If the Performance Schema is available, the output includes several variables with names that begin with `performance_schema`:

```
shell> mysqld --verbose --help
...
  --performance_schema
                      Enable the performance schema.
  --performance_schema_events_waits_history_long_size=#
                      Number of rows in events_waits_history_long.
...
```

If such variables do not appear in the output, your server has not been built to support the Performance Schema. In this case, see Section 22.2, “Performance Schema Build Configuration”.

Assuming that the Performance Schema is available, it is enabled by default. To enable or disable it explicitly, start the server with the `performance_schema` variable set to an appropriate value. For example, use these lines in your `my.cnf` file:

```
[mysqld]
performance_schema=ON
```

When the server starts, it sees `performance_schema` and attempts to initialize the Performance Schema. To verify successful initialization, use this statement:

```
mysql> SHOW VARIABLES LIKE 'performance_schema';
+--------------------+-------+
| Variable_name      | Value |
+--------------------+-------+
| performance_schema | ON    |
+--------------------+-------+
```

A value of `ON` means that the Performance Schema initialized successfully and is ready for use. A value of `OFF` means that some error occurred. Check the server error log for information about what went wrong.

The Performance Schema is implemented as a storage engine. If this engine is available (which you should already have checked earlier), you should see it listed with a `SUPPORT` value of `YES` in the output from the `INFORMATION_SCHEMA.ENGINES` table or the `SHOW ENGINES` statement:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.ENGINES
       WHERE ENGINE='PERFORMANCE_SCHEMA'\G
*************************** 1. row ***************************
     ENGINE: PERFORMANCE_SCHEMA
```

```
     SUPPORT: YES
     COMMENT: Performance Schema
TRANSACTIONS: NO
          XA: NO
  SAVEPOINTS: NO

mysql> SHOW ENGINES\G
...
      Engine: PERFORMANCE_SCHEMA
     Support: YES
     Comment: Performance Schema
Transactions: NO
          XA: NO
  Savepoints: NO
...
```

The `PERFORMANCE_SCHEMA` storage engine operates on tables in the `performance_schema` database. You can make `performance_schema` the default database so that references to its tables need not be qualified with the database name:

```
mysql> USE performance_schema;
```

Performance Schema tables are stored in the `performance_schema` database. Information about the structure of this database and its tables can be obtained, as for any other database, by selecting from the `INFORMATION_SCHEMA` database or by using `SHOW` statements. For example, use either of these statements to see what Performance Schema tables exist:

```
mysql> SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
       WHERE TABLE_SCHEMA = 'performance_schema';
+----------------------------------------------------+
| TABLE_NAME                                         |
+----------------------------------------------------+
| accounts                                           |
| cond_instances                                     |
| events_stages_current                              |
| events_stages_history                              |
| events_stages_history_long                         |
| events_stages_summary_by_account_by_event_name     |
| events_stages_summary_by_host_by_event_name        |
| events_stages_summary_by_thread_by_event_name      |
| events_stages_summary_by_user_by_event_name        |
| events_stages_summary_global_by_event_name         |
| events_statements_current                          |
| events_statements_history                          |
| events_statements_history_long                     |
...
| file_instances                                     |
| file_summary_by_event_name                         |
| file_summary_by_instance                           |
| host_cache                                         |
| hosts                                              |
| mutex_instances                                    |
| objects_summary_global_by_type                     |
| performance_timers                                 |
| rwlock_instances                                   |
| session_account_connect_attrs                      |
| session_connect_attrs                              |
| setup_actors                                       |
| setup_consumers                                    |
| setup_instruments                                  |
| setup_objects                                      |
| setup_timers                                       |
| socket_instances                                   |
| socket_summary_by_event_name                       |
| socket_summary_by_instance                         |
| table_io_waits_summary_by_index_usage              |
| table_io_waits_summary_by_table                    |
| table_lock_waits_summary_by_table                  |
| threads                                            |
| users                                              |
+----------------------------------------------------+
```

```
mysql> SHOW TABLES FROM performance_schema;
+----------------------------------------------------+
| Tables_in_performance_schema                       |
+----------------------------------------------------+
| accounts                                           |
| cond_instances                                     |
| events_stages_current                              |
| events_stages_history                              |
| events_stages_history_long                         |
...
```

The number of Performance Schema tables increases over time as implementation of additional instrumentation proceeds.

The name of the `performance_schema` database is lowercase, as are the names of tables within it. Queries should specify the names in lowercase.

To see the structure of individual tables, use `SHOW CREATE TABLE`:

```
mysql> SHOW CREATE TABLE performance_schema.setup_consumers\G
*************************** 1. row ***************************
       Table: setup_consumers
Create Table: CREATE TABLE `setup_consumers` (
  `NAME` varchar(64) NOT NULL,
  `ENABLED` enum('YES','NO') NOT NULL
) ENGINE=PERFORMANCE_SCHEMA DEFAULT CHARSET=utf8
```

Table structure is also available by selecting from tables such as `INFORMATION_SCHEMA.COLUMNS` or by using statements such as `SHOW COLUMNS`.

Tables in the `performance_schema` database can be grouped according to the type of information in them: Current events, event histories and summaries, object instances, and setup (configuration) information. The following examples illustrate a few uses for these tables. For detailed information about the tables in each group, see Section 22.12, “Performance Schema Table Descriptions”.

Initially, not all instruments and consumers are enabled, so the performance schema does not collect all events. To turn all of these on and enable event timing, execute two statements (the row counts may differ depending on MySQL version):

```
mysql> UPDATE performance_schema.setup_instruments
       SET ENABLED = 'YES', TIMED = 'YES';
Query OK, 338 rows affected (0.12 sec)
mysql> UPDATE performance_schema.setup_consumers
       SET ENABLED = 'YES';
Query OK, 8 rows affected (0.00 sec)
```

To see what the server is doing at the moment, examine the `events_waits_current` table. It contains one row per thread showing each thread's most recent monitored event:

```
mysql> SELECT *
       FROM performance_schema.events_waits_current\G
*************************** 1. row ***************************
            THREAD_ID: 0
             EVENT_ID: 5523
         END_EVENT_ID: 5523
           EVENT_NAME: wait/synch/mutex/mysys/THR_LOCK::mutex
               SOURCE: thr_lock.c:525
          TIMER_START: 201660494489586
            TIMER_END: 201660494576112
           TIMER_WAIT: 86526
                SPINS: NULL
        OBJECT_SCHEMA: NULL
          OBJECT_NAME: NULL
           INDEX_NAME: NULL
          OBJECT_TYPE: NULL
OBJECT_INSTANCE_BEGIN: 142270668
     NESTING_EVENT_ID: NULL
   NESTING_EVENT_TYPE: NULL
```

```
            OPERATION: lock
      NUMBER_OF_BYTES: NULL
                FLAGS: 0
...
```

This event indicates that thread 0 was waiting for 86,526 picoseconds to acquire a lock on `THR_LOCK::mutex`, a mutex in the `mysys` subsystem. The first few columns provide the following information:

- The ID columns indicate which thread the event comes from and the event number.
- `EVENT_NAME` indicates what was instrumented and `SOURCE` indicates which source file contains the instrumented code.
- The timer columns show when the event started and stopped and how long it took. If an event is still in progress, the `TIMER_END` and `TIMER_WAIT` values are `NULL`. Timer values are approximate and expressed in picoseconds. For information about timers and event time collection, see Section 22.4.1, “Performance Schema Event Timing”.

The history tables contain the same kind of rows as the current-events table but have more rows and show what the server has been doing “recently” rather than “currently.” The `events_waits_history` and `events_waits_history_long` tables contain the most recent 10 events per thread and most recent 10,000 events, respectively. For example, to see information for recent events produced by thread 13, do this:

```
mysql> SELECT EVENT_ID, EVENT_NAME, TIMER_WAIT
       FROM performance_schema.events_waits_history
       WHERE THREAD_ID = 13
       ORDER BY EVENT_ID;
+----------+----------------------------------------+------------+
| EVENT_ID | EVENT_NAME                             | TIMER_WAIT |
+----------+----------------------------------------+------------+
|       86 | wait/synch/mutex/mysys/THR_LOCK::mutex |     686322 |
|       87 | wait/synch/mutex/mysys/THR_LOCK_malloc |     320535 |
|       88 | wait/synch/mutex/mysys/THR_LOCK_malloc |     339390 |
|       89 | wait/synch/mutex/mysys/THR_LOCK_malloc |     377100 |
|       90 | wait/synch/mutex/sql/LOCK_plugin       |     614673 |
|       91 | wait/synch/mutex/sql/LOCK_open         |     659925 |
|       92 | wait/synch/mutex/sql/THD::LOCK_thd_data |     494001 |
|       93 | wait/synch/mutex/mysys/THR_LOCK_malloc |     222489 |
|       94 | wait/synch/mutex/mysys/THR_LOCK_malloc |     214947 |
|       95 | wait/synch/mutex/mysys/LOCK_alarm      |     312993 |
+----------+----------------------------------------+------------+
```

As new events are added to a history table, older events are discarded if the table is full.

Summary tables provide aggregated information for all events over time. The tables in this group summarize event data in different ways. To see which instruments have been executed the most times or have taken the most wait time, sort the `events_waits_summary_global_by_event_name` table on the `COUNT_STAR` or `SUM_TIMER_WAIT` column, which correspond to a `COUNT(*)` or `SUM(TIMER_WAIT)` value, respectively, calculated over all events:

```
mysql> SELECT EVENT_NAME, COUNT_STAR
       FROM performance_schema.events_waits_summary_global_by_event_name
       ORDER BY COUNT_STAR DESC LIMIT 10;
+----------------------------------------------------+------------+
| EVENT_NAME                                         | COUNT_STAR |
+----------------------------------------------------+------------+
| wait/synch/mutex/mysys/THR_LOCK_malloc             |       6419 |
| wait/io/file/sql/FRM                               |        452 |
| wait/synch/mutex/sql/LOCK_plugin                   |        337 |
| wait/synch/mutex/mysys/THR_LOCK_open               |        187 |
| wait/synch/mutex/mysys/LOCK_alarm                  |        147 |
| wait/synch/mutex/sql/THD::LOCK_thd_data            |        115 |
| wait/io/file/myisam/kfile                          |        102 |
| wait/synch/mutex/sql/LOCK_global_system_variables |         89 |
| wait/synch/mutex/mysys/THR_LOCK::mutex             |         89 |
| wait/synch/mutex/sql/LOCK_open                     |         88 |
```

```
+----------------------------------------------------+------------+

mysql> SELECT EVENT_NAME, SUM_TIMER_WAIT
       FROM performance_schema.events_waits_summary_global_by_event_name
       ORDER BY SUM_TIMER_WAIT DESC LIMIT 10;
+----------------------------------------+----------------+
| EVENT_NAME                             | SUM_TIMER_WAIT |
+----------------------------------------+----------------+
| wait/io/file/sql/MYSQL_LOG             |     1599816582 |
| wait/synch/mutex/mysys/THR_LOCK_malloc |     1530083250 |
| wait/io/file/sql/binlog_index          |     1385291934 |
| wait/io/file/sql/FRM                   |     1292823243 |
| wait/io/file/myisam/kfile              |      411193611 |
| wait/io/file/myisam/dfile              |      322401645 |
| wait/synch/mutex/mysys/LOCK_alarm      |      145126935 |
| wait/io/file/sql/casetest              |      104324715 |
| wait/synch/mutex/sql/LOCK_plugin       |       86027823 |
| wait/io/file/sql/pid                   |       72591750 |
+----------------------------------------+----------------+
```

These results show that the `THR_LOCK_malloc` mutex is “hot,” both in terms of how often it is used and amount of time that threads wait attempting to acquire it.

Note

The `THR_LOCK_malloc` mutex is used only in debug builds. In production builds it is not hot because it is nonexistent.

Instance tables document what types of objects are instrumented. An instrumented object, when used by the server, produces an event. These tables provide event names and explanatory notes or status information. For example, the `file_instances` table lists instances of instruments for file I/O operations and their associated files:

```
mysql> SELECT *
       FROM performance_schema.file_instances\G
*************************** 1. row ***************************
 FILE_NAME: /opt/mysql-log/60500/binlog.000007
EVENT_NAME: wait/io/file/sql/binlog
OPEN_COUNT: 0
*************************** 2. row ***************************
 FILE_NAME: /opt/mysql/60500/data/mysql/tables_priv.MYI
EVENT_NAME: wait/io/file/myisam/kfile
OPEN_COUNT: 1
*************************** 3. row ***************************
 FILE_NAME: /opt/mysql/60500/data/mysql/columns_priv.MYI
EVENT_NAME: wait/io/file/myisam/kfile
OPEN_COUNT: 1
...
```

Setup tables are used to configure and display monitoring characteristics. For example, `setup_instruments` lists the set of instruments for which events can be collected and shows which of them are enabled:

```
mysql> SELECT * FROM performance_schema.setup_instruments;
+----------------------------------------------------+---------+-------+
| NAME                                               | ENABLED | TIMED |
+----------------------------------------------------+---------+-------+
...
| stage/sql/end                                      | NO      | NO    |
| stage/sql/executing                                | NO      | NO    |
| stage/sql/init                                     | NO      | NO    |
| stage/sql/insert                                   | NO      | NO    |
...
| statement/sql/load                                 | YES     | YES   |
| statement/sql/grant                                | YES     | YES   |
| statement/sql/check                                | YES     | YES   |
| statement/sql/flush                                | YES     | YES   |
...
| wait/synch/mutex/sql/LOCK_global_read_lock         | YES     | YES   |
| wait/synch/mutex/sql/LOCK_global_system_variables | YES     | YES   |
```

```
| wait/synch/mutex/sql/LOCK_lock_db                 | YES     | YES   |
| wait/synch/mutex/sql/LOCK_manager                 | YES     | YES   |
...
| wait/synch/rwlock/sql/LOCK_grant                  | YES     | YES   |
| wait/synch/rwlock/sql/LOGGER::LOCK_logger         | YES     | YES   |
| wait/synch/rwlock/sql/LOCK_sys_init_connect       | YES     | YES   |
| wait/synch/rwlock/sql/LOCK_sys_init_slave         | YES     | YES   |
...
| wait/io/file/sql/binlog                           | YES     | YES   |
| wait/io/file/sql/binlog_index                     | YES     | YES   |
| wait/io/file/sql/casetest                         | YES     | YES   |
| wait/io/file/sql/dbopt                            | YES     | YES   |
...
```

To understand how to interpret instrument names, see Section 22.6, “Performance Schema Instrument Naming Conventions”.

To control whether events are collected for an instrument, set its `ENABLED` value to `YES` or `NO`. For example:

```
mysql> UPDATE performance_schema.setup_instruments
       SET ENABLED = 'NO'
       WHERE NAME = 'wait/synch/mutex/sql/LOCK_mysql_create_db';
```

The Performance Schema uses collected events to update tables in the `performance_schema` database, which act as “consumers” of event information. The `setup_consumers` table lists the available consumers and which are enabled:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+--------------------------------+---------+
| NAME                           | ENABLED |
+--------------------------------+---------+
| events_stages_current          | NO      |
| events_stages_history          | NO      |
| events_stages_history_long     | NO      |
| events_statements_current      | YES     |
| events_statements_history      | NO      |
| events_statements_history_long | NO      |
| events_waits_current           | NO      |
| events_waits_history           | NO      |
| events_waits_history_long      | NO      |
| global_instrumentation         | YES     |
| thread_instrumentation         | YES     |
| statements_digest              | YES     |
+--------------------------------+---------+
```

To control whether the Performance Schema maintains a consumer as a destination for event information, set its `ENABLED` value.

For more information about the setup tables and how to use them to control event collection, see Section 22.4.2, “Performance Schema Event Filtering”.

There are some miscellaneous tables that do not fall into any of the previous groups. For example, `performance_timers` lists the available event timers and their characteristics. For information about timers, see Section 22.4.1, “Performance Schema Event Timing”.

22.2 Performance Schema Build Configuration

For the Performance Schema to be available, it must be configured into the MySQL server at build time. Binary MySQL distributions provided by Oracle Corporation are configured to support the Performance Schema. If you use a binary MySQL distribution from another provider, check with the provider whether the distribution has been appropriately configured.

If you build MySQL from a source distribution, enable the Performance Schema by running `CMake` with the `WITH_PERFSCHEMA_STORAGE_ENGINE` option enabled:

```
shell> cmake . -DWITH_PERFSCHEMA_STORAGE_ENGINE=1
```

Configuring MySQL with the `-DWITHOUT_PERFSCHEMA_STORAGE_ENGINE=1` option prevents inclusion of the Performance Schema, so if you want it included, do not use this option. See Section 2.9.7, "MySQL Source-Configuration Options".

If you install MySQL over a previous installation that was configured without the Performance Schema (or with an older version of the Performance Schema that has missing or out-of-date tables). One indication of this issue is the presence of messages such as the following in the error log:

```
[ERROR] Native table 'performance_schema'.'events_waits_history'
has the wrong structure
[ERROR] Native table 'performance_schema'.'events_waits_history_long'
has the wrong structure
...
```

To correct that problem, perform the MySQL upgrade procedure. See Section 2.11, "Upgrading MySQL".

To verify whether a server was built with Performance Schema support, check its help output. If the Performance Schema is available, the output includes several variables with names that begin with `performance_schema`:

```
shell> mysqld --verbose --help
...
  --performance_schema
                      Enable the performance schema.
  --performance_schema_events_waits_history_long_size=#
                      Number of rows in events_waits_history_long.
...
```

You can also connect to the server and look for a line that names the `PERFORMANCE_SCHEMA` storage engine in the output from `SHOW ENGINES`:

```
mysql> SHOW ENGINES\G
...
      Engine: PERFORMANCE_SCHEMA
     Support: YES
     Comment: Performance Schema
Transactions: NO
          XA: NO
  Savepoints: NO
...
```

If the Performance Schema was not configured into the server at build time, no row for `PERFORMANCE_SCHEMA` appears in the output from `SHOW ENGINES`. You might see `performance_schema` listed in the output from `SHOW DATABASES`, but it contains no tables and you cannot use it.

A line for `PERFORMANCE_SCHEMA` in the `SHOW ENGINES` output means that the Performance Schema is available, not that it is enabled. To enable it, you must do so at server startup, as described in the next section.

22.3 Performance Schema Startup Configuration

To use the MySQL Performance Schema, it must be enabled at server startup to enable event collection to occur.

Assuming that the Performance Schema is available, it is enabled by default. To enable or disable it explicitly, start the server with the `performance_schema` variable set to an appropriate value. For example, use these lines in your `my.cnf` file:

```
[mysqld]
performance_schema=ON
```

If the server is unable to allocate any internal buffer during Performance Schema initialization, the Performance Schema disables itself and sets `performance_schema` to `OFF`, and the server runs without instrumentation.

The Performance Schema also permits instrument and consumer configuration at server startup.

To control an instrument at server startup, use an option of this form:

```
--performance-schema-instrument='instrument_name=value'
```

Here, *`instrument_name`* is an instrument name such as `wait/synch/mutex/sql/LOCK_open`, and *`value`* is one of these values:

- `OFF`, `FALSE`, or `0`: Disable the instrument
- `ON`, `TRUE`, or `1`: Enable and time the instrument
- `COUNTED`: Enable and count (rather than time) the instrument

Each `--performance-schema-instrument` option can specify only one instrument name, but multiple instances of the option can be given to configure multiple instruments. In addition, patterns are permitted in instrument names to configure instruments that match the pattern. To configure all condition synchronization instruments as enabled and counted, use this option:

```
--performance-schema-instrument='wait/synch/cond/%=COUNTED'
```

To disable all instruments, use this option:

```
--performance-schema-instrument='%=OFF'
```

Longer instrument name strings take precedence over shorter pattern names, regardless of order. For information about specifying patterns to select instruments, see Section 22.4.9, “Naming Instruments or Consumers for Filtering Operations”.

An unrecognized instrument name is ignored. It is possible that a plugin installed later may create the instrument, at which time the name is recognized and configured.

To control a consumer at server startup, use an option of this form:

```
--performance-schema-consumer-consumer_name=value
```

Here, *`consumer_name`* is a consumer name such as `events_waits_history`, and *`value`* is one of these values:

- `OFF`, `FALSE`, or `0`: Do not collect events for the consumer
- `ON`, `TRUE`, or `1`: Collect events for the consumer

For example, to enable the `events_waits_history` consumer, use this option:

```
--performance-schema-consumer-events-waits-history=ON
```

The permitted consumer names can be found by examining the `setup_consumers` table. Patterns are not permitted. Consumer names in the `setup_consumers` table use underscores, but for consumers set at startup, dashes and underscores within the name are equivalent.

The Performance Schema includes several system variables that provide configuration information:

```
mysql> SHOW VARIABLES LIKE 'perf%';
+----------------------------------------------------+---------+
| Variable_name                                      | Value   |
+----------------------------------------------------+---------+
| performance_schema                                 | ON      |
| performance_schema_accounts_size                   | 100     |
| performance_schema_digests_size                    | 200     |
| performance_schema_events_stages_history_long_size | 10000   |
| performance_schema_events_stages_history_size      | 10      |
| performance_schema_events_statements_history_long_size | 10000   |
| performance_schema_events_statements_history_size  | 10      |
| performance_schema_events_waits_history_long_size  | 10000   |
```

```
| performance_schema_events_waits_history_size           | 10      |
| performance_schema_hosts_size                          | 100     |
| performance_schema_max_cond_classes                    | 80      |
| performance_schema_max_cond_instances                  | 1000    |
...
```

The `performance_schema` variable is `ON` or `OFF` to indicate whether the Performance Schema is enabled or disabled. The other variables indicate table sizes (number of rows) or memory allocation values.

Note

With the Performance Schema enabled, the number of Performance Schema instances affects the server memory footprint, perhaps to a large extent. It may be necessary to tune the values of Performance Schema system variables to find the number of instances that balances insufficient instrumentation against excessive memory consumption.

To change the value of Performance Schema system variables, set them at server startup. For example, put the following lines in a `my.cnf` file to change the sizes of the history tables for wait events:

```
[mysqld]
performance_schema
performance_schema_events_waits_history_size=20
performance_schema_events_waits_history_long_size=15000
```

The Performance Schema automatically sizes the values of several of its parameters at server startup if they are not set explicitly. For example, it sizes the parameters that control the sizes of the events waits tables this way. To see which parameters are autosized under this policy, use `mysqld --verbose --help` and look for those with a default value of −1, or see Section 22.15, “Performance Schema System Variables”.

For each autosized parameter that is not set at server startup, the Performance Schema determines how to set its value based on the value of the following system values, which are considered as “hints” about how you have configured your MySQL server:

```
max_connections
open_files_limit
table_definition_cache
table_open_cache
```

To override autosizing for a given parameter, set it to a value other than −1 at startup. In this case, the Performance Schema assigns it the specified value.

At runtime, `SHOW VARIABLES` displays the actual values that autosized parameters were set to.

If the Performance Schema is disabled, its autosized parameters remain set to −1 and `SHOW VARIABLES` displays −1.

22.4 Performance Schema Runtime Configuration

Specific Performance Schema features can be enabled at runtime to control which types of event collection occur.

Performance Schema setup tables contain information about monitoring configuration:

```
mysql> SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
       WHERE TABLE_SCHEMA = 'performance_schema'
       AND TABLE_NAME LIKE 'setup%';
+--------------------+
| TABLE_NAME         |
+--------------------+
| setup_actors       |
```

```
| setup_consumers   |
| setup_instruments |
| setup_objects     |
| setup_timers      |
+-------------------+
```

You can examine the contents of these tables to obtain information about Performance Schema monitoring characteristics. If you have the `UPDATE` privilege, you can change Performance Schema operation by modifying setup tables to affect how monitoring occurs. For additional details about these tables, see Section 22.12.2, “Performance Schema Setup Tables”.

To see which event timers are selected, query the `setup_timers` tables:

```
mysql> SELECT * FROM performance_schema.setup_timers;
+-----------+-------------+
| NAME      | TIMER_NAME  |
+-----------+-------------+
| idle      | MICROSECOND |
| wait      | CYCLE       |
| stage     | NANOSECOND  |
| statement | NANOSECOND  |
+-----------+-------------+
```

The `NAME` value indicates the type of instrument to which the timer applies, and `TIMER_NAME` indicates which timer applies to those instruments. The timer applies to instruments where their name begins with an element matching the `NAME` value.

To change the timer, update the `NAME` value. For example, to use the `NANOSECOND` timer for the `wait` timer:

```
mysql> UPDATE performance_schema.setup_timers
       SET TIMER_NAME = 'NANOSECOND'
       WHERE NAME = 'wait';
mysql> SELECT * FROM performance_schema.setup_timers;
+-----------+-------------+
| NAME      | TIMER_NAME  |
+-----------+-------------+
| idle      | MICROSECOND |
| wait      | NANOSECOND  |
| stage     | NANOSECOND  |
| statement | NANOSECOND  |
+-----------+-------------+
```

For discussion of timers, see Section 22.4.1, “Performance Schema Event Timing”.

The `setup_instruments` and `setup_consumers` tables list the instruments for which events can be collected and the types of consumers for which event information actually is collected, respectively. Other setup tables enable further modification of the monitoring configuration. Section 22.4.2, “Performance Schema Event Filtering”, discusses how you can modify these tables to affect event collection.

If there are Performance Schema configuration changes that must be made at runtime using SQL statements and you would like these changes to take effect each time the server starts, put the statements in a file and start the server with the `init_file` system variable set to name the file. This strategy can also be useful if you have multiple monitoring configurations, each tailored to produce a different kind of monitoring, such as casual server health monitoring, incident investigation, application behavior troubleshooting, and so forth. Put the statements for each monitoring configuration into their own file and specify the appropriate file as the `init_file` value when you start the server.

22.4.1 Performance Schema Event Timing

Events are collected by means of instrumentation added to the server source code. Instruments time events, which is how the Performance Schema provides an idea of how long events take. It is also possible to configure instruments not to collect timing information. This section discusses the available timers and their characteristics, and how timing values are represented in events.

Performance Schema Timers

Two Performance Schema tables provide timer information:

- `performance_timers` lists the available timers and their characteristics.
- `setup_timers` indicates which timers are used for which instruments.

Each timer row in `setup_timers` must refer to one of the timers listed in `performance_timers`.

Timers vary in precision and amount of overhead. To see what timers are available and their characteristics, check the `performance_timers` table:

```
mysql> SELECT * FROM performance_schema.performance_timers;
+-------------+-----------------+------------------+----------------+
| TIMER_NAME  | TIMER_FREQUENCY | TIMER_RESOLUTION | TIMER_OVERHEAD |
+-------------+-----------------+------------------+----------------+
| CYCLE       |      2389029850 |                1 |             72 |
| NANOSECOND  |      1000000000 |                1 |            112 |
| MICROSECOND |         1000000 |                1 |            136 |
| MILLISECOND |            1036 |                1 |            168 |
| TICK        |             105 |                1 |           2416 |
+-------------+-----------------+------------------+----------------+
```

If the values associated with a given timer name are `NULL`, that timer is not supported on your platform. The rows that do not contain `NULL` indicate which timers you can use in `setup_timers`.

The columns have these meanings:

- The `TIMER_NAME` column shows the names of the available timers. `CYCLE` refers to the timer that is based on the CPU (processor) cycle counter. The timers in `setup_timers` that you can use are those that do not have `NULL` in the other columns. If the values associated with a given timer name are `NULL`, that timer is not supported on your platform.
- `TIMER_FREQUENCY` indicates the number of timer units per second. For a cycle timer, the frequency is generally related to the CPU speed. The value shown was obtained on a system with a 2.4GHz processor. The other timers are based on fixed fractions of seconds. For `TICK`, the frequency may vary by platform (for example, some use 100 ticks/second, others 1000 ticks/second).
- `TIMER_RESOLUTION` indicates the number of timer units by which timer values increase at a time. If a timer has a resolution of 10, its value increases by 10 each time.
- `TIMER_OVERHEAD` is the minimal number of cycles of overhead to obtain one timing with the given timer. The overhead per event is twice the value displayed because the timer is invoked at the beginning and end of the event.

To see which timers are in effect or to change timers, access the `setup_timers` table:

```
mysql> SELECT * FROM performance_schema.setup_timers;
+-----------+-------------+
| NAME      | TIMER_NAME  |
+-----------+-------------+
| idle      | MICROSECOND |
| wait      | CYCLE       |
| stage     | NANOSECOND  |
| statement | NANOSECOND  |
+-----------+-------------+

mysql> UPDATE performance_schema.setup_timers
       SET TIMER_NAME = 'MICROSECOND'
       WHERE NAME = 'idle';

mysql> SELECT * FROM performance_schema.setup_timers;
+-----------+-------------+
| NAME      | TIMER_NAME  |
+-----------+-------------+
```

```
| idle      | MICROSECOND |
| wait      | CYCLE       |
| stage     | NANOSECOND  |
| statement | NANOSECOND  |
+-----------+-------------+
```

By default, the Performance Schema uses the best timer available for each instrument type, but you can select a different one.

To time wait events, the most important criterion is to reduce overhead, at the possible expense of the timer accuracy, so using the `CYCLE` timer is the best.

The time a statement (or stage) takes to execute is in general orders of magnitude larger than the time it takes to execute a single wait. To time statements, the most important criterion is to have an accurate measure, which is not affected by changes in processor frequency, so using a timer which is not based on cycles is the best. The default timer for statements is `NANOSECOND`. The extra “overhead” compared to the `CYCLE` timer is not significant, because the overhead caused by calling a timer twice (once when the statement starts, once when it ends) is orders of magnitude less compared to the CPU time used to execute the statement itself. Using the `CYCLE` timer has no benefit here, only drawbacks.

The precision offered by the cycle counter depends on processor speed. If the processor runs at 1 GHz (one billion cycles/second) or higher, the cycle counter delivers sub-nanosecond precision. Using the cycle counter is much cheaper than getting the actual time of day. For example, the standard `gettimeofday()` function can take hundreds of cycles, which is an unacceptable overhead for data gathering that may occur thousands or millions of times per second.

Cycle counters also have disadvantages:

- End users expect to see timings in wall-clock units, such as fractions of a second. Converting from cycles to fractions of seconds can be expensive. For this reason, the conversion is a quick and fairly rough multiplication operation.
- Processor cycle rate might change, such as when a laptop goes into power-saving mode or when a CPU slows down to reduce heat generation. If a processor's cycle rate fluctuates, conversion from cycles to real-time units is subject to error.
- Cycle counters might be unreliable or unavailable depending on the processor or the operating system. For example, on Pentiums, the instruction is `RDTSC` (an assembly-language rather than a C instruction) and it is theoretically possible for the operating system to prevent user-mode programs from using it.
- Some processor details related to out-of-order execution or multiprocessor synchronization might cause the counter to seem fast or slow by up to 1000 cycles.

MySQL works with cycle counters on x386 (Windows, macOS, Linux, Solaris, and other Unix flavors), PowerPC, and IA-64.

Performance Schema Timer Representation in Events

Rows in Performance Schema tables that store current events and historical events have three columns to represent timing information: `TIMER_START` and `TIMER_END` indicate when an event started and finished, and `TIMER_WAIT` indicates event duration.

The `setup_instruments` table has an `ENABLED` column to indicate the instruments for which to collect events. The table also has a `TIMED` column to indicate which instruments are timed. If an instrument is not enabled, it produces no events. If an enabled instrument is not timed, events produced by the instrument have `NULL` for the `TIMER_START`, `TIMER_END`, and `TIMER_WAIT` timer values. This in turn causes those values to be ignored when calculating aggregate time values in summary tables (sum, minimum, maximum, and average).

Internally, times within events are stored in units given by the timer in effect when event timing begins. For display when events are retrieved from Performance Schema tables, times are shown in

picoseconds (trillionths of a second) to normalize them to a standard unit, regardless of which timer is selected.

Modifications to the `setup_timers` table affect monitoring immediately. Events already in progress may use the original timer for the begin time and the new timer for the end time. To avoid unpredictable results after you make timer changes, use `TRUNCATE TABLE` to reset Performance Schema statistics.

The timer baseline (“time zero”) occurs at Performance Schema initialization during server startup. `TIMER_START` and `TIMER_END` values in events represent picoseconds since the baseline. `TIMER_WAIT` values are durations in picoseconds.

Picosecond values in events are approximate. Their accuracy is subject to the usual forms of error associated with conversion from one unit to another. If the `CYCLE` timer is used and the processor rate varies, there might be drift. For these reasons, it is not reasonable to look at the `TIMER_START` value for an event as an accurate measure of time elapsed since server startup. On the other hand, it is reasonable to use `TIMER_START` or `TIMER_WAIT` values in `ORDER BY` clauses to order events by start time or duration.

The choice of picoseconds in events rather than a value such as microseconds has a performance basis. One implementation goal was to show results in a uniform time unit, regardless of the timer. In an ideal world this time unit would look like a wall-clock unit and be reasonably precise; in other words, microseconds. But to convert cycles or nanoseconds to microseconds, it would be necessary to perform a division for every instrumentation. Division is expensive on many platforms. Multiplication is not expensive, so that is what is used. Therefore, the time unit is an integer multiple of the highest possible `TIMER_FREQUENCY` value, using a multiplier large enough to ensure that there is no major precision loss. The result is that the time unit is “picoseconds.” This precision is spurious, but the decision enables overhead to be minimized.

Before MySQL 5.6.26, while a wait, stage, or statement event is executing, the respective current-event tables display the event with `TIMER_START` populated, but with `TIMER_END` and `TIMER_WAIT` set to `NULL`:

```
events_waits_current
events_stages_current
events_statements_current
```

As of MySQL 5.6.26, current-event timing provides more information. To make it possible to determine how long a not-yet-completed event has been running, the timer columns are set as follows:

- `TIMER_START` is populated.
- `TIMER_END` is populated with the current timer value.
- `TIMER_WAIT` is populated with the time elapsed so far (`TIMER_END` − `TIMER_START`).

Events that have not yet completed have an `END_EVENT_ID` value of `NULL`. To assess time elapsed so far for an event, use the `TIMER_WAIT` column. Therefore, to identify events that have not yet completed and have taken longer than *`N`* picoseconds thus far, monitoring applications can use this expression in queries:

```
WHERE END_EVENT_ID IS NULL AND TIMER_WAIT > N
```

Event identification as just described assumes that the corresponding instruments have `ENABLED` and `TIMED` set to `YES` and that the relevant consumers are enabled.

22.4.2 Performance Schema Event Filtering

Events are processed in a producer/consumer fashion:

- Instrumented code is the source for events and produces events to be collected. The `setup_instruments` table lists the instruments for which events can be collected, whether they are enabled, and (for enabled instruments) whether to collect timing information:

```
mysql> SELECT * FROM performance_schema.setup_instruments;
+-----------------------------------------------------+---------+-------+
| NAME                                                | ENABLED | TIMED |
+-----------------------------------------------------+---------+-------+
...
| wait/synch/mutex/sql/LOCK_global_read_lock          | YES     | YES   |
| wait/synch/mutex/sql/LOCK_global_system_variables | YES     | YES   |
| wait/synch/mutex/sql/LOCK_lock_db                   | YES     | YES   |
| wait/synch/mutex/sql/LOCK_manager                   | YES     | YES   |
...
```

The `setup_instruments` table provides the most basic form of control over event production. To further refine event production based on the type of object or thread being monitored, other tables may be used as described in Section 22.4.3, “Event Pre-Filtering”.

- Performance Schema tables are the destinations for events and consume events. The `setup_consumers` table lists the types of consumers to which event information can be sent and whether they are enabled:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+--------------------------------+---------+
| NAME                           | ENABLED |
+--------------------------------+---------+
| events_stages_current          | NO      |
| events_stages_history          | NO      |
| events_stages_history_long     | NO      |
| events_statements_current      | YES     |
| events_statements_history      | NO      |
| events_statements_history_long | NO      |
| events_waits_current           | NO      |
| events_waits_history           | NO      |
| events_waits_history_long      | NO      |
| global_instrumentation         | YES     |
| thread_instrumentation         | YES     |
| statements_digest              | YES     |
+--------------------------------+---------+
```

Filtering can be done at different stages of performance monitoring:

- **Pre-filtering.** This is done by modifying Performance Schema configuration so that only certain types of events are collected from producers, and collected events update only certain consumers. To do this, enable or disable instruments or consumers. Pre-filtering is done by the Performance Schema and has a global effect that applies to all users.

  Reasons to use pre-filtering:

  - To reduce overhead. Performance Schema overhead should be minimal even with all instruments enabled, but perhaps you want to reduce it further. Or you do not care about timing events and want to disable the timing code to eliminate timing overhead.
  - To avoid filling the current-events or history tables with events in which you have no interest. Pre-filtering leaves more “room” in these tables for instances of rows for enabled instrument types. If you enable only file instruments with pre-filtering, no rows are collected for nonfile instruments. With post-filtering, nonfile events are collected, leaving fewer rows for file events.
  - To avoid maintaining some kinds of event tables. If you disable a consumer, the server does not spend time maintaining destinations for that consumer. For example, if you do not care about event histories, you can disable the history table consumers to improve performance.

- **Post-filtering.** This involves the use of `WHERE` clauses in queries that select information from Performance Schema tables, to specify which of the available events you want to see. Post-filtering is performed on a per-user basis because individual users select which of the available events are of interest.

  Reasons to use post-filtering:

- To avoid making decisions for individual users about which event information is of interest.
- To use the Performance Schema to investigate a performance issue when the restrictions to impose using pre-filtering are not known in advance.

The following sections provide more detail about pre-filtering and provide guidelines for naming instruments or consumers in filtering operations. For information about writing queries to retrieve information (post-filtering), see Section 22.5, “Performance Schema Queries”.

22.4.3 Event Pre-Filtering

Pre-filtering is done by the Performance Schema and has a global effect that applies to all users. Pre-filtering can be applied to either the producer or consumer stage of event processing:

- To configure pre-filtering at the producer stage, several tables can be used:
  - `setup_instruments` indicates which instruments are available. An instrument disabled in this table produces no events regardless of the contents of the other production-related setup tables. An instrument enabled in this table is permitted to produce events, subject to the contents of the other tables.
  - `setup_objects` controls whether the Performance Schema monitors particular table objects.
  - `threads` indicates whether monitoring is enabled for each server thread.
  - `setup_actors` determines the initial monitoring state for new foreground threads.
- To configure pre-filtering at the consumer stage, modify the `setup_consumers` table. This determines the destinations to which events are sent. `setup_consumers` also implicitly affects event production. If a given event is not sent to any destination (is not consumed), the Performance Schema does not produce it.

Modifications to any of these tables affect monitoring immediately, with some exceptions:

- Modifications to some instruments in the `setup_instruments` table are effective only at server startup; changing them at runtime has no effect. This affects primarily mutexes, conditions, and rwlocks in the server, although there may be other instruments for which this is true.
- Modifications to the `setup_actors` table affect only foreground threads created subsequent to the modification, not existing threads.

When you change the monitoring configuration, the Performance Schema does not flush the history tables. Events already collected remain in the current-events and history tables until displaced by newer events. If you disable instruments, you might need to wait a while before events for them are displaced by newer events of interest. Alternatively, use `TRUNCATE TABLE` to empty the history tables.

After making instrumentation changes, you might want to truncate the summary tables. Generally, the effect is to reset the summary columns to 0 or `NULL`, not to remove rows. This enables you to clear collected values and restart aggregation. That might be useful, for example, after you have made a runtime configuration change. Exceptions to this truncation behavior are noted in individual summary table sections.

The following sections describe how to use specific tables to control Performance Schema pre-filtering.

22.4.4 Pre-Filtering by Instrument

The `setup_instruments` table lists the available instruments:

```
mysql> SELECT * FROM performance_schema.setup_instruments;
+-----------------------------------------------------+---------+-------+
| NAME                                                | ENABLED | TIMED |
```

```
+-----------------------------------------------------+---------+-------+
...
| stage/sql/end                                       | NO      | NO    |
| stage/sql/executing                                 | NO      | NO    |
| stage/sql/init                                      | NO      | NO    |
| stage/sql/insert                                    | NO      | NO    |
...
| statement/sql/load                                  | YES     | YES   |
| statement/sql/grant                                 | YES     | YES   |
| statement/sql/check                                 | YES     | YES   |
| statement/sql/flush                                 | YES     | YES   |
...
| wait/synch/mutex/sql/LOCK_global_read_lock          | YES     | YES   |
| wait/synch/mutex/sql/LOCK_global_system_variables | YES     | YES   |
| wait/synch/mutex/sql/LOCK_lock_db                   | YES     | YES   |
| wait/synch/mutex/sql/LOCK_manager                   | YES     | YES   |
...
| wait/synch/rwlock/sql/LOCK_grant                    | YES     | YES   |
| wait/synch/rwlock/sql/LOGGER::LOCK_logger           | YES     | YES   |
| wait/synch/rwlock/sql/LOCK_sys_init_connect         | YES     | YES   |
| wait/synch/rwlock/sql/LOCK_sys_init_slave           | YES     | YES   |
...
| wait/io/file/sql/binlog                             | YES     | YES   |
| wait/io/file/sql/binlog_index                       | YES     | YES   |
| wait/io/file/sql/casetest                           | YES     | YES   |
| wait/io/file/sql/dbopt                              | YES     | YES   |
...
```

To control whether an instrument is enabled, set its `ENABLED` column to `YES` or `NO`. To configure whether to collect timing information for an enabled instrument, set its `TIMED` value to `YES` or `NO`. Setting the `TIMED` column affects Performance Schema table contents as described in Section 22.4.1, “Performance Schema Event Timing”.

Modifications to most `setup_instruments` rows affect monitoring immediately. For some instruments, modifications are effective only at server startup; changing them at runtime has no effect. This affects primarily mutexes, conditions, and rwlocks in the server, although there may be other instruments for which this is true.

The `setup_instruments` table provides the most basic form of control over event production. To further refine event production based on the type of object or thread being monitored, other tables may be used as described in Section 22.4.3, “Event Pre-Filtering”.

The following examples demonstrate possible operations on the `setup_instruments` table. These changes, like other pre-filtering operations, affect all users. Some of these queries use the `LIKE` operator and a pattern match instrument names. For additional information about specifying patterns to select instruments, see Section 22.4.9, “Naming Instruments or Consumers for Filtering Operations”.

- Disable all instruments:

  ```
  UPDATE performance_schema.setup_instruments
  SET ENABLED = 'NO';
  ```

  Now no events are collected.

- Disable all file instruments, adding them to the current set of disabled instruments:

  ```
  UPDATE performance_schema.setup_instruments
  SET ENABLED = 'NO'
  WHERE NAME LIKE 'wait/io/file/%';
  ```

- Disable only file instruments, enable all other instruments:

  ```
  UPDATE performance_schema.setup_instruments
  SET ENABLED = IF(NAME LIKE 'wait/io/file/%', 'NO', 'YES');
  ```

- Enable all but those instruments in the `mysys` library:

  ```
  UPDATE performance_schema.setup_instruments
  ```

```
SET ENABLED = CASE WHEN NAME LIKE '%/mysys/%' THEN 'YES' ELSE 'NO' END;
```

- Disable a specific instrument:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO'
WHERE NAME = 'wait/synch/mutex/mysys/TMPDIR_mutex';
```

- To toggle the state of an instrument, “flip” its `ENABLED` value:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = IF(ENABLED = 'YES', 'NO', 'YES')
WHERE NAME = 'wait/synch/mutex/mysys/TMPDIR_mutex';
```

- Disable timing for all events:

```
UPDATE performance_schema.setup_instruments
SET TIMED = 'NO';
```

22.4.5 Pre-Filtering by Object

The `setup_objects` table controls whether the Performance Schema monitors particular table objects. The initial `setup_objects` contents look like this:

```
mysql> SELECT * FROM performance_schema.setup_objects;
+-------------+--------------------+-------------+---------+-------+
| OBJECT_TYPE | OBJECT_SCHEMA      | OBJECT_NAME | ENABLED | TIMED |
+-------------+--------------------+-------------+---------+-------+
| TABLE       | mysql              | %           | NO      | NO    |
| TABLE       | performance_schema | %           | NO      | NO    |
| TABLE       | information_schema | %           | NO      | NO    |
| TABLE       | %                  | %           | YES     | YES   |
+-------------+--------------------+-------------+---------+-------+
```

Modifications to the `setup_objects` table affect object monitoring immediately.

The `OBJECT_TYPE` column indicates the type of object to which a row applies. `TABLE` filtering affects table I/O events (`wait/io/table/sql/handler` instrument) and table lock events (`wait/lock/table/sql/handler` instrument).

The `OBJECT_SCHEMA` and `OBJECT_NAME` columns should contain a literal schema or table name, or `'%'` to match any name.

The `ENABLED` column indicates whether matching objects are monitored, and `TIMED` indicates whether to collect timing information. Setting the `TIMED` column affects Performance Schema table contents as described in Section 22.4.1, “Performance Schema Event Timing”.

The effect of the default object configuration is to instrument all tables except those in the `mysql`, `INFORMATION_SCHEMA`, and `performance_schema` databases. (Tables in the `INFORMATION_SCHEMA` database are not instrumented regardless of the contents of `setup_objects`; the row for `information_schema.%` simply makes this default explicit.)

When the Performance Schema checks for a match in `setup_objects`, it tries to find more specific matches first. For rows that match a given `OBJECT_TYPE`, the Performance Schema checks rows in this order:

- Rows with `OBJECT_SCHEMA='literal'` and `OBJECT_NAME='literal'`.
- Rows with `OBJECT_SCHEMA='literal'` and `OBJECT_NAME='%'`.
- Rows with `OBJECT_SCHEMA='%'` and `OBJECT_NAME='%'`.

For example, with a table `db1.t1`, the Performance Schema looks in `TABLE` rows for a match for `'db1'` and `'t1'`, then for `'db1'` and `'%'`, then for `'%'` and `'%'`. The order in which matching occurs matters because different matching `setup_objects` rows can have different `ENABLED` and `TIMED` values.

For table-related events, the Performance Schema combines the contents of `setup_objects` with `setup_instruments` to determine whether to enable instruments and whether to time enabled instruments:

- For tables that match a row in `setup_objects`, table instruments produce events only if `ENABLED` is `YES` in both `setup_instruments` and `setup_objects`.
- The `TIMED` values in the two tables are combined, so that timing information is collected only when both values are `YES`.

Suppose that `setup_objects` contains the following `TABLE` rows that apply to `db1`, `db2`, and `db3`:

```
+-------------+---------------+-------------+---------+-------+
| OBJECT_TYPE | OBJECT_SCHEMA | OBJECT_NAME | ENABLED | TIMED |
+-------------+---------------+-------------+---------+-------+
| TABLE       | db1           | t1          | YES     | YES   |
| TABLE       | db1           | t2          | NO      | NO    |
| TABLE       | db2           | %           | YES     | YES   |
| TABLE       | db3           | %           | NO      | NO    |
| TABLE       | %             | %           | YES     | YES   |
+-------------+---------------+-------------+---------+-------+
```

If a table-related instrument in `setup_instruments` has an `ENABLED` value of `NO`, events for the object are not monitored. If the `ENABLED` value is `YES`, event monitoring occurs according to the `ENABLED` value in the relevant `setup_objects` row:

- `db1.t1` events are monitored
- `db1.t2` events are not monitored
- `db2.t3` events are monitored
- `db3.t4` events are not monitored
- `db4.t5` events are monitored

Similar logic applies for combining the `TIMED` columns from the `setup_instruments` and `setup_objects` tables to determine whether to collect event timing information.

If a persistent table and a temporary table have the same name, matching against `setup_objects` rows occurs the same way for both. It is not possible to enable monitoring for one table but not the other. However, each table is instrumented separately.

22.4.6 Pre-Filtering by Thread

The `threads` table contains a row for each server thread. Each row contains information about a thread and indicates whether monitoring is enabled for it. For the Performance Schema to monitor a thread, these things must be true:

- The `thread_instrumentation` consumer in the `setup_consumers` table must be `YES`.
- The `threads.INSTRUMENTED` column must be `YES`.
- Monitoring occurs only for those thread events produced from instruments that are enabled in the `setup_instruments` table.

The `INSTRUMENTED` column in the `threads` table indicates the monitoring state for each thread. For foreground threads (resulting from client connections), the initial `INSTRUMENTED` value is determined by whether the user account associated with the thread matches any row in the `setup_actors` table.

For background threads, there is no associated user. `INSTRUMENTED` is `YES` by default and `setup_actors` is not consulted.

The initial `setup_actors` contents look like this:

```
mysql> SELECT * FROM performance_schema.setup_actors;
+------+------+------+
| HOST | USER | ROLE |
+------+------+------+
| %    | %    | %    |
+------+------+------+
```

The `HOST` and `USER` columns should contain a literal host or user name, or `'%'` to match any name.

The Performance Schema uses the `HOST` and `USER` columns to match each new foreground thread. (`ROLE` is unused.) The `INSTRUMENTED` value for the thread becomes `YES` if any row matches, `NO` otherwise. This enables instrumenting to be applied selectively per host, user, or combination of host and user.

By default, monitoring is enabled for all new foreground threads because the `setup_actors` table initially contains a row with `'%'` for both `HOST` and `USER`. To perform more limited matching such as to enable monitoring only for some foreground threads, you must delete this row because it matches any connection.

Suppose that you modify `setup_actors` as follows:

```
TRUNCATE TABLE performance_schema.setup_actors;
```

Now `setup_actors` is empty and there are no rows that could match incoming connections. Consequently, the Performance Schema sets the `INSTRUMENTED` column to `NO` for all new foreground threads.

Suppose that you further modify `setup_actors`:

```
INSERT INTO performance_schema.setup_actors
(HOST,USER,ROLE) VALUES('localhost','joe','%');
INSERT INTO performance_schema.setup_actors
(HOST,USER,ROLE) VALUES('%','sam','%');
```

Now the Performance Schema determines how to set the `INSTRUMENTED` value for new connection threads as follows:

- If `joe` connects from the local host, the connection matches the first inserted row.
- If `joe` connects from any other host, there is no match.
- If `sam` connects from any host, the connection matches the second inserted row.
- For any other connection, there is no match.

Modifications to the `setup_actors` table affect only foreground threads created subsequent to the modification, not existing threads. To affect existing threads, modify the `INSTRUMENTED` column of `threads` table rows.

22.4.7 Pre-Filtering by Consumer

The `setup_consumers` table lists the available consumer types and which are enabled:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+--------------------------------+---------+
| NAME                           | ENABLED |
+--------------------------------+---------+
| events_stages_current          | NO      |
| events_stages_history          | NO      |
| events_stages_history_long     | NO      |
| events_statements_current      | YES     |
| events_statements_history      | NO      |
| events_statements_history_long | NO      |
| events_waits_current           | NO      |
| events_waits_history           | NO      |
```

```
| events_waits_history_long      | NO      |
| global_instrumentation         | YES     |
| thread_instrumentation         | YES     |
| statements_digest              | YES     |
+--------------------------------+---------+
```

Modify the `setup_consumers` table to affect pre-filtering at the consumer stage and determine the destinations to which events are sent. To enable or disable a consumer, set its `ENABLED` value to `YES` or `NO`.

Modifications to the `setup_consumers` table affect monitoring immediately.

If you disable a consumer, the server does not spend time maintaining destinations for that consumer. For example, if you do not care about historical event information, disable the history consumers:

```
UPDATE performance_schema.setup_consumers
SET ENABLED = 'NO'
WHERE NAME LIKE '%history%';
```

The consumer settings in the `setup_consumers` table form a hierarchy from higher levels to lower. The following principles apply:

- Destinations associated with a consumer receive no events unless the Performance Schema checks the consumer and the consumer is enabled.
- A consumer is checked only if all consumers it depends on (if any) are enabled.
- If a consumer is not checked, or is checked but is disabled, other consumers that depend on it are not checked.
- Dependent consumers may have their own dependent consumers.
- If an event would not be sent to any destination, the Performance Schema does not produce it.

The following lists describe the available consumer values. For discussion of several representative consumer configurations and their effect on instrumentation, see Section 22.4.8, “Example Consumer Configurations”.

- Global and Thread Consumers
- Wait Event Consumers
- Stage Event Consumers
- Statement Event Consumers
- Statement Digest Consumer

Global and Thread Consumers

- `global_instrumentation` is the highest level consumer. If `global_instrumentation` is `NO`, it disables global instrumentation. All other settings are lower level and are not checked; it does not matter what they are set to. No global or per thread information is maintained and no individual events are collected in the current-events or event-history tables. If `global_instrumentation` is `YES`, the Performance Schema maintains information for global states and also checks the `thread_instrumentation` consumer.
- `thread_instrumentation` is checked only if `global_instrumentation` is `YES`. Otherwise, if `thread_instrumentation` is `NO`, it disables thread-specific instrumentation and all lower-level settings are ignored. No information is maintained per thread and no individual events are collected in the current-events or event-history tables. If `thread_instrumentation` is `YES`, the Performance Schema maintains thread-specific information and also checks `events_xxx_current` consumers.

Wait Event Consumers

These consumers require both `global_instrumentation` and `thread_instrumentation` to be `YES` or they are not checked. If checked, they act as follows:

- `events_waits_current`, if `NO`, disables collection of individual wait events in the `events_waits_current` table. If `YES`, it enables wait event collection and the Performance Schema checks the `events_waits_history` and `events_waits_history_long` consumers.
- `events_waits_history` is not checked if `event_waits_current` is `NO`. Otherwise, an `events_waits_history` value of `NO` or `YES` disables or enables collection of wait events in the `events_waits_history` table.
- `events_waits_history_long` is not checked if `event_waits_current` is `NO`. Otherwise, an `events_waits_history_long` value of `NO` or `YES` disables or enables collection of wait events in the `events_waits_history_long` table.

Stage Event Consumers

These consumers require both `global_instrumentation` and `thread_instrumentation` to be `YES` or they are not checked. If checked, they act as follows:

- `events_stages_current`, if `NO`, disables collection of individual stage events in the `events_stages_current` table. If `YES`, it enables stage event collection and the Performance Schema checks the `events_stages_history` and `events_stages_history_long` consumers.
- `events_stages_history` is not checked if `event_stages_current` is `NO`. Otherwise, an `events_stages_history` value of `NO` or `YES` disables or enables collection of stage events in the `events_stages_history` table.
- `events_stages_history_long` is not checked if `event_stages_current` is `NO`. Otherwise, an `events_stages_history_long` value of `NO` or `YES` disables or enables collection of stage events in the `events_stages_history_long` table.

Statement Event Consumers

These consumers require both `global_instrumentation` and `thread_instrumentation` to be `YES` or they are not checked. If checked, they act as follows:

- `events_statements_current`, if `NO`, disables collection of individual statement events in the `events_statements_current` table. If `YES`, it enables statement event collection and the Performance Schema checks the `events_statements_history` and `events_statements_history_long` consumers.
- `events_statements_history` is not checked if `events_statements_current` is `NO`. Otherwise, an `events_statements_history` value of `NO` or `YES` disables or enables collection of statement events in the `events_statements_history` table.
- `events_statements_history_long` is not checked if `events_statements_current` is `NO`. Otherwise, an `events_statements_history_long` value of `NO` or `YES` disables or enables collection of statement events in the `events_statements_history_long` table.

Statement Digest Consumer

The `statements_digest` consumer requires `global_instrumentation` to be `YES` or it is not checked. There is no dependency on the statement event consumers, so you can obtain statistics per digest without having to collect statistics in `events_statements_current`, which is advantageous in terms of overhead. Conversely, you can get detailed statements in `events_statements_current` without digests (the `DIGEST` and `DIGEST_TEXT` columns are `NULL`).

For more information about statement digesting, see Section 22.10, “Performance Schema Statement Digests”.

22.4.8 Example Consumer Configurations

The consumer settings in the `setup_consumers` table form a hierarchy from higher levels to lower. The following discussion describes how consumers work, showing specific configurations and their effects as consumer settings are enabled progressively from high to low. The consumer values shown are representative. The general principles described here apply to other consumer values that may be available.

The configuration descriptions occur in order of increasing functionality and overhead. If you do not need the information provided by enabling lower-level settings, disable them and the Performance Schema executes less code on your behalf and you have less information to sift through.

The `setup_consumers` table contains the following hierarchy of values:

```
global_instrumentation
 thread_instrumentation
   events_waits_current
     events_waits_history
     events_waits_history_long
   events_stages_current
     events_stages_history
     events_stages_history_long
   events_statements_current
     events_statements_history
     events_statements_history_long
 statements_digest
```

Note

In the consumer hierarchy, the consumers for waits, stages, and statements are all at the same level. This differs from the event nesting hierarchy, for which wait events nest within stage events, which nest within statement events.

If a given consumer setting is `NO`, the Performance Schema disables the instrumentation associated with the consumer and ignores all lower-level settings. If a given setting is `YES`, the Performance Schema enables the instrumentation associated with it and checks the settings at the next lowest level. For a description of the rules for each consumer, see Section 22.4.7, “Pre-Filtering by Consumer”.

For example, if `global_instrumentation` is enabled, `thread_instrumentation` is checked. If `thread_instrumentation` is enabled, the `events_xxx_current` consumers are checked. If of these `events_waits_current` is enabled, `events_waits_history` and `events_waits_history_long` are checked.

Each of the following configuration descriptions indicates which setup elements the Performance Schema checks and which output tables it maintains (that is, for which tables it collects information).

- No Instrumentation
- Global Instrumentation Only
- Global and Thread Instrumentation Only
- Global, Thread, and Current-Event Instrumentation
- Global, Thread, Current-Event, and Event-History instrumentation

No Instrumentation

Server configuration state:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
```

```
+---------------------------+---------+
| NAME                      | ENABLED |
+---------------------------+---------+
| global_instrumentation    | NO      |
...
+---------------------------+---------+
```

In this configuration, nothing is instrumented.

Setup elements checked:

- Table `setup_consumers`, consumer `global_instrumentation`

Output tables maintained:

- None

Global Instrumentation Only

Server configuration state:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+---------------------------+---------+
| NAME                      | ENABLED |
+---------------------------+---------+
| global_instrumentation    | YES     |
| thread_instrumentation    | NO      |
...
+---------------------------+---------+
```

In this configuration, instrumentation is maintained only for global states. Per-thread instrumentation is disabled.

Additional setup elements checked, relative to the preceding configuration:

- Table `setup_consumers`, consumer `thread_instrumentation`
- Table `setup_instruments`
- Table `setup_objects`
- Table `setup_timers`

Additional output tables maintained, relative to the preceding configuration:

- `mutex_instances`
- `rwlock_instances`
- `cond_instances`
- `file_instances`
- `users`
- `hosts`
- `accounts`
- `socket_summary_by_event_name`
- `file_summary_by_instance`
- `file_summary_by_event_name`
- `objects_summary_global_by_type`

- table_lock_waits_summary_by_table
- table_io_waits_summary_by_index_usage
- table_io_waits_summary_by_table
- events_waits_summary_by_instance
- events_waits_summary_global_by_event_name
- events_stages_summary_global_by_event_name
- events_statements_summary_global_by_event_name

Global and Thread Instrumentation Only

Server configuration state:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+--------------------------------+---------+
| NAME                           | ENABLED |
+--------------------------------+---------+
| global_instrumentation         | YES     |
| thread_instrumentation         | YES     |
| events_waits_current           | NO      |
...
| events_stages_current          | NO      |
...
| events_statements_current      | YES     |
...
+--------------------------------+---------+
```

In this configuration, instrumentation is maintained globally and per thread. No individual events are collected in the current-events or event-history tables.

Additional setup elements checked, relative to the preceding configuration:

- Table setup_consumers, consumers events_*xxx*_current, where *xxx* is waits, stages, statements
- Table setup_actors
- Column threads.instrumented

Additional output tables maintained, relative to the preceding configuration:

- events_*xxx*_summary_by_*yyy*_by_event_name, where *xxx* is waits, stages, statements; and *yyy* is thread, user, host, account

Global, Thread, and Current-Event Instrumentation

Server configuration state:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+--------------------------------+---------+
| NAME                           | ENABLED |
+--------------------------------+---------+
| global_instrumentation         | YES     |
| thread_instrumentation         | YES     |
| events_waits_current           | YES     |
| events_waits_history           | NO      |
| events_waits_history_long      | NO      |
| events_stages_current          | YES     |
| events_stages_history          | NO      |
| events_stages_history_long     | NO      |
| events_statements_current      | YES     |
| events_statements_history      | NO      |
```

```
| events_statements_history_long | NO      |
...
+--------------------------------+---------+
```

In this configuration, instrumentation is maintained globally and per thread. Individual events are collected in the current-events table, but not in the event-history tables.

Additional setup elements checked, relative to the preceding configuration:

- Consumers `events_xxx_history`, where *xxx* is `waits`, `stages`, `statements`
- Consumers `events_xxx_history_long`, where *xxx* is `waits`, `stages`, `statements`

Additional output tables maintained, relative to the preceding configuration:

- `events_xxx_current`, where *xxx* is `waits`, `stages`, `statements`

Global, Thread, Current-Event, and Event-History instrumentation

The preceding configuration collects no event history because the `events_xxx_history` and `events_xxx_history_long` consumers are disabled. Those consumers can be enabled separately or together to collect event history per thread, globally, or both.

This configuration collects event history per thread, but not globally:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+--------------------------------+---------+
| NAME                           | ENABLED |
+--------------------------------+---------+
| global_instrumentation         | YES     |
| thread_instrumentation         | YES     |
| events_waits_current           | YES     |
| events_waits_history           | YES     |
| events_waits_history_long      | NO      |
| events_stages_current          | YES     |
| events_stages_history          | YES     |
| events_stages_history_long     | NO      |
| events_statements_current      | YES     |
| events_statements_history      | YES     |
| events_statements_history_long | NO      |
...
+--------------------------------+---------+
```

Event-history tables maintained for this configuration:

- `events_xxx_history`, where *xxx* is `waits`, `stages`, `statements`

This configuration collects event history globally, but not per thread:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+--------------------------------+---------+
| NAME                           | ENABLED |
+--------------------------------+---------+
| global_instrumentation         | YES     |
| thread_instrumentation         | YES     |
| events_waits_current           | YES     |
| events_waits_history           | NO      |
| events_waits_history_long      | YES     |
| events_stages_current          | YES     |
| events_stages_history          | NO      |
| events_stages_history_long     | YES     |
| events_statements_current      | YES     |
| events_statements_history      | NO      |
| events_statements_history_long | YES     |
...
+--------------------------------+---------+
```

Event-history tables maintained for this configuration:

- `events_xxx_history_long`, where *xxx* is `waits`, `stages`, `statements`

This configuration collects event history per thread and globally:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+--------------------------------+---------+
| NAME                           | ENABLED |
+--------------------------------+---------+
| global_instrumentation         | YES     |
| thread_instrumentation         | YES     |
| events_waits_current           | YES     |
| events_waits_history           | YES     |
| events_waits_history_long      | YES     |
| events_stages_current          | YES     |
| events_stages_history          | YES     |
| events_stages_history_long     | YES     |
| events_statements_current      | YES     |
| events_statements_history      | YES     |
| events_statements_history_long | YES     |
...
+--------------------------------+---------+
```

Event-history tables maintained for this configuration:

- `events_xxx_history`, where *xxx* is `waits`, `stages`, `statements`
- `events_xxx_history_long`, where *xxx* is `waits`, `stages`, `statements`

22.4.9 Naming Instruments or Consumers for Filtering Operations

Names given for filtering operations can be as specific or general as required. To indicate a single instrument or consumer, specify its name in full:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO'
WHERE NAME = 'wait/synch/mutex/myisammrg/MYRG_INFO::mutex';

UPDATE performance_schema.setup_consumers
SET ENABLED = 'NO'
WHERE NAME = 'events_waits_current';
```

To specify a group of instruments or consumers, use a pattern that matches the group members:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO'
WHERE NAME LIKE 'wait/synch/mutex/%';

UPDATE performance_schema.setup_consumers
SET ENABLED = 'NO'
WHERE NAME LIKE '%history%';
```

If you use a pattern, it should be chosen so that it matches all the items of interest and no others. For example, to select all file I/O instruments, it is better to use a pattern that includes the entire instrument name prefix:

```
... WHERE NAME LIKE 'wait/io/file/%';
```

The pattern `'%/file/%'` matches other instruments that have an element of `'/file/'` anywhere in the name. Even less suitable is the pattern `'%file%'` because it matches instruments with `'file'` anywhere in the name, such as `wait/synch/mutex/sql/LOCK_des_key_file`.

To check which instrument or consumer names a pattern matches, perform a simple test:

```
SELECT NAME FROM performance_schema.setup_instruments
WHERE NAME LIKE 'pattern';

SELECT NAME FROM performance_schema.setup_consumers
WHERE NAME LIKE 'pattern';
```

For information about the types of names that are supported, see Section 22.6, “Performance Schema Instrument Naming Conventions”.

22.4.10 Determining What Is Instrumented

It is always possible to determine what instruments the Performance Schema includes by checking the `setup_instruments` table. For example, to see what file-related events are instrumented for the `InnoDB` storage engine, use this query:

```
mysql> SELECT * FROM performance_schema.setup_instruments
       WHERE NAME LIKE 'wait/io/file/innodb/%';
+-------------------------------------+---------+-------+
| NAME                                | ENABLED | TIMED |
+-------------------------------------+---------+-------+
| wait/io/file/innodb/innodb_data_file | YES     | YES   |
| wait/io/file/innodb/innodb_log_file  | YES     | YES   |
| wait/io/file/innodb/innodb_temp_file | YES     | YES   |
+-------------------------------------+---------+-------+
```

An exhaustive description of precisely what is instrumented is not given in this documentation, for several reasons:

- What is instrumented is the server code. Changes to this code occur often, which also affects the set of instruments.
- It is not practical to list all the instruments because there are hundreds of them.
- As described earlier, it is possible to find out by querying the `setup_instruments` table. This information is always up to date for your version of MySQL, also includes instrumentation for instrumented plugins you might have installed that are not part of the core server, and can be used by automated tools.

22.5 Performance Schema Queries

Pre-filtering limits which event information is collected and is independent of any particular user. By contrast, post-filtering is performed by individual users through the use of queries with appropriate `WHERE` clauses that restrict what event information to select from the events available after pre-filtering has been applied.

In Section 22.4.3, “Event Pre-Filtering”, an example showed how to pre-filter for file instruments. If the event tables contain both file and nonfile information, post-filtering is another way to see information only for file events. Add a `WHERE` clause to queries to restrict event selection appropriately:

```
mysql> SELECT THREAD_ID, NUMBER_OF_BYTES
       FROM performance_schema.events_waits_history
       WHERE EVENT_NAME LIKE 'wait/io/file/%'
       AND NUMBER_OF_BYTES IS NOT NULL;
+-----------+-----------------+
| THREAD_ID | NUMBER_OF_BYTES |
+-----------+-----------------+
|        11 |              66 |
|        11 |              47 |
|        11 |             139 |
|         5 |              24 |
|         5 |             834 |
+-----------+-----------------+
```

22.6 Performance Schema Instrument Naming Conventions

An instrument name consists of a sequence of elements separated by `'/'` characters. Example names:

```
wait/io/file/myisam/log
wait/io/file/mysys/charset
```

```
wait/lock/table/sql/handler
wait/synch/cond/mysys/COND_alarm
wait/synch/cond/sql/BINLOG::update_cond
wait/synch/mutex/mysys/BITMAP_mutex
wait/synch/mutex/sql/LOCK_delete
wait/synch/rwlock/sql/Query_cache_query::lock
stage/sql/closing tables
stage/sql/Sorting result
statement/com/Execute
statement/com/Query
statement/sql/create_table
statement/sql/lock_tables
```

The instrument name space has a tree-like structure. The elements of an instrument name from left to right provide a progression from more general to more specific. The number of elements a name has depends on the type of instrument.

The interpretation of a given element in a name depends on the elements to the left of it. For example, `myisam` appears in both of the following names, but `myisam` in the first name is related to file I/O, whereas in the second it is related to a synchronization instrument:

```
wait/io/file/myisam/log
wait/synch/cond/myisam/MI_SORT_INFO::cond
```

Instrument names consist of a prefix with a structure defined by the Performance Schema implementation and a suffix defined by the developer implementing the instrument code. The top-level element of an instrument prefix indicates the type of instrument. This element also determines which event timer in the `setup_timers` table applies to the instrument. For the prefix part of instrument names, the top level indicates the type of instrument.

The suffix part of instrument names comes from the code for the instruments themselves. Suffixes may include levels such as these:

- A name for the major element (a server module such as `myisam`, `innodb`, `mysys`, or `sql`) or a plugin name.
- The name of a variable in the code, in the form *`XXX`* (a global variable) or *`CCC`*`::`*`MMM`* (a member *`MMM`* in class *`CCC`*). Examples: `COND_thread_cache`, `THR_LOCK_myisam`, `BINLOG::LOCK_index`.

- Top-Level Instrument Elements
- Idle Instrument Elements
- Stage Instrument Elements
- Statement Instrument Elements
- Wait Instrument Elements

Top-Level Instrument Elements

- `idle`: An instrumented idle event. This instrument has no further elements.
- `stage`: An instrumented stage event.
- `statement`: An instrumented statement event.
- `wait`: An instrumented wait event.

Idle Instrument Elements

The `idle` instrument is used for idle events, which The Performance Schema generates as discussed in the description of the `socket_instances.STATE` column in Section 22.12.3.5, “The socket_instances Table”.

Stage Instrument Elements

Stage instruments have names of the form `stage/code_area/stage_name`, where *`code_area`* is a value such as `sql` or `myisam`, and *`stage_name`* indicates the stage of statement processing, such as `Sorting result` or `Sending data`. Stages correspond to the thread states displayed by `SHOW PROCESSLIST` or that are visible in the `INFORMATION_SCHEMA.PROCESSLIST` table.

Statement Instrument Elements

- `statement/abstract/*`: An abstract instrument for statement operations. Abstract instruments are used during the early stages of statement classification before the exact statement type is known, then changed to a more specific statement instrument when the type is known. For a description of this process, see Section 22.12.6, “Performance Schema Statement Event Tables”.
- `statement/com`: An instrumented command operation. These have names corresponding to `COM_`*`xxx`* operations (see the `mysql_com.h` header file and `sql/sql_parse.cc`. For example, the `statement/com/Connect` and `statement/com/Init DB` instruments correspond to the `COM_CONNECT` and `COM_INIT_DB` commands.
- `statement/sql`: An instrumented SQL statement operation. For example, the `statement/sql/create_db` and `statement/sql/select` instruments are used for `CREATE DATABASE` and `SELECT` statements.

Wait Instrument Elements

- `wait/io`

  An instrumented I/O operation.

  - `wait/io/file`

    An instrumented file I/O operation. For files, the wait is the time waiting for the file operation to complete (for example, a call to `fwrite()`). Due to caching, the physical file I/O on the disk might not happen within this call.

  - `wait/io/socket`

    An instrumented socket operation. Socket instruments have names of the form `wait/io/socket/sql/`*`socket_type`*. The server has a listening socket for each network protocol that it supports. The instruments associated with listening sockets for TCP/IP or Unix socket file connections have a *`socket_type`* value of `server_tcpip_socket` or `server_unix_socket`, respectively. When a listening socket detects a connection, the server transfers the connection to a new socket managed by a separate thread. The instrument for the new connection thread has a *`socket_type`* value of `client_connection`.

  - `wait/io/table`

    An instrumented table I/O operation. These include row-level accesses to persistent base tables or temporary tables. Operations that affect rows are fetch, insert, update, and delete. For a view, waits are associated with base tables referenced by the view.

    Unlike most waits, a table I/O wait can include other waits. For example, table I/O might include file I/O or memory operations. Thus, `events_waits_current` for a table I/O wait usually has two rows. For more information, see Section 22.8, “Performance Schema Atom and Molecule Events”.

    Some row operations might cause multiple table I/O waits. For example, an insert might activate a trigger that causes an update.

- `wait/lock`

  An instrumented lock operation.

  - `wait/lock/table`

    An instrumented table lock operation.

- `wait/synch`

  An instrumented synchronization object. For synchronization objects, the `TIMER_WAIT` time includes the amount of time blocked while attempting to acquire a lock on the object, if any.

  - `wait/synch/cond`

    A condition is used by one thread to signal to other threads that something they were waiting for has happened. If a single thread was waiting for a condition, it can wake up and proceed with its execution. If several threads were waiting, they can all wake up and compete for the resource for which they were waiting.

  - `wait/synch/mutex`

    A mutual exclusion object used to permit access to a resource (such as a section of executable code) while preventing other threads from accessing the resource.

  - `wait/synch/rwlock`

    A read/write lock object used to lock a specific variable for access while preventing its use by other threads. A shared read lock can be acquired simultaneously by multiple threads. An exclusive write lock can be acquired by only one thread at a time.

22.7 Performance Schema Status Monitoring

There are several status variables associated with the Performance Schema:

```
mysql> SHOW STATUS LIKE 'perf%';
+-----------------------------------------------+-------+
| Variable_name                                 | Value |
+-----------------------------------------------+-------+
| Performance_schema_accounts_lost              | 0     |
| Performance_schema_cond_classes_lost          | 0     |
| Performance_schema_cond_instances_lost        | 0     |
| Performance_schema_digest_lost                | 0     |
| Performance_schema_file_classes_lost          | 0     |
| Performance_schema_file_handles_lost          | 0     |
| Performance_schema_file_instances_lost        | 0     |
| Performance_schema_hosts_lost                 | 0     |
| Performance_schema_locker_lost                | 0     |
| Performance_schema_mutex_classes_lost         | 0     |
| Performance_schema_mutex_instances_lost       | 0     |
| Performance_schema_rwlock_classes_lost        | 0     |
| Performance_schema_rwlock_instances_lost      | 0     |
| Performance_schema_session_connect_attrs_lost | 0     |
| Performance_schema_socket_classes_lost        | 0     |
| Performance_schema_socket_instances_lost      | 0     |
| Performance_schema_stage_classes_lost         | 0     |
| Performance_schema_statement_classes_lost     | 0     |
| Performance_schema_table_handles_lost         | 0     |
| Performance_schema_table_instances_lost       | 0     |
| Performance_schema_thread_classes_lost        | 0     |
| Performance_schema_thread_instances_lost      | 0     |
| Performance_schema_users_lost                 | 0     |
+-----------------------------------------------+-------+
```

The Performance Schema status variables provide information about instrumentation that could not be loaded or created due to memory constraints. Names for these variables have several forms:

- `Performance_schema_`*`xxx`*`_classes_lost` indicates how many instruments of type *`xxx`* could not be loaded.

- `Performance_schema_xxx_instances_lost` indicates how many instances of object type *`xxx`* could not be created.
- `Performance_schema_xxx_handles_lost` indicates how many instances of object type *`xxx`* could not be opened.
- `Performance_schema_locker_lost` indicates how many events are “lost” or not recorded.

For example, if a mutex is instrumented in the server source but the server cannot allocate memory for the instrumentation at runtime, it increments `Performance_schema_mutex_classes_lost`. The mutex still functions as a synchronization object (that is, the server continues to function normally), but performance data for it is not collected. If the instrument can be allocated, it can be used for initializing instrumented mutex instances. For a singleton mutex such as a global mutex, there is only one instance. Other mutexes have an instance per connection, or per page in various caches and data buffers, so the number of instances varies over time. Increasing the maximum number of connections or the maximum size of some buffers increases the maximum number of instances that might be allocated at once. If the server cannot create a given instrumented mutex instance, it increments `Performance_schema_mutex_instances_lost`.

Suppose that the following conditions hold:

- The server was started with the `--performance_schema_max_mutex_classes=200` option and thus has room for 200 mutex instruments.
- 150 mutex instruments have been loaded already.
- The plugin named `plugin_a` contains 40 mutex instruments.
- The plugin named `plugin_b` contains 20 mutex instruments.

The server allocates mutex instruments for the plugins depending on how many they need and how many are available, as illustrated by the following sequence of statements:

```
INSTALL PLUGIN plugin_a
```

The server now has 150+40 = 190 mutex instruments.

```
UNINSTALL PLUGIN plugin_a;
```

The server still has 190 instruments. All the historical data generated by the plugin code is still available, but new events for the instruments are not collected.

```
INSTALL PLUGIN plugin_a;
```

The server detects that the 40 instruments are already defined, so no new instruments are created, and previously assigned internal memory buffers are reused. The server still has 190 instruments.

```
INSTALL PLUGIN plugin_b;
```

The server has room for 200-190 = 10 instruments (in this case, mutex classes), and sees that the plugin contains 20 new instruments. 10 instruments are loaded, and 10 are discarded or “lost.” The `Performance_schema_mutex_classes_lost` indicates the number of instruments (mutex classes) lost:

```
mysql> SHOW STATUS LIKE "perf%mutex_classes_lost";
+---------------------------------------+-------+
| Variable_name                         | Value |
+---------------------------------------+-------+
| Performance_schema_mutex_classes_lost | 10    |
+---------------------------------------+-------+
1 row in set (0.10 sec)
```

The instrumentation still works and collects (partial) data for `plugin_b`.

When the server cannot create a mutex instrument, these results occur:

- No row for the instrument is inserted into the `setup_instruments` table.
- `Performance_schema_mutex_classes_lost` increases by 1.
- `Performance_schema_mutex_instances_lost` does not change. (When the mutex instrument is not created, it cannot be used to create instrumented mutex instances later.)

The pattern just described applies to all types of instruments, not just mutexes.

A value of `Performance_schema_mutex_classes_lost` greater than 0 can happen in two cases:

- To save a few bytes of memory, you start the server with `--performance_schema_max_mutex_classes=N`, where *N* is less than the default value. The default value is chosen to be sufficient to load all the plugins provided in the MySQL distribution, but this can be reduced if some plugins are never loaded. For example, you might choose not to load some of the storage engines in the distribution.
- You load a third-party plugin that is instrumented for the Performance Schema but do not allow for the plugin's instrumentation memory requirements when you start the server. Because it comes from a third party, the instrument memory consumption of this engine is not accounted for in the default value chosen for `performance_schema_max_mutex_classes`.

  If the server has insufficient resources for the plugin's instruments and you do not explicitly allocate more using `--performance_schema_max_mutex_classes=N`, loading the plugin leads to starvation of instruments.

If the value chosen for `performance_schema_max_mutex_classes` is too small, no error is reported in the error log and there is no failure at runtime. However, the content of the tables in the `performance_schema` database misses events. The `Performance_schema_mutex_classes_lost` status variable is the only visible sign to indicate that some events were dropped internally due to failure to create instruments.

If an instrument is not lost, it is known to the Performance Schema, and is used when instrumenting instances. For example, `wait/synch/mutex/sql/LOCK_delete` is the name of a mutex instrument in the `setup_instruments` table. This single instrument is used when creating a mutex in the code (in `THD::LOCK_delete`) however many instances of the mutex are needed as the server runs. In this case, `LOCK_delete` is a mutex that is per connection (`THD`), so if a server has 1000 connections, there are 1000 threads, and 1000 instrumented `LOCK_delete` mutex instances (`THD::LOCK_delete`).

If the server does not have room for all these 1000 instrumented mutexes (instances), some mutexes are created with instrumentation, and some are created without instrumentation. If the server can create only 800 instances, 200 instances are lost. The server continues to run, but increments `Performance_schema_mutex_instances_lost` by 200 to indicate that instances could not be created.

A value of `Performance_schema_mutex_instances_lost` greater than 0 can happen when the code initializes more mutexes at runtime than were allocated for `--performance_schema_max_mutex_instances=N`.

The bottom line is that if `SHOW STATUS LIKE 'perf%'` says that nothing was lost (all values are zero), the Performance Schema data is accurate and can be relied upon. If something was lost, the data is incomplete, and the Performance Schema could not record everything given the insufficient amount of memory it was given to use. In this case, the specific `Performance_schema_xxx_lost` variable indicates the problem area.

It might be appropriate in some cases to cause deliberate instrument starvation. For example, if you do not care about performance data for file I/O, you can start the server with all Performance Schema

parameters related to file I/O set to 0. No memory is allocated for file-related classes, instances, or handles, and all file events are lost.

Use `SHOW ENGINE PERFORMANCE_SCHEMA STATUS` to inspect the internal operation of the Performance Schema code:

```
mysql> SHOW ENGINE PERFORMANCE_SCHEMA STATUS\G
...
*************************** 3. row ***************************
  Type: performance_schema
  Name: events_waits_history.row_size
Status: 76
*************************** 4. row ***************************
  Type: performance_schema
  Name: events_waits_history.row_count
Status: 10000
*************************** 5. row ***************************
  Type: performance_schema
  Name: events_waits_history.memory
Status: 760000
...
*************************** 57. row ***************************
  Type: performance_schema
  Name: performance_schema.memory
Status: 26459600
...
```

This statement is intended to help the DBA understand the effects that different Performance Schema options have on memory requirements. For a description of the field meanings, see Section 13.7.5.16, “SHOW ENGINE Statement”.

22.8 Performance Schema Atom and Molecule Events

For a table I/O event, there are usually two rows in `events_waits_current`, not one. For example, a row fetch might result in rows like this:

```
Row# EVENT_NAME                 TIMER_START TIMER_END
---- ----------                 ----------- ---------
   1 wait/io/file/myisam/dfile        10001 10002
   2 wait/io/table/sql/handler        10000 NULL
```

The row fetch causes a file read. In the example, the table I/O fetch event started before the file I/O event but has not finished (its `TIMER_END` value is `NULL`). The file I/O event is “nested” within the table I/O event.

This occurs because, unlike other “atomic” wait events such as for mutexes or file I/O, table I/O events are “molecular” and include (overlap with) other events. In `events_waits_current`, the table I/O event usually has two rows:

- One row for the most recent table I/O wait event
- One row for the most recent wait event of any kind

Usually, but not always, the “of any kind” wait event differs from the table I/O event. As each subsidiary event completes, it disappears from `events_waits_current`. At this point, and until the next subsidiary event begins, the table I/O wait is also the most recent wait of any kind.

22.9 Performance Schema Tables for Current and Historical Events

For wait, stage, and statement events, the Performance Schema can monitor and store current events. In addition, when events end, the Performance Schema can store them in history tables. For each event type, the Performance Schema uses three tables for storing current and historical events. The

tables have names of the following forms, where *xxx* indicates the event type (`waits`, `stages`, `statements`):

- `events_xxx_current`: The “current events” table stores the current monitored event for each thread (one row per thread).
- `events_xxx_history`: The “recent history” table stores the most recent events that have ended per thread (up to a maximum number of rows per thread).
- `events_xxx_history_long`: The “long history” table stores the most recent events that have ended globally (across all threads, up to a maximum number of rows per table).

The `_current` table for each event type contains one row per thread, so there is no system variable for configuring its maximum size. The Performance Schema autosizes the history tables, or the sizes can be configured explicitly at server startup using table-specific system variables, as indicated in the sections that describe the individual history tables. Typical autosized values are 10 rows per thread for `_history` tables, and 10,000 rows total for `_history_long` tables.

For each event type, the `_current`, `_history`, and `_history_long` tables have the same columns.

The `_current` tables show what is currently happening within the server. When a current event ends, it is removed from its `_current` table.

The `_history` and `_history_long` tables show what has happened in the recent past. When the history tables become full, old events are discarded as new events are added. Rows expire from the `_history` and `_history_long` tables in different ways because the tables serve different purposes:

- `_history` is meant to investigate individual threads, independently of the global server load.
- `_history_long` is meant to investigate the server globally, not each thread.

The difference between the two types of history tables relates to the data retention policy. Both tables contains the same data when an event is first seen. However, data within each table expires differently over time, so that data might be preserved for a longer or shorter time in each table:

- For `_history`, when the table contains the maximum number of rows for a given thread, the oldest thread row is discarded when a new row for that thread is added.
- For `_history_long`, when the table becomes full, the oldest row is discarded when a new row is added, regardless of which thread generated either row.

When a thread ends, all its rows are discarded from the `_history` table but not from the `_history_long` table.

The following example illustrates the differences in how events are added to and discarded from the two types of history tables. The principles apply equally to all event types. The example is based on these assumptions:

- The Performance Schema is configured to retain 10 rows per thread in the `_history` table and 10,000 rows total in the `_history_long` table.
- Thread A generates 1 event per second.

  Thread B generates 100 events per second.
- No other threads are running.

After 5 seconds of execution:

- A and B have generated 5 and 500 events, respectively.
- `_history` contains 5 rows for A and 10 rows for B. Because storage per thread is limited to 10 rows, no rows have been discarded for A, whereas 490 rows have been discarded for B.

- `_history_long` contains 5 rows for A and 500 rows for B. Because the table has a maximum size of 10,000 rows, no rows have been discarded for either thread.

After 5 minutes (300 seconds) of execution:

- A and B have generated 300 and 30,000 events, respectively.
- `_history` contains 10 rows for A and 10 rows for B. Because storage per thread is limited to 10 rows, 290 rows have been discarded for A, whereas 29,990 rows have been discarded for B. Rows for A include data up to 10 seconds old, whereas rows for B include data up to only .1 seconds old.
- `_history_long` contains 10,000 rows. Because A and B together generate 101 events per second, the table contains data up to approximately 10,000/101 = 99 seconds old, with a mix of rows approximately 100 to 1 from B as opposed to A.

22.10 Performance Schema Statement Digests

The MySQL server is capable of maintaining statement digest information. The digesting process converts each SQL statement to normalized form (the statement digest) and computes an MD5 hash value (the digest hash value) from the normalized result. Normalization permits statements that are similar to be grouped and summarized to expose information about the types of statements the server is executing and how often they occur. This section describes how statement digesting occurs and how it can be useful.

As of 5.6.24, digesting occurs in the parser regardless of whether the Performance Schema is available, so that other server components such as MySQL Enterprise Firewall have access to statement digests. Before MySQL 5.6.24, statement digesting was a function of the Performance Schema.

- Statement Digest General Concepts
- Statement Digests in the Performance Schema
- Statement Digest Memory Use

Statement Digest General Concepts

When the parser receives an SQL statement, it computes a statement digest if that digest is needed, which is true if any of the following conditions are true:

- Performance Schema digest instrumentation is enabled
- MySQL Enterprise Firewall is enabled

The `max_digest_length` system variable value determines the maximum number of bytes available per session for computation of normalized statement digests. Once that amount of space is used during digest computation, truncation occurs: no further tokens from a parsed statement are collected or figure into its digest value. Statements that differ only after that many bytes of parsed tokens produce the same normalized statement digest and are considered identical if compared or if aggregated for digest statistics.

After the normalized statement has been computed, an MD5 hash value is computed from it. In addition:

- If MySQL Enterprise Firewall is enabled, it is called and the digest as computed is available to it.
- If the Performance Schema has digest instrumentation enabled, it makes a copy of the normalized statement digest, allocating a maximum of `performance_schema_max_digest_length` bytes for it. Consequently, if `performance_schema_max_digest_length` is less than `max_digest_length`, the copy is truncated relative to the original. The copy of the normalized

statement digest is stored in the appropriate Performance Schema tables, along with the MD5 hash value computed from the original normalized statement. (If the Performance Schema truncates its copy of the normalized statement digest relative to the original, it does not recompute the MD5 hash value.)

In MySQL 5.6.24 and 5.6.25, `performance_schema_max_digest_length` is not available and `max_digest_length` applies to all digest computation. Before MySQL 5.6.24, neither `max_digest_length` nor `performance_schema_max_digest_length` are available and a fixed maximum of 1024 bytes applies to all digest computation.

Statement normalization transforms the statement text to a more standardized digest string representation that preserves the general statement structure while removing information not essential to the structure:

- Object identifiers such as database and table names are preserved.
- Literal values are converted to parameter markers. A normalized statement does not retain information such as names, passwords, dates, and so forth.
- Comments are removed and whitespace is adjusted.

Consider these statements:

```
SELECT * FROM orders WHERE customer_id=10 AND quantity>20
SELECT * FROM orders WHERE customer_id = 20 AND quantity > 100
```

To normalize these statements, the parser replaces data values by `?` and adjusts whitespace. Both statements yield the same normalized form and thus are considered “the same”:

```
SELECT * FROM orders WHERE customer_id = ? AND quantity > ?
```

The normalized statement contains less information but is still representative of the original statement. Other similar statements that have different data values have the same normalized form.

Now consider these statements:

```
SELECT * FROM customers WHERE customer_id = 1000
SELECT * FROM orders WHERE customer_id = 1000
```

In this case, the normalized statements differ because the object identifiers differ:

```
SELECT * FROM customers WHERE customer_id = ?
SELECT * FROM orders WHERE customer_id = ?
```

If normalization produces a statement that exceeds the space available in the digest buffer (as determined by `max_digest_length`), truncation occurs and the text ends with “...”. Long normalized statements that differ only in the part that occurs following the “...” are considered the same. Consider these statements:

```
SELECT * FROM mytable WHERE cola = 10 AND colb = 20
SELECT * FROM mytable WHERE cola = 10 AND colc = 20
```

If the cutoff happens to be right after the `AND`, both statements have this normalized form:

```
SELECT * FROM mytable WHERE cola = ? AND ...
```

In this case, the difference in the second column name is lost and both statements are considered the same.

Statement Digests in the Performance Schema

In the Performance Schema, statement digesting involves these elements:

- A `statements_digest` consumer in the `setup_consumers` table controls whether the Performance Schema maintains digest information. See Statement Digest Consumer.
- The statement event tables (`events_statements_current`, `events_statements_history`, and `events_statements_history_long`) have columns for storing normalized statement digests and the corresponding digest MD5 hash values:
  - `DIGEST_TEXT` is the text of the normalized statement digest. This is a copy of the original normalized statement that was computed to a maximum of `max_digest_length` bytes, further truncated as necessary to `performance_schema_max_digest_length` bytes.
  - `DIGEST` is the digest MD5 hash value computed from the original normalized statement.

  See Section 22.12.6, “Performance Schema Statement Event Tables”.
- The `events_statements_summary_by_digest` summary table provides aggregated statement digest information. This table aggregates information for statements per `SCHEMA_NAME` and `DIGEST` combination. The Performance Schema uses MD5 hash values for aggregation because they are fast to compute and have a favorable statistical distribution that minimizes collisions. See Section 22.12.9.3, “Statement Summary Tables”.

The statement event tables also have an `SQL_TEXT` column that contains the original SQL statement. The maximum space available for statement display is 1024 bytes.

The `performance_schema_max_digest_length` system variable determines the maximum number of bytes available per statement for digest value storage in the Performance Schema. However, the display length of statement digests may be longer than the available buffer size due to internal encoding of statement elements such as keywords and literal values. Consequently, values selected from the `DIGEST_TEXT` column of statement event tables may appear to exceed the `performance_schema_max_digest_length` value.

The `events_statements_summary_by_digest` summary table provides a profile of the statements executed by the server. It shows what kinds of statements an application is executing and how often. An application developer can use this information together with other information in the table to assess the application's performance characteristics. For example, table columns that show wait times, lock times, or index use may highlight types of queries that are inefficient. This gives the developer insight into which parts of the application need attention.

The `events_statements_summary_by_digest` summary table has a fixed size. By default the Performance Schema estimates the size to use at startup. To specify the table size explicitly, set the `performance_schema_digests_size` system variable at server startup. If the table becomes full, the Performance Schema groups statements that have `SCHEMA_NAME` and `DIGEST` values not matching existing values in the table in a special row with `SCHEMA_NAME` and `DIGEST` set to `NULL`. This permits all statements to be counted. However, if the special row accounts for a significant percentage of the statements executed, it might be desirable to increase the summary table size by increasing `performance_schema_digests_size`.

Statement Digest Memory Use

For applications that generate very long statements that differ only at the end, increasing `max_digest_length` enables computation of digests that distinguish statements that would otherwise aggregate to the same digest. Conversely, decreasing `max_digest_length` causes the server to devote less memory to digest storage but increases the likelihood of longer statements aggregating to the same digest. Administrators should keep in mind that larger values result in correspondingly increased memory requirements, particularly for workloads that involve large numbers of simultaneous sessions (the server allocates `max_digest_length` bytes per session).

As described previously, normalized statement digests as computed by the parser are constrained to a maximum of `max_digest_length` bytes, whereas normalized statement digests stored in the

Performance Schema use `performance_schema_max_digest_length` bytes. The following memory-use considerations apply regarding the relative values of `max_digest_length` and `performance_schema_max_digest_length`:

- If `max_digest_length` is less than `performance_schema_max_digest_length`:
  - Server components other than the Performance Schema use normalized statement digests that take up to `max_digest_length` bytes.
  - The Performance Schema does not further truncate normalized statement digests that it stores, but allocates more memory than `max_digest_length` bytes per digest, which is unnecessary.
- If `max_digest_length` equals `performance_schema_max_digest_length`:
  - Server components other than the Performance Schema use normalized statement digests that take up to `max_digest_length` bytes.
  - The Performance Schema does not further truncate normalized statement digests that it stores, and allocates the same amount of memory as `max_digest_length` bytes per digest.
- If `max_digest_length` is greater than `performance_schema_max_digest_length`:
  - Server components other than the Performance Schema use normalized statement digests that take up to `max_digest_length` bytes.
  - The Performance Schema further truncates normalized statement digests that it stores, and allocates less memory than `max_digest_length` bytes per digest.

Because the Performance Schema statement event tables might store many digests, setting `performance_schema_max_digest_length` smaller than `max_digest_length` enables administrators to balance these factors:

- The need to have long normalized statement digests available for server components outside the Performance Schema
- Many concurrent sessions, each of which allocates digest-computation memory
- The need to limit memory consumption by the Performance Schema statement event tables when storing many statement digests

The `performance_schema_max_digest_length` setting is not per session, it is per statement, and a session can store multiple statements in the `events_statements_history` table. A typical number of statements in this table is 10 per session, so each session consumes 10 times the memory indicated by the `performance_schema_max_digest_length` value, for this table alone.

There are also many statements (and digests) collected globally, most notably in the `events_statements_history_long` table. Here, too, *`N`* statements stored consume *`N`* times the memory indicated by the `performance_schema_max_digest_length` value.

22.11 Performance Schema General Table Characteristics

The name of the `performance_schema` database is lowercase, as are the names of tables within it. Queries should specify the names in lowercase.

Many tables in the `performance_schema` database are read only and cannot be modified:

```
mysql> TRUNCATE TABLE performance_schema.setup_instruments;
ERROR 1683 (HY000): Invalid performance_schema usage.
```

Some of the setup tables have columns that can be modified to affect Performance Schema operation; some also permit rows to be inserted or deleted. Truncation is permitted to clear collected events, so

`TRUNCATE TABLE` can be used on tables containing those kinds of information, such as tables named with a prefix of `events_waits_`.

Summary tables can be truncated with `TRUNCATE TABLE`. Generally, the effect is to reset the summary columns to 0 or `NULL`, not to remove rows. This enables you to clear collected values and restart aggregation. That might be useful, for example, after you have made a runtime configuration change. Exceptions to this truncation behavior are noted in individual summary table sections.

Privileges are as for other databases and tables:

- To retrieve from `performance_schema` tables, you must have the `SELECT` privilege.
- To change those columns that can be modified, you must have the `UPDATE` privilege.
- To truncate tables that can be truncated, you must have the `DROP` privilege.

Because only a limited set of privileges apply to Performance Schema tables, attempts to use `GRANT ALL` as shorthand for granting privileges at the database or table leval fail with an error:

```
mysql> GRANT ALL ON performance_schema.*
       TO 'u1'@'localhost';
ERROR 1044 (42000): Access denied for user 'root'@'localhost'
to database 'performance_schema'
mysql> GRANT ALL ON performance_schema.setup_instruments
       TO 'u2'@'localhost';
ERROR 1044 (42000): Access denied for user 'root'@'localhost'
to database 'performance_schema'
```

Instead, grant exactly the desired privileges:

```
mysql> GRANT SELECT ON performance_schema.*
       TO 'u1'@'localhost';
Query OK, 0 rows affected (0.03 sec)

mysql> GRANT SELECT, UPDATE ON performance_schema.setup_instruments
       TO 'u2'@'localhost';
Query OK, 0 rows affected (0.02 sec)
```

22.12 Performance Schema Table Descriptions

Tables in the `performance_schema` database can be grouped as follows:

- Setup tables. These tables are used to configure and display monitoring characteristics.
- Current events tables. The `events_waits_current` table contains the most recent event for each thread. Other similar tables contain current events at different levels of the event hierarchy: `events_stages_current` for stage events, and `events_statements_current` for statement events.
- History tables. These tables have the same structure as the current events tables, but contain more rows. For example, for wait events, `events_waits_history` table contains the most recent 10 events per thread. `events_waits_history_long` contains the most recent 10,000 events. Other similar tables exist for stage and statement histories.

  To change the sizes of the history tables, set the appropriate system variables at server startup. For example, to set the sizes of the wait event history tables, set `performance_schema_events_waits_history_size` and `performance_schema_events_waits_history_long_size`.
- Summary tables. These tables contain information aggregated over groups of events, including those that have been discarded from the history tables.
- Instance tables. These tables document what types of objects are instrumented. An instrumented object, when used by the server, produces an event. These tables provide event names and explanatory notes or status information.

- Miscellaneous tables. These do not fall into any of the other table groups.

22.12.1 Performance Schema Table Index

The following table lists each Performance Schema table and provides a short description of each one.

Table 22.1 Performance Schema Tables

Table Name	Description
accounts	Connection statistics per client account
cond_instances	synchronization object instances
events_stages_current	Current stage events
events_stages_history	Most recent stage events per thread
events_stages_history_long	Most recent stage events overall
events_stages_summary_by_account_by_event_name	Stage events per account and event name
events_stages_summary_by_host_by_event_name	Stage events per host name and event name
events_stages_summary_by_thread_by_event_name	Stage waits per thread and event name
events_stages_summary_by_user_by_event_name	Stage events per user name and event name
events_stages_summary_global_by_event_name	Stage waits per event name
events_statements_current	Current statement events
events_statements_history	Most recent statement events per thread
events_statements_history_long	Most recent statement events overall
events_statements_summary_by_account_by_event_name	Statement events per account and event name
events_statements_summary_by_digest	Statement events per schema and digest value
events_statements_summary_by_host_by_event_name	Statement events per host name and event name
events_statements_summary_by_thread_by_event_name	Statement events per thread and event name
events_statements_summary_by_user_by_event_name	Statement events per user name and event name
events_statements_summary_global_by_event_name	Statement events per event name
events_waits_current	Current wait events
events_waits_history	Most recent wait events per thread
events_waits_history_long	Most recent wait events overall
events_waits_summary_by_account_by_event_name	Wait events per account and event name
events_waits_summary_by_host_by_event_name	Wait events per host name and event name
events_waits_summary_by_instance	Wait events per instance
events_waits_summary_by_thread_by_event_name	Wait events per thread and event name
events_waits_summary_by_user_by_event_name	Wait events per user name and event name
events_waits_summary_global_by_event_name	Wait events per event name
file_instances	File instances
file_summary_by_event_name	File events per event name
file_summary_by_instance	File events per file instance
host_cache	Information from the internal host cache
hosts	Connection statistics per client host name

Table Name	Description
mutex_instances	Mutex synchronization object instances
objects_summary_global_by_type	Object summaries
performance_timers	Which event timers are available
rwlock_instances	Lock synchronization object instances
session_account_connect_attrs	Connection attributes per for the current session
session_connect_attrs	Connection attributes for all sessions
setup_actors	How to initialize monitoring for new foreground threads
setup_consumers	Consumers for which event information can be stored
setup_instruments	Classes of instrumented objects for which events can be collected
setup_objects	Which objects should be monitored
setup_timers	Current event timer
socket_instances	Active connection instances
socket_summary_by_event_name	Socket waits and I/O per event name
socket_summary_by_instance	Socket waits and I/O per instance
table_io_waits_summary_by_index_usage	Table I/O waits per index
table_io_waits_summary_by_table	Table I/O waits per table
table_lock_waits_summary_by_table	Table lock waits per table
threads	Information about server threads
users	Connection statistics per client user name

22.12.2 Performance Schema Setup Tables

The setup tables provide information about the current instrumentation and enable the monitoring configuration to be changed. For this reason, some columns in these tables can be changed if you have the UPDATE privilege.

The use of tables rather than individual variables for setup information provides a high degree of flexibility in modifying Performance Schema configuration. For example, you can use a single statement with standard SQL syntax to make multiple simultaneous configuration changes.

These setup tables are available:

- setup_actors: How to initialize monitoring for new foreground threads
- setup_consumers: The destinations to which event information can be sent and stored
- setup_instruments: The classes of instrumented objects for which events can be collected
- setup_objects: Which objects should be monitored
- setup_timers: The current event timer

22.12.2.1 The setup_actors Table

The setup_actors table contains information that determines whether to enable monitoring for new foreground server threads (threads associated with client connections). This

table has a maximum size of 100 rows by default. To change the table size, modify the `performance_schema_setup_actors_size` system variable at server startup.

For each new foreground thread, the Performance Schema matches the user and host for the thread against the rows of the `setup_actors` table. If a row from that table matches, its `ENABLED` column value is used to set the `INSTRUMENTED` column of the `threads` table row for the thread. This enables instrumenting to be applied selectively per host, user, or account (user and host combination). If there is no match, the `INSTRUMENTED` column for the thread is set to `NO`.

For background threads, there is no associated user. `INSTRUMENTED` is `YES` by default and `setup_actors` is not consulted.

The initial contents of the `setup_actors` table match any user and host combination, so monitoring is enabled by default for all foreground threads:

```
mysql> SELECT * FROM performance_schema.setup_actors;
+------+------+------+
| HOST | USER | ROLE |
+------+------+------+
| %    | %    | %    |
+------+------+------+
```

For information about how to use the `setup_actors` table to affect event monitoring, see Section 22.4.6, “Pre-Filtering by Thread”.

Modifications to the `setup_actors` table affect only foreground threads created subsequent to the modification, not existing threads. To affect existing threads, modify the `INSTRUMENTED` column of `threads` table rows.

The `setup_actors` table has these columns:

- `HOST`

  The host name. This should be a literal name, or `'%'` to mean “any host.”

- `USER`

  The user name. This should be a literal name, or `'%'` to mean “any user.”

- `ROLE`

  Unused.

`TRUNCATE TABLE` is permitted for the `setup_actors` table. It removes the rows.

22.12.2.2 The setup_consumers Table

The `setup_consumers` table lists the types of consumers for which event information can be stored and which are enabled:

```
mysql> SELECT * FROM performance_schema.setup_consumers;
+--------------------------------+---------+
| NAME                           | ENABLED |
+--------------------------------+---------+
| events_stages_current          | NO      |
| events_stages_history          | NO      |
| events_stages_history_long     | NO      |
| events_statements_current      | YES     |
| events_statements_history      | NO      |
| events_statements_history_long | NO      |
| events_waits_current           | NO      |
| events_waits_history           | NO      |
| events_waits_history_long      | NO      |
| global_instrumentation         | YES     |
```

```
| thread_instrumentation         | YES     |
| statements_digest              | YES     |
+--------------------------------+---------+
```

The consumer settings in the `setup_consumers` table form a hierarchy from higher levels to lower. For detailed information about the effect of enabling different consumers, see Section 22.4.7, “Pre-Filtering by Consumer”.

Modifications to the `setup_consumers` table affect monitoring immediately.

The `setup_consumers` table has these columns:

- `NAME`

  The consumer name.

- `ENABLED`

  Whether the consumer is enabled. The value is `YES` or `NO`. This column can be modified. If you disable a consumer, the server does not spend time adding event information to it.

`TRUNCATE TABLE` is not permitted for the `setup_consumers` table.

22.12.2.3 The setup_instruments Table

The `setup_instruments` table lists classes of instrumented objects for which events can be collected:

```
mysql> SELECT * FROM performance_schema.setup_instruments;
+-----------------------------------------------------+---------+-------+
| NAME                                                | ENABLED | TIMED |
+-----------------------------------------------------+---------+-------+
...
| stage/sql/end                                       | NO      | NO    |
| stage/sql/executing                                 | NO      | NO    |
| stage/sql/init                                      | NO      | NO    |
| stage/sql/insert                                    | NO      | NO    |
...
| statement/sql/load                                  | YES     | YES   |
| statement/sql/grant                                 | YES     | YES   |
| statement/sql/check                                 | YES     | YES   |
| statement/sql/flush                                 | YES     | YES   |
...
| wait/synch/mutex/sql/LOCK_global_read_lock          | YES     | YES   |
| wait/synch/mutex/sql/LOCK_global_system_variables | YES     | YES   |
| wait/synch/mutex/sql/LOCK_lock_db                   | YES     | YES   |
| wait/synch/mutex/sql/LOCK_manager                   | YES     | YES   |
...
| wait/synch/rwlock/sql/LOCK_grant                    | YES     | YES   |
| wait/synch/rwlock/sql/LOGGER::LOCK_logger           | YES     | YES   |
| wait/synch/rwlock/sql/LOCK_sys_init_connect         | YES     | YES   |
| wait/synch/rwlock/sql/LOCK_sys_init_slave           | YES     | YES   |
...
| wait/io/file/sql/binlog                             | YES     | YES   |
| wait/io/file/sql/binlog_index                       | YES     | YES   |
| wait/io/file/sql/casetest                           | YES     | YES   |
| wait/io/file/sql/dbopt                              | YES     | YES   |
...
```

Each instrument added to the source code provides a row for the `setup_instruments` table, even when the instrumented code is not executed. When an instrument is enabled and executed, instrumented instances are created, which are visible in the *`xxx`*`_instances` tables, such as `file_instances` or `rwlock_instances`.

Modifications to most `setup_instruments` rows affect monitoring immediately. For some instruments, modifications are effective only at server startup; changing them at runtime has no effect.

This affects primarily mutexes, conditions, and rwlocks in the server, although there may be other instruments for which this is true.

For more information about the role of the `setup_instruments` table in event filtering, see Section 22.4.3, “Event Pre-Filtering”.

The `setup_instruments` table has these columns:

- `NAME`

  The instrument name. Instrument names may have multiple parts and form a hierarchy, as discussed in Section 22.6, “Performance Schema Instrument Naming Conventions”. Events produced from execution of an instrument have an `EVENT_NAME` value that is taken from the instrument `NAME` value. (Events do not really have a “name,” but this provides a way to associate events with instruments.)

- `ENABLED`

  Whether the instrument is enabled. The value is `YES` or `NO`. A disabled instrument produces no events. This column can be modified, although setting `ENABLED` has no effect for instruments that have already been created.

- `TIMED`

  Whether the instrument is timed. The value is `YES` or `NO`. This column can be modified, although setting `TIMED` has no effect for instruments that have already been created.

  If an enabled instrument is not timed, the instrument code is enabled, but the timer is not. Events produced by the instrument have `NULL` for the `TIMER_START`, `TIMER_END`, and `TIMER_WAIT` timer values. This in turn causes those values to be ignored when calculating the sum, minimum, maximum, and average time values in summary tables.

`TRUNCATE TABLE` is not permitted for the `setup_instruments` table.

22.12.2.4 The setup_objects Table

The `setup_objects` table controls whether the Performance Schema monitors particular objects. This table has a maximum size of 100 rows by default. To change the table size, modify the `performance_schema_setup_objects_size` system variable at server startup.

The initial `setup_objects` contents look like this:

```
mysql> SELECT * FROM performance_schema.setup_objects;
+-------------+--------------------+-------------+---------+-------+
| OBJECT_TYPE | OBJECT_SCHEMA      | OBJECT_NAME | ENABLED | TIMED |
+-------------+--------------------+-------------+---------+-------+
| TABLE       | mysql              | %           | NO      | NO    |
| TABLE       | performance_schema | %           | NO      | NO    |
| TABLE       | information_schema | %           | NO      | NO    |
| TABLE       | %                  | %           | YES     | YES   |
+-------------+--------------------+-------------+---------+-------+
```

Modifications to the `setup_objects` table affect object monitoring immediately.

For object types listed in `setup_objects`, the Performance Schema uses the table to how to monitor them. Object matching is based on the `OBJECT_SCHEMA` and `OBJECT_NAME` columns. Objects for which there is no match are not monitored.

The effect of the default object configuration is to instrument all tables except those in the `mysql`, `INFORMATION_SCHEMA`, and `performance_schema` databases. (Tables in the `INFORMATION_SCHEMA` database are not instrumented regardless of the contents of `setup_objects`; the row for `information_schema.%` simply makes this default explicit.)

When the Performance Schema checks for a match in `setup_objects`, it tries to find more specific matches first. For example, with a table `db1.t1`, it looks for a match for `'db1'` and `'t1'`, then for

'db1' and '%', then for '%' and '%'. The order in which matching occurs matters because different matching setup_objects rows can have different ENABLED and TIMED values.

Rows can be inserted into or deleted from setup_objects by users with the INSERT or DELETE privilege on the table. For existing rows, only the ENABLED and TIMED columns can be modified, by users with the UPDATE privilege on the table.

For more information about the role of the setup_objects table in event filtering, see Section 22.4.3, "Event Pre-Filtering".

The setup_objects table has these columns:

- OBJECT_TYPE

  The type of object to instrument. This is always 'TABLE' (base table).

  TABLE filtering affects table I/O events (wait/io/table/sql/handler instrument) and table lock events (wait/lock/table/sql/handler instrument).

- OBJECT_SCHEMA

  The schema that contains the object. This should be a literal name, or '%' to mean "any schema."

- OBJECT_NAME

  The name of the instrumented object. This should be a literal name, or '%' to mean "any object."

- ENABLED

  Whether events for the object are instrumented. The value is YES or NO. This column can be modified.

- TIMED

  Whether events for the object are timed. The value is YES or NO. This column can be modified.

TRUNCATE TABLE is permitted for the setup_objects table. It removes the rows.

22.12.2.5 The setup_timers Table

The setup_timers table shows the currently selected event timers:

```
mysql> SELECT * FROM performance_schema.setup_timers;
+-----------+-------------+
| NAME      | TIMER_NAME  |
+-----------+-------------+
| idle      | MICROSECOND |
| wait      | CYCLE       |
| stage     | NANOSECOND  |
| statement | NANOSECOND  |
+-----------+-------------+
```

The setup_timers.TIMER_NAME value can be changed to select a different timer. The value can be any of the values in the performance_timers.TIMER_NAME column. For an explanation of how event timing occurs, see Section 22.4.1, "Performance Schema Event Timing".

Modifications to the setup_timers table affect monitoring immediately. Events already in progress may use the original timer for the begin time and the new timer for the end time. To avoid unpredictable results after you make timer changes, use TRUNCATE TABLE to reset Performance Schema statistics.

The setup_timers table has these columns:

- NAME

The type of instrument the timer is used for.

- `TIMER_NAME`

  The timer that applies to the instrument type. This column can be modified.

`TRUNCATE TABLE` is not permitted for the `setup_timers` table.

22.12.3 Performance Schema Instance Tables

Instance tables document what types of objects are instrumented. They provide event names and explanatory notes or status information:

- `cond_instances`: Condition synchronization object instances
- `file_instances`: File instances
- `mutex_instances`: Mutex synchronization object instances
- `rwlock_instances`: Lock synchronization object instances
- `socket_instances`: Active connection instances

These tables list instrumented synchronization objects, files, and connections. There are three types of synchronization objects: `cond`, `mutex`, and `rwlock`. Each instance table has an `EVENT_NAME` or `NAME` column to indicate the instrument associated with each row. Instrument names may have multiple parts and form a hierarchy, as discussed in Section 22.6, “Performance Schema Instrument Naming Conventions”.

The `mutex_instances.LOCKED_BY_THREAD_ID` and `rwlock_instances.WRITE_LOCKED_BY_THREAD_ID` columns are extremely important for investigating performance bottlenecks or deadlocks. For examples of how to use them for this purpose, see Section 22.18, “Using the Performance Schema to Diagnose Problems”

22.12.3.1 The cond_instances Table

The `cond_instances` table lists all the conditions seen by the Performance Schema while the server executes. A condition is a synchronization mechanism used in the code to signal that a specific event has happened, so that a thread waiting for this condition can resume work.

When a thread is waiting for something to happen, the condition name is an indication of what the thread is waiting for, but there is no immediate way to tell which other threads cause the condition to happen.

The `cond_instances` table has these columns:

- `NAME`

  The instrument name associated with the condition.

- `OBJECT_INSTANCE_BEGIN`

  The address in memory of the instrumented condition.

`TRUNCATE TABLE` is not permitted for the `cond_instances` table.

22.12.3.2 The file_instances Table

The `file_instances` table lists all the files seen by the Performance Schema when executing file I/O instrumentation. If a file on disk has never been opened, it is not listed in `file_instances`. When a file is deleted from the disk, it is also removed from the `file_instances` table.

The `file_instances` table has these columns:

- `FILE_NAME`

  The file name.

- `EVENT_NAME`

  The instrument name associated with the file.

- `OPEN_COUNT`

  The count of open handles on the file. If a file was opened and then closed, it was opened 1 time, but `OPEN_COUNT` is 0. To list all the files currently opened by the server, use `WHERE OPEN_COUNT > 0`.

`TRUNCATE TABLE` is not permitted for the `file_instances` table.

22.12.3.3 The mutex_instances Table

The `mutex_instances` table lists all the mutexes seen by the Performance Schema while the server executes. A mutex is a synchronization mechanism used in the code to enforce that only one thread at a given time can have access to some common resource. The resource is said to be “protected” by the mutex.

When two threads executing in the server (for example, two user sessions executing a query simultaneously) do need to access the same resource (a file, a buffer, or some piece of data), these two threads compete against each other, so that the first query to obtain a lock on the mutex causes the other query to wait until the first is done and unlocks the mutex.

The work performed while holding a mutex is said to be in a “critical section,” and multiple queries do execute this critical section in a serialized way (one at a time), which is a potential bottleneck.

The `mutex_instances` table has these columns:

- `NAME`

  The instrument name associated with the mutex.

- `OBJECT_INSTANCE_BEGIN`

  The address in memory of the instrumented mutex.

- `LOCKED_BY_THREAD_ID`

  When a thread currently has a mutex locked, `LOCKED_BY_THREAD_ID` is the `THREAD_ID` of the locking thread, otherwise it is `NULL`.

`TRUNCATE TABLE` is not permitted for the `mutex_instances` table.

For every mutex instrumented in the code, the Performance Schema provides the following information.

- The `setup_instruments` table lists the name of the instrumentation point, with the prefix `wait/synch/mutex/`.
- When some code creates a mutex, a row is added to the `mutex_instances` table. The `OBJECT_INSTANCE_BEGIN` column is a property that uniquely identifies the mutex.
- When a thread attempts to lock a mutex, the `events_waits_current` table shows a row for that thread, indicating that it is waiting on a mutex (in the `EVENT_NAME` column), and indicating which mutex is waited on (in the `OBJECT_INSTANCE_BEGIN` column).
- When a thread succeeds in locking a mutex:

  - `events_waits_current` shows that the wait on the mutex is completed (in the `TIMER_END` and `TIMER_WAIT` columns)
  - The completed wait event is added to the `events_waits_history` and `events_waits_history_long` tables
  - `mutex_instances` shows that the mutex is now owned by the thread (in the `THREAD_ID` column).
- When a thread unlocks a mutex, `mutex_instances` shows that the mutex now has no owner (the `THREAD_ID` column is `NULL`).
- When a mutex object is destroyed, the corresponding row is removed from `mutex_instances`.

By performing queries on both of the following tables, a monitoring application or a DBA can detect bottlenecks or deadlocks between threads that involve mutexes:

- `events_waits_current`, to see what mutex a thread is waiting for
- `mutex_instances`, to see which other thread currently owns a mutex

22.12.3.4 The rwlock_instances Table

The `rwlock_instances` table lists all the `rwlock` instances (read write locks) seen by the Performance Schema while the server executes. An `rwlock` is a synchronization mechanism used in the code to enforce that threads at a given time can have access to some common resource following certain rules. The resource is said to be “protected” by the `rwlock`. The access is either shared (many threads can have a read lock at the same time) or exclusive (only one thread can have a write lock at a given time).

Depending on how many threads are requesting a lock, and the nature of the locks requested, access can be either granted in shared mode, granted in exclusive mode, or not granted at all, waiting for other threads to finish first.

The `rwlock_instances` table has these columns:

- `NAME`

  The instrument name associated with the lock.

- `OBJECT_INSTANCE_BEGIN`

  The address in memory of the instrumented lock.

- `WRITE_LOCKED_BY_THREAD_ID`

  When a thread currently has an `rwlock` locked in exclusive (write) mode, `WRITE_LOCKED_BY_THREAD_ID` is the `THREAD_ID` of the locking thread, otherwise it is `NULL`.

- `READ_LOCKED_BY_COUNT`

  When a thread currently has an `rwlock` locked in shared (read) mode, `READ_LOCKED_BY_COUNT` is incremented by 1. This is a counter only, so it cannot be used directly to find which thread holds a read lock, but it can be used to see whether there is a read contention on an `rwlock`, and see how many readers are currently active.

`TRUNCATE TABLE` is not permitted for the `rwlock_instances` table.

By performing queries on both of the following tables, a monitoring application or a DBA may detect some bottlenecks or deadlocks between threads that involve locks:

- `events_waits_current`, to see what `rwlock` a thread is waiting for

- `rwlock_instances`, to see which other thread currently owns an `rwlock`

There is a limitation: The `rwlock_instances` can be used only to identify the thread holding a write lock, but not the threads holding a read lock.

22.12.3.5 The socket_instances Table

The `socket_instances` table provides a real-time snapshot of the active connections to the MySQL server. The table contains one row per TCP/IP or Unix socket file connection. Information available in this table provides a real-time snapshot of the active connections to the server. (Additional information is available in socket summary tables, including network activity such as socket operations and number of bytes transmitted and received; see Section 22.12.9.7, "Socket Summary Tables").

```
mysql> SELECT * FROM performance_schema.socket_instances\G
*************************** 1. row ***************************
           EVENT_NAME: wait/io/socket/sql/server_unix_socket
OBJECT_INSTANCE_BEGIN: 4316619408
            THREAD_ID: 1
            SOCKET_ID: 16
                   IP:
                 PORT: 0
                STATE: ACTIVE
*************************** 2. row ***************************
           EVENT_NAME: wait/io/socket/sql/client_connection
OBJECT_INSTANCE_BEGIN: 4316644608
            THREAD_ID: 21
            SOCKET_ID: 39
                   IP: 127.0.0.1
                 PORT: 55233
                STATE: ACTIVE
*************************** 3. row ***************************
           EVENT_NAME: wait/io/socket/sql/server_tcpip_socket
OBJECT_INSTANCE_BEGIN: 4316699040
            THREAD_ID: 1
            SOCKET_ID: 14
                   IP: 0.0.0.0
                 PORT: 50603
                STATE: ACTIVE
```

Socket instruments have names of the form `wait/io/socket/sql/socket_type` and are used like this:

1. The server has a listening socket for each network protocol that it supports. The instruments associated with listening sockets for TCP/IP or Unix socket file connections have a *`socket_type`* value of `server_tcpip_socket` or `server_unix_socket`, respectively.

2. When a listening socket detects a connection, the server transfers the connection to a new socket managed by a separate thread. The instrument for the new connection thread has a *`socket_type`* value of `client_connection`.

3. When a connection terminates, the row in `socket_instances` corresponding to it is deleted.

The `socket_instances` table has these columns:

- `EVENT_NAME`

  The name of the `wait/io/socket/*` instrument that produced the event. This is a `NAME` value from the `setup_instruments` table. Instrument names may have multiple parts and form a hierarchy, as discussed in Section 22.6, "Performance Schema Instrument Naming Conventions".

- `OBJECT_INSTANCE_BEGIN`

  This column uniquely identifies the socket. The value is the address of an object in memory.

- `THREAD_ID`

The internal thread identifier assigned by the server. Each socket is managed by a single thread, so each socket can be mapped to a thread which can be mapped to a server process.

- `SOCKET_ID`

  The internal file handle assigned to the socket.

- `IP`

  The client IP address. The value may be either an IPv4 or IPv6 address, or blank to indicate a Unix socket file connection.

- `PORT`

  The TCP/IP port number, in the range from 0 to 65535.

- `STATE`

  The socket status, either `IDLE` or `ACTIVE`. Wait times for active sockets are tracked using the corresponding socket instrument. Wait times for idle sockets are tracked using the `idle` instrument.

  A socket is idle if it is waiting for a request from the client. When a socket becomes idle, the event row in `socket_instances` that is tracking the socket switches from a status of `ACTIVE` to `IDLE`. The `EVENT_NAME` value remains `wait/io/socket/*`, but timing for the instrument is suspended. Instead, an event is generated in the `events_waits_current` table with an `EVENT_NAME` value of `idle`.

  When the next request is received, the `idle` event terminates, the socket instance switches from `IDLE` to `ACTIVE`, and timing of the socket instrument resumes.

`TRUNCATE TABLE` is not permitted for the `socket_instances` table.

The `IP:PORT` column combination value identifies the connection. This combination value is used in the `OBJECT_NAME` column of the `events_waits_xxx` tables, to identify the connection from which socket events come:

- For the Unix domain listener socket (`server_unix_socket`), the port is 0, and the IP is `''`.
- For client connections via the Unix domain listener (`client_connection`), the port is 0, and the IP is `''`.
- For the TCP/IP server listener socket (`server_tcpip_socket`), the port is always the master port (for example, 3306), and the IP is always `0.0.0.0`.
- For client connections via the TCP/IP listener (`client_connection`), the port is whatever the server assigns, but never 0. The IP is the IP of the originating host (`127.0.0.1` or `::1` for the local host)

22.12.4 Performance Schema Wait Event Tables

The Performance Schema instruments waits, which are events that take time. Within the event hierarchy, wait events nest within stage events, which nest within statement events.

These tables store wait events:

- `events_waits_current`: The current wait event for each thread.
- `events_waits_history`: The most recent wait events that have ended per thread.
- `events_waits_history_long`: The most recent wait events that have ended globally (across all threads).

The following sections describe the wait event tables. There are also summary tables that aggregate information about wait events; see Section 22.12.9.1, “Wait Event Summary Tables”.

For more information about the relationship between the three wait event tables, see Section 22.9, “Performance Schema Tables for Current and Historical Events”.

Configuring Wait Event Collection

To control whether to collect wait events, set the state of the relevant instruments and consumers:

- The `setup_instruments` table contains instruments with names that begin with `wait`. Use these instruments to enable or disable collection of individual wait event classes.
- The `setup_consumers` table contains consumer values with names corresponding to the current and historical wait event table names. Use these consumers to filter collection of wait events.

Some wait instruments are enabled by default; others are disabled. For example:

```
mysql> SELECT * FROM performance_schema.setup_instruments
       WHERE NAME LIKE 'wait/io/file/innodb%';
+-------------------------------------+---------+-------+
| NAME                                | ENABLED | TIMED |
+-------------------------------------+---------+-------+
| wait/io/file/innodb/innodb_data_file | YES     | YES   |
| wait/io/file/innodb/innodb_log_file  | YES     | YES   |
| wait/io/file/innodb/innodb_temp_file | YES     | YES   |
+-------------------------------------+---------+-------+
mysql> SELECT * FROM performance_schema.setup_instruments
       WHERE NAME LIKE 'wait/io/socket/%';
+---------------------------------------+---------+-------+
| NAME                                  | ENABLED | TIMED |
+---------------------------------------+---------+-------+
| wait/io/socket/sql/server_tcpip_socket | NO      | NO    |
| wait/io/socket/sql/server_unix_socket  | NO      | NO    |
| wait/io/socket/sql/client_connection   | NO      | NO    |
+---------------------------------------+---------+-------+
```

The wait consumers are disabled by default:

```
mysql> SELECT *
       FROM performance_schema.setup_consumers
       WHERE NAME LIKE 'events_waits%';
+--------------------------+---------+
| NAME                     | ENABLED |
+--------------------------+---------+
| events_waits_current     | NO      |
| events_waits_history     | NO      |
| events_waits_history_long | NO      |
+--------------------------+---------+
```

To control wait event collection at server startup, use lines like these in your `my.cnf` file:

- Enable:

```
[mysqld]
performance-schema-instrument='wait/%=ON'
performance-schema-consumer-events-waits-current=ON
performance-schema-consumer-events-waits-history=ON
performance-schema-consumer-events-waits-history-long=ON
```

- Disable:

```
[mysqld]
performance-schema-instrument='wait/%=OFF'
performance-schema-consumer-events-waits-current=OFF
performance-schema-consumer-events-waits-history=OFF
performance-schema-consumer-events-waits-history-long=OFF
```

To control wait event collection at runtime, update the `setup_instruments` and `setup_consumers` tables:

- Enable:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'YES', TIMED = 'YES'
WHERE NAME LIKE 'wait/%';

UPDATE performance_schema.setup_consumers
SET ENABLED = 'YES'
WHERE NAME LIKE 'events_waits%';
```

- Disable:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO', TIMED = 'NO'
WHERE NAME LIKE 'wait/%';

UPDATE performance_schema.setup_consumers
SET ENABLED = 'NO'
WHERE NAME LIKE 'events_waits%';
```

To collect only specific wait events, enable only the corresponding wait instruments. To collect wait events only for specific wait event tables, enable the wait instruments but only the wait consumers corresponding to the desired tables.

The `setup_timers` table contains a row with a `NAME` value of `wait` that indicates the unit for wait event timing. The default unit is `CYCLE`:

```
mysql> SELECT *
       FROM performance_schema.setup_timers
       WHERE NAME = 'wait';
+------+------------+
| NAME | TIMER_NAME |
+------+------------+
| wait | CYCLE      |
+------+------------+
```

To change the timing unit, modify the `TIMER_NAME` value:

```
UPDATE performance_schema.setup_timers
SET TIMER_NAME = 'NANOSECOND'
WHERE NAME = 'wait';
```

For additional information about configuring event collection, see Section 22.3, “Performance Schema Startup Configuration”, and Section 22.4, “Performance Schema Runtime Configuration”.

22.12.4.1 The events_waits_current Table

The `events_waits_current` table contains current wait events. The table stores one row per thread showing the current status of the thread's most recent monitored wait event, so there is no system variable for configuring the table size.

Of the tables that contain wait event rows, `events_waits_current` is the most fundamental. Other tables that contain wait event rows are logically derived from the current events. For example, the `events_waits_history` and `events_waits_history_long` tables are collections of the most recent wait events that have ended, up to a maximum number of rows per thread and globally across all threads, respectively.

For more information about the relationship between the three wait event tables, see Section 22.9, “Performance Schema Tables for Current and Historical Events”.

For information about configuring whether to collect wait events, see Section 22.12.4, “Performance Schema Wait Event Tables”.

The `events_waits_current` table has these columns:

- `THREAD_ID`, `EVENT_ID`

  The thread associated with the event and the thread current event number when the event starts. The `THREAD_ID` and `EVENT_ID` values taken together uniquely identify the row. No two rows have the same pair of values.

- `END_EVENT_ID`

  This column is set to `NULL` when the event starts and updated to the thread current event number when the event ends.

- `EVENT_NAME`

  The name of the instrument that produced the event. This is a `NAME` value from the `setup_instruments` table. Instrument names may have multiple parts and form a hierarchy, as discussed in Section 22.6, “Performance Schema Instrument Naming Conventions”.

- `SOURCE`

  The name of the source file containing the instrumented code that produced the event and the line number in the file at which the instrumentation occurs. This enables you to check the source to determine exactly what code is involved. For example, if a mutex or lock is being blocked, you can check the context in which this occurs.

- `TIMER_START`, `TIMER_END`, `TIMER_WAIT`

  Timing information for the event. The unit for these values is picoseconds (trillionths of a second). The `TIMER_START` and `TIMER_END` values indicate when event timing started and ended. `TIMER_WAIT` is the event elapsed time (duration).

  If an event has not finished, `TIMER_END` and `TIMER_WAIT` are `NULL` before MySQL 5.6.26. As of 5.6.26, `TIMER_END` is the current timer value and `TIMER_WAIT` is the time elapsed so far (`TIMER_END` − `TIMER_START`).

  If an event is produced from an instrument that has `TIMED = NO`, timing information is not collected, and `TIMER_START`, `TIMER_END`, and `TIMER_WAIT` are all `NULL`.

  For discussion of picoseconds as the unit for event times and factors that affect time values, see Section 22.4.1, “Performance Schema Event Timing”.

- `SPINS`

  For a mutex, the number of spin rounds. If the value is `NULL`, the code does not use spin rounds or spinning is not instrumented.

- `OBJECT_SCHEMA`, `OBJECT_NAME`, `OBJECT_TYPE`, `OBJECT_INSTANCE_BEGIN`

  These columns identify the object “being acted on.” What that means depends on the object type.

  For a synchronization object (`cond`, `mutex`, `rwlock`):

  - `OBJECT_SCHEMA`, `OBJECT_NAME`, and `OBJECT_TYPE` are `NULL`.
  - `OBJECT_INSTANCE_BEGIN` is the address of the synchronization object in memory.

  For a file I/O object:

  - `OBJECT_SCHEMA` is `NULL`.
  - `OBJECT_NAME` is the file name.

- `OBJECT_TYPE` is `FILE`.
- `OBJECT_INSTANCE_BEGIN` is an address in memory.

For a socket object:

- `OBJECT_NAME` is the `IP:PORT` value for the socket.
- `OBJECT_INSTANCE_BEGIN` is an address in memory.

For a table I/O object:

- `OBJECT_SCHEMA` is the name of the schema that contains the table.
- `OBJECT_NAME` is the table name.
- `OBJECT_TYPE` is `TABLE` for a persistent base table or `TEMPORARY TABLE` for a temporary table.
- `OBJECT_INSTANCE_BEGIN` is an address in memory.

An `OBJECT_INSTANCE_BEGIN` value itself has no meaning, except that different values indicate different objects. `OBJECT_INSTANCE_BEGIN` can be used for debugging. For example, it can be used with `GROUP BY OBJECT_INSTANCE_BEGIN` to see whether the load on 1,000 mutexes (that protect, say, 1,000 pages or blocks of data) is spread evenly or just hitting a few bottlenecks. This can help you correlate with other sources of information if you see the same object address in a log file or another debugging or performance tool.

- `INDEX_NAME`

  The name of the index used. `PRIMARY` indicates the table primary index. `NULL` means that no index was used.

- `NESTING_EVENT_ID`

  The `EVENT_ID` value of the event within which this event is nested.

- `NESTING_EVENT_TYPE`

  The nesting event type. The value is `STATEMENT`, `STAGE`, or `WAIT`.

- `OPERATION`

  The type of operation performed, such as `lock`, `read`, or `write`.

- `NUMBER_OF_BYTES`

  The number of bytes read or written by the operation. For table I/O waits (events for the `wait/io/table/sql/handler` instrument), `NUMBER_OF_BYTES` is `NULL`.

- `FLAGS`

  Reserved for future use.

`TRUNCATE TABLE` is permitted for the `events_waits_current` table. It removes the rows.

22.12.4.2 The events_waits_history Table

The `events_waits_history` table contains the *`N`* most recent wait events that have ended per thread. Wait events are not added to the table until they have ended. When the table contains the maximum number of rows for a given thread, the oldest thread row is discarded when a new row for that thread is added. When a thread ends, all its rows are discarded.

The Performance Schema autosizes the value of *N* during server startup. To set the number of rows per thread explicitly, set the `performance_schema_events_waits_history_size` system variable at server startup.

The `events_waits_history` table has the same columns as `events_waits_current`. See Section 22.12.4.1, “The events_waits_current Table”.

`TRUNCATE TABLE` is permitted for the `events_waits_history` table. It removes the rows.

For more information about the relationship between the three wait event tables, see Section 22.9, “Performance Schema Tables for Current and Historical Events”.

For information about configuring whether to collect wait events, see Section 22.12.4, “Performance Schema Wait Event Tables”.

22.12.4.3 The events_waits_history_long Table

The `events_waits_history_long` table contains *N* the most recent wait events that have ended globally, across all threads. Wait events are not added to the table until they have ended. When the table becomes full, the oldest row is discarded when a new row is added, regardless of which thread generated either row.

The Performance Schema autosizes the value of *N* during server startup. To set the table size explicitly, set the `performance_schema_events_waits_history_long_size` system variable at server startup.

The `events_waits_history_long` table has the same columns as `events_waits_current`. See Section 22.12.4.1, “The events_waits_current Table”.

`TRUNCATE TABLE` is permitted for the `events_waits_history_long` table. It removes the rows.

For more information about the relationship between the three wait event tables, see Section 22.9, “Performance Schema Tables for Current and Historical Events”.

For information about configuring whether to collect wait events, see Section 22.12.4, “Performance Schema Wait Event Tables”.

22.12.5 Performance Schema Stage Event Tables

The Performance Schema instruments stages, which are steps during the statement-execution process, such as parsing a statement, opening a table, or performing a `filesort` operation. Stages correspond to the thread states displayed by `SHOW PROCESSLIST` or that are visible in the `INFORMATION_SCHEMA.PROCESSLIST` table. Stages begin and end when state values change.

Within the event hierarchy, wait events nest within stage events, which nest within statement events.

These tables store stage events:

- `events_stages_current`: The current stage event for each thread.
- `events_stages_history`: The most recent stage events that have ended per thread.
- `events_stages_history_long`: The most recent stage events that have ended globally (across all threads).

The following sections describe the stage event tables. There are also summary tables that aggregate information about stage events; see Section 22.12.9.2, “Stage Summary Tables”.

For more information about the relationship between the three stage event tables, see Section 22.9, “Performance Schema Tables for Current and Historical Events”.

- Configuring Stage Event Collection

Configuring Stage Event Collection

To control whether to collect stage events, set the state of the relevant instruments and consumers:

- The `setup_instruments` table contains instruments with names that begin with `stage`. Use these instruments to enable or disable collection of individual stage event classes.
- The `setup_consumers` table contains consumer values with names corresponding to the current and historical stage event table names. Use these consumers to filter collection of stage events.

The stage instruments are disabled by default. For example:

```
mysql> SELECT *
       FROM performance_schema.setup_instruments
       WHERE NAME RLIKE 'stage/sql/[a-c]';
+---------------------------------------------------+---------+-------+
| NAME                                              | ENABLED | TIMED |
+---------------------------------------------------+---------+-------+
| stage/sql/After create                            | NO      | NO    |
| stage/sql/allocating local table                  | NO      | NO    |
| stage/sql/altering table                          | NO      | NO    |
| stage/sql/committing alter table to storage engine | NO      | NO    |
| stage/sql/Changing master                         | NO      | NO    |
| stage/sql/Checking master version                 | NO      | NO    |
| stage/sql/checking permissions                    | NO      | NO    |
| stage/sql/checking privileges on cached query     | NO      | NO    |
| stage/sql/checking query cache for query          | NO      | NO    |
| stage/sql/cleaning up                             | NO      | NO    |
| stage/sql/closing tables                          | NO      | NO    |
| stage/sql/Connecting to master                    | NO      | NO    |
| stage/sql/converting HEAP to MyISAM               | NO      | NO    |
| stage/sql/Copying to group table                  | NO      | NO    |
| stage/sql/Copying to tmp table                    | NO      | NO    |
| stage/sql/copy to tmp table                       | NO      | NO    |
| stage/sql/Creating delayed handler                | NO      | NO    |
| stage/sql/Creating sort index                     | NO      | NO    |
| stage/sql/creating table                          | NO      | NO    |
| stage/sql/Creating tmp table                      | NO      | NO    |
+---------------------------------------------------+---------+-------+
```

The stage consumers are disabled by default:

```
mysql> SELECT *
       FROM performance_schema.setup_consumers
       WHERE NAME LIKE 'events_stages%';
+----------------------------+---------+
| NAME                       | ENABLED |
+----------------------------+---------+
| events_stages_current      | NO      |
| events_stages_history      | NO      |
| events_stages_history_long | NO      |
+----------------------------+---------+
```

To control stage event collection at server startup, use lines like these in your `my.cnf` file:

- Enable:

```
[mysqld]
performance-schema-instrument='stage/%=ON'
performance-schema-consumer-events-stages-current=ON
performance-schema-consumer-events-stages-history=ON
performance-schema-consumer-events-stages-history-long=ON
```

- Disable:

```
[mysqld]
performance-schema-instrument='stage/%=OFF'
performance-schema-consumer-events-stages-current=OFF
performance-schema-consumer-events-stages-history=OFF
performance-schema-consumer-events-stages-history-long=OFF
```

To control stage event collection at runtime, update the `setup_instruments` and `setup_consumers` tables:

- Enable:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'YES', TIMED = 'YES'
WHERE NAME LIKE 'stage/%';

UPDATE performance_schema.setup_consumers
SET ENABLED = 'YES'
WHERE NAME LIKE 'events_stages%';
```

- Disable:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO', TIMED = 'NO'
WHERE NAME LIKE 'stage/%';

UPDATE performance_schema.setup_consumers
SET ENABLED = 'NO'
WHERE NAME LIKE 'events_stages%';
```

To collect only specific stage events, enable only the corresponding stage instruments. To collect stage events only for specific stage event tables, enable the stage instruments but only the stage consumers corresponding to the desired tables.

The `setup_timers` table contains a row with a `NAME` value of `stage` that indicates the unit for stage event timing. The default unit is `NANOSECOND`:

```
mysql> SELECT *
       FROM performance_schema.setup_timers
       WHERE NAME = 'stage';
+-------+------------+
| NAME  | TIMER_NAME |
+-------+------------+
| stage | NANOSECOND |
+-------+------------+
```

To change the timing unit, modify the `TIMER_NAME` value:

```
UPDATE performance_schema.setup_timers
SET TIMER_NAME = 'MICROSECOND'
WHERE NAME = 'stage';
```

For additional information about configuring event collection, see Section 22.3, “Performance Schema Startup Configuration”, and Section 22.4, “Performance Schema Runtime Configuration”.

22.12.5.1 The events_stages_current Table

The `events_stages_current` table contains current stage events. The table stores one row per thread showing the current status of the thread's most recent monitored stage event, so there is no system variable for configuring the table size.

Of the tables that contain stage event rows, `events_stages_current` is the most fundamental. Other tables that contain stage event rows are logically derived from the current events. For example, the `events_stages_history` and `events_stages_history_long` tables are collections of the most recent stage events that have ended, up to a maximum number of rows per thread and globally across all threads, respectively.

For more information about the relationship between the three stage event tables, see Section 22.9, “Performance Schema Tables for Current and Historical Events”.

For information about configuring whether to collect stage events, see Section 22.12.5, “Performance Schema Stage Event Tables”.

The `events_stages_current` table has these columns:

- `THREAD_ID`, `EVENT_ID`

  The thread associated with the event and the thread current event number when the event starts. The `THREAD_ID` and `EVENT_ID` values taken together uniquely identify the row. No two rows have the same pair of values.

- `END_EVENT_ID`

  This column is set to `NULL` when the event starts and updated to the thread current event number when the event ends.

- `EVENT_NAME`

  The name of the instrument that produced the event. This is a `NAME` value from the `setup_instruments` table. Instrument names may have multiple parts and form a hierarchy, as discussed in Section 22.6, “Performance Schema Instrument Naming Conventions”.

- `SOURCE`

  The name of the source file containing the instrumented code that produced the event and the line number in the file at which the instrumentation occurs. This enables you to check the source to determine exactly what code is involved.

- `TIMER_START`, `TIMER_END`, `TIMER_WAIT`

  Timing information for the event. The unit for these values is picoseconds (trillionths of a second). The `TIMER_START` and `TIMER_END` values indicate when event timing started and ended. `TIMER_WAIT` is the event elapsed time (duration).

  If an event has not finished, `TIMER_END` and `TIMER_WAIT` are `NULL` before MySQL 5.6.26. As of 5.6.26, `TIMER_END` is the current timer value and `TIMER_WAIT` is the time elapsed so far (`TIMER_END` − `TIMER_START`).

  If an event is produced from an instrument that has `TIMED = NO`, timing information is not collected, and `TIMER_START`, `TIMER_END`, and `TIMER_WAIT` are all `NULL`.

  For discussion of picoseconds as the unit for event times and factors that affect time values, see Section 22.4.1, “Performance Schema Event Timing”.

- `NESTING_EVENT_ID`

  The `EVENT_ID` value of the event within which this event is nested. The nesting event for a stage event is usually a statement event.

- `NESTING_EVENT_TYPE`

  The nesting event type. The value is `STATEMENT`, `STAGE`, or `WAIT`.

`TRUNCATE TABLE` is permitted for the `events_stages_current` table. It removes the rows.

22.12.5.2 The events_stages_history Table

The `events_stages_history` table contains the *`N`* most recent stage events that have ended per thread. Stage events are not added to the table until they have ended. When the table contains the maximum number of rows for a given thread, the oldest thread row is discarded when a new row for that thread is added. When a thread ends, all its rows are discarded.

The Performance Schema autosizes the value of *`N`* during server startup. To set the number of rows per thread explicitly, set the `performance_schema_events_stages_history_size` system variable at server startup.

The `events_stages_history` table has the same columns as `events_stages_current`. See Section 22.12.5.1, “The events_stages_current Table”.

`TRUNCATE TABLE` is permitted for the `events_stages_history` table. It removes the rows.

For more information about the relationship between the three stage event tables, see Section 22.9, “Performance Schema Tables for Current and Historical Events”.

For information about configuring whether to collect stage events, see Section 22.12.5, “Performance Schema Stage Event Tables”.

22.12.5.3 The events_stages_history_long Table

The `events_stages_history_long` table contains the *`N`* most recent stage events that have ended globally, across all threads. Stage events are not added to the table until they have ended. When the table becomes full, the oldest row is discarded when a new row is added, regardless of which thread generated either row.

The Performance Schema autosizes the value of *`N`* during server startup. To set the table size explicitly, set the `performance_schema_events_stages_history_long_size` system variable at server startup.

The `events_stages_history_long` table has the same columns as `events_stages_current`. See Section 22.12.5.1, “The events_stages_current Table”.

`TRUNCATE TABLE` is permitted for the `events_stages_history_long` table. It removes the rows.

For more information about the relationship between the three stage event tables, see Section 22.9, “Performance Schema Tables for Current and Historical Events”.

For information about configuring whether to collect stage events, see Section 22.12.5, “Performance Schema Stage Event Tables”.

22.12.6 Performance Schema Statement Event Tables

The Performance Schema instruments statement execution. Statement events occur at a high level of the event hierarchy: Wait events nest within stage events, which nest within statement events.

These tables store statement events:

- `events_statements_current`: The current statement event for each thread.
- `events_statements_history`: The most recent statement events that have ended per thread.
- `events_statements_history_long`: The most recent statement events that have ended globally (across all threads).

The following sections describe the statement event tables. There are also summary tables that aggregate information about statement events; see Section 22.12.9.3, “Statement Summary Tables”.

For more information about the relationship between the three `events_statements_xxx` event tables, see Section 22.9, “Performance Schema Tables for Current and Historical Events”.

- Configuring Statement Event Collection
- Statement Monitoring

Configuring Statement Event Collection

To control whether to collect statement events, set the state of the relevant instruments and consumers:

- The `setup_instruments` table contains instruments with names that begin with `statement`. Use these instruments to enable or disable collection of individual statement event classes.
- The `setup_consumers` table contains consumer values with names corresponding to the current and historical statement event table names, and the statement digest consumer. Use these consumers to filter collection of statement events and statement digesting.

The statement instruments are enabled by default, and the `events_statements_current` and `statements_digest` statement consumers are enabled by default:

```
mysql> SELECT *
       FROM performance_schema.setup_instruments
       WHERE NAME LIKE 'statement/%';
+----------------------------------------+---------+-------+
| NAME                                   | ENABLED | TIMED |
+----------------------------------------+---------+-------+
| statement/sql/select                   | YES     | YES   |
| statement/sql/create_table             | YES     | YES   |
| statement/sql/create_index             | YES     | YES   |
...
| statement/sp/stmt                      | YES     | YES   |
| statement/sp/set                       | YES     | YES   |
| statement/sp/set_trigger_field         | YES     | YES   |
| statement/scheduler/event              | YES     | YES   |
| statement/com/Sleep                    | YES     | YES   |
| statement/com/Quit                     | YES     | YES   |
| statement/com/Init DB                  | YES     | YES   |
...
| statement/abstract/Query               | YES     | YES   |
| statement/abstract/new_packet          | YES     | YES   |
| statement/abstract/relay_log           | YES     | YES   |
+----------------------------------------+---------+-------+
```

```
mysql> SELECT *
       FROM performance_schema.setup_consumers
       WHERE NAME LIKE '%statements%';
+--------------------------------+---------+
| NAME                           | ENABLED |
+--------------------------------+---------+
| events_statements_current      | YES     |
| events_statements_history      | NO      |
| events_statements_history_long | NO      |
| statements_digest              | YES     |
+--------------------------------+---------+
```

To control statement event collection at server startup, use lines like these in your `my.cnf` file:

- Enable:

```
[mysqld]
performance-schema-instrument='statement/%=ON'
performance-schema-consumer-events-statements-current=ON
performance-schema-consumer-events-statements-history=ON
performance-schema-consumer-events-statements-history-long=ON
performance-schema-consumer-statements-digest=ON
```

- Disable:

```
[mysqld]
performance-schema-instrument='statement/%=OFF'
performance-schema-consumer-events-statements-current=OFF
performance-schema-consumer-events-statements-history=OFF
performance-schema-consumer-events-statements-history-long=OFF
performance-schema-consumer-statements-digest=OFF
```

To control statement event collection at runtime, update the `setup_instruments` and `setup_consumers` tables:

- Enable:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'YES', TIMED = 'YES'
WHERE NAME LIKE 'statement/%';

UPDATE performance_schema.setup_consumers
SET ENABLED = 'YES'
WHERE NAME LIKE '%statements%';
```

- Disable:

```
UPDATE performance_schema.setup_instruments
SET ENABLED = 'NO', TIMED = 'NO'
WHERE NAME LIKE 'statement/%';

UPDATE performance_schema.setup_consumers
SET ENABLED = 'NO'
WHERE NAME LIKE '%statements%';
```

To collect only specific statement events, enable only the corresponding statement instruments. To collect statement events only for specific statement event tables, enable the statement instruments but only the statement consumers corresponding to the desired tables.

The `setup_timers` table contains a row with a `NAME` value of `statement` that indicates the unit for statement event timing. The default unit is `NANOSECOND`:

```
mysql> SELECT *
       FROM performance_schema.setup_timers
       WHERE NAME = 'statement';
+-----------+------------+
| NAME      | TIMER_NAME |
+-----------+------------+
| statement | NANOSECOND |
+-----------+------------+
```

To change the timing unit, modify the `TIMER_NAME` value:

```
UPDATE performance_schema.setup_timers
SET TIMER_NAME = 'MICROSECOND'
WHERE NAME = 'statement';
```

For additional information about configuring event collection, see Section 22.3, “Performance Schema Startup Configuration”, and Section 22.4, “Performance Schema Runtime Configuration”.

Statement Monitoring

Statement monitoring begins from the moment the server sees that activity is requested on a thread, to the moment when all activity has ceased. Typically, this means from the time the server gets the first packet from the client to the time the server has finished sending the response. Monitoring occurs only for top-level statements. Statements within stored programs and subqueries are not seen separately.

When the Performance Schema instruments a request (server command or SQL statement), it uses instrument names that proceed in stages from more general (or “abstract”) to more specific until it arrives at a final instrument name.

Final instrument names correspond to server commands and SQL statements:

- Server commands correspond to the `COM_xxx codes` defined in the `mysql_com.h` header file and processed in `sql/sql_parse.cc`. Examples are `COM_PING` and `COM_QUIT`. Instruments for commands have names that begin with `statement/com`, such as `statement/com/Ping` and `statement/com/Quit`.
- SQL statements are expressed as text, such as `DELETE FROM t1` or `SELECT * FROM t2`. Instruments for SQL statements have names that begin with `statement/sql`, such as `statement/sql/delete` and `statement/sql/select`.

Some final instrument names are specific to error handling:

- `statement/com/Error` accounts for messages received by the server that are out of band. It can be used to detect commands sent by clients that the server does not understand. This may be helpful for purposes such as identifying clients that are misconfigured or using a version of MySQL more recent than that of the server, or clients that are attempting to attack the server.
- `statement/sql/error` accounts for SQL statements that fail to parse. It can be used to detect malformed queries sent by clients. A query that fails to parse differs from a query that parses but fails due to an error during execution. For example, `SELECT * FROM` is malformed, and the `statement/sql/error` instrument is used. By contrast, `SELECT *` parses but fails with a `No tables used` error. In this case, `statement/sql/select` is used and the statement event contains information to indicate the nature of the error.

A request can be obtained from any of these sources:

- As a command or statement request from a client, which sends the request as packets
- As a statement string read from the relay log on a replica

The details for a request are not initially known and the Performance Schema proceeds from abstract to specific instrument names in a sequence that depends on the source of the request.

For a request received from a client:

1. When the server detects a new packet at the socket level, a new statement is started with an abstract instrument name of `statement/abstract/new_packet`.
2. When the server reads the packet number, it knows more about the type of request received, and the Performance Schema refines the instrument name. For example, if the request is a `COM_PING` packet, the instrument name becomes `statement/com/Ping` and that is the final name. If the request is a `COM_QUERY` packet, it is known to correspond to an SQL statement but not the particular type of statement. In this case, the instrument changes from one abstract name to a more specific but still abstract name, `statement/abstract/Query`, and the request requires further classification.
3. If the request is a statement, the statement text is read and given to the parser. After parsing, the exact statement type is known. If the request is, for example, an `INSERT` statement, the Performance Schema refines the instrument name from `statement/abstract/Query` to `statement/sql/insert`, which is the final name.

For a request read as a statement from the relay log on a replica:

1. Statements in the relay log are stored as text and are read as such. There is no network protocol, so the `statement/abstract/new_packet` instrument is not used. Instead, the initial instrument is `statement/abstract/relay_log`.
2. When the statement is parsed, the exact statement type is known. If the request is, for example, an `INSERT` statement, the Performance Schema refines the instrument name from `statement/abstract/Query` to `statement/sql/insert`, which is the final name.

The preceding description applies only for statement-based replication. For row-based replication, table I/O done on the replica as it processes row changes can be instrumented, but row events in the relay log do not appear as discrete statements.

For statistics to be collected for statements, it is not sufficient to enable only the final `statement/sql/*` instruments used for individual statement types. The abtract `statement/abstract/*` instruments must be enabled as well. This should not normally be an issue because all statement instruments are enabled by default. However, an application that enables or disables statement instruments selectively must take into account that disabling abstract instruments also disables statistics collection for the individual statement instruments. For example, to collect statistics for `INSERT` statements, `statement/sql/insert` must be enabled, but also `statement/abstract/new_packet` and `statement/abstract/Query`. Similarly, for replicated statements to be instrumented, `statement/abstract/relay_log` must be enabled.

No statistics are aggregated for abstract instruments such as `statement/abstract/Query` because no statement is ever classified with an abstract instrument as the final statement name.

The abstract instrument names in the preceding discussion are as of MySQL 5.6.15. In earlier 5.6 versions, there was some renaming before those names were settled on:

- `statement/abstract/new_packet` was `statement/com/` in MySQL 5.6.14, `statement/com/new_packet` in MySQL 5.6.13, and `statement/com/` before that.
- `statement/abstract/Query` was `statement/com/Query` before MySQL 5.6.15.
- `statement/abstract/relay_log` was `statement/rpl/relay_log` from MySQL 5.6.13 to 5.6.14 and did not exist before that.

22.12.6.1 The events_statements_current Table

The `events_statements_current` table contains current statement events. The table stores one row per thread showing the current status of the thread's most recent monitored statement event, so there is no system variable for configuring the table size.

Of the tables that contain statement event rows, `events_statements_current` is the most fundamental. Other tables that contain statement event rows are logically derived from the current events. For example, the `events_statements_history` and `events_statements_history_long` tables are collections of the most recent statement events that have ended, up to a maximum number of rows per thread and globally across all threads, respectively.

For more information about the relationship between the three `events_statements_xxx` event tables, see Section 22.9, “Performance Schema Tables for Current and Historical Events”.

For information about configuring whether to collect statement events, see Section 22.12.6, “Performance Schema Statement Event Tables”.

The `events_statements_current` table has these columns:

- `THREAD_ID`, `EVENT_ID`

  The thread associated with the event and the thread current event number when the event starts. The `THREAD_ID` and `EVENT_ID` values taken together uniquely identify the row. No two rows have the same pair of values.

- `END_EVENT_ID`

  This column is set to `NULL` when the event starts and updated to the thread current event number when the event ends.

- `EVENT_NAME`

  The name of the instrument from which the event was collected. This is a `NAME` value from the `setup_instruments` table. Instrument names may have multiple parts and form a hierarchy, as discussed in Section 22.6, “Performance Schema Instrument Naming Conventions”.

  For SQL statements, the `EVENT_NAME` value initially is `statement/com/Query` until the statement is parsed, then changes to a more appropriate value, as described in Section 22.12.6, “Performance Schema Statement Event Tables”.

- `SOURCE`

  The name of the source file containing the instrumented code that produced the event and the line number in the file at which the instrumentation occurs. This enables you to check the source to determine exactly what code is involved.

- `TIMER_START`, `TIMER_END`, `TIMER_WAIT`

Timing information for the event. The unit for these values is picoseconds (trillionths of a second). The `TIMER_START` and `TIMER_END` values indicate when event timing started and ended. `TIMER_WAIT` is the event elapsed time (duration).

If an event has not finished, `TIMER_END` and `TIMER_WAIT` are `NULL` before MySQL 5.6.26. As of 5.6.26, `TIMER_END` is the current timer value and `TIMER_WAIT` is the time elapsed so far (`TIMER_END` − `TIMER_START`).

If an event is produced from an instrument that has `TIMED = NO`, timing information is not collected, and `TIMER_START`, `TIMER_END`, and `TIMER_WAIT` are all `NULL`.

For discussion of picoseconds as the unit for event times and factors that affect time values, see Section 22.4.1, “Performance Schema Event Timing”.

- `LOCK_TIME`

  The time spent waiting for table locks. This value is computed in microseconds but normalized to picoseconds for easier comparison with other Performance Schema timers.

- `SQL_TEXT`

  The text of the SQL statement. For a command not associated with an SQL statement, the value is `NULL`. The maximum space available for statement display is 1024 bytes.

- `DIGEST`

  The statement digest MD5 value as a string of 32 hexadecimal characters, or `NULL` if the `statements_digest` consumer is `no`. For more information about statement digesting, see Section 22.10, “Performance Schema Statement Digests”.

- `DIGEST_TEXT`

  The normalized statement digest text, or `NULL` if the `statements_digest` consumer is `no`. For more information about statement digesting, see Section 22.10, “Performance Schema Statement Digests”.

  The `performance_schema_max_digest_length` system variable determines the maximum number of bytes available per session for digest value storage. However, the display length of statement digests may be longer than the available buffer size due to encoding of statement elements such as keywords and literal values in digest buffer. Consequently, values selected from the `DIGEST_TEXT` column of statement event tables may appear to exceed the `performance_schema_max_digest_length` value.

  `performance_schema_max_digest_length` was added in MySQL 5.6.26. In MySQL 5.6.24 and 5.6.25, `performance_schema_max_digest_length` is not available and the `max_digest_length` value determines the maximum number of bytes available for digest value storage. Before MySQL 5.6.24, neither `max_digest_length` nor `performance_schema_max_digest_length` are available and a fixed maximum of 1024 bytes is available for digest value storage.

- `CURRENT_SCHEMA`

  The default database for the statement, `NULL` if there is none.

- `OBJECT_SCHEMA`, `OBJECT_NAME`, `OBJECT_TYPE`

  Reserved. Always `NULL`.

- `OBJECT_INSTANCE_BEGIN`

  This column identifies the statement. The value is the address of an object in memory.

- `MYSQL_ERRNO`

  The statement error number, from the statement diagnostics area.

- `RETURNED_SQLSTATE`

  The statement SQLSTATE value, from the statement diagnostics area.

- `MESSAGE_TEXT`

  The statement error message, from the statement diagnostics area.

- `ERRORS`

  Whether an error occurred for the statement. The value is 0 if the SQLSTATE value begins with `00` (completion) or `01` (warning). The value is 1 is the SQLSTATE value is anything else.

- `WARNINGS`

  The number of warnings, from the statement diagnostics area.

- `ROWS_AFFECTED`

  The number of rows affected by the statement. For a description of the meaning of “affected,” see mysql_affected_rows().

- `ROWS_SENT`

  The number of rows returned by the statement.

- `ROWS_EXAMINED`

  The number of rows examined by the server layer (not counting any processing internal to storage engines).

- `CREATED_TMP_DISK_TABLES`

  Like the `Created_tmp_disk_tables` status variable, but specific to the statement.

- `CREATED_TMP_TABLES`

  Like the `Created_tmp_tables` status variable, but specific to the statement.

- `SELECT_FULL_JOIN`

  Like the `Select_full_join` status variable, but specific to the statement.

- `SELECT_FULL_RANGE_JOIN`

  Like the `Select_full_range_join` status variable, but specific to the statement.

- `SELECT_RANGE`

  Like the `Select_range` status variable, but specific to the statement.

- `SELECT_RANGE_CHECK`

  Like the `Select_range_check` status variable, but specific to the statement.

- `SELECT_SCAN`

  Like the `Select_scan` status variable, but specific to the statement.

- `SORT_MERGE_PASSES`

Like the `Sort_merge_passes` status variable, but specific to the statement.

- `SORT_RANGE`

  Like the `Sort_range` status variable, but specific to the statement.

- `SORT_ROWS`

  Like the `Sort_rows` status variable, but specific to the statement.

- `SORT_SCAN`

  Like the `Sort_scan` status variable, but specific to the statement.

- `NO_INDEX_USED`

  1 if the statement performed a table scan without using an index, 0 otherwise.

- `NO_GOOD_INDEX_USED`

  1 if the server found no good index to use for the statement, 0 otherwise. For additional information, see the description of the `Extra` column from `EXPLAIN` output for the `Range checked for each record` value in Section 8.8.2, "EXPLAIN Output Format".

- `NESTING_EVENT_ID`, `NESTING_EVENT_TYPE`

  Reserved. Always `NULL`.

`TRUNCATE TABLE` is permitted for the `events_statements_current` table. It removes the rows.

22.12.6.2 The events_statements_history Table

The `events_statements_history` table contains the *`N`* most recent statement events that have ended per thread. Statement events are not added to the table until they have ended. When the table contains the maximum number of rows for a given thread, the oldest thread row is discarded when a new row for that thread is added. When a thread ends, all its rows are discarded.

The Performance Schema autosizes the value of *`N`* during server startup. To set the number of rows per thread explicitly, set the `performance_schema_events_statements_history_size` system variable at server startup.

The `events_statements_history` table has the same columns as `events_statements_current`. See Section 22.12.6.1, "The events_statements_current Table".

`TRUNCATE TABLE` is permitted for the `events_statements_history` table. It removes the rows.

For more information about the relationship between the three `events_statements_`*`xxx`* event tables, see Section 22.9, "Performance Schema Tables for Current and Historical Events".

For information about configuring whether to collect statement events, see Section 22.12.6, "Performance Schema Statement Event Tables".

22.12.6.3 The events_statements_history_long Table

The `events_statements_history_long` table contains the *`N`* most recent statement events that have ended globally, across all threads. Statement events are not added to the table until they have ended. When the table becomes full, the oldest row is discarded when a new row is added, regardless of which thread generated either row.

The value of *`N`* is autosized at server startup. To set the table size explicitly, set the `performance_schema_events_statements_history_long_size` system variable at server startup.

The `events_statements_history_long` table has the same columns as `events_statements_current`. See Section 22.12.6.1, “The events_statements_current Table”.

`TRUNCATE TABLE` is permitted for the `events_statements_history_long` table. It removes the rows.

For more information about the relationship between the three `events_statements_xxx` event tables, see Section 22.9, “Performance Schema Tables for Current and Historical Events”.

For information about configuring whether to collect statement events, see Section 22.12.6, “Performance Schema Statement Event Tables”.

22.12.7 Performance Schema Connection Tables

When a client connects to the MySQL server, it does so under a particular user name and from a particular host. The Performance Schema provides statistics about these connections, tracking them per account (user and host combination) as well as separately per user name and host name, using these tables:

- `accounts`: Connection statistics per client account
- `hosts`: Connection statistics per client host name
- `users`: Connection statistics per client user name

The meaning of “account” in the connection tables is similar to its meaning in the MySQL grant tables in the `mysql` system database, in the sense that the term refers to a combination of user and host values. They differ in that, for grant tables, the host part of an account can be a pattern, whereas for Performance Schema tables, the host value is always a specific nonpattern host name.

Each connection table has `CURRENT_CONNECTIONS` and `TOTAL_CONNECTIONS` columns to track the current and total number of connections per “tracking value” on which its statistics are based. The tables differ in what they use for the tracking value. The `accounts` table has `USER` and `HOST` columns to track connections per user and host combination. The `users` and `hosts` tables have a `USER` and `HOST` column, respectively, to track connections per user name and host name.

The Performance Schema also counts internal threads and threads for user sessions that failed to authenticate, using rows with `USER` and `HOST` column values of `NULL`.

Suppose that clients named `user1` and `user2` each connect one time from `hosta` and `hostb`. The Performance Schema tracks the connections as follows:

- The `accounts` table has four rows, for the `user1`/`hosta`, `user1`/`hostb`, `user2`/`hosta`, and `user2`/`hostb` account values, each row counting one connection per account.
- The `hosts` table has two rows, for `hosta` and `hostb`, each row counting two connections per host name.
- The `users` table has two rows, for `user1` and `user2`, each row counting two connections per user name.

When a client connects, the Performance Schema determines which row in each connection table applies, using the tracking value appropriate to each table. If there is no such row, one is added. Then the Performance Schema increments by one the `CURRENT_CONNECTIONS` and `TOTAL_CONNECTIONS` columns in that row.

When a client disconnects, the Performance Schema decrements by one the `CURRENT_CONNECTIONS` column in the row and leaves the `TOTAL_CONNECTIONS` column unchanged.

`TRUNCATE TABLE` is permitted for connection tables. It has these effects:

- Rows are removed for accounts, hosts, or users that have no current connections (rows with `CURRENT_CONNECTIONS = 0`).
- Nonremoved rows are reset to count only current connections: For rows with `CURRENT_CONNECTIONS > 0`, `TOTAL_CONNECTIONS` is reset to `CURRENT_CONNECTIONS`.
- Summary tables that depend on the connection table are implicitly truncated, as described later in this section.

The Performance Schema maintains summary tables that aggregate connection statistics for various event types by account, host, or user. These tables have `_summary_by_account`, `_summary_by_host`, or `_summary_by_user` in the name. To identify them, use this query:

```
mysql> SELECT TABLE_NAME FROM INFORMATION_SCHEMA.TABLES
       WHERE TABLE_SCHEMA = 'performance_schema'
       AND TABLE_NAME REGEXP '_summary_by_(account|host|user)'
       ORDER BY TABLE_NAME;
+-----------------------------------------------------+
| TABLE_NAME                                          |
+-----------------------------------------------------+
| events_stages_summary_by_account_by_event_name      |
| events_stages_summary_by_host_by_event_name         |
| events_stages_summary_by_user_by_event_name         |
| events_statements_summary_by_account_by_event_name  |
| events_statements_summary_by_host_by_event_name     |
| events_statements_summary_by_user_by_event_name     |
| events_waits_summary_by_account_by_event_name       |
| events_waits_summary_by_host_by_event_name          |
| events_waits_summary_by_user_by_event_name          |
+-----------------------------------------------------+
```

For details about individual connection summary tables, consult the section that describes tables for the summarized event type:

- Wait event summaries: Section 22.12.9.1, “Wait Event Summary Tables”
- Stage event summaries: Section 22.12.9.2, “Stage Summary Tables”
- Statement event summaries: Section 22.12.9.3, “Statement Summary Tables”

`TRUNCATE TABLE` is permitted for connection summary tables. It removes rows for accounts, hosts, or users with no connections, and resets the summary columns to zero for the remaining rows. In addition, each summary table that is aggregated by account, host, user, or thread is implicitly truncated by truncation of the connection table on which it depends. The following table describes the relationship between connection table truncation and implicitly truncated tables.

Table 22.2 Implicit Effects of Connection Table Truncation

Truncated Connection Table	Implicitly Truncated Summary Tables
`accounts`	Tables with names containing `_summary_by_account`, `_summary_by_thread`
`hosts`	Tables with names containing `_summary_by_account`, `_summary_by_host`, `_summary_by_thread`
`users`	Tables with names containing `_summary_by_account`, `_summary_by_user`, `_summary_by_thread`

Truncating a `_summary_global` summary table also implicitly truncates its corresponding connection and thread summary tables. For example, truncating `events_waits_summary_global_by_event_name` implicitly truncates the wait event summary tables that are aggregated by account, host, user, or thread.

22.12.7.1 The accounts Table

The `accounts` table contains a row for each account that has connected to the MySQL server. For each account, the table counts the current and total number of connections. The table size is autosized at server startup. To set the table size explicitly, set the `performance_schema_accounts_size` system variable at server startup. To disable account statistics, set this variable to 0.

The `accounts` table has the following columns. For a description of how the Performance Schema maintains rows in this table, including the effect of `TRUNCATE TABLE`, see Section 22.12.7, "Performance Schema Connection Tables".

- `USER`

  The client user name for the connection. This is `NULL` for an internal thread, or for a user session that failed to authenticate.

- `HOST`

  The host from which the client connected. This is `NULL` for an internal thread, or for a user session that failed to authenticate.

- `CURRENT_CONNECTIONS`

  The current number of connections for the account.

- `TOTAL_CONNECTIONS`

  The total number of connections for the account.

22.12.7.2 The hosts Table

The `hosts` table contains a row for each host from which clients have connected to the MySQL server. For each host name, the table counts the current and total number of connections. The table size is autosized at server startup. To set the table size explicitly, set the `performance_schema_hosts_size` system variable at server startup. To disable host statistics, set this variable to 0.

The `hosts` table has the following columns. For a description of how the Performance Schema maintains rows in this table, including the effect of `TRUNCATE TABLE`, see Section 22.12.7, "Performance Schema Connection Tables".

- `HOST`

  The host from which the client connected. This is `NULL` for an internal thread, or for a user session that failed to authenticate.

- `CURRENT_CONNECTIONS`

  The current number of connections for the host.

- `TOTAL_CONNECTIONS`

  The total number of connections for the host.

22.12.7.3 The users Table

The `users` table contains a row for each user who has connected to the MySQL server. For each user name, the table counts the current and total number of connections. The table size is autosized at server startup. To set the table size explicitly, set the `performance_schema_users_size` system variable at server startup. To disable user statistics, set this variable to 0.

The `users` table has the following columns. For a description of how the Performance Schema maintains rows in this table, including the effect of `TRUNCATE TABLE`, see Section 22.12.7, "Performance Schema Connection Tables".

- `USER`

  The client user name for the connection. This is `NULL` for an internal thread, or for a user session that failed to authenticate.

- `CURRENT_CONNECTIONS`

  The current number of connections for the user.

- `TOTAL_CONNECTIONS`

  The total number of connections for the user.

22.12.8 Performance Schema Connection Attribute Tables

Connection attributes are key-value pairs that application programs can pass to the server at connect time. For applications based on the C API implemented by the `libmysqlclient` client library, the `mysql_options()` and `mysql_options4()` functions define the connection attribute set. Other MySQL Connectors may provide their own attribute-definition methods.

These Performance Schema tables expose attribute information:

- `session_account_connect_attrs`: Connection attributes for the current session, and other sessions associated with the session account
- `session_connect_attrs`: Connection attributes for all sessions

Attribute names that begin with an underscore (_) are reserved for internal use and should not be created by application programs. This convention permits new attributes to be introduced by MySQL without colliding with application attributes, and enables application programs to define their own attributes that do not collide with internal attributes.

- Available Connection Atrributes
- Connection Atrribute Limits

Available Connection Atrributes

The set of connection attributes visible within a given connection varies depending on factors such as your platform, MySQL Connector used to establish the connection, or client program.

The `libmysqlclient` client library sets these attributes:

- `_client_name`: The client name (`libmysql` for the client library).
- `_client_version`: The client library version.
- `_os`: The operating system (for example, `Linux`, `Win64`).
- `_pid`: The client process ID.
- `_platform`: The machine platform (for example, `x86_64`).
- `_thread`: The client thread ID (Windows only).

Other MySQL Connectors may define their own connection attributes.

MySQL Connector/J defines these attributes:

- `_client_license`: The connector license type.
- `_runtime_vendor`: The Java runtime environment (JRE) vendor.
- `_runtime_version`: The Java runtime environment (JRE) version.

MySQL Connector/NET defines these attributes:

- `_client_version`: The client library version.
- `_os`: The operating system (for example, `Linux`, `Win64`).
- `_pid`: The client process ID.
- `_platform`: The machine platform (for example, `x86_64`).
- `_program_name`: The client name.
- `_thread`: The client thread ID (Windows only).

PHP defines attributes that depend on how it was compiled:

- Compiled using `libmysqlclient`: The standard `libmysqlclient` attributes, described previously.
- Compiled using `mysqlnd`: Only the `_client_name` attribute, with a value of `mysqlnd`.

Many MySQL client programs set a `program_name` attribute with a value equal to the client name. For example, `mysqladmin` and `mysqldump` set `program_name` to `mysqladmin` and `mysqldump`, respectively.

Some MySQL client programs define additional attributes:

- `mysqlbinlog`:
  - `_client_role`: `binary_log_listener`
- Replica connections:
  - `program_name`: `mysqld`
  - `_client_role`: `binary_log_listener`
- `FEDERATED` storage engine connections:
  - `program_name`: `mysqld`
  - `_client_role`: `federated_storage`

Connection Atrribute Limits

There are limits on the amount of connection attribute data transmitted from client to server:

- A fixed limit imposed by the client prior to connect time.
- A fixed limit imposed by the server at connect time.
- A configurable limit imposed by the Performance Schema at connect time.

For connections initiated using the C API, the `libmysqlclient` library imposes a limit of 64KB on the aggregate size of connection attribute data on the client side: Calls to `mysql_options()` that cause this limit to be exceeded produce a `CR_INVALID_PARAMETER_NO` error. Other MySQL Connectors may impose their own client-side limits on how much connection attribute data can be transmitted to the server.

On the server side, these size checks on connection attribute data occur:

- The server imposes a limit of 64KB on the aggregate size of connection attribute data it can accept. If a client attempts to send more than 64KB of attribute data, the server rejects the connection.

- For accepted connections, the Performance Schema checks aggregate attribute size against the value of the `performance_schema_session_connect_attrs_size` system variable. If attribute size exceeds this value, these actions take place:
  - The Performance Schema truncates the attribute data and increments the `Performance_schema_session_connect_attrs_lost` status variable, which indicates the number of connections for which attribute truncation occurred.
  - The Performance Schema writes a message to the error log if the `log_warnings` system variable is greater than zero:

    ```
    [Warning] Connection attributes of length N were truncated
    ```

22.12.8.1 The session_account_connect_attrs Table

Application programs can provide key-value connection attributes to be passed to the server at connect time. For descriptions of common attributes, see Section 22.12.8, “Performance Schema Connection Attribute Tables”.

The `session_account_connect_attrs` table contains connection attributes only for the current session, and other sessions associated with the session account. To see connection attributes for all sessions, use the `session_connect_attrs` table.

The `session_account_connect_attrs` table has these columns:

- `PROCESSLIST_ID`

  The connection identifier for the session.

- `ATTR_NAME`

  The attribute name.

- `ATTR_VALUE`

  The attribute value.

- `ORDINAL_POSITION`

  The order in which the attribute was added to the set of connection attributes.

`TRUNCATE TABLE` is not permitted for the `session_account_connect_attrs` table.

22.12.8.2 The session_connect_attrs Table

Application programs can provide key-value connection attributes to be passed to the server at connect time. For descriptions of common attributes, see Section 22.12.8, “Performance Schema Connection Attribute Tables”.

The `session_connect_attrs` table contains connection attributes for all sessions. To see connection attributes only for the current session, and other sessions associated with the session account, use the `session_account_connect_attrs` table.

The `session_connect_attrs` table has these columns:

- `PROCESSLIST_ID`

  The connection identifier for the session.

- `ATTR_NAME`

  The attribute name.

- `ATTR_VALUE`

  The attribute value.

- `ORDINAL_POSITION`

  The order in which the attribute was added to the set of connection attributes.

`TRUNCATE TABLE` is not permitted for the `session_connect_attrs` table.

22.12.9 Performance Schema Summary Tables

Summary tables provide aggregated information for terminated events over time. The tables in this group summarize event data in different ways.

Wait Event Summaries

- `events_waits_summary_by_account_by_event_name`: Wait events per account and event name
- `events_waits_summary_by_host_by_event_name`: Wait events per host name and event name
- `events_waits_summary_by_instance`: Wait events per instance
- `events_waits_summary_by_thread_by_event_name`: Wait events per thread and event name
- `events_waits_summary_by_user_by_event_name`: Wait events per user name and event name
- `events_waits_summary_global_by_event_name`: Wait events per event name

Stage Summaries

- `events_stages_summary_by_account_by_event_name`: Stage events per account and event name
- `events_stages_summary_by_host_by_event_name`: Stage events per host name and event name
- `events_stages_summary_by_thread_by_event_name`: Stage waits per thread and event name
- `events_stages_summary_by_user_by_event_name`: Stage events per user name and event name
- `events_stages_summary_global_by_event_name`: Stage waits per event name

Statement Summaries

- `events_statements_summary_by_account_by_event_name`: Statement events per account and event name
- `events_statements_summary_by_digest`: Statement events per schema and digest value
- `events_statements_summary_by_host_by_event_name`: Statement events per host name and event name
- `events_statements_summary_by_thread_by_event_name`: Statement events per thread and event name
- `events_statements_summary_by_user_by_event_name`: Statement events per user name and event name

- `events_statements_summary_global_by_event_name`: Statement events per event name

Object Wait Summaries

- `objects_summary_global_by_type`: Object summaries

File I/O Summaries

- `file_summary_by_event_name`: File events per event name
- `file_summary_by_instance`: File events per file instance

Table I/O and Lock Wait Summaries

- `table_io_waits_summary_by_index_usage`: Table I/O waits per index
- `table_io_waits_summary_by_table`: Table I/O waits per table
- `table_lock_waits_summary_by_table`: Table lock waits per table

Socket Summaries

- `socket_summary_by_instance`: Socket waits and I/O per instance
- `socket_summary_by_event_name`: Socket waits and I/O per event name

Each summary table has grouping columns that determine how to group the data to be aggregated, and summary columns that contain the aggregated values. Tables that summarize events in similar ways often have similar sets of summary columns and differ only in the grouping columns used to determine how events are aggregated.

Summary tables can be truncated with `TRUNCATE TABLE`. Generally, the effect is to reset the summary columns to 0 or `NULL`, not to remove rows. This enables you to clear collected values and restart aggregation. That might be useful, for example, after you have made a runtime configuration change. Exceptions to this truncation behavior are noted in individual summary table sections.

22.12.9.1 Wait Event Summary Tables

The Performance Schema maintains tables for collecting current and recent wait events, and aggregates that information in summary tables. Section 22.12.4, “Performance Schema Wait Event Tables” describes the events on which wait summaries are based. See that discussion for information about the content of wait events, the current and recent wait event tables, and how to control wait event collection, which is disabled by default.

Example wait event summary information:

```
mysql> SELECT *
       FROM performance_schema.events_waits_summary_global_by_event_name\G
...
*************************** 6. row ***************************
    EVENT_NAME: wait/synch/mutex/sql/BINARY_LOG::LOCK_index
    COUNT_STAR: 8
SUM_TIMER_WAIT: 2119302
MIN_TIMER_WAIT: 196092
AVG_TIMER_WAIT: 264912
MAX_TIMER_WAIT: 569421
...
*************************** 9. row ***************************
    EVENT_NAME: wait/synch/mutex/sql/hash_filo::lock
    COUNT_STAR: 69
SUM_TIMER_WAIT: 16848828
MIN_TIMER_WAIT: 0
AVG_TIMER_WAIT: 244185
MAX_TIMER_WAIT: 735345
...
```

Each wait event summary table has one or more grouping columns to indicate how the table aggregates events. Event names refer to names of event instruments in the `setup_instruments` table:

- `events_waits_summary_by_account_by_event_name` has `EVENT_NAME`, `USER`, and `HOST` columns. Each row summarizes events for a given account (user and host combination) and event name.
- `events_waits_summary_by_host_by_event_name` has `EVENT_NAME` and `HOST` columns. Each row summarizes events for a given host and event name.
- `events_waits_summary_by_instance` has `EVENT_NAME` and `OBJECT_INSTANCE_BEGIN` columns. Each row summarizes events for a given event name and object. If an instrument is used to create multiple instances, each instance has a unique `OBJECT_INSTANCE_BEGIN` value and is summarized separately in this table.
- `events_waits_summary_by_thread_by_event_name` has `THREAD_ID` and `EVENT_NAME` columns. Each row summarizes events for a given thread and event name.
- `events_waits_summary_by_user_by_event_name` has `EVENT_NAME` and `USER` columns. Each row summarizes events for a given user and event name.
- `events_waits_summary_global_by_event_name` has an `EVENT_NAME` column. Each row summarizes events for a given event name. An instrument might be used to create multiple instances of the instrumented object. For example, if there is an instrument for a mutex that is created for each connection, there are as many instances as there are connections. The summary row for the instrument summarizes over all these instances.

Each wait event summary table has these summary columns containing aggregated values:

- `COUNT_STAR`

  The number of summarized events. This value includes all events, whether timed or nontimed.

- `SUM_TIMER_WAIT`

  The total wait time of the summarized timed events. This value is calculated only for timed events because nontimed events have a wait time of `NULL`. The same is true for the other *`xxx`*`_TIMER_WAIT` values.

- `MIN_TIMER_WAIT`

  The minimum wait time of the summarized timed events.

- `AVG_TIMER_WAIT`

  The average wait time of the summarized timed events.

- `MAX_TIMER_WAIT`

  The maximum wait time of the summarized timed events.

`TRUNCATE TABLE` is permitted for wait summary tables. It has these effects:

- For summary tables not aggregated by account, host, or user, truncation resets the summary columns to zero rather than removing rows.
- For summary tables aggregated by account, host, or user, truncation removes rows for accounts, hosts, or users with no connections, and resets the summary columns to zero for the remaining rows.

In addition, each wait summary table that is aggregated by account, host, user, or thread is implicitly truncated by truncation of the connection table on which it depends, or truncation

of events_waits_summary_global_by_event_name. For details, see Section 22.12.7, “Performance Schema Connection Tables”.

22.12.9.2 Stage Summary Tables

The Performance Schema maintains tables for collecting current and recent stage events, and aggregates that information in summary tables. Section 22.12.5, “Performance Schema Stage Event Tables” describes the events on which stage summaries are based. See that discussion for information about the content of stage events, the current and historical stage event tables, and how to control stage event collection, which is disabled by default.

Example stage event summary information:

```
mysql> SELECT *
       FROM performance_schema.events_stages_summary_global_by_event_name\G
...
*************************** 5. row ***************************
    EVENT_NAME: stage/sql/checking permissions
    COUNT_STAR: 57
SUM_TIMER_WAIT: 26501888880
MIN_TIMER_WAIT: 7317456
AVG_TIMER_WAIT: 464945295
MAX_TIMER_WAIT: 12858936792
...
*************************** 9. row ***************************
    EVENT_NAME: stage/sql/closing tables
    COUNT_STAR: 37
SUM_TIMER_WAIT: 662606568
MIN_TIMER_WAIT: 1593864
AVG_TIMER_WAIT: 17907891
MAX_TIMER_WAIT: 437977248
...
```

Each stage summary table has one or more grouping columns to indicate how the table aggregates events. Event names refer to names of event instruments in the setup_instruments table:

- events_stages_summary_by_account_by_event_name has EVENT_NAME, USER, and HOST columns. Each row summarizes events for a given account (user and host combination) and event name.
- events_stages_summary_by_host_by_event_name has EVENT_NAME and HOST columns. Each row summarizes events for a given host and event name.
- events_stages_summary_by_thread_by_event_name has THREAD_ID and EVENT_NAME columns. Each row summarizes events for a given thread and event name.
- events_stages_summary_by_user_by_event_name has EVENT_NAME and USER columns. Each row summarizes events for a given user and event name.
- events_stages_summary_global_by_event_name has an EVENT_NAME column. Each row summarizes events for a given event name.

Each stage summary table has these summary columns containing aggregated values: COUNT_STAR, SUM_TIMER_WAIT, MIN_TIMER_WAIT, AVG_TIMER_WAIT, and MAX_TIMER_WAIT. These columns are analogous to the columns of the same names in the wait event summary tables (see Section 22.12.9.1, “Wait Event Summary Tables”), except that the stage summary tables aggregate events from events_stages_current rather than events_waits_current.

TRUNCATE TABLE is permitted for stage summary tables. It has these effects:

- For summary tables not aggregated by account, host, or user, truncation resets the summary columns to zero rather than removing rows.
- For summary tables aggregated by account, host, or user, truncation removes rows for accounts, hosts, or users with no connections, and resets the summary columns to zero for the remaining rows.

In addition, each stage summary table that is aggregated by account, host, user, or thread is implicitly truncated by truncation of the connection table on which it depends, or truncation of `events_stages_summary_global_by_event_name`. For details, see Section 22.12.7, “Performance Schema Connection Tables”.

22.12.9.3 Statement Summary Tables

The Performance Schema maintains tables for collecting current and recent statement events, and aggregates that information in summary tables. Section 22.12.6, “Performance Schema Statement Event Tables” describes the events on which statement summaries are based. See that discussion for information about the content of statement events, the current and historical statement event tables, and how to control statement event collection, which is partially disabled by default.

Example statement event summary information:

```
mysql> SELECT *
       FROM performance_schema.events_statements_summary_global_by_event_name\G
*************************** 1. row ***************************
                EVENT_NAME: statement/sql/select
                COUNT_STAR: 25
            SUM_TIMER_WAIT: 1535983999000
            MIN_TIMER_WAIT: 209823000
            AVG_TIMER_WAIT: 61439359000
            MAX_TIMER_WAIT: 1363397650000
             SUM_LOCK_TIME: 20186000000
                SUM_ERRORS: 0
              SUM_WARNINGS: 0
         SUM_ROWS_AFFECTED: 0
             SUM_ROWS_SENT: 388
         SUM_ROWS_EXAMINED: 370
SUM_CREATED_TMP_DISK_TABLES: 0
     SUM_CREATED_TMP_TABLES: 0
       SUM_SELECT_FULL_JOIN: 0
 SUM_SELECT_FULL_RANGE_JOIN: 0
           SUM_SELECT_RANGE: 0
     SUM_SELECT_RANGE_CHECK: 0
            SUM_SELECT_SCAN: 6
      SUM_SORT_MERGE_PASSES: 0
             SUM_SORT_RANGE: 0
              SUM_SORT_ROWS: 0
              SUM_SORT_SCAN: 0
          SUM_NO_INDEX_USED: 6
     SUM_NO_GOOD_INDEX_USED: 0
...
```

Each statement summary table has one or more grouping columns to indicate how the table aggregates events. Event names refer to names of event instruments in the `setup_instruments` table:

- `events_statements_summary_by_account_by_event_name` has `EVENT_NAME`, `USER`, and `HOST` columns. Each row summarizes events for a given account (user and host combination) and event name.

- `events_statements_summary_by_digest` has `SCHEMA_NAME` and `DIGEST` columns. Each row summarizes events per schema and digest value. (The `DIGEST_TEXT` column contains the corresponding normalized statement digest text, but is neither a grouping nor a summary column.)

  The maximum number of rows in the table is autosized at server startup. To set this maximum explicitly, set the `performance_schema_digests_size` system variable at server startup.

- `events_statements_summary_by_host_by_event_name` has `EVENT_NAME` and `HOST` columns. Each row summarizes events for a given host and event name.

- `events_statements_summary_by_thread_by_event_name` has `THREAD_ID` and `EVENT_NAME` columns. Each row summarizes events for a given thread and event name.

- `events_statements_summary_by_user_by_event_name` has `EVENT_NAME` and `USER` columns. Each row summarizes events for a given user and event name.
- `events_statements_summary_global_by_event_name` has an `EVENT_NAME` column. Each row summarizes events for a given event name.

Each statement summary table has these summary columns containing aggregated values:

- `COUNT_STAR`, `SUM_TIMER_WAIT`, `MIN_TIMER_WAIT`, `AVG_TIMER_WAIT`, `MAX_TIMER_WAIT`

  These columns are analogous to the columns of the same names in the wait event summary tables (see Section 22.12.9.1, “Wait Event Summary Tables”), except that the statement summary tables aggregate events from `events_statements_current` rather than `events_waits_current`.

- `SUM_xxx`

  The aggregate of the corresponding *`xxx`* column in the `events_statements_current` table. For example, the `SUM_LOCK_TIME` and `SUM_ERRORS` columns in statement summary tables are the aggregates of the `LOCK_TIME` and `ERRORS` columns in `events_statements_current` table.

The `events_statements_summary_by_digest` table has these additional summary columns:

- `FIRST_SEEN`, `LAST_SEEN`

  Timestamps indicating when statements with the given digest value were first seen and most recently seen.

`TRUNCATE TABLE` is permitted for statement summary tables. It has these effects:

- For `events_statements_summary_by_digest`, it removes the rows.
- For other summary tables not aggregated by account, host, or user, truncation resets the summary columns to zero rather than removing rows.
- For other summary tables aggregated by account, host, or user, truncation removes rows for accounts, hosts, or users with no connections, and resets the summary columns to zero for the remaining rows.

In addition, each statement summary table that is aggregated by account, host, user, or thread is implicitly truncated by truncation of the connection table on which it depends, or truncation of `events_statements_summary_global_by_event_name`. For details, see Section 22.12.7, “Performance Schema Connection Tables”.

Statement Digest Aggregation Rules

If the `statements_digest` consumer is enabled, aggregation into `events_statements_summary_by_digest` occurs as follows when a statement completes. Aggregation is based on the `DIGEST` value computed for the statement.

- If a `events_statements_summary_by_digest` row already exists with the digest value for the statement that just completed, statistics for the statement are aggregated to that row. The `LAST_SEEN` column is updated to the current time.
- If no row has the digest value for the statement that just completed, and the table is not full, a new row is created for the statement. The `FIRST_SEEN` and `LAST_SEEN` columns are initialized with the current time.
- If no row has the statement digest value for the statement that just completed, and the table is full, the statistics for the statement that just completed are added to a special “catch-all” row with `DIGEST` = `NULL`, which is created if necessary. If the row is created, the `FIRST_SEEN` and `LAST_SEEN` columns are initialized with the current time. Otherwise, the `LAST_SEEN` column is updated with the current time.

The row with `DIGEST` = `NULL` is maintained because Performance Schema tables have a maximum size due to memory constraints. The `DIGEST` = `NULL` row permits digests that do not match other rows to be counted even if the summary table is full, using a common “other” bucket. This row helps you estimate whether the digest summary is representative:

- A `DIGEST` = `NULL` row that has a `COUNT_STAR` value that represents 5% of all digests shows that the digest summary table is very representative; the other rows cover 95% of the statements seen.
- A `DIGEST` = `NULL` row that has a `COUNT_STAR` value that represents 50% of all digests shows that the digest summary table is not very representative; the other rows cover only half the statements seen. Most likely the DBA should increase the maximum table size so that more of the rows counted in the `DIGEST` = `NULL` row would be counted using more specific rows instead. To do this, set the `performance_schema_digests_size` system variable to a larger value at server startup. The default size is 200.

22.12.9.4 Object Wait Summary Table

The Performance Schema maintains the `objects_summary_global_by_type` table for aggregating object wait events.

Example object wait event summary information:

```
mysql> SELECT * FROM performance_schema.objects_summary_global_by_type\G
...
*************************** 3. row ***************************
   OBJECT_TYPE: TABLE
 OBJECT_SCHEMA: test
   OBJECT_NAME: t
    COUNT_STAR: 3
SUM_TIMER_WAIT: 263126976
MIN_TIMER_WAIT: 1522272
AVG_TIMER_WAIT: 87708678
MAX_TIMER_WAIT: 258428280
...
*************************** 10. row ***************************
   OBJECT_TYPE: TABLE
 OBJECT_SCHEMA: mysql
   OBJECT_NAME: user
    COUNT_STAR: 14
SUM_TIMER_WAIT: 365567592
MIN_TIMER_WAIT: 1141704
AVG_TIMER_WAIT: 26111769
MAX_TIMER_WAIT: 334783032
...
```

The `objects_summary_global_by_type` table has these grouping columns to indicate how the table aggregates events: `OBJECT_TYPE`, `OBJECT_SCHEMA`, and `OBJECT_NAME`. Each row summarizes events for the given object.

`objects_summary_global_by_type` has the same summary columns as the `events_waits_summary_by_xxx` tables. See Section 22.12.9.1, “Wait Event Summary Tables”.

`TRUNCATE TABLE` is permitted for the object summary table. It resets the summary columns to zero rather than removing rows.

22.12.9.5 File I/O Summary Tables

The Performance Schema maintains file I/O summary tables that aggregate information about I/O operations.

Example file I/O event summary information:

```
mysql> SELECT * FROM performance_schema.file_summary_by_event_name\G
...
*************************** 2. row ***************************
               EVENT_NAME: wait/io/file/sql/binlog
```

```
               COUNT_STAR: 31
           SUM_TIMER_WAIT: 8243784888
           MIN_TIMER_WAIT: 0
           AVG_TIMER_WAIT: 265928484
           MAX_TIMER_WAIT: 6490658832
...
mysql> SELECT * FROM performance_schema.file_summary_by_instance\G
...
*************************** 2. row ***************************
                FILE_NAME: /var/mysql/share/english/errmsg.sys
               EVENT_NAME: wait/io/file/sql/ERRMSG
               EVENT_NAME: wait/io/file/sql/ERRMSG
    OBJECT_INSTANCE_BEGIN: 4686193384
               COUNT_STAR: 5
           SUM_TIMER_WAIT: 13990154448
           MIN_TIMER_WAIT: 26349624
           AVG_TIMER_WAIT: 2798030607
           MAX_TIMER_WAIT: 8150662536
...
```

Each file I/O summary table has one or more grouping columns to indicate how the table aggregates events. Event names refer to names of event instruments in the `setup_instruments` table:

- `file_summary_by_event_name` has an `EVENT_NAME` column. Each row summarizes events for a given event name.
- `file_summary_by_instance` has `FILE_NAME`, `EVENT_NAME`, and `OBJECT_INSTANCE_BEGIN` columns. Each row summarizes events for a given file and event name.

Each file I/O summary table has the following summary columns containing aggregated values. Some columns are more general and have values that are the same as the sum of the values of more fine-grained columns. In this way, aggregations at higher levels are available directly without the need for user-defined views that sum lower-level columns.

- `COUNT_STAR`, `SUM_TIMER_WAIT`, `MIN_TIMER_WAIT`, `AVG_TIMER_WAIT`, `MAX_TIMER_WAIT`

  These columns aggregate all I/O operations.

- `COUNT_READ`, `SUM_TIMER_READ`, `MIN_TIMER_READ`, `AVG_TIMER_READ`, `MAX_TIMER_READ`, `SUM_NUMBER_OF_BYTES_READ`

  These columns aggregate all read operations, including `FGETS`, `FGETC`, `FREAD`, and `READ`.

- `COUNT_WRITE`, `SUM_TIMER_WRITE`, `MIN_TIMER_WRITE`, `AVG_TIMER_WRITE`, `MAX_TIMER_WRITE`, `SUM_NUMBER_OF_BYTES_WRITE`

  These columns aggregate all write operations, including `FPUTS`, `FPUTC`, `FPRINTF`, `VFPRINTF`, `FWRITE`, and `PWRITE`.

- `COUNT_MISC`, `SUM_TIMER_MISC`, `MIN_TIMER_MISC`, `AVG_TIMER_MISC`, `MAX_TIMER_MISC`

  These columns aggregate all other I/O operations, including `CREATE`, `DELETE`, `OPEN`, `CLOSE`, `STREAM_OPEN`, `STREAM_CLOSE`, `SEEK`, `TELL`, `FLUSH`, `STAT`, `FSTAT`, `CHSIZE`, `RENAME`, and `SYNC`. There are no byte counts for these operations.

`TRUNCATE TABLE` is permitted for file I/O summary tables. It resets the summary columns to zero rather than removing rows.

The MySQL server uses several techniques to avoid I/O operations by caching information read from files, so it is possible that statements you might expect to result in I/O events do not. You may be able to ensure that I/O does occur by flushing caches or restarting the server to reset its state.

22.12.9.6 Table I/O and Lock Wait Summary Tables

The following sections describe the table I/O and lock wait summary tables:

- `table_io_waits_summary_by_index_usage`: Table I/O waits per index
- `table_io_waits_summary_by_table`: Table I/O waits per table
- `table_lock_waits_summary_by_table`: Table lock waits per table

The table_io_waits_summary_by_table Table

The `table_io_waits_summary_by_table` table aggregates all table I/O wait events, as generated by the `wait/io/table/sql/handler` instrument. The grouping is by table.

The `table_io_waits_summary_by_table` table has these grouping columns to indicate how the table aggregates events: `OBJECT_TYPE`, `OBJECT_SCHEMA`, and `OBJECT_NAME`. These columns have the same meaning as in the `events_waits_current` table. They identify the table to which the row applies.

`table_io_waits_summary_by_table` has the following summary columns containing aggregated values. As indicated in the column descriptions, some columns are more general and have values that are the same as the sum of the values of more fine-grained columns. For example, columns that aggregate all writes hold the sum of the corresponding columns that aggregate inserts, updates, and deletes. In this way, aggregations at higher levels are available directly without the need for user-defined views that sum lower-level columns.

- `COUNT_STAR`, `SUM_TIMER_WAIT`, `MIN_TIMER_WAIT`, `AVG_TIMER_WAIT`, `MAX_TIMER_WAIT`

  These columns aggregate all I/O operations. They are the same as the sum of the corresponding *`xxx`*`_READ` and *`xxx`*`_WRITE` columns.

- `COUNT_READ`, `SUM_TIMER_READ`, `MIN_TIMER_READ`, `AVG_TIMER_READ`, `MAX_TIMER_READ`

  These columns aggregate all read operations. They are the same as the sum of the corresponding *`xxx`*`_FETCH` columns.

- `COUNT_WRITE`, `SUM_TIMER_WRITE`, `MIN_TIMER_WRITE`, `AVG_TIMER_WRITE`, `MAX_TIMER_WRITE`

  These columns aggregate all write operations. They are the same as the sum of the corresponding *`xxx`*`_INSERT`, *`xxx`*`_UPDATE`, and *`xxx`*`_DELETE` columns.

- `COUNT_FETCH`, `SUM_TIMER_FETCH`, `MIN_TIMER_FETCH`, `AVG_TIMER_FETCH`, `MAX_TIMER_FETCH`

  These columns aggregate all fetch operations.

- `COUNT_INSERT`, `SUM_TIMER_INSERT`, `MIN_TIMER_INSERT`, `AVG_TIMER_INSERT`, `MAX_TIMER_INSERT`

  These columns aggregate all insert operations.

- `COUNT_UPDATE`, `SUM_TIMER_UPDATE`, `MIN_TIMER_UPDATE`, `AVG_TIMER_UPDATE`, `MAX_TIMER_UPDATE`

  These columns aggregate all update operations.

- `COUNT_DELETE`, `SUM_TIMER_DELETE`, `MIN_TIMER_DELETE`, `AVG_TIMER_DELETE`, `MAX_TIMER_DELETE`

  These columns aggregate all delete operations.

`TRUNCATE TABLE` is permitted for table I/O summary tables. It resets the summary columns to zero rather than removing rows. Truncating this table also truncates the `table_io_waits_summary_by_index_usage` table.

The table_io_waits_summary_by_index_usage Table

The `table_io_waits_summary_by_index_usage` table aggregates all table index I/O wait events, as generated by the `wait/io/table/sql/handler` instrument. The grouping is by table index.

The columns of `table_io_waits_summary_by_index_usage` are nearly identical to `table_io_waits_summary_by_table`. The only difference is the additional group column, `INDEX_NAME`, which corresponds to the name of the index that was used when the table I/O wait event was recorded:

- A value of `PRIMARY` indicates that table I/O used the primary index.
- A value of `NULL` means that table I/O used no index.
- Inserts are counted against `INDEX_NAME = NULL`.

`TRUNCATE TABLE` is permitted for table I/O summary tables. It resets the summary columns to zero rather than removing rows. This table is also truncated by truncation of the `table_io_waits_summary_by_table` table. A DDL operation that changes the index structure of a table may cause the per-index statistics to be reset.

The table_lock_waits_summary_by_table Table

The `table_lock_waits_summary_by_table` table aggregates all table lock wait events, as generated by the `wait/lock/table/sql/handler` instrument. The grouping is by table.

This table contains information about internal and external locks:

- An internal lock corresponds to a lock in the SQL layer. This is currently implemented by a call to `thr_lock()`. In event rows, these locks are distinguished by the `OPERATION` column, which has one of these values:

```
read normal
read with shared locks
read high priority
read no insert
write allow write
write concurrent insert
write delayed
write low priority
write normal
```

- An external lock corresponds to a lock in the storage engine layer. This is currently implemented by a call to `handler::external_lock()`. In event rows, these locks are distinguished by the `OPERATION` column, which has one of these values:

```
read external
write external
```

The `table_lock_waits_summary_by_table` table has these grouping columns to indicate how the table aggregates events: `OBJECT_TYPE`, `OBJECT_SCHEMA`, and `OBJECT_NAME`. These columns have the same meaning as in the `events_waits_current` table. They identify the table to which the row applies.

`table_lock_waits_summary_by_table` has the following summary columns containing aggregated values. As indicated in the column descriptions, some columns are more general and have values that are the same as the sum of the values of more fine-grained columns. For example, columns that aggregate all locks hold the sum of the corresponding columns that aggregate read and write locks. In this way, aggregations at higher levels are available directly without the need for user-defined views that sum lower-level columns.

- `COUNT_STAR`, `SUM_TIMER_WAIT`, `MIN_TIMER_WAIT`, `AVG_TIMER_WAIT`, `MAX_TIMER_WAIT`

These columns aggregate all lock operations. They are the same as the sum of the corresponding *xxx*_READ and *xxx*_WRITE columns.

- COUNT_READ, SUM_TIMER_READ, MIN_TIMER_READ, AVG_TIMER_READ, MAX_TIMER_READ

  These columns aggregate all read-lock operations. They are the same as the sum of the corresponding *xxx*_READ_NORMAL, *xxx*_READ_WITH_SHARED_LOCKS, *xxx*_READ_HIGH_PRIORITY, and *xxx*_READ_NO_INSERT columns.

- COUNT_WRITE, SUM_TIMER_WRITE, MIN_TIMER_WRITE, AVG_TIMER_WRITE, MAX_TIMER_WRITE

  These columns aggregate all write-lock operations. They are the same as the sum of the corresponding *xxx*_WRITE_ALLOW_WRITE, *xxx*_WRITE_CONCURRENT_INSERT, *xxx*_WRITE_DELAYED, *xxx*_WRITE_LOW_PRIORITY, and *xxx*_WRITE_NORMAL columns.

- COUNT_READ_NORMAL, SUM_TIMER_READ_NORMAL, MIN_TIMER_READ_NORMAL, AVG_TIMER_READ_NORMAL, MAX_TIMER_READ_NORMAL

  These columns aggregate internal read locks.

- COUNT_READ_WITH_SHARED_LOCKS, SUM_TIMER_READ_WITH_SHARED_LOCKS, MIN_TIMER_READ_WITH_SHARED_LOCKS, AVG_TIMER_READ_WITH_SHARED_LOCKS, MAX_TIMER_READ_WITH_SHARED_LOCKS

  These columns aggregate internal read locks.

- COUNT_READ_HIGH_PRIORITY, SUM_TIMER_READ_HIGH_PRIORITY, MIN_TIMER_READ_HIGH_PRIORITY, AVG_TIMER_READ_HIGH_PRIORITY, MAX_TIMER_READ_HIGH_PRIORITY

  These columns aggregate internal read locks.

- COUNT_READ_NO_INSERT, SUM_TIMER_READ_NO_INSERT, MIN_TIMER_READ_NO_INSERT, AVG_TIMER_READ_NO_INSERT, MAX_TIMER_READ_NO_INSERT

  These columns aggregate internal read locks.

- COUNT_READ_EXTERNAL, SUM_TIMER_READ_EXTERNAL, MIN_TIMER_READ_EXTERNAL, AVG_TIMER_READ_EXTERNAL, MAX_TIMER_READ_EXTERNAL

  These columns aggregate external read locks.

- COUNT_WRITE_ALLOW_WRITE, SUM_TIMER_WRITE_ALLOW_WRITE, MIN_TIMER_WRITE_ALLOW_WRITE, AVG_TIMER_WRITE_ALLOW_WRITE, MAX_TIMER_WRITE_ALLOW_WRITE

  These columns aggregate internal write locks.

- COUNT_WRITE_CONCURRENT_INSERT, SUM_TIMER_WRITE_CONCURRENT_INSERT, MIN_TIMER_WRITE_CONCURRENT_INSERT, AVG_TIMER_WRITE_CONCURRENT_INSERT, MAX_TIMER_WRITE_CONCURRENT_INSERT

  These columns aggregate internal write locks.

- COUNT_WRITE_DELAYED, SUM_TIMER_WRITE_DELAYED, MIN_TIMER_WRITE_DELAYED, AVG_TIMER_WRITE_DELAYED, MAX_TIMER_WRITE_DELAYED

  These columns aggregate internal write locks.

  DELAYED inserts are deprecated; expect these columns to be removed in a future release.

- `COUNT_WRITE_LOW_PRIORITY`, `SUM_TIMER_WRITE_LOW_PRIORITY`, `MIN_TIMER_WRITE_LOW_PRIORITY`, `AVG_TIMER_WRITE_LOW_PRIORITY`, `MAX_TIMER_WRITE_LOW_PRIORITY`

  These columns aggregate internal write locks.

- `COUNT_WRITE_NORMAL`, `SUM_TIMER_WRITE_NORMAL`, `MIN_TIMER_WRITE_NORMAL`, `AVG_TIMER_WRITE_NORMAL`, `MAX_TIMER_WRITE_NORMAL`

  These columns aggregate internal write locks.

- `COUNT_WRITE_EXTERNAL`, `SUM_TIMER_WRITE_EXTERNAL`, `MIN_TIMER_WRITE_EXTERNAL`, `AVG_TIMER_WRITE_EXTERNAL`, `MAX_TIMER_WRITE_EXTERNAL`

  These columns aggregate external write locks.

`TRUNCATE TABLE` is permitted for table lock summary tables. It resets the summary columns to zero rather than removing rows.

22.12.9.7 Socket Summary Tables

The Performance Schema maintains socket summary tables, which aggregate timer and byte count information for socket operations:

- `socket_summary_by_event_name`: Aggregate timer and byte count statistics generated by the `wait/io/socket/*` instruments for all socket I/O operations, per socket instrument.
- `socket_summary_by_instance`: Aggregate timer and byte count statistics generated by the `wait/io/socket/*` instruments for all socket I/O operations, per socket instance. When a connection terminates, the row in `socket_summary_by_instance` corresponding to it is deleted.

The socket summary tables do not aggregate waits generated by `idle` events while sockets are waiting for the next request from the client. For `idle` event aggregations, use the wait-event summary tables; see Section 22.12.9.1, “Wait Event Summary Tables”.

Each socket summary table has one or more grouping columns to indicate how the table aggregates events. Event names refer to names of event instruments in the `setup_instruments` table:

- `socket_summary_by_event_name` has an `EVENT_NAME` column. Each row summarizes events for a given event name.
- `socket_summary_by_instance` has an `OBJECT_INSTANCE_BEGIN` column. Each row summarizes events for a given object.

Each socket summary table has these summary columns containing aggregated values:

- `COUNT_STAR`, `SUM_TIMER_WAIT`, `MIN_TIMER_WAIT`, `AVG_TIMER_WAIT`, `MAX_TIMER_WAIT`

  These columns aggregate all operations.

- `COUNT_READ`, `SUM_TIMER_READ`, `MIN_TIMER_READ`, `AVG_TIMER_READ`, `MAX_TIMER_READ`, `SUM_NUMBER_OF_BYTES_READ`

  These columns aggregate all receive operations (`RECV`, `RECVFROM`, and `RECVMSG`).

- `COUNT_WRITE`, `SUM_TIMER_WRITE`, `MIN_TIMER_WRITE`, `AVG_TIMER_WRITE`, `MAX_TIMER_WRITE`, `SUM_NUMBER_OF_BYTES_WRITE`

  These columns aggregate all send operations (`SEND`, `SENDTO`, and `SENDMSG`).

- `COUNT_MISC`, `SUM_TIMER_MISC`, `MIN_TIMER_MISC`, `AVG_TIMER_MISC`, `MAX_TIMER_MISC`

These columns aggregate all other socket operations, such as `CONNECT`, `LISTEN`, `ACCEPT`, `CLOSE`, and `SHUTDOWN`. There are no byte counts for these operations.

The `socket_summary_by_instance` table also has an `EVENT_NAME` column that indicates the class of the socket: `client_connection`, `server_tcpip_socket`, `server_unix_socket`. This column can be grouped on to isolate, for example, client activity from that of the server listening sockets.

`TRUNCATE TABLE` is permitted for socket summary tables. Except for `events_statements_summary_by_digest`, tt resets the summary columns to zero rather than removing rows.

22.12.10 Performance Schema Miscellaneous Tables

The following sections describe tables that do not fall into the table categories discussed in the preceding sections:

- `host_cache`: Information from the internal host cache.
- `performance_timers`: Which event timers are available.
- `threads`: Information about server threads.

22.12.10.1 The host_cache Table

The MySQL server maintains an in-memory host cache that contains client host name and IP address information and is used to avoid Domain Name System (DNS) lookups. The `host_cache` table exposes the contents of this cache. The `host_cache_size` system variable controls the size of the host cache, as well as the size of the `host_cache` table. For operational and configuration information about the host cache, see Section 5.1.11.2, “DNS Lookups and the Host Cache”.

Because the `host_cache` table exposes the contents of the host cache, it can be examined using `SELECT` statements. This may help you diagnose the causes of connection problems. The Performance Schema must be enabled or this table is empty.

The `host_cache` table has these columns:

- `IP`

  The IP address of the client that connected to the server, expressed as a string.

- `HOST`

  The resolved DNS host name for that client IP, or `NULL` if the name is unknown.

- `HOST_VALIDATED`

  Whether the IP-to-host name-to-IP DNS resolution was performed successfully for the client IP. If `HOST_VALIDATED` is `YES`, the `HOST` column is used as the host name corresponding to the IP so that additional calls to DNS can be avoided. While `HOST_VALIDATED` is `NO`, DNS resolution is attempted for each connection attempt, until it eventually completes with either a valid result or a permanent error. This information enables the server to avoid caching bad or missing host names during temporary DNS failures, which would negatively affect clients forever.

- `SUM_CONNECT_ERRORS`

  The number of connection errors that are deemed “blocking” (assessed against the `max_connect_errors` system variable). Only protocol handshake errors are counted, and only for hosts that passed validation (`HOST_VALIDATED = YES`).

  Once `SUM_CONNECT_ERRORS` for a given host reaches the value of `max_connect_errors`, new connections from that host are blocked. The `SUM_CONNECT_ERRORS` value can exceed

the `max_connect_errors` value because multiple connection attempts from a host can occur simultaneously while the host is not blocked. Any or all of them can fail, independently incrementing `SUM_CONNECT_ERRORS`, possibly beyond the value of `max_connect_errors`.

Suppose that `max_connect_errors` is 200 and `SUM_CONNECT_ERRORS` for a given host is 199. If 10 clients attempt to connect from that host simultaneously, none of them are blocked because `SUM_CONNECT_ERRORS` has not reached 200. If blocking errors occur for five of the clients, `SUM_CONNECT_ERRORS` is increased by one for each client, for a resulting `SUM_CONNECT_ERRORS` value of 204. The other five clients succeed and are not blocked because the value of `SUM_CONNECT_ERRORS` when their connection attempts began had not reached 200. New connections from the host that begin after `SUM_CONNECT_ERRORS` reaches 200 are blocked.

- `COUNT_HOST_BLOCKED_ERRORS`

  The number of connections that were blocked because `SUM_CONNECT_ERRORS` exceeded the value of the `max_connect_errors` system variable.

- `COUNT_NAMEINFO_TRANSIENT_ERRORS`

  The number of transient errors during IP-to-host name DNS resolution.

- `COUNT_NAMEINFO_PERMANENT_ERRORS`

  The number of permanent errors during IP-to-host name DNS resolution.

- `COUNT_FORMAT_ERRORS`

  The number of host name format errors. MySQL does not perform matching of `Host` column values in the `mysql.user` system table against host names for which one or more of the initial components of the name are entirely numeric, such as `1.2.example.com`. The client IP address is used instead. For the rationale why this type of matching does not occur, see Section 6.2.4, “Specifying Account Names”.

- `COUNT_ADDRINFO_TRANSIENT_ERRORS`

  The number of transient errors during host name-to-IP reverse DNS resolution.

- `COUNT_ADDRINFO_PERMANENT_ERRORS`

  The number of permanent errors during host name-to-IP reverse DNS resolution.

- `COUNT_FCRDNS_ERRORS`

  The number of forward-confirmed reverse DNS errors. These errors occur when IP-to-host name-to-IP DNS resolution produces an IP address that does not match the client originating IP address.

- `COUNT_HOST_ACL_ERRORS`

  The number of errors that occur because no users are permitted to connect from the client host. In such cases, the server returns `ER_HOST_NOT_PRIVILEGED` and does not even ask for a user name or password.

- `COUNT_NO_AUTH_PLUGIN_ERRORS`

  The number of errors due to requests for an unavailable authentication plugin. A plugin can be unavailable if, for example, it was never loaded or a load attempt failed.

- `COUNT_AUTH_PLUGIN_ERRORS`

  The number of errors reported by authentication plugins.

  An authentication plugin can report different error codes to indicate the root cause of a failure. Depending on the type of error, one of these columns is

incremented: `COUNT_AUTHENTICATION_ERRORS`, `COUNT_AUTH_PLUGIN_ERRORS`, `COUNT_HANDSHAKE_ERRORS`. New return codes are an optional extension to the existing plugin API. Unknown or unexpected plugin errors are counted in the `COUNT_AUTH_PLUGIN_ERRORS` column.

- `COUNT_HANDSHAKE_ERRORS`

  The number of errors detected at the wire protocol level.

- `COUNT_PROXY_USER_ERRORS`

  The number of errors detected when proxy user A is proxied to another user B who does not exist.

- `COUNT_PROXY_USER_ACL_ERRORS`

  The number of errors detected when proxy user A is proxied to another user B who does exist but for whom A does not have the `PROXY` privilege.

- `COUNT_AUTHENTICATION_ERRORS`

  The number of errors caused by failed authentication.

- `COUNT_SSL_ERRORS`

  The number of errors due to SSL problems.

- `COUNT_MAX_USER_CONNECTIONS_ERRORS`

  The number of errors caused by exceeding per-user connection quotas. See Section 6.2.13, “Setting Account Resource Limits”.

- `COUNT_MAX_USER_CONNECTIONS_PER_HOUR_ERRORS`

  The number of errors caused by exceeding per-user connections-per-hour quotas. See Section 6.2.13, “Setting Account Resource Limits”.

- `COUNT_DEFAULT_DATABASE_ERRORS`

  The number of errors related to the default database. For example, the database does not exist or the user has no privileges to access it.

- `COUNT_INIT_CONNECT_ERRORS`

  The number of errors caused by execution failures of statements in the `init_connect` system variable value.

- `COUNT_LOCAL_ERRORS`

  The number of errors local to the server implementation and not related to the network, authentication, or authorization. For example, out-of-memory conditions fall into this category.

- `COUNT_UNKNOWN_ERRORS`

  The number of other, unknown errors not accounted for by other columns in this table. This column is reserved for future use, in case new error conditions must be reported, and if preserving the backward compatibility and structure of the `host_cache` table is required.

- `FIRST_SEEN`

  The timestamp of the first connection attempt seen from the client in the `IP` column.

- `LAST_SEEN`

  The timestamp of the most recent connection attempt seen from the client in the `IP` column.

- `FIRST_ERROR_SEEN`

  The timestamp of the first error seen from the client in the `IP` column.

- `LAST_ERROR_SEEN`

  The timestamp of the most recent error seen from the client in the `IP` column.

`TRUNCATE TABLE` is permitted for the `host_cache` table. It requires the `DROP` privilege for the table. Truncating the table flushes the host cache, which has the effects described in Flushing the Host Cache.

22.12.10.2 The performance_timers Table

The `performance_timers` table shows which event timers are available:

```
mysql> SELECT * FROM performance_schema.performance_timers;
+-------------+-----------------+------------------+----------------+
| TIMER_NAME  | TIMER_FREQUENCY | TIMER_RESOLUTION | TIMER_OVERHEAD |
+-------------+-----------------+------------------+----------------+
| CYCLE       |      2389029850 |                1 |             72 |
| NANOSECOND  |      1000000000 |                1 |            112 |
| MICROSECOND |         1000000 |                1 |            136 |
| MILLISECOND |            1036 |                1 |            168 |
| TICK        |             105 |                1 |           2416 |
+-------------+-----------------+------------------+----------------+
```

If the values associated with a given timer name are `NULL`, that timer is not supported on your platform. The rows that do not contain `NULL` indicate which timers you can use in `setup_timers`. For an explanation of how event timing occurs, see Section 22.4.1, “Performance Schema Event Timing”.

The `performance_timers` table has these columns:

- `TIMER_NAME`

  The name by which to refer to the timer when configuring the `setup_timers` table.

- `TIMER_FREQUENCY`

  The number of timer units per second. For a cycle timer, the frequency is generally related to the CPU speed. For example, on a system with a 2.4GHz processor, the `CYCLE` may be close to 2400000000.

- `TIMER_RESOLUTION`

  Indicates the number of timer units by which timer values increase. If a timer has a resolution of 10, its value increases by 10 each time.

- `TIMER_OVERHEAD`

  The minimal number of cycles of overhead to obtain one timing with the given timer. The Performance Schema determines this value by invoking the timer 20 times during initialization and picking the smallest value. The total overhead really is twice this amount because the instrumentation invokes the timer at the start and end of each event. The timer code is called only for timed events, so this overhead does not apply for nontimed events.

`TRUNCATE TABLE` is not permitted for the `performance_timers` table.

22.12.10.3 The threads Table

The `threads` table contains a row for each server thread. Each row contains information about a thread and indicates whether monitoring is enabled for it:

```
mysql> SELECT * FROM performance_schema.threads\G
*************************** 1. row ***************************
```

```
         THREAD_ID: 1
              NAME: thread/sql/main
              TYPE: BACKGROUND
     PROCESSLIST_ID: NULL
   PROCESSLIST_USER: NULL
   PROCESSLIST_HOST: NULL
     PROCESSLIST_DB: NULL
PROCESSLIST_COMMAND: NULL
   PROCESSLIST_TIME: 80284
  PROCESSLIST_STATE: NULL
   PROCESSLIST_INFO: NULL
   PARENT_THREAD_ID: NULL
              ROLE: NULL
       INSTRUMENTED: YES
...
*************************** 4. row ***************************
         THREAD_ID: 51
              NAME: thread/sql/one_connection
              TYPE: FOREGROUND
     PROCESSLIST_ID: 34
   PROCESSLIST_USER: isabella
   PROCESSLIST_HOST: localhost
     PROCESSLIST_DB: performance_schema
PROCESSLIST_COMMAND: Query
   PROCESSLIST_TIME: 0
  PROCESSLIST_STATE: Sending data
   PROCESSLIST_INFO: SELECT * FROM performance_schema.threads
   PARENT_THREAD_ID: 1
              ROLE: NULL
       INSTRUMENTED: YES
...
```

When the Performance Schema initializes, it populates the `threads` table based on the threads in existence then. Thereafter, a new row is added each time the server creates a thread.

The `INSTRUMENTED` column value for new threads is determined by the contents of the `setup_actors` table. For information about how to use the `setup_actors` table to control this column, see Section 22.4.6, “Pre-Filtering by Thread”.

Removal of rows from the `threads` table occurs when threads end. For a thread associated with a client session, removal occurs when the session ends. If a client has auto-reconnect enabled and the session reconnects after a disconnect, the session becomes associated with a new row in the `threads` table that has a different `PROCESSLIST_ID` value. The initial `INSTRUMENTED` value for the new thread may be different from that of the original thread: The `setup_actors` table may have changed in the meantime, and if the `INSTRUMENTED` value for the original thread was changed after it was initialized, that change does not carry over to the new thread.

You can enable or disable thread monitoring (that is, whether events executed by the thread are instrumented). To control the initial `INSTRUMENTED` value for new foreground threads, use the `setup_actors` table. To control monitoring of existing threads, set the `INSTRUMENTED` column of `threads` table rows. (For more information about the conditions under which thread monitoring occurs, see the description of the `INSTRUMENTED` column.)

For a comparison of the `threads` table columns with names having a prefix of `PROCESSLIST_` to other process information sources, see Sources of Process Information.

Important

For thread information sources other than the `threads` table, information about threads for other users is shown only if the current user has the `PROCESS` privilege. That is not true of the `threads` table; all rows are shown to any user who has the `SELECT` privilege for the table. Users who should not be able to see threads for other users by accessing the `threads` table should not be given the `SELECT` privilege for it.

The `threads` table has these columns:

- `THREAD_ID`

  A unique thread identifier.

- `NAME`

  The name associated with the thread instrumentation code in the server. For example, `thread/sql/one_connection` corresponds to the thread function in the code responsible for handling a user connection, and `thread/sql/main` stands for the `main()` function of the server.

- `TYPE`

  The thread type, either `FOREGROUND` or `BACKGROUND`. User connection threads are foreground threads. Threads associated with internal server activity are background threads. Examples are internal `InnoDB` threads, “binlog dump” threads sending information to replicas, and replica I/O and SQL threads.

- `PROCESSLIST_ID`

  For a foreground thread (associated with a user connection), this is the connection identifier. This is the same value displayed in the `ID` column of the `INFORMATION_SCHEMA PROCESSLIST` table, displayed in the `Id` column of `SHOW PROCESSLIST` output, and returned by the `CONNECTION_ID()` function within the thread.

  For a background thread (not associated with a user connection), `PROCESSLIST_ID` is `NULL`, so the values are not unique.

- `PROCESSLIST_USER`

  The user associated with a foreground thread, `NULL` for a background thread.

- `PROCESSLIST_HOST`

  The host name of the client associated with a foreground thread, `NULL` for a background thread.

  Unlike the `HOST` column of the `INFORMATION_SCHEMA PROCESSLIST` table or the `Host` column of `SHOW PROCESSLIST` output, the `PROCESSLIST_HOST` column does not include the port number for TCP/IP connections. To obtain this information from the Performance Schema, enable the socket instrumentation (which is not enabled by default) and examine the `socket_instances` table:

```
mysql> SELECT * FROM performance_schema.setup_instruments
       WHERE NAME LIKE 'wait/io/socket%';
+---------------------------------------+---------+-------+
| NAME                                  | ENABLED | TIMED |
+---------------------------------------+---------+-------+
| wait/io/socket/sql/server_tcpip_socket | NO      | NO    |
| wait/io/socket/sql/server_unix_socket  | NO      | NO    |
| wait/io/socket/sql/client_connection   | NO      | NO    |
+---------------------------------------+---------+-------+
3 rows in set (0.01 sec)

mysql> UPDATE performance_schema.setup_instruments
       SET ENABLED='YES'
       WHERE NAME LIKE 'wait/io/socket%';
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0

mysql> SELECT * FROM performance_schema.socket_instances\G
*************************** 1. row ***************************
           EVENT_NAME: wait/io/socket/sql/client_connection
OBJECT_INSTANCE_BEGIN: 140612577298432
            THREAD_ID: 31
            SOCKET_ID: 53
                   IP: ::ffff:127.0.0.1
                 PORT: 55642
                STATE: ACTIVE
```

```
...
```

- `PROCESSLIST_DB`

  The default database for the thread, or `NULL` if none has been selected.

- `PROCESSLIST_COMMAND`

  For foreground threads, the type of command the thread is executing on behalf of the client, or `Sleep` if the session is idle. For descriptions of thread commands, see Section 8.14, “Examining Server Thread (Process) Information”. The value of this column corresponds to the `COM_xxx` commands of the client/server protocol and `Com_xxx` status variables. See Section 5.1.9, “Server Status Variables”

  Background threads do not execute commands on behalf of clients, so this column may be `NULL`.

- `PROCESSLIST_TIME`

  The time in seconds that the thread has been in its current state. For a replica SQL thread, the value is the number of seconds between the timestamp of the last replicated event and the real time of the replica host. See Section 17.2.1, “Replication Threads”.

- `PROCESSLIST_STATE`

  An action, event, or state that indicates what the thread is doing. For descriptions of `PROCESSLIST_STATE` values, see Section 8.14, “Examining Server Thread (Process) Information”. If the value if `NULL`, the thread may correspond to an idle client session or the work it is doing is not instrumented with stages.

  Most states correspond to very quick operations. If a thread stays in a given state for many seconds, there might be a problem that bears investigation.

- `PROCESSLIST_INFO`

  The statement the thread is executing, or `NULL` if it is executing no statement. The statement might be the one sent to the server, or an innermost statement if the statement executes other statements. For example, if a `CALL` statement executes a stored procedure that is executing a `SELECT` statement, the `PROCESSLIST_INFO` value shows the `SELECT` statement.

- `PARENT_THREAD_ID`

  If this thread is a subthread (spawned by another thread), this is the `THREAD_ID` value of the spawning thread. Thread spawning occurs, for example, to handle insertion of rows from `INSERT DELAYED` statements.

- `ROLE`

  Unused.

- `INSTRUMENTED`

  Whether events executed by the thread are instrumented. The value is `YES` or `NO`.

  - For foreground threads, the initial `INSTRUMENTED` value is determined by whether the user account associated with the thread matches any row in the `setup_actors` table. Matching is based on the values of the `PROCESSLIST_USER` and `PROCESSLIST_HOST` columns.

    If the thread spawns a subthread, matching occurs again for the `threads` table row created for the subthread.

  - For background threads, `INSTRUMENTED` is `YES` by default. `setup_actors` is not consulted because there is no associated user for background threads.

- For any thread, its `INSTRUMENTED` value can be changed during the lifetime of the thread. This is the only `threads` table column that can be modified.

For monitoring of events executed by the thread to occur, these things must be true:

- The `thread_instrumentation` consumer in the `setup_consumers` table must be `YES`.
- The `threads.INSTRUMENTED` column must be `YES`.
- Monitoring occurs only for those thread events produced from instruments that have the `ENABLED` column set to `YES` in the `setup_instruments` table.

`TRUNCATE TABLE` is not permitted for the `threads` table.

22.13 Performance Schema Option and Variable Reference

Table 22.3 Performance Schema Variable Reference

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
performance_schema	Yes	Yes	Yes		Global	No
Performance_schema_accounts_lost				Yes	Global	No
performance_schema_accounts_size	Yes	Yes	Yes		Global	No
Performance_schema_cond_classes_lost				Yes	Global	No
Performance_schema_cond_instances_lost				Yes	Global	No
performance-schema-consumer-events-stages-current	Yes	Yes				
performance-schema-consumer-events-stages-history	Yes	Yes				
performance-schema-consumer-events-stages-history-long	Yes	Yes				
performance-schema-consumer-events-statements-current	Yes	Yes				
performance-schema-consumer-events-statements-history	Yes	Yes				
performance-schema-consumer-events-statements-history-long	Yes	Yes				
performance-schema-consumer-events-waits-current	Yes	Yes				
performance-schema-consumer-events-waits-history	Yes	Yes				

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dy
performance-schema-consumer-events-waits-history-long	Yes	Yes				
performance-schema-consumer-global-instrumentation	Yes	Yes				
performance-schema-consumer-statements-digest	Yes	Yes				
performance-schema-consumer-thread-instrumentation	Yes	Yes				
Performance_schema_digest_lost				Yes	Global	No
performance_schema_digests_size	Yes	Yes	Yes		Global	No
performance_schema_events_stages_history_long_size	Yes	Yes	Yes		Global	No
performance_schema_events_stages_history_size	Yes	Yes	Yes		Global	No
performance_schema_events_statements_history_long_size	Yes	Yes	Yes		Global	No
performance_schema_events_statements_history_size	Yes	Yes	Yes		Global	No
performance_schema_events_waits_history_long_size	Yes	Yes	Yes		Global	No
performance_schema_events_waits_history_size	Yes	Yes	Yes		Global	No
Performance_schema_file_classes_lost				Yes	Global	No
Performance_schema_file_handles_lost				Yes	Global	No
Performance_schema_file_instances_lost				Yes	Global	No
Performance_schema_hosts_lost				Yes	Global	No
performance_schema_hosts_size	Yes	Yes	Yes		Global	No
performance-schema-instrument	Yes	Yes				
Performance_schema_locker_lost				Yes	Global	No
performance_schema_max_cond_classes	Yes	Yes	Yes		Global	No
performance_schema_max_cond_instances	Yes	Yes	Yes		Global	No
performance_schema_max_digest_length	Yes	Yes	Yes		Global	No
performance_schema_max_file_classes	Yes	Yes	Yes		Global	No
performance_schema_max_file_handles	Yes	Yes	Yes		Global	No
performance_schema_max_file_instances	Yes	Yes	Yes		Global	No
performance_schema_max_mutex_classes	Yes	Yes	Yes		Global	No
performance_schema_max_mutex_instances	Yes	Yes	Yes		Global	No
performance_schema_max_rwlock_classes	Yes	Yes	Yes		Global	No
performance_schema_max_rwlock_instances	Yes	Yes	Yes		Global	No
performance_schema_max_socket_classes	Yes	Yes	Yes		Global	No
performance_schema_max_socket_instances	Yes	Yes	Yes		Global	No
performance_schema_max_stage_classes	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynam
performance_schema_max_statement_classes	Yes	Yes	Yes		Global	No
performance_schema_max_table_handles	Yes	Yes	Yes		Global	No
performance_schema_max_table_instances	Yes	Yes	Yes		Global	No
performance_schema_max_thread_classes	Yes	Yes	Yes		Global	No
performance_schema_max_thread_instances	Yes	Yes	Yes		Global	No
Performance_schema_mutex_classes_lost				Yes	Global	No
Performance_schema_mutex_instances_lost				Yes	Global	No
Performance_schema_rwlock_classes_lost				Yes	Global	No
Performance_schema_rwlock_instances_lost				Yes	Global	No
Performance_schema_session_connect_attrs_lost				Yes	Global	No
performance_schema_session_connect_attrs_size	Yes	Yes	Yes		Global	No
performance_schema_setup_actors_size	Yes	Yes	Yes		Global	No
performance_schema_setup_objects_size	Yes	Yes	Yes		Global	No
Performance_schema_socket_classes_lost				Yes	Global	No
Performance_schema_socket_instances_lost				Yes	Global	No
Performance_schema_stage_classes_lost				Yes	Global	No
Performance_schema_statement_classes_lost				Yes	Global	No
Performance_schema_table_handles_lost				Yes	Global	No
Performance_schema_table_instances_lost				Yes	Global	No
Performance_schema_thread_classes_lost				Yes	Global	No
Performance_schema_thread_instances_lost				Yes	Global	No
Performance_schema_users_lost				Yes	Global	No
performance_schema_users_size	Yes	Yes	Yes		Global	No

22.14 Performance Schema Command Options

Performance Schema parameters can be specified at server startup on the command line or in option files to configure Performance Schema instruments and consumers. Runtime configuration is also possible in many cases (see Section 22.4, “Performance Schema Runtime Configuration”), but startup configuration must be used when runtime configuration is too late to affect instruments that have already been initialized during the startup process.

Performance Schema consumers and instruments can be configured at startup using the following syntax. For additional details, see Section 22.3, “Performance Schema Startup Configuration”.

- `--performance-schema-consumer-`*`consumer_name`*`=value`

  Configure a Performance Schema consumer. Consumer names in the `setup_consumers` table use underscores, but for consumers set at startup, dashes and underscores within the name are equivalent. Options for configuring individual consumers are detailed later in this section.

- `--performance-schema-instrument=`*`instrument_name`*`=value`

  Configure a Performance Schema instrument. The name may be given as a pattern to configure instruments that match the pattern.

The following items configure individual consumers:

- `--performance-schema-consumer-events-stages-current=value`

Configure the `events-stages-current` consumer.

- `--performance-schema-consumer-events-stages-history=value`

  Configure the `events-stages-history` consumer.

- `--performance-schema-consumer-events-stages-history-long=value`

  Configure the `events-stages-history-long` consumer.

- `--performance-schema-consumer-events-statements-current=value`

  Configure the `events-statements-current` consumer.

- `--performance-schema-consumer-events-statements-history=value`

  Configure the `events-statements-history` consumer.

- `--performance-schema-consumer-events-statements-history-long=value`

  Configure the `events-statements-history-long` consumer.

- `--performance-schema-consumer-events-waits-current=value`

  Configure the `events-waits-current` consumer.

- `--performance-schema-consumer-events-waits-history=value`

  Configure the `events-waits-history` consumer.

- `--performance-schema-consumer-events-waits-history-long=value`

  Configure the `events-waits-history-long` consumer.

- `--performance-schema-consumer-global-instrumentation=value`

  Configure the `global-instrumentation` consumer.

- `--performance-schema-consumer-statements-digest=value`

  Configure the `statements-digest` consumer.

- `--performance-schema-consumer-thread-instrumentation=value`

  Configure the `thread-instrumentation` consumer.

22.15 Performance Schema System Variables

The Performance Schema implements several system variables that provide configuration information:

```
mysql> SHOW VARIABLES LIKE 'perf%';
+----------------------------------------------------+---------+
| Variable_name                                      | Value   |
+----------------------------------------------------+---------+
| performance_schema                                 | ON      |
| performance_schema_accounts_size                   | 100     |
| performance_schema_digests_size                    | 200     |
| performance_schema_events_stages_history_long_size | 10000   |
| performance_schema_events_stages_history_size      | 10      |
| performance_schema_events_statements_history_long_size | 10000   |
| performance_schema_events_statements_history_size  | 10      |
| performance_schema_events_waits_history_long_size  | 10000   |
| performance_schema_events_waits_history_size       | 10      |
| performance_schema_hosts_size                      | 100     |
| performance_schema_max_cond_classes                | 80      |
| performance_schema_max_cond_instances              | 1000    |
```

```
| performance_schema_max_file_classes                   | 50      |
| performance_schema_max_file_handles                   | 32768   |
| performance_schema_max_file_instances                 | 10000   |
| performance_schema_max_mutex_classes                  | 200     |
| performance_schema_max_mutex_instances                | 1000000 |
| performance_schema_max_rwlock_classes                 | 30      |
| performance_schema_max_rwlock_instances               | 1000000 |
| performance_schema_max_socket_classes                 | 10      |
| performance_schema_max_socket_instances               | 1000    |
| performance_schema_max_stage_classes                  | 150     |
| performance_schema_max_statement_classes              | 165     |
| performance_schema_max_table_handles                  | 10000   |
| performance_schema_max_table_instances                | 1000    |
| performance_schema_max_thread_classes                 | 50      |
| performance_schema_max_thread_instances               | 1000    |
| performance_schema_session_connect_attrs_size         | 512     |
| performance_schema_setup_actors_size                  | 100     |
| performance_schema_setup_objects_size                 | 100     |
| performance_schema_users_size                         | 100     |
+-------------------------------------------------------+---------+
```

Performance Schema system variables can be set at server startup on the command line or in option files, and many can be set at runtime. See Section 22.13, “Performance Schema Option and Variable Reference”.

The Performance Schema automatically sizes the values of several of its parameters at server startup if they are not set explicitly. For more information, see Section 22.3, “Performance Schema Startup Configuration”.

Performance Schema system variables have the following meanings:

- `performance_schema`

Command-Line Format	`--performance-schema[={OFF|ON}]`
System Variable	`performance_schema`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`ON`

  The value of this variable is `ON` or `OFF` to indicate whether the Performance Schema is enabled. By default, the value is `ON`. At server startup, you can specify this variable with no value or a value of `ON` or 1 to enable it, or with a value of `OFF` or 0 to disable it.

- `performance_schema_accounts_size`

Command-Line Format	`--performance-schema-accounts-size=#`
System Variable	`performance_schema_accounts_size`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)
Minimum Value	`-1` (signifies autosizing; do not assign this literal value)
Maximum Value	`1048576`

  The number of rows in the `accounts` table. If this variable is 0, the Performance Schema does not maintain connection statistics in the `accounts` table.

- `performance_schema_digests_size`

Command-Line Format	`--performance-schema-digests-size=#`
System Variable	`performance_schema_digests_size`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)
Minimum Value	`-1`
Maximum Value	`1048576`

The maximum number of rows in the `events_statements_summary_by_digest` table. If this maximum is exceeded such that a digest cannot be instrumented, the Performance Schema increments the `Performance_schema_digest_lost` status variable.

For more information about statement digesting, see Section 22.10, "Performance Schema Statement Digests".

- `performance_schema_events_stages_history_long_size`

Command-Line Format	`--performance-schema-events-stages-history-long-size=#`
System Variable	`performance_schema_events_stages_history_long_size`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)

The number of rows in the `events_stages_history_long` table.

- `performance_schema_events_stages_history_size`

Command-Line Format	`--performance-schema-events-stages-history-size=#`
System Variable	`performance_schema_events_stages_history_size`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)

The number of rows per thread in the `events_stages_history` table.

- `performance_schema_events_statements_history_long_size`

Command-Line Format	`--performance-schema-events-statements-history-long-size=#`
System Variable	`performance_schema_events_statements_history_long_size`
Scope	Global
Dynamic	No
Type	Integer

Default Value	-1 (signifies autosizing; do not assign this literal value)

The number of rows in the events_statements_history_long table.

- performance_schema_events_statements_history_size

Command-Line Format	--performance-schema-events-statements-history-size=#
System Variable	performance_schema_events_statements_history_size
Scope	Global
Dynamic	No
Type	Integer
Default Value	-1 (signifies autosizing; do not assign this literal value)

The number of rows per thread in the events_statements_history table.

- performance_schema_events_waits_history_long_size

Command-Line Format	--performance-schema-events-waits-history-long-size=#
System Variable	performance_schema_events_waits_history_long_size
Scope	Global
Dynamic	No
Type	Integer
Default Value	-1 (signifies autosizing; do not assign this literal value)

The number of rows in the events_waits_history_long table.

- performance_schema_events_waits_history_size

Command-Line Format	--performance-schema-events-waits-history-size=#
System Variable	performance_schema_events_waits_history_size
Scope	Global
Dynamic	No
Type	Integer
Default Value	-1 (signifies autosizing; do not assign this literal value)

The number of rows per thread in the events_waits_history table.

- performance_schema_hosts_size

Command-Line Format	--performance-schema-hosts-size=#
System Variable	performance_schema_hosts_size
Scope	Global
Dynamic	No
Type	Integer
Default Value	-1 (signifies autosizing; do not assign this literal value)
Minimum Value	-1 (signifies autosizing; do not assign this literal value)

Maximum Value	1048576

The number of rows in the hosts table. If this variable is 0, the Performance Schema does not maintain connection statistics in the hosts table.

- performance_schema_max_cond_classes

Command-Line Format	--performance-schema-max-cond-classes=#
System Variable	performance_schema_max_cond_classes
Scope	Global
Dynamic	No
Type	Integer
Default Value	80
Minimum Value	0
Maximum Value	256

The maximum number of condition instruments. For information about how to set and use this variable, see Section 22.7, “Performance Schema Status Monitoring”.

- performance_schema_max_cond_instances

Command-Line Format	--performance-schema-max-cond-instances=#
System Variable	performance_schema_max_cond_instances
Scope	Global
Dynamic	No
Type	Integer
Default Value	-1 (signifies autosizing; do not assign this literal value)

The maximum number of instrumented condition objects. For information about how to set and use this variable, see Section 22.7, “Performance Schema Status Monitoring”.

- performance_schema_max_digest_length

Command-Line Format	--performance-schema-max-digest-length=#
Introduced	5.6.26
System Variable	performance_schema_max_digest_length
Scope	Global
Dynamic	No
Type	Integer
Default Value	1024
Minimum Value	0
Maximum Value	1048576

The maximum number of bytes of memory reserved per statement for computation of normalized statement digest values in the Performance Schema. This variable is related to max_digest_length; see the description of that variable in Section 5.1.7, “Server System Variables”.

For more information about statement digesting, including considerations regarding memory use, see Section 22.10, “Performance Schema Statement Digests”.

- `performance_schema_max_file_classes`

Command-Line Format	`--performance-schema-max-file-classes=#`
System Variable	`performance_schema_max_file_classes`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`50`
Minimum Value	`0`
Maximum Value	`256`

The maximum number of file instruments. For information about how to set and use this variable, see Section 22.7, “Performance Schema Status Monitoring”.

- `performance_schema_max_file_handles`

Command-Line Format	`--performance-schema-max-file-handles=#`
System Variable	`performance_schema_max_file_handles`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`32768`

The maximum number of opened file objects. For information about how to set and use this variable, see Section 22.7, “Performance Schema Status Monitoring”.

The value of `performance_schema_max_file_handles` should be greater than the value of `open_files_limit`: `open_files_limit` affects the maximum number of open file handles the server can support and `performance_schema_max_file_handles` affects how many of these file handles can be instrumented.

- `performance_schema_max_file_instances`

Command-Line Format	`--performance-schema-max-file-instances=#`
System Variable	`performance_schema_max_file_instances`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)

The maximum number of instrumented file objects. For information about how to set and use this variable, see Section 22.7, “Performance Schema Status Monitoring”.

- `performance_schema_max_mutex_classes`

Command-Line Format	`--performance-schema-max-mutex-classes=#`
System Variable	`performance_schema_max_mutex_classes`
Scope	Global
Dynamic	No
Type	Integer

Default Value	200
Minimum Value	0
Maximum Value	256

The maximum number of mutex instruments. For information about how to set and use this variable, see Section 22.7, “Performance Schema Status Monitoring”.

- performance_schema_max_mutex_instances

Command-Line Format	--performance-schema-max-mutex-instances=#
System Variable	performance_schema_max_mutex_instances
Scope	Global
Dynamic	No
Type	Integer
Default Value	-1 (signifies autosizing; do not assign this literal value)

The maximum number of instrumented mutex objects. For information about how to set and use this variable, see Section 22.7, “Performance Schema Status Monitoring”.

- performance_schema_max_rwlock_classes

Command-Line Format	--performance-schema-max-rwlock-classes=#
System Variable	performance_schema_max_rwlock_classes
Scope	Global
Dynamic	No
Type	Integer
Default Value (≥ 5.6.15)	40
Default Value (≤ 5.6.14)	30
Minimum Value	0
Maximum Value	256

The maximum number of rwlock instruments. For information about how to set and use this variable, see Section 22.7, “Performance Schema Status Monitoring”.

- performance_schema_max_rwlock_instances

Command-Line Format	--performance-schema-max-rwlock-instances=#
System Variable	performance_schema_max_rwlock_instances
Scope	Global
Dynamic	No
Type	Integer
Default Value	-1 (signifies autosizing; do not assign this literal value)

The maximum number of instrumented rwlock objects. For information about how to set and use this variable, see Section 22.7, “Performance Schema Status Monitoring”.

- performance_schema_max_socket_classes

Command-Line Format	--performance-schema-max-socket-classes=#
System Variable	performance_schema_max_socket_classes

Scope	Global
Dynamic	No
Type	Integer
Default Value	`10`
Minimum Value	`0`
Maximum Value	`256`

The maximum number of socket instruments. For information about how to set and use this variable, see Section 22.7, “Performance Schema Status Monitoring”.

- `performance_schema_max_socket_instances`

Command-Line Format	`--performance-schema-max-socket-instances=#`
System Variable	`performance_schema_max_socket_instances`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)

The maximum number of instrumented socket objects. For information about how to set and use this variable, see Section 22.7, “Performance Schema Status Monitoring”.

- `performance_schema_max_stage_classes`

Command-Line Format	`--performance-schema-max-stage-classes=#`
System Variable	`performance_schema_max_stage_classes`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`150`
Minimum Value	`0`
Maximum Value	`256`

The maximum number of stage instruments. For information about how to set and use this variable, see Section 22.7, “Performance Schema Status Monitoring”.

- `performance_schema_max_statement_classes`

Command-Line Format	`--performance-schema-max-statement-classes=#`
System Variable	`performance_schema_max_statement_classes`
Scope	Global
Dynamic	No
Type	Integer

Default Value	`-1` (signifies autosizing; do not assign this literal value)

The maximum number of statement instruments. For information about how to set and use this variable, see Section 22.7, “Performance Schema Status Monitoring”.

The default value is calculated at server build time based on the number of commands in the client/server protocol and the number of SQL statement types supported by the server.

This variable should not be changed, unless to set it to 0 to disable all statement instrumentation and save all memory associated with it. Setting the variable to nonzero values other than the default has no benefit; in particular, values larger than the default cause more memory to be allocated then is needed.

- `performance_schema_max_table_handles`

Command-Line Format	`--performance-schema-max-table-handles=#`
System Variable	`performance_schema_max_table_handles`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)

The maximum number of opened table objects. For information about how to set and use this variable, see Section 22.7, “Performance Schema Status Monitoring”.

- `performance_schema_max_table_instances`

Command-Line Format	`--performance-schema-max-table-instances=#`
System Variable	`performance_schema_max_table_instances`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)

The maximum number of instrumented table objects. For information about how to set and use this variable, see Section 22.7, “Performance Schema Status Monitoring”.

- `performance_schema_max_thread_classes`

Command-Line Format	`--performance-schema-max-thread-classes=#`
System Variable	`performance_schema_max_thread_classes`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`50`
Minimum Value	`0`
Maximum Value	`256`

The maximum number of thread instruments. For information about how to set and use this variable, see Section 22.7, “Performance Schema Status Monitoring”.

- `performance_schema_max_thread_instances`

Command-Line Format	`--performance-schema-max-thread-instances=#`
System Variable	`performance_schema_max_thread_instances`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)

The maximum number of instrumented thread objects. The value controls the size of the `threads` table. If this maximum is exceeded such that a thread cannot be instrumented, the Performance Schema increments the `Performance_schema_thread_instances_lost` status variable. For information about how to set and use this variable, see Section 22.7, "Performance Schema Status Monitoring".

The `max_connections` system variable affects how many threads are run in the server. `performance_schema_max_thread_instances` affects how many of these running threads can be instrumented. The default value of `performance_schema_max_thread_instances` is autosized based on the value of `max_connections`.

- `performance_schema_session_connect_attrs_size`

Command-Line Format	`--performance-schema-session-connect-attrs-size=#`
System Variable	`performance_schema_session_connect_attrs_size`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`-1` (signifies autosizing; do not assign this literal value)
Minimum Value	`-1`
Maximum Value	`1048576`

The amount of preallocated memory per thread reserved to hold connection attribute key-value pairs. If the aggregate size of connection attribute data sent by a client is larger than this amount, the Performance Schema truncates the attribute data, increments the `Performance_schema_session_connect_attrs_lost` status variable, and writes a message to the error log indicating that truncation occurred if the `log_warnings` system variable value is greater than zero.

The default value of `performance_schema_session_connect_attrs_size` is autosized at server startup. This value may be small, so if truncation occurs (`Performance_schema_session_connect_attrs_lost` becomes nonzero), you may wish to set `performance_schema_session_connect_attrs_size` explicitly to a larger value.

Although the maximum permitted `performance_schema_session_connect_attrs_size` value is 1MB, the effective maximum is 64KB because the server imposes a limit of 64KB on the aggregate size of connection attribute data it can accept. If a client attempts to send more than 64KB of attribute data, the server rejects the connection. For more information, see Section 22.12.8, "Performance Schema Connection Attribute Tables".

- `performance_schema_setup_actors_size`

Command-Line Format	`--performance-schema-setup-actors-size=#`
System Variable	`performance_schema_setup_actors_size`

Scope	Global
Dynamic	No
Type	Integer
Default Value	100

The number of rows in the setup_actors table.

- performance_schema_setup_objects_size

Command-Line Format	--performance-schema-setup-objects-size=#
System Variable	performance_schema_setup_objects_size
Scope	Global
Dynamic	No
Type	Integer
Default Value	100

The number of rows in the setup_objects table.

- performance_schema_users_size

Command-Line Format	--performance-schema-users-size=#
System Variable	performance_schema_users_size
Scope	Global
Dynamic	No
Type	Integer
Default Value	-1 (signifies autosizing; do not assign this literal value)
Minimum Value	-1 (signifies autosizing; do not assign this literal value)
Maximum Value	1048576

The number of rows in the users table. If this variable is 0, the Performance Schema does not maintain connection statistics in the users table.

22.16 Performance Schema Status Variables

The Performance Schema implements several status variables that provide information about instrumentation that could not be loaded or created due to memory constraints:

```
mysql> SHOW STATUS LIKE 'perf%';
+-------------------------------------------+-------+
| Variable_name                             | Value |
+-------------------------------------------+-------+
| Performance_schema_accounts_lost          | 0     |
| Performance_schema_cond_classes_lost      | 0     |
| Performance_schema_cond_instances_lost    | 0     |
| Performance_schema_file_classes_lost      | 0     |
| Performance_schema_file_handles_lost      | 0     |
| Performance_schema_file_instances_lost    | 0     |
| Performance_schema_hosts_lost             | 0     |
| Performance_schema_locker_lost            | 0     |
| Performance_schema_mutex_classes_lost     | 0     |
| Performance_schema_mutex_instances_lost   | 0     |
| Performance_schema_rwlock_classes_lost    | 0     |
| Performance_schema_rwlock_instances_lost  | 0     |
| Performance_schema_socket_classes_lost    | 0     |
| Performance_schema_socket_instances_lost  | 0     |
| Performance_schema_stage_classes_lost     | 0     |
```

```
| Performance_schema_statement_classes_lost | 0     |
| Performance_schema_table_handles_lost     | 0     |
| Performance_schema_table_instances_lost   | 0     |
| Performance_schema_thread_classes_lost    | 0     |
| Performance_schema_thread_instances_lost  | 0     |
| Performance_schema_users_lost             | 0     |
+-------------------------------------------+-------+
```

For information on using these variables to check Performance Schema status, see Section 22.7, “Performance Schema Status Monitoring”.

Performance Schema status variables have the following meanings:

- `Performance_schema_accounts_lost`

  The number of times a row could not be added to the `accounts` table because it was full.

- `Performance_schema_cond_classes_lost`

  How many condition instruments could not be loaded.

- `Performance_schema_cond_instances_lost`

  How many condition instrument instances could not be created.

- `Performance_schema_digest_lost`

  The number of digest instances that could not be instrumented in the `events_statements_summary_by_digest` table. This can be nonzero if the value of `performance_schema_digests_size` is too small.

- `Performance_schema_file_classes_lost`

  How many file instruments could not be loaded.

- `Performance_schema_file_handles_lost`

  How many file instrument instances could not be opened.

- `Performance_schema_file_instances_lost`

  How many file instrument instances could not be created.

- `Performance_schema_hosts_lost`

  The number of times a row could not be added to the `hosts` table because it was full.

- `Performance_schema_locker_lost`

  How many events are “lost” or not recorded, due to the following conditions:

  - Events are recursive (for example, waiting for A caused a wait on B, which caused a wait on C).
  - The depth of the nested events stack is greater than the limit imposed by the implementation.

  Events recorded by the Performance Schema are not recursive, so this variable should always be 0.

- `Performance_schema_mutex_classes_lost`

  How many mutex instruments could not be loaded.

- `Performance_schema_mutex_instances_lost`

  How many mutex instrument instances could not be created.

- `Performance_schema_rwlock_classes_lost`

  How many rwlock instruments could not be loaded.

- `Performance_schema_rwlock_instances_lost`

  How many rwlock instrument instances could not be created.

- `Performance_schema_session_connect_attrs_lost`

  The number of connections for which connection attribute truncation has occurred. For a given connection, if the client sends connection attribute key-value pairs for which the aggregate size is larger than the reserved storage permitted by the value of the `performance_schema_session_connect_attrs_size` system variable, the Performance Schema truncates the attribute data and increments `Performance_schema_session_connect_attrs_lost`. If this value is nonzero, you may wish to set `performance_schema_session_connect_attrs_size` to a larger value.

  For more information about connection attributes, see Section 22.12.8, “Performance Schema Connection Attribute Tables”.

- `Performance_schema_socket_classes_lost`

  How many socket instruments could not be loaded.

- `Performance_schema_socket_instances_lost`

  How many socket instrument instances could not be created.

- `Performance_schema_stage_classes_lost`

  How many stage instruments could not be loaded.

- `Performance_schema_statement_classes_lost`

  How many statement instruments could not be loaded.

- `Performance_schema_table_handles_lost`

  How many table instrument instances could not be opened.

- `Performance_schema_table_instances_lost`

  How many table instrument instances could not be created.

- `Performance_schema_thread_classes_lost`

  How many thread instruments could not be loaded.

- `Performance_schema_thread_instances_lost`

  The number of thread instances that could not be instrumented in the `threads` table. This can be nonzero if the value of `performance_schema_max_thread_instances` is too small.

- `Performance_schema_users_lost`

  The number of times a row could not be added to the `users` table because it was full.

22.17 Performance Schema and Plugins

Removing a plugin with `UNINSTALL PLUGIN` does not affect information already collected for code in that plugin. Time spent executing the code while the plugin was loaded was still spent even if the

plugin is unloaded later. The associated event information, including aggregate information, remains readable in `performance_schema` database tables. For additional information about the effect of plugin installation and removal, see Section 22.7, “Performance Schema Status Monitoring”.

A plugin implementor who instruments plugin code should document its instrumentation characteristics to enable those who load the plugin to account for its requirements. For example, a third-party storage engine should include in its documentation how much memory the engine needs for mutex and other instruments.

22.18 Using the Performance Schema to Diagnose Problems

The Performance Schema is a tool to help a DBA do performance tuning by taking real measurements instead of “wild guesses.” This section demonstrates some ways to use the Performance Schema for this purpose. The discussion here relies on the use of event filtering, which is described in Section 22.4.2, “Performance Schema Event Filtering”.

The following example provides one methodology that you can use to analyze a repeatable problem, such as investigating a performance bottleneck. To begin, you should have a repeatable use case where performance is deemed “too slow” and needs optimization, and you should enable all instrumentation (no pre-filtering at all).

1. Run the use case.
2. Using the Performance Schema tables, analyze the root cause of the performance problem. This analysis relies heavily on post-filtering.
3. For problem areas that are ruled out, disable the corresponding instruments. For example, if analysis shows that the issue is not related to file I/O in a particular storage engine, disable the file I/O instruments for that engine. Then truncate the history and summary tables to remove previously collected events.
4. Repeat the process at step 1.

   At each iteration, the Performance Schema output, particularly the `events_waits_history_long` table, contains less and less “noise” caused by nonsignificant instruments, and given that this table has a fixed size, contains more and more data relevant to the analysis of the problem at hand.

   At each iteration, investigation should lead closer and closer to the root cause of the problem, as the signal-to-noise ratio improves, making analysis easier.
5. Once a root cause of performance bottleneck is identified, take the appropriate corrective action, such as:
   - Tune the server parameters (cache sizes, memory, and so forth).
   - Tune a query by writing it differently,
   - Tune the database schema (tables, indexes, and so forth).
   - Tune the code (this applies to storage engine or server developers only).
6. Start again at step 1, to see the effects of the changes on performance.

The `mutex_instances.LOCKED_BY_THREAD_ID` and `rwlock_instances.WRITE_LOCKED_BY_THREAD_ID` columns are extremely important for investigating performance bottlenecks or deadlocks. This is made possible by Performance Schema instrumentation as follows:

1. Suppose that thread 1 is stuck waiting for a mutex.

2. You can determine what the thread is waiting for:

```
SELECT * FROM performance_schema.events_waits_current
WHERE THREAD_ID = thread_1;
```

   Say the query result identifies that the thread is waiting for mutex A, found in `events_waits_current.OBJECT_INSTANCE_BEGIN`.

3. You can determine which thread is holding mutex A:

```
SELECT * FROM performance_schema.mutex_instances
WHERE OBJECT_INSTANCE_BEGIN = mutex_A;
```

   Say the query result identifies that it is thread 2 holding mutex A, as found in `mutex_instances.LOCKED_BY_THREAD_ID`.

4. You can see what thread 2 is doing:

```
SELECT * FROM performance_schema.events_waits_current
WHERE THREAD_ID = thread_2;
```

22.18.1 Query Profiling Using Performance Schema

The following example demonstrates how to use Performance Schema statement events and stage events to retrieve data comparable to profiling information provided by `SHOW PROFILES` and `SHOW PROFILE` statements.

In this example, statement and stage event data is collected in the `events_statements_history_long` and `events_stages_history_long` tables. On a busy server with many active foreground threads, data could age out of the history tables before you are able to retrieve the information you want to analyze. If you encounter this problem, options include:

- Running the query on a test instance where there is less foreground thread activity.
- Disabling instrumentation for other existing foreground threads by setting the `INSTRUMENTED` field of the `threads` table to `NO` for other thread records. For example, the following statement disables instrumentation for all foreground threads except the `test_user` thread:

```
mysql> UPDATE performance_schema.threads
       SET INSTRUMENTED = 'NO'
       WHERE TYPE='FOREGROUND' AND PROCESSLIST_USER NOT LIKE 'test_user';
```

  However, be aware that new threads are always instrumented by default.

- Increasing the number of rows in the `events_statements_history_long` and `events_stages_history_long` tables. The `performance_schema_events_statements_history_size` and `performance_schema_events_stages_history_size` configuration options are autosized by default but can also be set explicitly at startup. You can view current settings by running `SHOW VARIABLES`. For information about autosized Performance Schema parameters, see Section 22.3, “Performance Schema Startup Configuration”.

Performance Schema displays event timer information in picoseconds (trillionths of a second) to normalize timing data to a standard unit. In the following example, `TIMER_WAIT` values are divided by 1000000000000 to show data in units of seconds. Values are also truncated to 6 decimal places to display data in the same format as `SHOW PROFILES` and `SHOW PROFILE` statements.

1. Ensure that statement and stage instrumentation is enabled by updating the `setup_instruments` table. Some instruments may already be enabled by default.

```
mysql> UPDATE performance_schema.setup_instruments
       SET ENABLED = 'YES', TIMED = 'YES'
       WHERE NAME LIKE '%statement/%';
```

```
mysql> UPDATE performance_schema.setup_instruments
       SET ENABLED = 'YES', TIMED = 'YES'
       WHERE NAME LIKE '%stage/%';
```

2. Ensure that `events_statements_*` and `events_stages_*` consumers are enabled. Some consumers may already be enabled by default.

```
mysql> UPDATE performance_schema.setup_consumers
       SET ENABLED = 'YES'
       WHERE NAME LIKE '%events_statements_%';

mysql> UPDATE performance_schema.setup_consumers
       SET ENABLED = 'YES'
       WHERE NAME LIKE '%events_stages_%';
```

3. Run the statement that you want to profile. For example:

```
mysql> SELECT * FROM employees.employees WHERE emp_no = 10001;
+--------+------------+------------+-----------+--------+------------+
| emp_no | birth_date | first_name | last_name | gender | hire_date |
+--------+------------+------------+-----------+--------+------------+
|  10001 | 1953-09-02 | Georgi     | Facello   | M      | 1986-06-26 |
+--------+------------+------------+-----------+--------+------------+
```

4. Identify the `EVENT_ID` of the statement by querying the `events_statements_history_long` table. This step is similar to running `SHOW PROFILES` to identify the `Query_ID`. The following query produces output similar to `SHOW PROFILES`:

```
mysql> SELECT EVENT_ID, TRUNCATE(TIMER_WAIT/1000000000000,6) as Duration, SQL_TEXT
       FROM performance_schema.events_statements_history_long WHERE SQL_TEXT like '%10001%';
+----------+----------+-----------------------------------------------------------+
| event_id | duration | sql_text                                                  |
+----------+----------+-----------------------------------------------------------+
|       31 | 0.028310 | SELECT * FROM employees.employees WHERE emp_no = 10001 |
+----------+----------+-----------------------------------------------------------+
```

5. Query the `events_stages_history_long` table to retrieve the statement's stage events. Stages are linked to statements using event nesting. Each stage event record has a `NESTING_EVENT_ID` column that contains the `EVENT_ID` of the parent statement.

```
mysql> SELECT event_name AS Stage, TRUNCATE(TIMER_WAIT/1000000000000,6) AS Duration
       FROM performance_schema.events_stages_history_long WHERE NESTING_EVENT_ID=31;
+--------------------------------+----------+
| Stage                          | Duration |
+--------------------------------+----------+
| stage/sql/starting             | 0.000080 |
| stage/sql/checking permissions | 0.000005 |
| stage/sql/Opening tables       | 0.027759 |
| stage/sql/init                 | 0.000052 |
| stage/sql/System lock          | 0.000009 |
| stage/sql/optimizing           | 0.000006 |
| stage/sql/statistics           | 0.000082 |
| stage/sql/preparing            | 0.000008 |
| stage/sql/executing            | 0.000000 |
| stage/sql/Sending data         | 0.000017 |
| stage/sql/end                  | 0.000001 |
| stage/sql/query end            | 0.000004 |
| stage/sql/closing tables       | 0.000006 |
| stage/sql/freeing items        | 0.000272 |
| stage/sql/cleaning up          | 0.000001 |
+--------------------------------+----------+
```

22.19 Restrictions on Performance Schema

The Performance Schema avoids using mutexes to collect or produce data, so there are no guarantees of consistency and results can sometimes be incorrect. Event values in `performance_schema` tables are nondeterministic and nonrepeatable.

If you save event information in another table, you should not assume that the original events remain available later. For example, if you select events from a `performance_schema` table into a temporary table, intending to join that table with the original table later, there might be no matches.

`mysqldump` and `BACKUP DATABASE` ignore tables in the `performance_schema` database.

Tables in the `performance_schema` database cannot be locked with `LOCK TABLES`, except the `setup_xxx` tables.

Tables in the `performance_schema` database cannot be indexed.

Results for queries that refer to tables in the `performance_schema` database are not saved in the query cache.

Tables in the `performance_schema` database are not replicated.

The Performance Schema is not available in `libmysqld`, the embedded server.

The types of timers might vary per platform. The `performance_timers` table shows which event timers are available. If the values in this table for a given timer name are `NULL`, that timer is not supported on your platform.

Instruments that apply to storage engines might not be implemented for all storage engines. Instrumentation of each third-party engine is the responsibility of the engine maintainer.

Chapter 23 Connectors and APIs

Table of Contents

23.1 MySQL Connector/C++ 3179
23.2 MySQL Connector/J 3179
23.3 MySQL Connector/NET 3180
23.4 MySQL Connector/ODBC 3180
23.5 MySQL Connector/Python 3180
23.6 libmysqld, the Embedded MySQL Server Library 3180
    23.6.1 Compiling Programs with libmysqld 3181
    23.6.2 Restrictions When Using the Embedded MySQL Server 3181
    23.6.3 Options with the Embedded Server 3182
    23.6.4 Embedded Server Examples 3182
23.7 MySQL C API 3185
23.8 MySQL PHP API 3185
23.9 MySQL Perl API 3185
23.10 MySQL Python API 3186
23.11 MySQL Ruby APIs 3186
    23.11.1 The MySQL/Ruby API 3187
    23.11.2 The Ruby/MySQL API 3187
23.12 MySQL Tcl API 3187
23.13 MySQL Eiffel Wrapper 3187

MySQL Connectors provide connectivity to the MySQL server for client programs. APIs provide low-level access to the MySQL protocol and MySQL resources. Both Connectors and the APIs enable you to connect and execute MySQL statements from another language or environment, including ODBC, Java (JDBC), Perl, Python, PHP, Ruby, and native C and embedded MySQL instances.

MySQL Connectors

Oracle develops a number of connectors:

- Connector/C++ enables C++ applications to connect to MySQL.
- Connector/J provides driver support for connecting to MySQL from Java applications using the standard Java Database Connectivity (JDBC) API.
- Connector/NET enables developers to create .NET applications that connect to MySQL. Connector/NET implements a fully functional ADO.NET interface and provides support for use with ADO.NET aware tools. Applications that use Connector/NET can be written in any supported .NET language.

  MySQL for Visual Studio works with Connector/NET and Microsoft Visual Studio 2012, 2013, 2015, and 2017. MySQL for Visual Studio provides access to MySQL objects and data from Visual Studio. As a Visual Studio package, it integrates directly into Server Explorer providing the ability to create new connections and work with MySQL database objects.
- Connector/ODBC provides driver support for connecting to MySQL using the Open Database Connectivity (ODBC) API. Support is available for ODBC connectivity from Windows, Unix, and macOS platforms.
- Connector/Python provides driver support for connecting to MySQL from Python applications using an API that is compliant with the Python DB API version 2.0. No additional Python modules or MySQL client libraries are required.

The MySQL C API

For direct access to using MySQL natively within a C application, there are two methods:

- The C API provides low-level access to the MySQL client/server protocol through the `libmysqlclient` client library. This is the primary method used to connect to an instance of the MySQL server, and is used both by MySQL command-line clients and many of the MySQL Connectors and third-party APIs detailed here.

  `libmysqlclient` is included in MySQL distributions.

- `libmysqld` is an embedded MySQL server library that enables you to embed an instance of the MySQL server into your C applications.

  `libmysqld` is included in MySQL distributions.

See also MySQL C API Implementations.

To access MySQL from a C application, or to build an interface to MySQL for a language not supported by the Connectors or APIs in this chapter, the C API is where to start. A number of programmer's utilities are available to help with the process; see Section 4.7, “Program Development Utilities”.

Third-Party MySQL APIs

The remaining APIs described in this chapter provide an interface to MySQL from specific application languages. These third-party solutions are not developed or supported by Oracle. Basic information on their usage and abilities is provided here for reference purposes only.

All the third-party language APIs are developed using one of two methods, using `libmysqlclient` or by implementing a *native driver*. The two solutions offer different benefits:

- Using *`libmysqlclient`* offers complete compatibility with MySQL because it uses the same libraries as the MySQL client applications. However, the feature set is limited to the implementation and interfaces exposed through `libmysqlclient` and the performance may be lower as data is copied between the native language, and the MySQL API components.

- *Native drivers* are an implementation of the MySQL network protocol entirely within the host language or environment. Native drivers are fast, as there is less copying of data between components, and they can offer advanced functionality not available through the standard MySQL API. Native drivers are also easier for end users to build and deploy because no copy of the MySQL client libraries is needed to build the native driver components.

Table 23.1, “MySQL APIs and Interfaces” lists many of the libraries and interfaces available for MySQL.

Table 23.1 MySQL APIs and Interfaces

Environment	API	Type	Notes
Ada	GNU Ada MySQL Bindings	`libmysqlclient`	See MySQL Bindings for GNU Ada
C	C API	`libmysqlclient`	See MySQL 5.6 C API Developer Guide.
C++	Connector/C++	`libmysqlclient`	See MySQL Connector/C++ 8.0 Developer Guide.
	MySQL++	`libmysqlclient`	See MySQL++ website.
	MySQL wrapped	`libmysqlclient`	See MySQL wrapped.
Cocoa	MySQL-Cocoa	`libmysqlclient`	Compatible with the Objective-C Cocoa environment. See http://mysql-cocoa.sourceforge.net/
D	MySQL for D	`libmysqlclient`	See MySQL for D.
Eiffel	Eiffel MySQL	`libmysqlclient`	See Section 23.13, “MySQL Eiffel Wrapper”.

Environment	API	Type	Notes
Erlang	`erlang-mysql-driver`	`libmysqlclient`	See `erlang-mysql-driver`.
Haskell	Haskell MySQL Bindings	Native Driver	See Brian O'Sullivan's pure Haskell MySQL bindings.
	`hsql-mysql`	`libmysqlclient`	See MySQL driver for Haskell.
Java/ JDBC	Connector/J	Native Driver	See MySQL Connector/J 5.1 Developer Guide.
Kaya	MyDB	`libmysqlclient`	See MyDB.
Lua	LuaSQL	`libmysqlclient`	See LuaSQL.
.NET/ Mono	Connector/NET	Native Driver	See MySQL Connector/NET Developer Guide.
Objective Caml	OBjective Caml MySQL Bindings	`libmysqlclient`	See MySQL Bindings for Objective Caml.
Octave	Database bindings for GNU Octave	`libmysqlclient`	See Database bindings for GNU Octave.
ODBC	Connector/ODBC	`libmysqlclient`	See MySQL Connector/ODBC Developer Guide.
Perl	`DBI`/`DBD::mysql`	`libmysqlclient`	See Section 23.9, “MySQL Perl API”.
	`Net::MySQL`	Native Driver	See `Net::MySQL` at CPAN
PHP	`mysql`, `ext/mysql` interface (deprecated)	`libmysqlclient`	See Original MySQL API.
	`mysqli`, `ext/mysqli` interface	`libmysqlclient`	See MySQL Improved Extension.
	`PDO_MYSQL`	`libmysqlclient`	See MySQL Functions (PDO_MYSQL).
	PDO mysqlnd	Native Driver	
Python	Connector/Python	Native Driver	See MySQL Connector/Python Developer Guide.
Python	Connector/Python C Extension	`libmysqlclient`	See MySQL Connector/Python Developer Guide.
	MySQLdb	`libmysqlclient`	See Section 23.10, “MySQL Python API”.
Ruby	MySQL/Ruby	`libmysqlclient`	Uses `libmysqlclient`. See Section 23.11.1, “The MySQL/Ruby API”.
	Ruby/MySQL	Native Driver	See Section 23.11.2, “The Ruby/ MySQL API”.
Scheme	`Myscsh`	`libmysqlclient`	See `Myscsh`.
SPL	`sql_mysql`	`libmysqlclient`	See `sql_mysql` for SPL.
Tcl	MySQLtcl	`libmysqlclient`	See Section 23.12, “MySQL Tcl API”.

23.1 MySQL Connector/C++

The MySQL Connector/C++ manual is published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/C++ 8.0 Developer Guide
- Release notes: MySQL Connector/C++ Release Notes

23.2 MySQL Connector/J

The MySQL Connector/J manual is published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/J Developer Guide
- Release notes: MySQL Connector/J Release Notes

23.3 MySQL Connector/NET

The MySQL Connector/NET manual is published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/NET Developer Guide
- Release notes: MySQL Connector/NET Release Notes

23.4 MySQL Connector/ODBC

The MySQL Connector/ODBC manual is published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/ODBC Developer Guide
- Release notes: MySQL Connector/ODBC Release Notes

23.5 MySQL Connector/Python

The MySQL Connector/Python manual is published in standalone form, not as part of the MySQL Reference Manual. For information, see these documents:

- Main manual: MySQL Connector/Python Developer Guide
- Release notes: MySQL Connector/Python Release Notes

23.6 libmysqld, the Embedded MySQL Server Library

The embedded MySQL server library makes it possible to run a full-featured MySQL server inside a client application. The main benefits are increased speed and more simple management for embedded applications.

The embedded server library is based on the client/server version of MySQL, which is written in C/C++. Consequently, the embedded server also is written in C/C++. There is no embedded server available in other languages.

The API is identical for the embedded MySQL version and the client/server version. To change a threaded application to use the embedded library, you normally only have to add calls to the following functions.

Table 23.2 MySQL Embedded Server Library Functions

Function	When to Call
`mysql_library_init()`	Call it before any other MySQL function is called, preferably early in the `main()` function.
`mysql_library_end()`	Call it before your program exits.
`mysql_thread_init()`	Call it in each thread you create that accesses MySQL.
`mysql_thread_end()`	Call it before calling `pthread_exit()`.

Then, link your code with `libmysqld.a` instead of `libmysqlclient.a`. To ensure binary compatibility between your application and the server library, always compile your application against headers for the same series of MySQL that was used to compile the server library. For example,

if `libmysqld` was compiled against MySQL 5.5 headers, do not compile your application against MySQL 5.6 headers, or vice versa.

Because the `mysql_library_xxx()` functions are also included in `libmysqlclient.a`, you can change between the embedded and the client/server version by just linking your application with the right library. See mysql_library_init().

One difference between the embedded server and the standalone server is that for the embedded server, authentication for connections is disabled by default.

23.6.1 Compiling Programs with libmysqld

In precompiled binary MySQL distributions that include `libmysqld`, the embedded server library, MySQL builds the library using the appropriate vendor compiler if there is one.

To get a `libmysqld` library if you build MySQL from source yourself, you should configure MySQL with the `-DWITH_EMBEDDED_SERVER=1` option. See Section 2.9.7, “MySQL Source-Configuration Options”.

When you link your program with `libmysqld`, you must also include the system-specific `pthread` libraries and some libraries that the MySQL server uses. You can get the full list of libraries by executing `mysql_config --libmysqld-libs`.

The correct flags for compiling and linking a threaded program must be used, even if you do not directly call any thread functions in your code.

To compile a C program to include the necessary files to embed the MySQL server library into an executable version of a program, the compiler needs to know where to find various files and needs instructions on how to compile the program. The following example shows how a program could be compiled from the command line, assuming that you are using `gcc`, use the GNU C compiler:

```
gcc mysql_test.c -o mysql_test \
`/usr/local/mysql/bin/mysql_config --include --libmysqld-libs`
```

Immediately following the `gcc` command is the name of the C program source file. After it, the `-o` option is given to indicate that the file name that follows is the name that the compiler is to give to the output file, the compiled program. The next line of code tells the compiler to obtain the location of the include files and libraries and other settings for the system on which it is compiled. The `mysql_config` command is contained in backticks, not single quotation marks.

On some non-`gcc` platforms, the embedded library depends on C++ runtime libraries and linking against the embedded library might result in missing-symbol errors. To solve this, link using a C++ compiler or explicitly list the required libraries on the link command line.

23.6.2 Restrictions When Using the Embedded MySQL Server

The embedded server has the following limitations:

- No user-defined functions (UDFs).
- No stack trace on core dump.
- You cannot set this up as a source or a replica (no replication).
- Very large result sets may be unusable on low memory systems.
- You cannot connect to an embedded server from an outside process with sockets or TCP/IP. However, you can connect to an intermediate application, which in turn can connect to an embedded server on the behalf of a remote client or outside process.
- `libmysqld` does not support encrypted connections. An implication is that if an application linked against `libmysqld` establishes a connection to a remote server, the connection cannot be encrypted.

- `InnoDB` is not reentrant in the embedded server and cannot be used for multiple connections, either successively or simultaneously.
- The Event Scheduler is not available. Because of this, the `event_scheduler` system variable is disabled.
- The Performance Schema is not available.

Some of these limitations can be changed by editing the `mysql_embed.h` include file and recompiling MySQL.

23.6.3 Options with the Embedded Server

Any options that may be given with the `mysqld` server daemon, may be used with an embedded server library. Server options may be given in an array as an argument to the `mysql_library_init()`, which initializes the server. They also may be given in an option file like `my.cnf`. To specify an option file for a C program, use the `--defaults-file` option as one of the elements of the second argument of the `mysql_library_init()` function. See mysql_library_init(), for more information on the `mysql_library_init()` function.

Using option files can make it easier to switch between a client/server application and one where MySQL is embedded. Put common options under the `[server]` group. These are read by both MySQL versions. Client/server-specific options should go under the `[mysqld]` section. Put options specific to the embedded MySQL server library in the `[embedded]` section. Options specific to applications go under section labeled `[ApplicationName_SERVER]`. See Section 4.2.2.2, “Using Option Files”.

23.6.4 Embedded Server Examples

These two example programs should work without any changes on a Linux or FreeBSD system. For other operating systems, minor changes are needed, mostly with file paths. These examples are designed to give enough details for you to understand the problem, without the clutter that is a necessary part of a real application. The first example is very straightforward. The second example is a little more advanced with some error checking. The first is followed by a command-line entry for compiling the program. The second is followed by a GNUmake file that may be used for compiling instead.

Example 1

`test1_libmysqld.c`

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include "mysql.h"

MYSQL *mysql;
MYSQL_RES *results;
MYSQL_ROW record;

static char *server_options[] = \
       { "mysql_test", "--defaults-file=my.cnf", NULL };
int num_elements = (sizeof(server_options) / sizeof(char *)) - 1;

static char *server_groups[] = { "libmysqld_server",
                                 "libmysqld_client", NULL };

int main(void)
{
   mysql_library_init(num_elements, server_options, server_groups);
   mysql = mysql_init(NULL);
   mysql_options(mysql, MYSQL_READ_DEFAULT_GROUP, "libmysqld_client");
   mysql_options(mysql, MYSQL_OPT_USE_EMBEDDED_CONNECTION, NULL);

   mysql_real_connect(mysql, NULL,NULL,NULL, "database1", 0,NULL,0);
```

```
   mysql_query(mysql, "SELECT column1, column2 FROM table1");

   results = mysql_store_result(mysql);

   while((record = mysql_fetch_row(results))) {
      printf("%s - %s \n", record[0], record[1]);
   }

   mysql_free_result(results);
   mysql_close(mysql);
   mysql_library_end();

   return 0;
}
```

Here is the command line for compiling the above program:

```
gcc test1_libmysqld.c -o test1_libmysqld \
 `/usr/local/mysql/bin/mysql_config --include --libmysqld-libs`
```

Example 2

To try the example, create an `test2_libmysqld` directory at the same level as the MySQL source directory. Save the `test2_libmysqld.c` source and the `GNUmakefile` in the directory, and run GNU `make` from inside the `test2_libmysqld` directory.

`test2_libmysqld.c`

```
/*
 * A simple example client, using the embedded MySQL server library
*/

#include <mysql.h>
#include <stdarg.h>
#include <stdio.h>
#include <stdlib.h>

MYSQL *db_connect(const char *dbname);
void db_disconnect(MYSQL *db);
void db_do_query(MYSQL *db, const char *query);

const char *server_groups[] = {
  "test2_libmysqld_SERVER", "embedded", "server", NULL
};

int
main(int argc, char **argv)
{
  MYSQL *one, *two;

  /* mysql_library_init() must be called before any other mysql
   * functions.
   *
   * You can use mysql_library_init(0, NULL, NULL), and it
   * initializes the server using groups = {
   *   "server", "embedded", NULL
   *  }.
   *
   * In your $HOME/.my.cnf file, you probably want to put:

[test2_libmysqld_SERVER]
language = /path/to/source/of/mysql/sql/share/english

   * You could, of course, modify argc and argv before passing
   * them to this function.  Or you could create new ones in any
   * way you like.  But all of the arguments in argv (except for
   * argv[0], which is the program name) should be valid options
   * for the MySQL server.
   *
   * If you link this client against the normal mysqlclient
   * library, this function is just a stub that does nothing.
   */
```

```
  mysql_library_init(argc, argv, (char **)server_groups);

  one = db_connect("test");
  two = db_connect(NULL);

  db_do_query(one, "SHOW TABLE STATUS");
  db_do_query(two, "SHOW DATABASES");

  mysql_close(two);
  mysql_close(one);

  /* This must be called after all other mysql functions */
  mysql_library_end();

  exit(EXIT_SUCCESS);
}

static void
die(MYSQL *db, char *fmt, ...)
{
  va_list ap;
  va_start(ap, fmt);
  vfprintf(stderr, fmt, ap);
  va_end(ap);
  (void)putc('\n', stderr);
  if (db)
    db_disconnect(db);
  exit(EXIT_FAILURE);
}

MYSQL *
db_connect(const char *dbname)
{
  MYSQL *db = mysql_init(NULL);
  if (!db)
    die(db, "mysql_init failed: no memory");
  /*
   * Notice that the client and server use separate group names.
   * This is critical, because the server does not accept the
   * client's options, and vice versa.
   */
  mysql_options(db, MYSQL_READ_DEFAULT_GROUP, "test2_libmysqld_CLIENT");
  if (!mysql_real_connect(db, NULL, NULL, NULL, dbname, 0, NULL, 0))
    die(db, "mysql_real_connect failed: %s", mysql_error(db));

  return db;
}

void
db_disconnect(MYSQL *db)
{
  mysql_close(db);
}

void
db_do_query(MYSQL *db, const char *query)
{
  if (mysql_query(db, query) != 0)
    goto err;

  if (mysql_field_count(db) > 0)
  {
    MYSQL_RES   *res;
    MYSQL_ROW    row, end_row;
    int num_fields;

    if (!(res = mysql_store_result(db)))
      goto err;
    num_fields = mysql_num_fields(res);
    while ((row = mysql_fetch_row(res)))
    {
      (void)fputs(">> ", stdout);
```

```
      for (end_row = row + num_fields; row < end_row; ++row)
        (void)printf("%s\t", row ? (char*)*row : "NULL");
      (void)fputc('\n', stdout);
    }
    (void)fputc('\n', stdout);
    mysql_free_result(res);
  }
  else
    (void)printf("Affected rows: %lld\n", mysql_affected_rows(db));

  return;

err:
  die(db, "db_do_query failed: %s [%s]", mysql_error(db), query);
}
```

GNUmakefile

```
# This assumes the MySQL software is installed in /usr/local/mysql
inc      := /usr/local/mysql/include/mysql
lib      := /usr/local/mysql/lib

# If you have not installed the MySQL software yet, try this instead
#inc      := $(HOME)/mysql-5.6/include
#lib      := $(HOME)/mysql-5.6/libmysqld

CC       := gcc
CPPFLAGS := -I$(inc) -D_THREAD_SAFE -D_REENTRANT
CFLAGS   := -g -W -Wall
LDFLAGS  := -static
# You can change -lmysqld to -lmysqlclient to use the
# client/server library
LDLIBS    = -L$(lib) -lmysqld -lm -ldl -lcrypt

ifneq (,$(shell grep FreeBSD /COPYRIGHT 2>/dev/null))
# FreeBSD
LDFLAGS += -pthread
else
# Assume Linux
LDLIBS += -lpthread
endif

# This works for simple one-file test programs
sources := $(wildcard *.c)
objects := $(patsubst %c,%o,$(sources))
targets := $(basename $(sources))

all: $(targets)

clean:
        rm -f $(targets) $(objects) *.core
```

23.7 MySQL C API

The MySQL C API Developer Guide is published in standalone form, not as part of the MySQL Reference Manual. See MySQL 5.6 C API Developer Guide.

23.8 MySQL PHP API

The MySQL PHP API manual is now published in standalone form, not as part of the MySQL Reference Manual. See MySQL and PHP.

23.9 MySQL Perl API

The Perl `DBI` module provides a generic interface for database access. You can write a DBI script that works with many different database engines without change. To use DBI with MySQL, install the following:

1. The `DBI` module.
2. The `DBD::mysql` module. This is the DataBase Driver (DBD) module for Perl.
3. Optionally, the DBD module for any other type of database server you want to access.

Perl DBI is the recommended Perl interface. It replaces an older interface called `mysqlperl`, which should be considered obsolete.

These sections contain information about using Perl with MySQL and writing MySQL applications in Perl:

- For installation instructions for Perl DBI support, see Section 2.13, “Perl Installation Notes”.
- For an example of reading options from option files, see Section 5.7.4, “Using Client Programs in a Multiple-Server Environment”.
- For secure coding tips, see Section 6.1.1, “Security Guidelines”.
- For debugging tips, see Section 5.8.1.4, “Debugging mysqld under gdb”.
- For some Perl-specific environment variables, see Section 4.9, “Environment Variables”.
- For considerations for running on macOS, see Section 2.4, “Installing MySQL on OS X”.
- For ways to quote string literals, see Section 9.1.1, “String Literals”.

DBI information is available at the command line, online, or in printed form:

- Once you have the `DBI` and `DBD::mysql` modules installed, you can get information about them at the command line with the `perldoc` command:

  ```
  shell> perldoc DBI
  shell> perldoc DBI::FAQ
  shell> perldoc DBD::mysql
  ```

  You can also use `pod2man`, `pod2html`, and so on to translate this information into other formats.

- For online information about Perl DBI, visit the DBI website, http://dbi.perl.org/. That site hosts a general DBI mailing list.
- For printed information, the official DBI book is *Programming the Perl DBI* (Alligator Descartes and Tim Bunce, O'Reilly & Associates, 2000). Information about the book is available at the DBI website, http://dbi.perl.org/.

23.10 MySQL Python API

`MySQLdb` is a third-party driver that provides MySQL support for Python, compliant with the Python DB API version 2.0. It can be found at http://sourceforge.net/projects/mysql-python/.

The new MySQL Connector/Python component provides an interface to the same Python API, and is built into the MySQL Server and supported by Oracle. See MySQL Connector/Python Developer Guide for details on the Connector, as well as coding guidelines for Python applications and sample Python code.

23.11 MySQL Ruby APIs

Two APIs are available for Ruby programmers developing MySQL applications:

- The MySQL/Ruby API is based on the `libmysqlclient` API library. For information on installing and using the MySQL/Ruby API, see Section 23.11.1, “The MySQL/Ruby API”.

- The Ruby/MySQL API is written to use the native MySQL network protocol (a native driver). For information on installing and using the Ruby/MySQL API, see Section 23.11.2, “The Ruby/MySQL API”.

For background and syntax information about the Ruby language, see Ruby Programming Language.

23.11.1 The MySQL/Ruby API

The MySQL/Ruby module provides access to MySQL databases using Ruby through `libmysqlclient`.

For information on installing the module, and the functions exposed, see MySQL/Ruby.

23.11.2 The Ruby/MySQL API

The Ruby/MySQL module provides access to MySQL databases using Ruby through a native driver interface using the MySQL network protocol.

For information on installing the module, and the functions exposed, see Ruby/MySQL.

23.12 MySQL Tcl API

`MySQLtcl` is a simple API for accessing a MySQL database server from the Tcl programming language. It can be found at http://www.xdobry.de/mysqltcl/.

23.13 MySQL Eiffel Wrapper

Eiffel MySQL is an interface to the MySQL database server using the Eiffel programming language, written by Michael Ravits. It can be found at http://efsa.sourceforge.net/archive/ravits/mysql.htm.

Chapter 24 MySQL Enterprise Edition

Table of Contents

24.1 MySQL Enterprise Monitor Overview 3189
24.2 MySQL Enterprise Backup Overview 3190
24.3 MySQL Enterprise Security Overview 3191
24.4 MySQL Enterprise Encryption Overview 3191
24.5 MySQL Enterprise Audit Overview 3191
24.6 MySQL Enterprise Firewall Overview 3192
24.7 MySQL Enterprise Thread Pool Overview 3192
24.8 MySQL Enterprise Data Masking and De-Identification Overview 3192

MySQL Enterprise Edition is a commercial product. Like MySQL Community Edition, MySQL Enterprise Edition includes MySQL Server, a fully integrated transaction-safe, ACID-compliant database with full commit, rollback, crash-recovery, and row-level locking capabilities. In addition, MySQL Enterprise Edition includes the following components designed to provide monitoring and online backup, as well as improved security and scalability:

The following sections briefly discuss each of these components and indicate where to find more detailed information. To learn more about commercial products, see https://www.mysql.com/products/.

- MySQL Enterprise Monitor
- MySQL Enterprise Backup
- MySQL Enterprise Security
- MySQL Enterprise Encryption
- MySQL Enterprise Audit
- MySQL Enterprise Firewall
- MySQL Enterprise Thread Pool
- MySQL Enterprise Data Masking and De-Identification

24.1 MySQL Enterprise Monitor Overview

MySQL Enterprise Monitor is an enterprise monitoring system for MySQL that keeps an eye on your MySQL servers, notifies you of potential issues and problems, and advises you how to fix the issues. MySQL Enterprise Monitor can monitor all kinds of configurations, from a single MySQL server that is important to your business, all the way up to a huge farm of MySQL servers powering a busy website.

The following discussion briefly summarizes the basic components that make up the MySQL Enterprise Monitor product. For more information, see the MySQL Enterprise Monitor manual, available at https://dev.mysql.com/doc/mysql-monitor/en/.

MySQL Enterprise Monitor components can be installed in various configurations depending on your database and network topology, to give you the best combination of reliable and responsive monitoring data, with minimal overhead on the database server machines. A typical MySQL Enterprise Monitor installation consists of:

- One or more MySQL servers to monitor. MySQL Enterprise Monitor can monitor both Community and Enterprise MySQL server releases.
- A MySQL Enterprise Monitor Agent for each monitored host.

- A single MySQL Enterprise Service Manager, which collates information from the agents and provides the user interface to the collected data.

MySQL Enterprise Monitor is designed to monitor one or more MySQL servers. The monitoring information is collected by using an agent, *MySQL Enterprise Monitor Agent*. The agent communicates with the hosts and MySQL servers that it monitors, collecting variables, status and health information, and sending this information to the MySQL Enterprise Service Manager.

The information collected by the agent about each MySQL server and host you are monitoring is sent to the *MySQL Enterprise Service Manager*. This server collates all of the information from the agents. As it collates the information sent by the agents, the MySQL Enterprise Service Manager continually tests the collected data, comparing the status of the server to reasonable values. When thresholds are reached, the server can trigger an event (including an alarm and notification) to highlight a potential issue, such as low memory, high CPU usage, or more complex conditions such insufficient buffer sizes and status information. We call each test, with its associated threshold value, a *rule*.

These rules, and the alarms and notifications, are each known as a *MySQL Enterprise Advisors*. Advisors form a critical part of the MySQL Enterprise Service Manager, as they provide warning information and troubleshooting advice about potential problems.

The MySQL Enterprise Service Manager includes a web server, and you interact with it through any web browser. This interface, the MySQL Enterprise Monitor User Interface, displays all of the information collected by the agents, and lets you view all of your servers and their current status as a group or individually. You control and configure all aspects of the service using the MySQL Enterprise Monitor User Interface.

The information supplied by the MySQL Enterprise Monitor Agent processes also includes statistical and query information, which you can view in the form of graphs. For example, you can view aspects such as server load, query numbers, or index usage information as a graph over time. The graph lets you pinpoint problems or potential issues on your server, and can help diagnose the impact from database or external problems (such as external system or network failure) by examining the data from a specific time interval.

The MySQL Enterprise Monitor Agent can also be configured to collect detailed information about the queries executed on your server, including the row counts and performance times for executing each query. You can correlate the detailed query data with the graphical information to identify which queries were executing when you experienced a particularly high load, index or other issue. The query data is supported by a system called Query Analyzer, and the data can be presented in different ways depending on your needs.

24.2 MySQL Enterprise Backup Overview

MySQL Enterprise Backup performs hot backup operations for MySQL databases. The product is architected for efficient and reliable backups of tables created by the InnoDB storage engine. For completeness, it can also back up tables from MyISAM and other storage engines.

The following discussion briefly summarizes MySQL Enterprise Backup. For more information, see the MySQL Enterprise Backup manual, available at https://dev.mysql.com/doc/mysql-enterprise-backup/en/.

Hot backups are performed while the database is running and applications are reading and writing to it. This type of backup does not block normal database operations, and it captures even changes that occur while the backup is happening. For these reasons, hot backups are desirable when your database “grows up” -- when the data is large enough that the backup takes significant time, and when your data is important enough to your business that you must capture every last change, without taking your application, website, or web service offline.

MySQL Enterprise Backup does a hot backup of all tables that use the InnoDB storage engine. For tables using MyISAM or other non-InnoDB storage engines, it does a “warm” backup, where the database continues to run, but those tables cannot be modified while being backed up. For efficient

backup operations, you can designate InnoDB as the default storage engine for new tables, or convert existing tables to use the InnoDB storage engine.

24.3 MySQL Enterprise Security Overview

MySQL Enterprise Edition provides plugins that implement security features using external services:

- MySQL Enterprise Edition includes an authentication plugin that enables MySQL Server to use PAM (Pluggable Authentication Modules) to authenticate MySQL users. PAM enables a system to use a standard interface to access various kinds of authentication methods, such as Unix passwords or an LDAP directory. For more information, see Section 6.4.1.6, “PAM Pluggable Authentication”.
- MySQL Enterprise Edition includes an authentication plugin that performs external authentication on Windows, enabling MySQL Server to use native Windows services to authenticate client connections. Users who have logged in to Windows can connect from MySQL client programs to the server based on the information in their environment without specifying an additional password. For more information, see Section 6.4.1.7, “Windows Pluggable Authentication”.
- MySQL Enterprise Edition includes a set of encryption functions based on the OpenSSL library that expose OpenSSL capabilities at the SQL level. These functions enable masking existing data using several methods such as obfuscation (removing identifying characteristics), generation of formatted random data, and data replacement or substitution. For more information, see Section 24.4, “MySQL Enterprise Encryption Overview”.
- MySQL Enterprise Edition 5.7 and higher includes a keyring plugin that uses Oracle Key Vault as a back end for keyring storage. For more information, see The MySQL Keyring.

For other related Enterprise security features, see Section 24.4, “MySQL Enterprise Encryption Overview”.

24.4 MySQL Enterprise Encryption Overview

MySQL Enterprise Edition includes a set of encryption functions based on the OpenSSL library that expose OpenSSL capabilities at the SQL level. These functions enable Enterprise applications to perform the following operations:

- Implement added data protection using public-key asymmetric cryptography
- Create public and private keys and digital signatures
- Perform asymmetric encryption and decryption
- Use cryptographic hashing for digital signing and data verification and validation

For more information, see Section 6.5, “MySQL Enterprise Encryption”.

For other related Enterprise security features, see Section 24.3, “MySQL Enterprise Security Overview”.

24.5 MySQL Enterprise Audit Overview

MySQL Enterprise Edition includes MySQL Enterprise Audit, implemented using a server plugin. MySQL Enterprise Audit uses the open MySQL Audit API to enable standard, policy-based monitoring and logging of connection and query activity executed on specific MySQL servers. Designed to meet the Oracle audit specification, MySQL Enterprise Audit provides an out of box, easy to use auditing and compliance solution for applications that are governed by both internal and external regulatory guidelines.

When installed, the audit plugin enables MySQL Server to produce a log file containing an audit record of server activity. The log contents include when clients connect and disconnect, and what actions they perform while connected, such as which databases and tables they access.

For more information, see Section 6.4.4, “MySQL Enterprise Audit”.

24.6 MySQL Enterprise Firewall Overview

MySQL Enterprise Edition includes MySQL Enterprise Firewall, an application-level firewall that enables database administrators to permit or deny SQL statement execution based on matching against allowlists of accepted statement patterns. This helps harden MySQL Server against attacks such as SQL injection or attempts to exploit applications by using them outside of their legitimate query workload characteristics.

Each MySQL account registered with the firewall has its own statement allowlist, enabling protection to be tailored per account. For a given account, the firewall can operate in recording or protecting mode, for training in the accepted statement patterns or protection against unacceptable statements.

For more information, see Section 6.4.5, “MySQL Enterprise Firewall”.

24.7 MySQL Enterprise Thread Pool Overview

MySQL Enterprise Edition includes MySQL Enterprise Thread Pool, implemented using a server plugin. The default thread-handling model in MySQL Server executes statements using one thread per client connection. As more clients connect to the server and execute statements, overall performance degrades. In MySQL Enterprise Edition, a thread pool plugin provides an alternative thread-handling model designed to reduce overhead and improve performance. The plugin implements a thread pool that increases server performance by efficiently managing statement execution threads for large numbers of client connections.

For more information, see Section 5.5.3, “MySQL Enterprise Thread Pool”.

24.8 MySQL Enterprise Data Masking and De-Identification Overview

MySQL Enterprise Edition 5.7 and higher includes MySQL Enterprise Data Masking and De-Identification, implemented as a plugin library containing a plugin and a set of user-defined functions. Data masking hides sensitive information by replacing real values with substitutes. MySQL Enterprise Data Masking and De-Identification functions enable masking existing data using several methods such as obfuscation (removing identifying characteristics), generation of formatted random data, and data replacement or substitution.

For more information, see MySQL Enterprise Data Masking and De-Identification.

Chapter 25 MySQL Workbench

MySQL Workbench provides a graphical tool for working with MySQL servers and databases. MySQL Workbench fully supports MySQL versions 5.5 and higher.

The following discussion briefly describes MySQL Workbench capabilities. For more information, see the MySQL Workbench manual, available at https://dev.mysql.com/doc/workbench/en/.

MySQL Workbench provides five main areas of functionality:

- **SQL Development**: Enables you to create and manage connections to database servers. As well as enabling you to configure connection parameters, MySQL Workbench provides the capability to execute SQL queries on the database connections using the built-in SQL Editor. This functionality replaces that previously provided by the Query Browser standalone application.
- **Data Modeling**: Enables you to create models of your database schema graphically, reverse and forward engineer between a schema and a live database, and edit all aspects of your database using the comprehensive Table Editor. The Table Editor provides easy-to-use facilities for editing Tables, Columns, Indexes, Triggers, Partitioning, Options, Inserts and Privileges, Routines and Views.
- **Server Administration**: Enables you to create and administer server instances.
- **Data Migration**: Allows you to migrate from Microsoft SQL Server, Sybase ASE, SQLite, SQL Anywhere, PostreSQL, and other RDBMS tables, objects and data to MySQL. Migration also supports migrating from earlier versions of MySQL to the latest releases.
- **MySQL Enterprise Support**: Support for Enterprise products such as MySQL Enterprise Backup and MySQL Audit.

MySQL Workbench is available in two editions, the Community Edition and the Commercial Edition. The Community Edition is available free of charge. The Commercial Edition provides additional Enterprise features, such as database documentation generation, at low cost.

Appendix A MySQL 5.6 Frequently Asked Questions

Table of Contents

A.1 MySQL 5.6 FAQ: General 3195
A.2 MySQL 5.6 FAQ: Storage Engines 3197
A.3 MySQL 5.6 FAQ: Server SQL Mode 3197
A.4 MySQL 5.6 FAQ: Stored Procedures and Functions 3198
A.5 MySQL 5.6 FAQ: Triggers 3202
A.6 MySQL 5.6 FAQ: Views 3204
A.7 MySQL 5.6 FAQ: INFORMATION_SCHEMA 3204
A.8 MySQL 5.6 FAQ: Migration 3205
A.9 MySQL 5.6 FAQ: Security 3205
A.10 MySQL 5.6 FAQ: NDB Cluster 3206
A.11 MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets 3219
A.12 MySQL 5.6 FAQ: Connectors & APIs 3231
A.13 MySQL 5.6 FAQ: C API, libmysql 3231
A.14 MySQL 5.6 FAQ: Replication 3232
A.15 MySQL 5.6 FAQ: MySQL Enterprise Thread Pool 3236
A.16 MySQL 5.6 FAQ: InnoDB Change Buffer 3237
A.17 MySQL 5.6 FAQ: Virtualization Support 3239

A.1 MySQL 5.6 FAQ: General

A.1.1 Which version of MySQL is production-ready (GA)? 3195
A.1.2 Can MySQL 5.6 do subqueries? 3195
A.1.3 Can MySQL 5.6 perform multiple-table inserts, updates, and deletes? 3196
A.1.4 Does MySQL 5.6 have a Query Cache? Does it work on Server, Instance or Database? 3196
A.1.5 Does MySQL 5.6 have Sequences? 3196
A.1.6 Does MySQL 5.6 have a `NOW()` function with fractions of seconds? 3196
A.1.7 Does MySQL 5.6 work with multi-core processors? 3196
A.1.8 Why do I see multiple processes for `mysqld`? 3196
A.1.9 Can MySQL 5.6 perform ACID transactions? 3196

A.1.1. Which version of MySQL is production-ready (GA)?

MySQL 8.0, 5.7, and MySQL 5.6 are supported for production use.

MySQL 8.0 achieved General Availability (GA) status with MySQL 8.0.11, which was released for production use on 19 April 2018.

MySQL 5.7 achieved General Availability (GA) status with MySQL 5.7.9, which was released for production use on 21 October 2015.

MySQL 5.6 achieved General Availability (GA) status with MySQL 5.6.10, which was released for production use on 5 February 2013.

MySQL 5.5 achieved General Availability (GA) status with MySQL 5.5.8, which was released for production use on 3 December 2010. Active development for MySQL 5.5 has ended.

MySQL 5.1 achieved General Availability (GA) status with MySQL 5.1.30, which was released for production use on 14 November 2008. Active development for MySQL 5.1 has ended.

MySQL 5.0 achieved General Availability (GA) status with MySQL 5.0.15, which was released for production use on 19 October 2005. Active development for MySQL 5.0 has ended.

A.1.2. Can MySQL 5.6 do subqueries?

Yes. See Section 13.2.10, “Subqueries”.

A.1.3. Can MySQL 5.6 perform multiple-table inserts, updates, and deletes?

Yes. For the syntax required to perform multiple-table updates, see Section 13.2.11, “UPDATE Statement”; for that required to perform multiple-table deletes, see Section 13.2.2, “DELETE Statement”.

A multiple-table insert can be accomplished using a trigger whose `FOR EACH ROW` clause contains multiple `INSERT` statements within a `BEGIN ... END` block. See Section 20.3, “Using Triggers”.

A.1.4. Does MySQL 5.6 have a Query Cache? Does it work on Server, Instance or Database?

Yes. The query cache operates on the server level, caching complete result sets matched with the original query string. If an exactly identical query is made (which often happens, particularly in web applications), no parsing or execution is necessary; the result is sent directly from the cache. Various tuning options are available. See Section 8.10.3, “The MySQL Query Cache”.

A.1.5. Does MySQL 5.6 have Sequences?

No. However, MySQL has an `AUTO_INCREMENT` system, which in MySQL 5.6 can also handle inserts in a multi-source replication setup. With the `auto_increment_increment` and `auto_increment_offset` system variables, you can set each server to generate auto-increment values that don't conflict with other servers. The `auto_increment_increment` value should be greater than the number of servers, and each server should have a unique offset.

A.1.6. Does MySQL 5.6 have a `NOW()` function with fractions of seconds?

Yes, see Section 11.2.7, “Fractional Seconds in Time Values”.

A.1.7. Does MySQL 5.6 work with multi-core processors?

Yes. MySQL is fully multithreaded, and makes use of all CPUs made available to it. Not all CPUs may be available; modern operating systems should be able to utilize all underlying CPUs, but also make it possible to restrict a process to a specific CPU or sets of CPUs.

On Windows, there is currently a limit to the number of (logical) processors that `mysqld` can use: a single processor group, which is limited to a maximum of 64 logical processors.

Use of multiple cores may be seen in these ways:

- A single core is usually used to service the commands issued from one session.
- A few background threads make limited use of extra cores; for example, to keep background I/O tasks moving.
- If the database is I/O-bound (indicated by CPU consumption less than capacity), adding more CPUs is futile. If the database is partitioned into an I/O-bound part and a CPU-bond part, adding CPUs may still be useful.

A.1.8. Why do I see multiple processes for `mysqld`?

`mysqld` is a single-process program, not a multi-process program, and does not fork or launch other processes. However, `mysqld` is multithreaded and some process-reporting system utilities display separate entries for each thread of multithreaded processes, which may lead to the appearance of multiple `mysqld` processes when in fact there is only one.

A.1.9. Can MySQL 5.6 perform ACID transactions?

Yes. All current MySQL versions support transactions. The `InnoDB` storage engine offers full ACID transactions with row-level locking, multi-versioning, nonlocking repeatable reads, and all four SQL standard isolation levels.

The NDB storage engine supports the READ COMMITTED transaction isolation level only.

A.2 MySQL 5.6 FAQ: Storage Engines

A.2.1 Where can I obtain complete documentation for MySQL storage engines? 3197
A.2.2 Are there any new storage engines in MySQL 5.6? 3197
A.2.3 Have any storage engines been removed in MySQL 5.6? 3197
A.2.4 What are the unique benefits of the ARCHIVE storage engine? 3197

A.2.1. Where can I obtain complete documentation for MySQL storage engines?

See Chapter 15, *Alternative Storage Engines*. That chapter contains information about all MySQL storage engines except for the InnoDB storage engine and the NDB storage engine (used for MySQL Cluster). InnoDB is covered in Chapter 14, *The InnoDB Storage Engine*. NDB is covered in Chapter 18, *MySQL NDB Cluster 7.3 and NDB Cluster 7.4*.

A.2.2. Are there any new storage engines in MySQL 5.6?

The features from the optional InnoDB Plugin from MySQL 5.1 are folded into the built-in InnoDB storage engine, so you can take advantage of features such as the Barracuda file format, InnoDB table compression, and the new configuration options for performance. See Chapter 14, *The InnoDB Storage Engine* for details. InnoDB also becomes the default storage engine for new tables. See Section 14.1, “Introduction to InnoDB” for details.

A.2.3. Have any storage engines been removed in MySQL 5.6?

No.

A.2.4. What are the unique benefits of the ARCHIVE storage engine?

The ARCHIVE storage engine stores large amounts of data without indexes; it has a small footprint, and performs selects using table scans. See Section 15.5, “The ARCHIVE Storage Engine”, for details.

A.3 MySQL 5.6 FAQ: Server SQL Mode

A.3.1 What are server SQL modes? 3197
A.3.2 How many server SQL modes are there? 3197
A.3.3 How do you determine the server SQL mode? 3197
A.3.4 Is the mode dependent on the database or connection? 3198
A.3.5 Can the rules for strict mode be extended? 3198
A.3.6 Does strict mode impact performance? 3198
A.3.7 What is the default server SQL mode when MySQL 5.6 is installed? 3198

A.3.1. What are server SQL modes?

Server SQL modes define what SQL syntax MySQL should support and what kind of data validation checks it should perform. This makes it easier to use MySQL in different environments and to use MySQL together with other database servers. The MySQL Server apply these modes individually to different clients. For more information, see Section 5.1.10, “Server SQL Modes”.

A.3.2. How many server SQL modes are there?

Each mode can be independently switched on and off. See Section 5.1.10, “Server SQL Modes”, for a complete list of available modes.

A.3.3. How do you determine the server SQL mode?

You can set the default SQL mode (for mysqld startup) with the --sql-mode option. Using the statement SET [GLOBAL|SESSION] sql_mode='*modes*', you can change the settings from within a connection, either locally to the connection, or to take effect globally. You can retrieve the current mode by issuing a SELECT @@sql_mode statement.

A.3.4. Is the mode dependent on the database or connection?

A mode is not linked to a particular database. Modes can be set locally to the session (connection), or globally for the server. you can change these settings using `SET [GLOBAL|SESSION] sql_mode='modes'`.

A.3.5. Can the rules for strict mode be extended?

When we refer to *strict mode*, we mean a mode where at least one of the modes `TRADITIONAL`, `STRICT_TRANS_TABLES`, or `STRICT_ALL_TABLES` is enabled. Options can be combined, so you can add restrictions to a mode. See Section 5.1.10, “Server SQL Modes”, for more information.

A.3.6. Does strict mode impact performance?

The intensive validation of input data that some settings requires more time than if the validation is not done. While the performance impact is not that great, if you do not require such validation (perhaps your application already handles all of this), then MySQL gives you the option of leaving strict mode disabled. However, if you do require it, strict mode can provide such validation.

A.3.7. What is the default server SQL mode when MySQL 5.6 is installed?

The default SQL mode is `NO_ENGINE_SUBSTITUTION`. For information about all available modes and default MySQL behavior, see Section 5.1.10, “Server SQL Modes”.

A.4 MySQL 5.6 FAQ: Stored Procedures and Functions

A.4.1 Does MySQL 5.6 support stored procedures and functions? 3199
A.4.2 Where can I find documentation for MySQL stored procedures and stored functions? 3199
A.4.3 Is there a discussion forum for MySQL stored procedures? 3199
A.4.4 Where can I find the ANSI SQL 2003 specification for stored procedures? 3199
A.4.5 How do you manage stored routines? 3199
A.4.6 Is there a way to view all stored procedures and stored functions in a given database? 3199
A.4.7 Where are stored procedures stored? 3199
A.4.8 Is it possible to group stored procedures or stored functions into packages? 3199
A.4.9 Can a stored procedure call another stored procedure? 3199
A.4.10 Can a stored procedure call a trigger? 3199
A.4.11 Can a stored procedure access tables? 3200
A.4.12 Do stored procedures have a statement for raising application errors? 3200
A.4.13 Do stored procedures provide exception handling? 3200
A.4.14 Can MySQL 5.6 stored routines return result sets? 3200
A.4.15 Is `WITH RECOMPILE` supported for stored procedures? 3200
A.4.16 Is there a MySQL equivalent to using `mod_plsql` as a gateway on Apache to talk directly to a stored procedure in the database? 3200
A.4.17 Can I pass an array as input to a stored procedure? 3200
A.4.18 Can I pass a cursor as an `IN` parameter to a stored procedure? 3200
A.4.19 Can I return a cursor as an `OUT` parameter from a stored procedure? 3200
A.4.20 Can I print out a variable's value within a stored routine for debugging purposes? 3200
A.4.21 Can I commit or roll back transactions inside a stored procedure? 3200
A.4.22 Do MySQL 5.6 stored procedures and functions work with replication? 3200
A.4.23 Are stored procedures and functions created on a replication source server replicated to a replica? 3200
A.4.24 How are actions that take place inside stored procedures and functions replicated? 3201
A.4.25 Are there special security requirements for using stored procedures and functions together with replication? 3201
A.4.26 What limitations exist for replicating stored procedure and function actions? 3201
A.4.27 Do the preceding limitations affect the ability of MySQL to do point-in-time recovery? 3201
A.4.28 What is being done to correct the aforementioned limitations? 3201

A.4.1. Does MySQL 5.6 support stored procedures and functions?

Yes. MySQL 5.6 supports two types of stored routines, stored procedures and stored functions.

A.4.2. Where can I find documentation for MySQL stored procedures and stored functions?

See Section 20.2, “Using Stored Routines”.

A.4.3. Is there a discussion forum for MySQL stored procedures?

Yes. See https://forums.mysql.com/list.php?98.

A.4.4. Where can I find the ANSI SQL 2003 specification for stored procedures?

Unfortunately, the official specifications are not freely available (ANSI makes them available for purchase). However, there are books, such as *SQL-99 Complete, Really* by Peter Gulutzan and Trudy Pelzer, that provide a comprehensive overview of the standard, including coverage of stored procedures.

A.4.5. How do you manage stored routines?

It is always good practice to use a clear naming scheme for your stored routines. You can manage stored procedures with `CREATE [FUNCTION|PROCEDURE]`, `ALTER [FUNCTION|PROCEDURE]`, `DROP [FUNCTION|PROCEDURE]`, and `SHOW CREATE [FUNCTION|PROCEDURE]`. You can obtain information about existing stored procedures using the `ROUTINES` table in the `INFORMATION_SCHEMA` database (see Section 21.19, “The INFORMATION_SCHEMA ROUTINES Table”).

A.4.6. Is there a way to view all stored procedures and stored functions in a given database?

Yes. For a database named *`dbname`*, use this query on the `INFORMATION_SCHEMA.ROUTINES` table:

```
SELECT ROUTINE_TYPE, ROUTINE_NAME
    FROM INFORMATION_SCHEMA.ROUTINES
    WHERE ROUTINE_SCHEMA='dbname';
```

For more information, see Section 21.19, “The INFORMATION_SCHEMA ROUTINES Table”.

The body of a stored routine can be viewed using `SHOW CREATE FUNCTION` (for a stored function) or `SHOW CREATE PROCEDURE` (for a stored procedure). See Section 13.7.5.11, “SHOW CREATE PROCEDURE Statement”, for more information.

A.4.7. Where are stored procedures stored?

In the `proc` table of the `mysql` system database. However, you should not access the tables in the system database directly. Instead, query the `INFORMATION_SCHEMA ROUTINES` and `PARAMETERS` tables. See Section 21.19, “The INFORMATION_SCHEMA ROUTINES Table”, and Section 21.13, “The INFORMATION_SCHEMA PARAMETERS Table”.

You can also use `SHOW CREATE FUNCTION` to obtain information about stored functions, and `SHOW CREATE PROCEDURE` to obtain information about stored procedures. See Section 13.7.5.11, “SHOW CREATE PROCEDURE Statement”.

A.4.8. Is it possible to group stored procedures or stored functions into packages?

No. This is not supported in MySQL 5.6.

A.4.9. Can a stored procedure call another stored procedure?

Yes.

A.4.10. Can a stored procedure call a trigger?

A stored procedure can execute an SQL statement, such as an `UPDATE`, that causes a trigger to activate.

A.4.11 Can a stored procedure access tables?

Yes. A stored procedure can access one or more tables as required.

A.4.12 Do stored procedures have a statement for raising application errors?

Yes. MySQL 5.6 implements the SQL standard `SIGNAL` and `RESIGNAL` statements. See Section 13.6.7, “Condition Handling”.

A.4.13 Do stored procedures provide exception handling?

MySQL implements `HANDLER` definitions according to the SQL standard. See Section 13.6.7.2, “DECLARE ... HANDLER Statement”, for details.

A.4.14 Can MySQL 5.6 stored routines return result sets?

Stored procedures can, but stored functions cannot. If you perform an ordinary `SELECT` inside a stored procedure, the result set is returned directly to the client. You need to use the MySQL 4.1 (or higher) client/server protocol for this to work. This means that, for example, in PHP, you need to use the `mysqli` extension rather than the old `mysql` extension.

A.4.15 Is `WITH RECOMPILE` supported for stored procedures?

Not in MySQL 5.6.

A.4.16 Is there a MySQL equivalent to using `mod_plsql` as a gateway on Apache to talk directly to a stored procedure in the database?

There is no equivalent in MySQL 5.6.

A.4.17 Can I pass an array as input to a stored procedure?

Not in MySQL 5.6.

A.4.18 Can I pass a cursor as an `IN` parameter to a stored procedure?

In MySQL 5.6, cursors are available inside stored procedures only.

A.4.19 Can I return a cursor as an `OUT` parameter from a stored procedure?

In MySQL 5.6, cursors are available inside stored procedures only. However, if you do not open a cursor on a `SELECT`, the result is sent directly to the client. You can also `SELECT INTO` variables. See Section 13.2.9, “SELECT Statement”.

A.4.20 Can I print out a variable's value within a stored routine for debugging purposes?

Yes, you can do this in a *stored procedure*, but not in a stored function. If you perform an ordinary `SELECT` inside a stored procedure, the result set is returned directly to the client. You must use the MySQL 4.1 (or above) client/server protocol for this to work. This means that, for example, in PHP, you need to use the `mysqli` extension rather than the old `mysql` extension.

A.4.21 Can I commit or roll back transactions inside a stored procedure?

Yes. However, you cannot perform transactional operations within a stored function.

A.4.22 Do MySQL 5.6 stored procedures and functions work with replication?

Yes, standard actions carried out in stored procedures and functions are replicated from a replication source server to a replica. There are a few limitations that are described in detail in Section 20.7, “Stored Program Binary Logging”.

A.4.23 Are stored procedures and functions created on a replication source server replicated to a replica?

Yes, creation of stored procedures and functions carried out through normal DDL statements on a replication source server are replicated to a replica, so that the objects exist on both servers. `ALTER` and `DROP` statements for stored procedures and functions are also replicated.

A.4.24 How are actions that take place inside stored procedures and functions replicated?

MySQL records each DML event that occurs in a stored procedure and replicates those individual actions to a replica. The actual calls made to execute stored procedures are not replicated.

Stored functions that change data are logged as function invocations, not as the DML events that occur inside each function.

A.4.25 Are there special security requirements for using stored procedures and functions together with replication?

Yes. Because a replica has authority to execute any statement read from a source's binary log, special security constraints exist for using stored functions with replication. If replication or binary logging in general (for the purpose of point-in-time recovery) is active, then MySQL DBAs have two security options open to them:

1. Any user wishing to create stored functions must be granted the `SUPER` privilege.
2. Alternatively, a DBA can set the `log_bin_trust_function_creators` system variable to 1, which enables anyone with the standard `CREATE ROUTINE` privilege to create stored functions.

A.4.26 What limitations exist for replicating stored procedure and function actions?

Nondeterministic (random) or time-based actions embedded in stored procedures may not replicate properly. By their very nature, randomly produced results are not predictable and cannot be exactly reproduced; therefore, random actions replicated to a replica do not mirror those performed on a source. Declaring stored functions to be `DETERMINISTIC` or setting the `log_bin_trust_function_creators` system variable to 0 keeps random operations producing random values from being invoked.

In addition, time-based actions cannot be reproduced on a replica because the timing of such actions in a stored procedure is not reproducible through the binary log used for replication. It records only DML events and does not factor in timing constraints.

Finally, nontransactional tables for which errors occur during large DML actions (such as bulk inserts) may experience replication issues in that a source may be partially updated from DML activity, but no updates are done to the replica because of the errors that occurred. A workaround is for a function's DML actions to be carried out with the `IGNORE` keyword so that updates on the source that cause errors are ignored and updates that do not cause errors are replicated to the replica.

A.4.27 Do the preceding limitations affect the ability of MySQL to do point-in-time recovery?

The same limitations that affect replication do affect point-in-time recovery.

A.4.28 What is being done to correct the aforementioned limitations?

You can choose either statement-based replication or row-based replication. The original replication implementation is based on statement-based binary logging. Row-based binary logging resolves the limitations mentioned earlier.

Mixed replication is also available (by starting the server with `--binlog-format=mixed`). This hybrid form of replication “knows” whether statement-level replication can safely be used, or row-level replication is required.

For additional information, see Section 17.1.2, “Replication Formats”.

A.5 MySQL 5.6 FAQ: Triggers

A.5.1 Where can I find the documentation for MySQL 5.6 triggers? 3202
A.5.2 Is there a discussion forum for MySQL Triggers? 3202
A.5.3 Does MySQL 5.6 have statement-level or row-level triggers? 3202
A.5.4 Are there any default triggers? 3202
A.5.5 How are triggers managed in MySQL? 3202
A.5.6 Is there a way to view all triggers in a given database? 3202
A.5.7 Where are triggers stored? 3202
A.5.8 Can a trigger call a stored procedure? 3203
A.5.9 Can triggers access tables? 3203
A.5.10 Can a table have multiple triggers with the same trigger event and action time? 3203
A.5.11 Can triggers call an external application through a UDF? 3203
A.5.12 Is it possible for a trigger to update tables on a remote server? 3203
A.5.13 Do triggers work with replication? 3203
A.5.14 How are actions carried out through triggers on a source replicated to a replica? 3203

A.5.1. Where can I find the documentation for MySQL 5.6 triggers?

See Section 20.3, “Using Triggers”.

A.5.2. Is there a discussion forum for MySQL Triggers?

Yes. It is available at https://forums.mysql.com/list.php?99.

A.5.3. Does MySQL 5.6 have statement-level or row-level triggers?

In MySQL 5.6, all triggers are `FOR EACH ROW`; that is, the trigger is activated for each row that is inserted, updated, or deleted. MySQL 5.6 does not support triggers using `FOR EACH STATEMENT`.

A.5.4. Are there any default triggers?

Not explicitly. MySQL does have specific special behavior for some `TIMESTAMP` columns, as well as for columns which are defined using `AUTO_INCREMENT`.

A.5.5. How are triggers managed in MySQL?

In MySQL 5.6, triggers can be created using the `CREATE TRIGGER` statement, and dropped using `DROP TRIGGER`. See Section 13.1.19, “CREATE TRIGGER Statement”, and Section 13.1.30, “DROP TRIGGER Statement”, for more about these statements.

Information about triggers can be obtained by querying the `INFORMATION_SCHEMA.TRIGGERS` table. See Section 21.27, “The INFORMATION_SCHEMA TRIGGERS Table”.

A.5.6. Is there a way to view all triggers in a given database?

Yes. You can obtain a listing of all triggers defined on database `dbname` using a query on the `INFORMATION_SCHEMA.TRIGGERS` table such as the one shown here:

```
SELECT TRIGGER_NAME, EVENT_MANIPULATION, EVENT_OBJECT_TABLE, ACTION_STATEMENT
    FROM INFORMATION_SCHEMA.TRIGGERS
    WHERE TRIGGER_SCHEMA='dbname';
```

For more information about this table, see Section 21.27, “The INFORMATION_SCHEMA TRIGGERS Table”.

You can also use the `SHOW TRIGGERS` statement, which is specific to MySQL. See Section 13.7.5.39, “SHOW TRIGGERS Statement”.

A.5.7. Where are triggers stored?

Triggers are stored in `.TRG` files, with one such file one per table.

A.5.8. Can a trigger call a stored procedure?

Yes.

A.5.9. Can triggers access tables?

A trigger can access both old and new data in its own table. A trigger can also affect other tables, but it is not permitted to modify a table that is already being used (for reading or writing) by the statement that invoked the function or trigger.

A.5.10. Can a table have multiple triggers with the same trigger event and action time?

In MySQL 5.6, there cannot be multiple triggers for a given table that have the same trigger event and action time. For example, you cannot have two `BEFORE UPDATE` triggers for a table. This limitation is lifted in MySQL 5.7.

A.5.11. Can triggers call an external application through a UDF?

Yes. For example, a trigger could invoke the `sys_exec()` UDF.

A.5.12. Is it possible for a trigger to update tables on a remote server?

Yes. A table on a remote server could be updated using the `FEDERATED` storage engine. (See Section 15.8, “The FEDERATED Storage Engine”).

A.5.13. Do triggers work with replication?

Yes. However, the way in which they work depends whether you are using MySQL's “classic” statement-based or row-based replication format.

When using statement-based replication, triggers on the replica are executed by statements that are executed on the source (and replicated to the replica).

When using row-based replication, triggers are not executed on the replica due to statements that were run on the source and then replicated to the replica. Instead, when using row-based replication, the changes caused by executing the trigger on the source are applied on the replica.

For more information, see Section 17.4.1.33, “Replication and Triggers”.

A.5.14. How are actions carried out through triggers on a source replicated to a replica?

Again, this depends on whether you are using statement-based or row-based replication.

Statement-based replication. First, the triggers that exist on a source must be re-created on the replica server. Once this is done, the replication flow works as any other standard DML statement that participates in replication. For example, consider a table `EMP` that has an `AFTER` insert trigger, which exists on a replication source server. The same `EMP` table and `AFTER` insert trigger exist on the replica server as well. The replication flow would be:

1. An `INSERT` statement is made to `EMP`.
2. The `AFTER` trigger on `EMP` activates.
3. The `INSERT` statement is written to the binary log.
4. The replica picks up the `INSERT` statement to `EMP` and executes it.
5. The `AFTER` trigger on `EMP` that exists on the replica activates.

Row-based replication. When you use row-based replication, the changes caused by executing the trigger on the source are applied on the replica. However, the triggers themselves are not actually executed on the replica under row-based replication. This is because, if both the

source and the replica applied the changes from the source and, in addition, the trigger causing these changes were applied on the replica, the changes would in effect be applied twice on the replica, leading to different data on the source and the replica.

In most cases, the outcome is the same for both row-based and statement-based replication. However, if you use different triggers on the source and replica, you cannot use row-based replication. (This is because the row-based format replicates the changes made by triggers executing on the source to the replicas, rather than the statements that caused the triggers to execute, and the corresponding triggers on the replica are not executed.) Instead, any statements causing such triggers to be executed must be replicated using statement-based replication.

For more information, see Section 17.4.1.33, “Replication and Triggers”.

A.6 MySQL 5.6 FAQ: Views

A.6.1 Where can I find documentation covering MySQL Views? 3204
A.6.2 Is there a discussion forum for MySQL Views? 3204
A.6.3 What happens to a view if an underlying table is dropped or renamed? 3204
A.6.4 Does MySQL 5.6 have table snapshots? 3204
A.6.5 Does MySQL 5.6 have materialized views? 3204
A.6.6 Can you insert into views that are based on joins? 3204

A.6.1. Where can I find documentation covering MySQL Views?

See Section 20.5, “Using Views”.

A.6.2. Is there a discussion forum for MySQL Views?

Yes. See https://forums.mysql.com/list.php?100

A.6.3. What happens to a view if an underlying table is dropped or renamed?

After a view has been created, it is possible to drop or alter a table or view to which the definition refers. To check a view definition for problems of this kind, use the `CHECK TABLE` statement. (See Section 13.7.2.2, “CHECK TABLE Statement”.)

A.6.4. Does MySQL 5.6 have table snapshots?

No.

A.6.5. Does MySQL 5.6 have materialized views?

No.

A.6.6. Can you insert into views that are based on joins?

It is possible, provided that your `INSERT` statement has a column list that makes it clear there is only one table involved.

You *cannot* insert into multiple tables with a single insert on a view.

A.7 MySQL 5.6 FAQ: INFORMATION_SCHEMA

A.7.1 Where can I find documentation for the MySQL `INFORMATION_SCHEMA` database? 3204
A.7.2 Is there a discussion forum for `INFORMATION_SCHEMA`? 3205
A.7.3 Where can I find the ANSI SQL 2003 specification for `INFORMATION_SCHEMA`? 3205
A.7.4 What is the difference between the Oracle Data Dictionary and MySQL `INFORMATION_SCHEMA`? 3205
A.7.5 Can I add to or otherwise modify the tables found in the `INFORMATION_SCHEMA` database? 3205

A.7.1. Where can I find documentation for the MySQL `INFORMATION_SCHEMA` database?

See Chapter 21, *INFORMATION_SCHEMA Tables*

A.7.2. Is there a discussion forum for `INFORMATION_SCHEMA`?

See https://forums.mysql.com/list.php?101.

A.7.3. Where can I find the ANSI SQL 2003 specification for `INFORMATION_SCHEMA`?

Unfortunately, the official specifications are not freely available. (ANSI makes them available for purchase.) However, there are books available, such as *SQL-99 Complete, Really* by Peter Gulutzan and Trudy Pelzer, that provide a comprehensive overview of the standard, including `INFORMATION_SCHEMA`.

A.7.4. What is the difference between the Oracle Data Dictionary and MySQL `INFORMATION_SCHEMA`?

Both Oracle and MySQL provide metadata in tables. However, Oracle and MySQL use different table names and column names. The MySQL implementation is more similar to those found in DB2 and SQL Server, which also support `INFORMATION_SCHEMA` as defined in the SQL standard.

A.7.5. Can I add to or otherwise modify the tables found in the `INFORMATION_SCHEMA` database?

No. Since applications may rely on a certain standard structure, this should not be modified. For this reason, *we cannot support bugs or other issues which result from modifying `INFORMATION_SCHEMA` tables or data*.

A.8 MySQL 5.6 FAQ: Migration

A.8.1 Where can I find information on how to migrate from MySQL 5.5 to MySQL 5.6? 3205
A.8.2 How has storage engine (table type) support changed in MySQL 5.6 from previous versions? 3205

A.8.1. Where can I find information on how to migrate from MySQL 5.5 to MySQL 5.6?

For detailed upgrade information, see Section 2.11, “Upgrading MySQL”. Do not skip a major version when upgrading, but rather complete the process in steps, upgrading from one major version to the next in each step. This may seem more complicated, but ultimately saves time and trouble. If you encounter problems during the upgrade, their origin is easier to identify, either by you or, if you have a MySQL Enterprise subscription, by MySQL support.

A.8.2. How has storage engine (table type) support changed in MySQL 5.6 from previous versions?

Storage engine support has changed as follows:

- Support for `ISAM` tables was removed in MySQL 5.0 and you should now use the `MyISAM` storage engine in place of `ISAM`. To convert a table *`tblname`* from `ISAM` to `MyISAM`, simply issue a statement such as this one:

```
ALTER TABLE tblname ENGINE=MYISAM;
```

- Internal `RAID` for `MyISAM` tables was also removed in MySQL 5.0. This was formerly used to allow large tables in file systems that did not support file sizes greater than 2GB. All modern file systems allow for larger tables; in addition, there are now other solutions such as `MERGE` tables and views.
- The `VARCHAR` column type now retains trailing spaces in all storage engines.
- `MEMORY` tables (formerly known as `HEAP` tables) can also contain `VARCHAR` columns.

A.9 MySQL 5.6 FAQ: Security

A.9.1 Where can I find documentation that addresses security issues for MySQL? 3206

A.9.2 What is the default authentication plugin in MySQL 5.6? 3206
A.9.3 Does MySQL 5.6 have native support for SSL? 3206
A.9.4 Is SSL support built into MySQL binaries, or must I recompile the binary myself to enable it? 3206
A.9.5 Does MySQL 5.6 have built-in authentication against LDAP directories? 3206
A.9.6 Does MySQL 5.6 include support for Roles Based Access Control (RBAC)? 3206

A.9.1. Where can I find documentation that addresses security issues for MySQL?

The best place to start is Chapter 6, *Security*.

Other portions of the MySQL Documentation which you may find useful with regard to specific security concerns include the following:

- Section 6.1.1, “Security Guidelines”.
- Section 6.1.3, “Making MySQL Secure Against Attackers”.
- Section B.3.3.2, “How to Reset the Root Password”.
- Section 6.1.5, “How to Run MySQL as a Normal User”.
- UDF Security Precautions.
- Section 6.1.4, “Security-Related mysqld Options and Variables”.
- Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”.
- Section 2.10, “Postinstallation Setup and Testing”.
- Section 6.3, “Using Encrypted Connections”.

A.9.2. What is the default authentication plugin in MySQL 5.6?

The default authentication plugin in MySQL 5.6 is `mysql_native_password`. For information about this plugin, see Section 6.4.1.1, “Native Pluggable Authentication”. For general information about pluggable authentication and other available authentication plugins, see Section 6.2.11, “Pluggable Authentication”, and Section 6.4.1, “Authentication Plugins”.

A.9.3. Does MySQL 5.6 have native support for SSL?

Most 5.6 binaries have support for SSL connections between the client and server. See Section 6.3, “Using Encrypted Connections”.

You can also tunnel a connection using SSH, if (for example) the client application does not support SSL connections. For an example, see Section 6.3.5, “Connecting to MySQL Remotely from Windows with SSH”.

A.9.4. Is SSL support built into MySQL binaries, or must I recompile the binary myself to enable it?

Most 5.6 binaries have SSL enabled for client/server connections that are secured, authenticated, or both. See Section 6.3, “Using Encrypted Connections”.

A.9.5. Does MySQL 5.6 have built-in authentication against LDAP directories?

The Enterprise edition includes a PAM Authentication Plugin that supports authentication against an LDAP directory.

A.9.6. Does MySQL 5.6 include support for Roles Based Access Control (RBAC)?

Not at this time.

A.10 MySQL 5.6 FAQ: NDB Cluster

In the following section, we answer questions that are frequently asked about NDB Cluster and the `NDB` storage engine.

A.10.1 Which versions of the MySQL software support NDB Cluster? Do I have to compile from source? 3207
A.10.2 What do “NDB” and “NDBCLUSTER” mean? 3208
A.10.3 What is the difference between using NDB Cluster versus using MySQL Replication? 3209
A.10.4 Do I need any special networking to run NDB Cluster? How do computers in a cluster communicate? 3209
A.10.5 How many computers do I need to run an NDB Cluster, and why? 3209
A.10.6 What do the different computers do in an NDB Cluster? 3209
A.10.7 When I run the `SHOW` command in the NDB Cluster management client, I see a line of output that looks like this: 3210
A.10.8 With which operating systems can I use NDB Cluster? 3210
A.10.9 What are the hardware requirements for running NDB Cluster? 3210
A.10.10 How much RAM do I need to use NDB Cluster? Is it possible to use disk memory at all? . 3211
A.10.11 What file systems can I use with NDB Cluster? What about network file systems or network shares? 3212
A.10.12 Can I run NDB Cluster nodes inside virtual machines (such as those created by VMWare, VirtualBox, Parallels, or Xen)? 3212
A.10.13 I am trying to populate an NDB Cluster database. The loading process terminates prematurely and I get an error message like this one: 3212
A.10.14 NDB Cluster uses TCP/IP. Does this mean that I can run it over the Internet, with one or more nodes in remote locations? 3213
A.10.15 Do I have to learn a new programming or query language to use NDB Cluster? 3213
A.10.16 What programming languages and APIs are supported by NDB Cluster? 3213
A.10.17 Does NDB Cluster include any management tools? 3213
A.10.18 How do I find out what an error or warning message means when using NDB Cluster? ... 3214
A.10.19 Is NDB Cluster transaction-safe? What isolation levels are supported? 3214
A.10.20 What storage engines are supported by NDB Cluster? 3214
A.10.21 In the event of a catastrophic failure— for example, the whole city loses power *and* my UPS fails—would I lose all my data? 3214
A.10.22 Is it possible to use `FULLTEXT` indexes with NDB Cluster? 3214
A.10.23 Can I run multiple nodes on a single computer? 3214
A.10.24 Can I add data nodes to an NDB Cluster without restarting it? 3215
A.10.25 Are there any limitations that I should be aware of when using NDB Cluster? 3215
A.10.26 Does NDB Cluster support foreign keys? 3216
A.10.27 How do I import an existing MySQL database into an NDB Cluster? 3216
A.10.28 How do NDB Cluster nodes communicate with one another? 3216
A.10.29 What is an *arbitrator*? 3216
A.10.30 What data types are supported by NDB Cluster? 3216
A.10.31 How do I start and stop NDB Cluster? 3217
A.10.32 What happens to NDB Cluster data when the NDB Cluster is shut down? 3218
A.10.33 Is it a good idea to have more than one management node for an NDB Cluster? 3218
A.10.34 Can I mix different kinds of hardware and operating systems in one NDB Cluster? 3218
A.10.35 Can I run two data nodes on a single host? Two SQL nodes? 3218
A.10.36 Can I use host names with NDB Cluster? 3218
A.10.37 Does NDB Cluster support IPv6? 3218
A.10.38 How do I handle MySQL users in an NDB Cluster having multiple MySQL servers? 3218
A.10.39 How do I continue to send queries in the event that one of the SQL nodes fails? 3219
A.10.40 How do I back up and restore an NDB Cluster? 3219
A.10.41 What is an “angel process”? 3219

A.10.1 Which versions of the MySQL software support NDB Cluster? Do I have to compile from source?

NDB Cluster is not supported in standard MySQL Server 5.6 releases. Instead, MySQL NDB Cluster is provided as a separate product. Available NDB Cluster release series include the following:

- **NDB Cluster 7.2.** This series is no longer supported for new deployments or maintained. Users of NDB Cluster 7.2 should upgrade to a newer release series as soon as possible. We recommend that new deployments use the latest NDB Cluster 8.0 release.
- **NDB Cluster 7.3.** This series is a previous General Availability (GA) version of NDB Cluster, still available for production use, although we recommend that new deployments use the latest NDB Cluster 8.0 release. The most recent NDB Cluster 7.3 release can be obtained from https://dev.mysql.com/downloads/cluster/.
- **NDB Cluster 7.4.** This series is a previous General Availability (GA) version of NDB Cluster, still available for production use, although we recommend that new deployments use the latest NDB Cluster 8.0 release. The most recent NDB Cluster 7.4 release can be obtained from https://dev.mysql.com/downloads/cluster/.
- **NDB Cluster 7.5.** This series is a previous General Availability (GA) version of NDB Cluster, still available for production use, although we recommend that new deployments use the latest NDB Cluster 7.6 release. The latest NDB Cluster 7.5 releases can be obtained from https://dev.mysql.com/downloads/cluster/.
- **NDB Cluster 7.6.** This series is a previous General Availability (GA) version of NDB Cluster, still available for production use, although we recommend that new deployments use the latest NDB Cluster 8.0 release. The latest NDB Cluster 7.6 releases can be obtained from https://dev.mysql.com/downloads/cluster/.
- **NDB Cluster 8.0.** This series is the most recent General Availability (GA) version of NDB Cluster, based on version 8.0 of the `NDB` storage engine and MySQL Server 8.0. NDB Cluster 8.0 is available for production use; new deployments intended for production should use the latest GA release in this series, which is currently NDB Cluster 8.0.23. You can obtain the most recent NDB Cluster 8.0 release from https://dev.mysql.com/downloads/cluster/. For information about new features and other important changes in this series, see What is New in NDB Cluster.

You can obtain and compile NDB Cluster from source (see Section 18.2.1.4, “Building NDB Cluster from Source on Linux”, and Section 18.2.2.2, “Compiling and Installing NDB Cluster from Source on Windows”), but for all but the most specialized cases, we recommend using one of the following installers provided by Oracle that is appropriate to your operating platform and circumstances:

- The web-based NDB Cluster Auto-Installer (works on all platforms supported by `NDB`)
- Linux binary release (`tar.gz` file)
- Linux RPM package
- Linux `.deb` file
- Windows binary “no-install” release
- Windows MSI Installer

Installation packages may also be available from your platform's package management system.

You can determine whether your MySQL Server has `NDB` support using one of the statements `SHOW VARIABLES LIKE 'have_%'`, `SHOW ENGINES`, or `SHOW PLUGINS`.

A.10.2 What do “NDB” and “NDBCLUSTER” mean?

“NDB” stands for “**N**etwork **D**ata**b**ase”. `NDB` and `NDBCLUSTER` are both names for the storage engine that enables clustering support with MySQL. `NDB` is preferred, but either name is correct.

A.10.3: What is the difference between using NDB Cluster versus using MySQL Replication?

In traditional MySQL replication, a source MySQL server updates one or more replicas. Transactions are committed sequentially, and a slow transaction can cause the replica to lag behind the source. This means that if the source fails, it is possible that the replica might not have recorded the last few transactions. If a transaction-safe engine such as `InnoDB` is being used, a transaction is either completed on the replica or not applied at all, but replication does not guarantee that all data on the source and the replica remains consistent at all times. In NDB Cluster, all data nodes are kept in synchrony, and a transaction committed by any one data node is committed for all data nodes. In the event of a data node failure, all remaining data nodes remain in a consistent state.

In short, whereas standard MySQL replication is *asynchronous*, NDB Cluster is *synchronous*.

Asynchronous replication is also available in NDB Cluster. *NDB Cluster Replication* (also sometimes known as “geo-replication”) includes the capability to replicate both between two NDB Clusters, and from an NDB Cluster to a non-Cluster MySQL server. See Section 18.6, “NDB Cluster Replication”.

A.10.4: Do I need any special networking to run NDB Cluster? How do computers in a cluster communicate?

NDB Cluster is intended to be used in a high-bandwidth environment, with computers connecting using TCP/IP. Its performance depends directly upon the connection speed between the cluster's computers. The minimum connectivity requirements for NDB Cluster include a typical 100-megabit Ethernet network or the equivalent. We recommend you use gigabit Ethernet whenever available.

A.10.5: How many computers do I need to run an NDB Cluster, and why?

A minimum of three computers is required to run a viable cluster. However, the minimum *recommended* number of computers in an NDB Cluster is four: one each to run the management and SQL nodes, and two computers to serve as data nodes. The purpose of the two data nodes is to provide redundancy; the management node must run on a separate machine to guarantee continued arbitration services in the event that one of the data nodes fails.

To provide increased throughput and high availability, you should use multiple SQL nodes (MySQL Servers connected to the cluster). It is also possible (although not strictly necessary) to run multiple management servers.

A.10.6: What do the different computers do in an NDB Cluster?

An NDB Cluster has both a physical and logical organization, with computers being the physical elements. The logical or functional elements of a cluster are referred to as *nodes*, and a computer housing a cluster node is sometimes referred to as a *cluster host*. There are three types of nodes, each corresponding to a specific role within the cluster. These are:

- **Management node.** This node provides management services for the cluster as a whole, including startup, shutdown, backups, and configuration data for the other nodes. The management node server is implemented as the application `ndb_mgmd`; the management client used to control NDB Cluster is `ndb_mgm`. See Section 18.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”, and Section 18.4.5, “`ndb_mgm` — The NDB Cluster Management Client”, for information about these programs.

- **Data node.** This type of node stores and replicates data. Data node functionality is handled by instances of the `NDB` data node process `ndbd`. For more information, see Section 18.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”.

- **SQL node.** This is simply an instance of MySQL Server (`mysqld`) that is built with support for the `NDBCLUSTER` storage engine and started with the `--ndb-cluster` option to enable the engine and the `--ndb-connectstring` option to enable it to connect to an NDB Cluster

management server. For more about these options, see MySQL Server Options for NDB Cluster.

Note

An *API node* is any application that makes direct use of Cluster data nodes for data storage and retrieval. An SQL node can thus be considered a type of API node that uses a MySQL Server to provide an SQL interface to the Cluster. You can write such applications (that do not depend on a MySQL Server) using the NDB API, which supplies a direct, object-oriented transaction and scanning interface to NDB Cluster data; see NDB Cluster API Overview: The NDB API, for more information.

A.10.7. When I run the `SHOW` command in the NDB Cluster management client, I see a line of output that looks like this:

```
id=2    @10.100.10.32  (Version: 8.0.23-ndb-8.0.23 Nodegroup: 0, *)
```

What does the `*` mean? How is this node different from the others?

The simplest answer is, “It's not something you can control, and it's nothing that you need to worry about in any case, unless you're a software engineer writing or analyzing the NDB Cluster source code”.

If you don't find that answer satisfactory, here's a longer and more technical version:

A number of mechanisms in NDB Cluster require distributed coordination among the data nodes. These distributed algorithms and protocols include global checkpointing, DDL (schema) changes, and node restart handling. To make this coordination simpler, the data nodes “elect” one of their number to act as leader. There is no user-facing mechanism for influencing this selection, which is completely automatic; the fact that it *is* automatic is a key part of NDB Cluster's internal architecture.

When a node acts as the “leader” for any of these mechanisms, it is usually the point of coordination for the activity, and the other nodes act as “followers”, carrying out their parts of the activity as directed by the leader. If the node acting as leader fails, then the remaining nodes elect a new leader. Tasks in progress that were being coordinated by the old leader may either fail or be continued by the new leader, depending on the actual mechanism involved.

It is possible for some of these different mechanisms and protocols to have different leader nodes, but in general the same leader is chosen for all of them. The node indicated as the leader in the output of `SHOW` in the management client is known internally as the `DICT` manager, responsible for coordinating DDL and metadata activity.

NDB Cluster is designed in such a way that the choice of leader has no discernible effect outside the cluster itself. For example, the current leader does not have significantly higher CPU or resource usage than the other data nodes, and failure of the leader should not have a significantly different impact on the cluster than the failure of any other data node.

A.10.8. With which operating systems can I use NDB Cluster?

NDB Cluster is supported on most Unix-like operating systems. NDB Cluster is also supported in production settings on Microsoft Windows operating systems.

For more detailed information concerning the level of support which is offered for NDB Cluster on various operating system versions, operating system distributions, and hardware platforms, please refer to https://www.mysql.com/support/supportedplatforms/cluster.html.

A.10.9. What are the hardware requirements for running NDB Cluster?

NDB Cluster should run on any platform for which NDB-enabled binaries are available. For data nodes and API nodes, faster CPUs and more memory are likely to improve performance, and 64-bit CPUs are likely to be more effective than 32-bit processors. There must be sufficient memory on machines used for data nodes to hold each node's share of the database (see *How much RAM do I Need?* for more information). For a computer which is used only for running the NDB Cluster management server, the requirements are minimal; a common desktop PC (or the equivalent) is generally sufficient for this task. Nodes can communicate through the standard TCP/IP network and hardware. They can also use the high-speed SCI protocol; however, special networking hardware and software are required to use SCI (see Section 18.3.4, "Using High-Speed Interconnects with NDB Cluster").

A.10.10. How much RAM do I need to use NDB Cluster? Is it possible to use disk memory at all?

NDB Cluster was originally implemented as in-memory only, but all versions currently available also provide the ability to store NDB Cluster on disk. See Section 18.5.10, "NDB Cluster Disk Data Tables", for more information.

For in-memory NDB tables, you can use the following formula for obtaining a rough estimate of how much RAM is needed for each data node in the cluster:

```
(SizeofDatabase × NumberOfReplicas × 1.1 ) / NumberOfDataNodes
```

To calculate the memory requirements more exactly requires determining, for each table in the cluster database, the storage space required per row (see Section 11.6, "Data Type Storage Requirements", for details), and multiplying this by the number of rows. You must also remember to account for any column indexes as follows:

- Each primary key or hash index created for an NDBCLUSTER table requires 21−25 bytes per record. These indexes use IndexMemory.
- Each ordered index requires 10 bytes storage per record, using DataMemory.
- Creating a primary key or unique index also creates an ordered index, unless this index is created with USING HASH. In other words:
  - A primary key or unique index on a Cluster table normally takes up 31 to 35 bytes per record.
  - However, if the primary key or unique index is created with USING HASH, then it requires only 21 to 25 bytes per record.

Creating NDB Cluster tables with USING HASH for all primary keys and unique indexes generally causes table updates to run more quickly—in some cases by a much as 20 to 30 percent faster than updates on tables where USING HASH was not used in creating primary and unique keys. This is due to the fact that less memory is required (because no ordered indexes are created), and that less CPU must be utilized (because fewer indexes must be read and possibly updated). However, it also means that queries that could otherwise use range scans must be satisfied by other means, which can result in slower selects.

When calculating Cluster memory requirements, you may find useful the ndb_size.pl utility which is available in recent MySQL 5.6 releases. This Perl script connects to a current (non-Cluster) MySQL database and creates a report on how much space that database would require if it used the NDBCLUSTER storage engine. For more information, see Section 18.4.27, "ndb_size.pl — NDBCLUSTER Size Requirement Estimator".

It is especially important to keep in mind that *every NDB Cluster table must have a primary key*. The NDB storage engine creates a primary key automatically if none is defined; this primary key is created without USING HASH.

You can determine how much memory is being used for storage of NDB Cluster data and indexes at any given time using the REPORT MEMORYUSAGE command in the ndb_mgm client;

see Section 18.5.1, “Commands in the NDB Cluster Management Client”, for more information. In addition, warnings are written to the cluster log when 80% of available `DataMemory` or (prior to NDB 7.6) `IndexMemory` is in use, and again when usage reaches 90%, 99%, and 100%.

A.10.11. What file systems can I use with NDB Cluster? What about network file systems or network shares?

Generally, any file system that is native to the host operating system should work well with NDB Cluster. If you find that a given file system works particularly well (or not so especially well) with NDB Cluster, we invite you to discuss your findings in the NDB Cluster Forums.

For Windows, we recommend that you use `NTFS` file systems for NDB Cluster, just as we do for standard MySQL. We do not test NDB Cluster with `FAT` or `VFAT` file systems. Because of this, we do not recommend their use with MySQL or NDB Cluster.

NDB Cluster is implemented as a shared-nothing solution; the idea behind this is that the failure of a single piece of hardware should not cause the failure of multiple cluster nodes, or possibly even the failure of the cluster as a whole. For this reason, the use of network shares or network file systems is not supported for NDB Cluster. This also applies to shared storage devices such as SANs.

A.10.12. Can I run NDB Cluster nodes inside virtual machines (such as those created by VMWare, VirtualBox, Parallels, or Xen)?

NDB Cluster is supported for use in virtual machines. We currently support and test using Oracle VM.

Some NDB Cluster users have successfully deployed NDB Cluster using other virtualization products; in such cases, Oracle can provide NDB Cluster support, but issues specific to the virtual environment must be referred to that product's vendor.

A.10.13. I am trying to populate an NDB Cluster database. The loading process terminates prematurely and I get an error message like this one:

```
ERROR 1114: The table 'my_cluster_table' is full
```

Why is this happening?

The cause is very likely to be that your setup does not provide sufficient RAM for all table data and all indexes, *including the primary key required by the `NDB` storage engine and automatically created in the event that the table definition does not include the definition of a primary key*.

It is also worth noting that all data nodes should have the same amount of RAM, since no data node in a cluster can use more memory than the least amount available to any individual data node. For example, if there are four computers hosting Cluster data nodes, and three of these have 3GB of RAM available to store Cluster data while the remaining data node has only 1GB RAM, then each data node can devote at most 1GB to NDB Cluster data and indexes.

In some cases it is possible to get `Table is full` errors in MySQL client applications even when `ndb_mgm -e "ALL REPORT MEMORYUSAGE"` shows significant free `DataMemory`. You can force `NDB` to create extra partitions for NDB Cluster tables and thus have more memory available for hash indexes by using the `MAX_ROWS` option for `CREATE TABLE`. In general, setting `MAX_ROWS` to twice the number of rows that you expect to store in the table should be sufficient.

For similar reasons, you can also sometimes encounter problems with data node restarts on nodes that are heavily loaded with data. The `MinFreePct` parameter can help with this issue by reserving a portion (5% by default) of `DataMemory` and (prior to NDB 7.6) `IndexMemory` for use in restarts. This reserved memory is not available for storing `NDB` tables or data.

A.10.14. NDB Cluster uses TCP/IP. Does this mean that I can run it over the Internet, with one or more nodes in remote locations?

It is *very* unlikely that a cluster would perform reliably under such conditions, as NDB Cluster was designed and implemented with the assumption that it would be run under conditions guaranteeing dedicated high-speed connectivity such as that found in a LAN setting using 100 Mbps or gigabit Ethernet—preferably the latter. We neither test nor warrant its performance using anything slower than this.

Also, it is extremely important to keep in mind that communications between the nodes in an NDB Cluster are not secure; they are neither encrypted nor safeguarded by any other protective mechanism. The most secure configuration for a cluster is in a private network behind a firewall, with no direct access to any Cluster data or management nodes from outside. (For SQL nodes, you should take the same precautions as you would with any other instance of the MySQL server.) For more information, see Section 18.5.16, “NDB Cluster Security Issues”.

A.10.15. Do I have to learn a new programming or query language to use NDB Cluster?

No. Although some specialized commands are used to manage and configure the cluster itself, only standard (My)SQL statements are required for the following operations:

- Creating, altering, and dropping tables
- Inserting, updating, and deleting table data
- Creating, changing, and dropping primary and unique indexes

Some specialized configuration parameters and files are required to set up an NDB Cluster—see Section 18.3.3, “NDB Cluster Configuration Files”, for information about these.

A few simple commands are used in the NDB Cluster management client (`ndb_mgm`) for tasks such as starting and stopping cluster nodes. See Section 18.5.1, “Commands in the NDB Cluster Management Client”.

A.10.16. What programming languages and APIs are supported by NDB Cluster?

NDB Cluster supports the same programming APIs and languages as the standard MySQL Server, including ODBC, .Net, the MySQL C API, and numerous drivers for popular scripting languages such as PHP, Perl, and Python. NDB Cluster applications written using these APIs behave similarly to other MySQL applications; they transmit SQL statements to a MySQL Server (in the case of NDB Cluster, an SQL node), and receive responses containing rows of data. For more information about these APIs, see Chapter 23, *Connectors and APIs*.

NDB Cluster also supports application programming using the NDB API, which provides a low-level C++ interface to NDB Cluster data without needing to go through a MySQL Server. See The NDB API. In addition, many `NDBCLUSTER` management functions are exposed by the C-language MGM API; see The MGM API, for more information.

NDB Cluster also supports Java application programming using ClusterJ, which supports a domain object model of data using sessions and transactions. See Java and NDB Cluster, for more information.

In addition, NDB Cluster provides support for `memcached`, allowing developers to access data stored in NDB Cluster using the `memcached` interface; for more information, see ndbmemcache—Memcache API for NDB Cluster (NO LONGER SUPPORTED).

NDB Cluster also includes adapters supporting NoSQL applications written against `Node.js`, with NDB Cluster as the data store. See MySQL NoSQL Connector for JavaScript, for more information.

A.10.17. Does NDB Cluster include any management tools?

NDB Cluster includes a command line client for performing basic management functions. See Section 18.4.5, “`ndb_mgm` — The NDB Cluster Management Client”, and Section 18.5.1, “Commands in the NDB Cluster Management Client”.

NDB Cluster 7.6 and earlier are also supported by MySQL Cluster Manager, a separate product providing an advanced command line interface that can automate many NDB Cluster management tasks such as rolling restarts and configuration changes. Beginning with version 1.4.8, MySQL Cluster Manager also provides experimental support for NDB Cluster 8.0. For more information about MySQL Cluster Manager, see MySQL™ Cluster Manager 1.4.8 User Manual.

NDB Cluster also provides a graphical, browser-based Auto-Installer for setting up and deploying NDB Cluster, as part of the NDB Cluster software distribution. For more information, see Section 18.2.8, “The NDB Cluster Auto-Installer (No longer supported)”.

A.10.18. How do I find out what an error or warning message means when using NDB Cluster?

There are two ways in which this can be done:

- From within the `mysql` client, use `SHOW ERRORS` or `SHOW WARNINGS` immediately upon being notified of the error or warning condition.
- From a system shell prompt, use `perror --ndb error_code`.

A.10.19. Is NDB Cluster transaction-safe? What isolation levels are supported?

Yes. For tables created with the `NDB` storage engine, transactions are supported. Currently, NDB Cluster supports only the `READ COMMITTED` transaction isolation level.

A.10.20. What storage engines are supported by NDB Cluster?

NDB Cluster requires the `NDB` storage engine. That is, in order for a table to be shared between nodes in an NDB Cluster, the table must be created using `ENGINE=NDB` (or the equivalent option `ENGINE=NDBCLUSTER`).

It is possible to create tables using other storage engines (such as `InnoDB` or `MyISAM`) on a MySQL server being used with NDB Cluster, but since these tables do not use `NDB`, they do not participate in clustering; each such table is strictly local to the individual MySQL server instance on which it is created.

An NDB Cluster cannot be created using the `InnoDB` storage engine. For information about the differences between the `NDB` and `InnoDB` storage engines, see Section 18.1.6, “MySQL Server Using InnoDB Compared with NDB Cluster”.

A.10.21. In the event of a catastrophic failure— for example, the whole city loses power *and* my UPS fails —would I lose all my data?

All committed transactions are logged. Therefore, although it is possible that some data could be lost in the event of a catastrophe, this should be quite limited. Data loss can be further reduced by minimizing the number of operations per transaction. (It is not a good idea to perform large numbers of operations per transaction in any case.)

A.10.22. Is it possible to use `FULLTEXT` indexes with NDB Cluster?

`FULLTEXT` indexing is currently supported only by the `InnoDB` and `MyISAM` storage engines. See Section 12.10, “Full-Text Search Functions”, for more information.

A.10.23. Can I run multiple nodes on a single computer?

It is possible but not always advisable. One of the chief reasons to run a cluster is to provide redundancy. To obtain the full benefits of this redundancy, each node should reside on a separate machine. If you place multiple nodes on a single machine and that machine fails, you

lose all of those nodes. For this reason, if you do run multiple data nodes on a single machine, it is *extremely* important that they be set up in such a way that the failure of this machine does not cause the loss of all the data nodes in a given node group.

Given that NDB Cluster can be run on commodity hardware loaded with a low-cost (or even no-cost) operating system, the expense of an extra machine or two is well worth it to safeguard mission-critical data. It also worth noting that the requirements for a cluster host running a management node are minimal. This task can be accomplished with a 300 MHz Pentium or equivalent CPU and sufficient RAM for the operating system, plus a small amount of overhead for the `ndb_mgmd` and `ndb_mgm` processes.

It is acceptable to run multiple cluster data nodes on a single host that has multiple CPUs, cores, or both. The NDB Cluster distribution also provides a multithreaded version of the data node binary intended for use on such systems. For more information, see Section 18.4.3, “`ndbmtd` — The NDB Cluster Data Node Daemon (Multi-Threaded)”.

It is also possible in some cases to run data nodes and SQL nodes concurrently on the same machine; how well such an arrangement performs is dependent on a number of factors such as number of cores and CPUs as well as the amount of disk and memory available to the data node and SQL node processes, and you must take these factors into account when planning such a configuration.

A.10.24. Can I add data nodes to an NDB Cluster without restarting it?

It is possible to add new data nodes to a running NDB Cluster without taking the cluster offline. For more information, see Section 18.5.7, “Adding NDB Cluster Data Nodes Online”.

For other types of NDB Cluster nodes, a rolling restart is all that is required (see Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”).

A.10.25. Are there any limitations that I should be aware of when using NDB Cluster?

Limitations on `NDB` tables in MySQL NDB Cluster include the following:

- Temporary tables are not supported; a `CREATE TEMPORARY TABLE` statement using `ENGINE=NDB` or `ENGINE=NDBCLUSTER` fails with an error.
- The only types of user-defined partitioning supported for `NDBCLUSTER` tables are `KEY` and `LINEAR KEY`. Trying to create an `NDB` table using any other partitioning type fails with an error.
- `FULLTEXT` indexes are not supported.
- Index prefixes are not supported. Only complete columns may be indexed.
- Spatial indexes are not supported (although spatial columns can be used). See Section 11.4, “Spatial Data Types”.
- Support for partial transactions and partial rollbacks is comparable to that of other transactional storage engines such as `InnoDB` that can roll back individual statements.
- The maximum number of attributes allowed per table is 512. Attribute names cannot be any longer than 31 characters. For each table, the maximum combined length of the table and database names is 122 characters.
- Priot to NDB 8.0, the maximum size for a table row is 14 kilobytes, not counting `BLOB` values. In NDB 8.0, this maximum is increased to 30000 bytes. See Section 18.1.7.5, “Limits Associated with Database Objects in NDB Cluster”, for more information.

  There is no set limit for the number of rows per `NDB` table. Limits on table size depend on a number of factors, in particular on the amount of RAM available to each data node.

For a complete listing of limitations in NDB Cluster, see Section 18.1.7, “Known Limitations of NDB Cluster”. See also Section 18.1.7.11, “Previous NDB Cluster Issues Resolved in NDB Cluster 7.3”.

A.10.26. Does NDB Cluster support foreign keys?

NDB Cluster provides support for foreign key constraints which is comparable to that found in the `InnoDB` storage engine; see Section 1.7.3.2, “FOREIGN KEY Constraints”, for more detailed information, as well as Section 13.1.17.5, “FOREIGN KEY Constraints”. Applications requiring foreign key support should use NDB Cluster 7.3, 7.4, 7.5, or later.

A.10.27. How do I import an existing MySQL database into an NDB Cluster?

You can import databases into NDB Cluster much as you would with any other version of MySQL. Other than the limitations mentioned elsewhere in this FAQ, the only other special requirement is that any tables to be included in the cluster must use the `NDB` storage engine. This means that the tables must be created with `ENGINE=NDB` or `ENGINE=NDBCLUSTER`.

It is also possible to convert existing tables that use other storage engines to `NDBCLUSTER` using one or more `ALTER TABLE` statement. However, the definition of the table must be compatible with the `NDBCLUSTER` storage engine prior to making the conversion. In MySQL 5.6, an additional workaround is also required; see Section 18.1.7, “Known Limitations of NDB Cluster”, for details.

A.10.28. How do NDB Cluster nodes communicate with one another?

Cluster nodes can communicate through any of three different transport mechanisms: TCP/IP, SHM (shared memory), and SCI (Scalable Coherent Interface). Where available, SHM is used by default between nodes residing on the same cluster host; however, this is considered experimental. SCI is a high-speed (1 gigabit per second and higher), high-availability protocol used in building scalable multi-processor systems; it requires special hardware and drivers. See Section 18.3.4, “Using High-Speed Interconnects with NDB Cluster”, for more about using SCI as a transport mechanism for NDB Cluster.

A.10.29. What is an *arbitrator*?

If one or more data nodes in a cluster fail, it is possible that not all cluster data nodes are able to “see” one another. In fact, it is possible that two sets of data nodes might become isolated from one another in a network partitioning, also known as a “split-brain” scenario. This type of situation is undesirable because each set of data nodes tries to behave as though it is the entire cluster. An arbitrator is required to decide between the competing sets of data nodes.

When all data nodes in at least one node group are alive, network partitioning is not an issue, because no single subset of the cluster can form a functional cluster on its own. The real problem arises when no single node group has all its nodes alive, in which case network partitioning (the “split-brain” scenario) becomes possible. Then an arbitrator is required. All cluster nodes recognize the same node as the arbitrator, which is normally the management server; however, it is possible to configure any of the MySQL Servers in the cluster to act as the arbitrator instead. The arbitrator accepts the first set of cluster nodes to contact it, and tells the remaining set to shut down. Arbitrator selection is controlled by the `ArbitrationRank` configuration parameter for MySQL Server and management server nodes. You can also use the `ArbitrationRank` configuration parameter to control the arbitrator selection process. For more information about these parameters, see Section 18.3.3.5, “Defining an NDB Cluster Management Server”.

The role of arbitrator does not in and of itself impose any heavy demands upon the host so designated, and thus the arbitrator host does not need to be particularly fast or to have extra memory especially for this purpose.

A.10.30. What data types are supported by NDB Cluster?

NDB Cluster supports all of the usual MySQL data types, including those associated with MySQL's spatial extensions; however, the `NDB` storage engine does not support spatial indexes. (Spatial indexes are supported only by `MyISAM`; see Section 11.4, “Spatial Data Types”, for more information.) In addition, there are some differences with regard to indexes when used with `NDB` tables.

Note

NDB Cluster Disk Data tables (that is, tables created with `TABLESPACE ... STORAGE DISK ENGINE=NDB` or `TABLESPACE ... STORAGE DISK ENGINE=NDBCLUSTER`) have only fixed-width rows. This means that (for example) each Disk Data table record containing a `VARCHAR(255)` column requires space for 255 characters (as required for the character set and collation being used for the table), regardless of the actual number of characters stored therein.

See Section 18.1.7, “Known Limitations of NDB Cluster”, for more information about these issues.

A.10.31. How do I start and stop NDB Cluster?

It is necessary to start each node in the cluster separately, in the following order:

1. Start the management node, using the `ndb_mgmd` command.

   You must include the `-f` or `--config-file` option to tell the management node where its configuration file can be found.

2. Start each data node with the `ndbd` command.

   Each data node must be started with the `-c` or `--ndb-connectstring` option so that the data node knows how to connect to the management server.

3. Start each MySQL Server (SQL node) using your preferred startup script, such as `mysqld_safe`.

   Each MySQL Server must be started with the `--ndbcluster` and `--ndb-connectstring` options. These options cause `mysqld` to enable `NDBCLUSTER` storage engine support and how to connect to the management server.

Each of these commands must be run from a system shell on the machine housing the affected node. (You do not have to be physically present at the machine—a remote login shell can be used for this purpose.) You can verify that the cluster is running by starting the `NDB` management client `ndb_mgm` on the machine housing the management node and issuing the `SHOW` or `ALL STATUS` command.

To shut down a running cluster, issue the command `SHUTDOWN` in the management client. Alternatively, you may enter the following command in a system shell:

```
shell> ndb_mgm -e "SHUTDOWN"
```

(The quotation marks in this example are optional, since there are no spaces in the command string following the `-e` option; in addition, the `SHUTDOWN` command, like other management client commands, is not case-sensitive.)

Either of these commands causes the `ndb_mgm`, `ndb_mgm`, and any `ndbd` processes to terminate gracefully. MySQL servers running as SQL nodes can be stopped using `mysqladmin shutdown`.

For more information, see Section 18.5.1, “Commands in the NDB Cluster Management Client”, and Section 18.2.6, “Safe Shutdown and Restart of NDB Cluster”.

MySQL Cluster Manager and the NDB Cluster Auto-Installer provide additional ways to handle starting ansd stopping of NDB Cluster nodes. See MySQL™ Cluster Manager 1.4.8 User Manual, and The NDB Cluster Auto-Installer (NDB 7.6) (No longer supported), for more information about these tools.

A.10.32. What happens to NDB Cluster data when the NDB Cluster is shut down?

The data that was held in memory by the cluster's data nodes is written to disk, and is reloaded into memory the next time that the cluster is started.

A.10.33. Is it a good idea to have more than one management node for an NDB Cluster?

It can be helpful as a fail-safe. Only one management node controls the cluster at any given time, but it is possible to configure one management node as primary, and one or more additional management nodes to take over in the event that the primary management node fails.

See Section 18.3.3, “NDB Cluster Configuration Files”, for information on how to configure NDB Cluster management nodes.

A.10.34. Can I mix different kinds of hardware and operating systems in one NDB Cluster?

Yes, as long as all machines and operating systems have the same “endianness” (all big-endian or all little-endian).

It is also possible to use software from different NDB Cluster releases on different nodes. However, we support such use only as part of a rolling upgrade procedure (see Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”).

A.10.35. Can I run two data nodes on a single host? Two SQL nodes?

Yes, it is possible to do this. In the case of multiple data nodes, it is advisable (but not required) for each node to use a different data directory. If you want to run multiple SQL nodes on one machine, each instance of `mysqld` must use a different TCP/IP port.

Running data nodes and SQL nodes together on the same host is possible, but you should be aware that the `ndbd` or `ndbmtd` processes may compete for memory with `mysqld`.

A.10.36. Can I use host names with NDB Cluster?

Yes, it is possible to use DNS and DHCP for cluster hosts. However, if your application requires “five nines” availability, you should use fixed (numeric) IP addresses, since making communication between Cluster hosts dependent on services such as DNS and DHCP introduces additional potential points of failure.

A.10.37. Does NDB Cluster support IPv6?

IPv6 is supported for connections between SQL nodes (MySQL servers), but connections between all other types of NDB Cluster nodes must use IPv4.

In practical terms, this means that you can use IPv6 for replication between NDB Clusters, but connections between nodes in the same NDB Cluster must use IPv4. For more information, see Section 18.6.3, “Known Issues in NDB Cluster Replication”.

A.10.38. How do I handle MySQL users in an NDB Cluster having multiple MySQL servers?

MySQL user accounts and privileges are normally not automatically propagated between different MySQL servers accessing the same NDB Cluster. MySQL NDB Cluster provides support for distributed privileges, which you can enable by following a procedure provided in the documentation; see Section 18.5.12, “Distributed Privileges Using Shared Grant Tables”, for more information.

Important

The mechanism for handling users distributed or shared between NDB Cluster SQL nodes has changed significantly in NDB 8.0; this implementation is not compatible with that in NDB 7.6 and earlier. See Distributed MySQL Privileges with NDB_STORED_USER, for details.

A.10.39 How do I continue to send queries in the event that one of the SQL nodes fails?

MySQL NDB Cluster does not provide any sort of automatic failover between SQL nodes. Your application must be prepared to handle the loss of SQL nodes and to fail over between them.

A.10.40 How do I back up and restore an NDB Cluster?

You can use the NDB Cluster native backup and restore functionality in the NDB management client and the `ndb_restore` program. See Section 18.5.8, “Online Backup of NDB Cluster”, and Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”.

You can also use the traditional functionality provided for this purpose in `mysqldump` and the MySQL server. See Section 4.5.4, “`mysqldump` — A Database Backup Program”, for more information.

A.10.41 What is an “angel process”?

This process monitors and, if necessary, attempts to restart the data node process. If you check the list of active processes on your system after starting `ndbd`, you can see that there are actually 2 processes running by that name, as shown here (we omit the output from `ndb_mgmd` and `ndbd` for brevity):

```
shell> ./ndb_mgmd

shell> ps aux | grep ndb
me      23002  0.0  0.0 122948  3104 ?        Ssl  14:14   0:00 ./ndb_mgmd
me      23025  0.0  0.0   5284   820 pts/2    S+   14:14   0:00 grep ndb

shell> ./ndbd -c 127.0.0.1 --initial

shell> ps aux | grep ndb
me      23002  0.0  0.0 123080  3356 ?        Ssl  14:14   0:00 ./ndb_mgmd
me      23096  0.0  0.0  35876  2036 ?        Ss   14:14   0:00 ./ndbmtd -c 127.0.0.1 --initial
me      23097  1.0  2.4 524116 91096 ?        Sl   14:14   0:00 ./ndbmtd -c 127.0.0.1 --initial
me      23168  0.0  0.0   5284   812 pts/2    R+   14:15   0:00 grep ndb
```

The `ndbd` process showing `0.0` for both memory and CPU usage is the angel process (although it actually does use a very small amount of each). This process merely checks to see if the main `ndbd` or `ndbmtd` process (the primary data node process which actually handles the data) is running. If permitted to do so (for example, if the `StopOnError` configuration parameter is set to `false`), the angel process tries to restart the primary data node process.

A.11 MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets

This set of Frequently Asked Questions derives from the experience of MySQL's Support and Development groups in handling many inquiries about CJK (Chinese-Japanese-Korean) issues.

A.11.1 What CJK character sets are available in MySQL? 3220
A.11.2 I have inserted CJK characters into my table. Why does `SELECT` display them as “?” characters? 3221
A.11.3 What problems should I be aware of when working with the Big5 Chinese character set? .. 3223
A.11.4 Why do Japanese character set conversions fail? 3223
A.11.5 What should I do if I want to convert SJIS `81CA` to `cp932`? 3224
A.11.6 How does MySQL represent the Yen (¥) sign? 3224

A.11.7 Of what issues should I be aware when working with Korean character sets in MySQL? 3224
A.11.8 Why do I get `Incorrect string value` error messages? 3224
A.11.9 Why does my GUI front end or browser display CJK characters incorrectly in my application using Access, PHP, or another API? 3225
A.11.10 I've upgraded to MySQL 5.6. How can I revert to behavior like that in MySQL 4.0 with regard to character sets? 3226
A.11.11 Why do some `LIKE` and `FULLTEXT` searches with CJK characters fail? 3227
A.11.12 How do I know whether character *X* is available in all character sets? 3227
A.11.13 Why do CJK strings sort incorrectly in Unicode? (I) 3228
A.11.14 Why do CJK strings sort incorrectly in Unicode? (II) 3229
A.11.15 Why are my supplementary characters rejected by MySQL? 3230
A.11.16 Should “CJK” be “CJKV”? 3231
A.11.17 Does MySQL permit CJK characters to be used in database and table names? 3231
A.11.18 Where can I find translations of the MySQL Manual into Chinese, Japanese, and Korean? 3231
A.11.19 Where can I get help with CJK and related issues in MySQL? 3231

A.11.1 What CJK character sets are available in MySQL?

The list of CJK character sets may vary depending on your MySQL version. For example, the `gb18030` character set is not supported prior to MySQL 5.7.4. However, since the name of the applicable language appears in the `DESCRIPTION` column for every entry in the `INFORMATION_SCHEMA.CHARACTER_SETS` table, you can obtain a current list of all the non-Unicode CJK character sets using this query:

```
mysql> SELECT CHARACTER_SET_NAME, DESCRIPTION
       FROM INFORMATION_SCHEMA.CHARACTER_SETS
       WHERE DESCRIPTION LIKE '%Chin%'
       OR DESCRIPTION LIKE '%Japanese%'
       OR DESCRIPTION LIKE '%Korean%'
       ORDER BY CHARACTER_SET_NAME;
+--------------------+---------------------------------+
| CHARACTER_SET_NAME | DESCRIPTION                     |
+--------------------+---------------------------------+
| big5               | Big5 Traditional Chinese        |
| cp932              | SJIS for Windows Japanese       |
| eucjpms            | UJIS for Windows Japanese       |
| euckr              | EUC-KR Korean                   |
| gb18030            | China National Standard GB18030 |
| gb2312             | GB2312 Simplified Chinese       |
| gbk                | GBK Simplified Chinese          |
| sjis               | Shift-JIS Japanese              |
| ujis               | EUC-JP Japanese                 |
+--------------------+---------------------------------+
```

(For more information, see Section 21.2, “The INFORMATION_SCHEMA CHARACTER_SETS Table”.)

MySQL supports three variants of the *GB* (*Guojia Biaozhun*, or *National Standard*, or *Simplified Chinese*) character sets which are official in the People's Republic of China: `gb2312`, `gbk`, and (as of MySQL 5.7.4) `gb18030`.

Sometimes people try to insert `gbk` characters into `gb2312`, and it works most of the time because `gbk` is a superset of `gb2312`. But eventually they try to insert a rarer Chinese character and it does not work. (For an example, see Bug #16072).

Here, we try to clarify exactly what characters are legitimate in `gb2312` or `gbk`, with reference to the official documents. Please check these references before reporting `gb2312` or `gbk` bugs:

- The MySQL `gbk` character set is in reality “Microsoft code page 936”. This differs from the official `gbk` for characters `A1A4` (middle dot), `A1AA` (em dash), `A6E0-A6F5`, and `A8BB-A8C0`.

- For a listing of `gbk`/Unicode mappings, see http://www.unicode.org/Public/MAPPINGS/VENDORS/MICSFT/WINDOWS/CP936.TXT.

It is also possible to store CJK characters in Unicode character sets, although the available collations may not sort characters quite as you expect:

- The `utf8` and `ucs2` character sets support the characters from Unicode Basic Multilingual Plane (BMP). These characters have code point values between `U+0000` and `U+FFFF`.
- The `utf8mb4`, `utf16`, `utf16le`, and `utf32` character sets support BMP characters, as well as supplementary characters that lie outside the BMP. Supplementary characters have code point values between `U+10000` and `U+10FFFF`.

The collation used for a Unicode character set determines the ability to sort (that is, distinguish) characters in the set:

- Collations based on Unicode Collation Algorithm (UCA) 4.0.0 distinguish only BMP characters.
- Collations based on UCA 5.2.0 or 9.0.0 distinguish BMP and supplementary characters.
- Non-UCA collations may not distinguish all Unicode characters. For example, the `utf8mb4` default collation is `utf8mb4_general_ci`, which distinguishes only BMP characters.

Moreover, distinguishing characters is not the same as ordering them per the conventions of a given CJK language. Currently, MySQL has only one CJK-specific UCA collation, `gb18030_unicode_520_ci` (which requires use of the non-Unicode `gb18030` character set).

For information about Unicode collations and their differentiating properties, including collation properties for supplementary characters, see Section 10.10.1, “Unicode Character Sets”.

A.11.2. I have inserted CJK characters into my table. Why does `SELECT` display them as “?” characters?

This problem is usually due to a setting in MySQL that does not match the settings for the application program or the operating system. Here are some common steps for correcting these types of issues:

- *Be certain of what MySQL version you are using*.

  Use the statement `SELECT VERSION();` to determine this.

- *Make sure that the database is actually using the desired character set*.

  People often think that the client character set is always the same as either the server character set or the character set used for display purposes. However, both of these are false assumptions. You can make sure by checking the result of `SHOW CREATE TABLE tablename` or, better yet, by using this statement:

```
SELECT character_set_name, collation_name
    FROM information_schema.columns
    WHERE table_schema = your_database_name
        AND table_name = your_table_name
        AND column_name = your_column_name;
```

- *Determine the hexadecimal value of the character or characters that are not being displayed correctly*.

  You can obtain this information for a column *`column_name`* in the table *`table_name`* using the following query:

```
SELECT HEX(column_name)
FROM table_name;
```

`3F` is the encoding for the `?` character; this means that `?` is the character actually stored in the column. This most often happens because of a problem converting a particular character from your client character set to the target character set.

- *Make sure that a round trip is possible. When you select `literal` (or `_introducer hexadecimal-value`), do you obtain `literal` as a result*?

  For example, the Japanese Katakana character *Pe* (`ペ`) exists in all CJK character sets, and has the code point value (hexadecimal coding) `0x30da`. To test a round trip for this character, use this query:

  ```
  SELECT 'ペ' AS `ペ`;         /* or SELECT _ucs2 0x30da; */
  ```

  If the result is not also `ペ`, the round trip failed.

  For bug reports regarding such failures, we might ask you to follow up with `SELECT HEX('ペ');`. Then we can determine whether the client encoding is correct.

- *Make sure that the problem is not with the browser or other application, rather than with MySQL*.

  Use the `mysql` client program to accomplish this task. If `mysql` displays characters correctly but your application does not, your problem is probably due to system settings.

  To determine your settings, use the `SHOW VARIABLES` statement, whose output should resemble what is shown here:

  ```
  mysql> SHOW VARIABLES LIKE 'char%';
  +--------------------------+----------------------------------------+
  | Variable_name            | Value                                  |
  +--------------------------+----------------------------------------+
  | character_set_client     | utf8                                   |
  | character_set_connection | utf8                                   |
  | character_set_database   | latin1                                 |
  | character_set_filesystem | binary                                 |
  | character_set_results    | utf8                                   |
  | character_set_server     | latin1                                 |
  | character_set_system     | utf8                                   |
  | character_sets_dir       | /usr/local/mysql/share/mysql/charsets/ |
  +--------------------------+----------------------------------------+
  ```

  These are typical character-set settings for an international-oriented client (notice the use of `utf8` Unicode) connected to a server in the West (`latin1` is a West Europe character set).

  Although Unicode (usually the `utf8` variant on Unix, and the `ucs2` variant on Windows) is preferable to Latin, it is often not what your operating system utilities support best. Many Windows users find that a Microsoft character set, such as `cp932` for Japanese Windows, is suitable.

  If you cannot control the server settings, and you have no idea what setting your underlying computer uses, try changing to a common character set for the country that you're in (`euckr` = Korea; `gb18030`, `gb2312` or `gbk` = People's Republic of China; `big5` = Taiwan; `sjis`, `ujis`, `cp932`, or `eucjpms` = Japan; `ucs2` or `utf8` = anywhere). Usually it is necessary to change only the client and connection and results settings. The `SET NAMES`. statement changes all three at once. For example:

  ```
  SET NAMES 'big5';
  ```

  Once the setting is correct, you can make it permanent by editing `my.cnf` or `my.ini`. For example you might add lines looking like these:

  ```
  [mysqld]
  character-set-server=big5
  ```

```
[client]
default-character-set=big5
```

It is also possible that there are issues with the API configuration setting being used in your application; see *Why does my GUI front end or browser not display CJK characters correctly...?* for more information.

A.11.3: What problems should I be aware of when working with the Big5 Chinese character set?

MySQL supports the Big5 character set which is common in Hong Kong and Taiwan (Republic of China). The MySQL `big5` character set is in reality Microsoft code page 950, which is very similar to the original `big5` character set.

A feature request for adding `HKSCS` extensions has been filed. People who need this extension may find the suggested patch for Bug #13577 to be of interest.

A.11.4: Why do Japanese character set conversions fail?

MySQL supports the `sjis`, `ujis`, `cp932`, and `eucjpms` character sets, as well as Unicode. A common need is to convert between character sets. For example, there might be a Unix server (typically with `sjis` or `ujis`) and a Windows client (typically with `cp932`).

In the following conversion table, the `ucs2` column represents the source, and the `sjis`, `cp932`, `ujis`, and `eucjpms` columns represent the destinations; that is, the last 4 columns provide the hexadecimal result when we use `CONVERT(ucs2)` or we assign a `ucs2` column containing the value to an `sjis`, `cp932`, `ujis`, or `eucjpms` column.

Character Name	ucs2	sjis	cp932	ujis	eucjpms
BROKEN BAR	00A6	3F	3F	8FA2C3	3F
FULLWIDTH BROKEN BAR	FFE4	3F	FA55	3F	8FA2
YEN SIGN	00A5	3F	3F	20	3F
FULLWIDTH YEN SIGN	FFE5	818F	818F	A1EF	3F
TILDE	007E	7E	7E	7E	7E
OVERLINE	203E	3F	3F	20	3F
HORIZONTAL BAR	2015	815C	815C	A1BD	A1BD
EM DASH	2014	3F	3F	3F	3F
REVERSE SOLIDUS	005C	815F	5C	5C	5C
FULLWIDTH REVERSE SOLIDUS	FF3C	3F	815F	3F	A1C0
WAVE DASH	301C	8160	3F	A1C1	3F
FULLWIDTH TILDE	FF5E	3F	8160	3F	A1C1
DOUBLE VERTICAL LINE	2016	8161	3F	A1C2	3F
PARALLEL TO	2225	3F	8161	3F	A1C2
MINUS SIGN	2212	817C	3F	A1DD	3F
FULLWIDTH HYPHEN-MINUS	FF0D	3F	817C	3F	A1DD
CENT SIGN	00A2	8191	3F	A1F1	3F
FULLWIDTH CENT SIGN	FFE0	3F	8191	3F	A1F1
POUND SIGN	00A3	8192	3F	A1F2	3F
FULLWIDTH POUND SIGN	FFE1	3F	8192	3F	A1F2
NOT SIGN	00AC	81CA	3F	A2CC	3F
FULLWIDTH NOT SIGN	FFE2	3F	81CA	3F	A2CC

Now consider the following portion of the table.

	ucs2	sjis	cp932
NOT SIGN	00AC	81CA	3F
FULLWIDTH NOT SIGN	FFE2	3F	81CA

This means that MySQL converts the `NOT SIGN` (Unicode `U+00AC`) to `sjis` code point `0x81CA` and to `cp932` code point `3F`. (`3F` is the question mark (“?”. This is what is always used when the conversion cannot be performed.)

A.11.5 What should I do if I want to convert SJIS `81CA` to `cp932`?

Our answer is: “?”. There are disadvantages to this, and many people would prefer a “loose” conversion, so that `81CA (NOT SIGN)` in `sjis` becomes `81CA (FULLWIDTH NOT SIGN)` in `cp932`.

A.11.6 How does MySQL represent the Yen (`¥`) sign?

A problem arises because some versions of Japanese character sets (both `sjis` and `euc`) treat `5C` as a *reverse solidus* (`\`, also known as a backslash), whereas others treat it as a yen sign (`¥`).

MySQL follows only one version of the JIS (Japanese Industrial Standards) standard description. In MySQL, *`5C` is always the reverse solidus (`\`)*.

A.11.7 Of what issues should I be aware when working with Korean character sets in MySQL?

In theory, while there have been several versions of the `euckr` (*Extended Unix Code Korea*) character set, only one problem has been noted. We use the “ASCII” variant of EUC-KR, in which the code point `0x5c` is REVERSE SOLIDUS, that is `\`, instead of the “KS-Roman” variant of EUC-KR, in which the code point `0x5c` is `WON SIGN` (`₩`). This means that you cannot convert Unicode `U+20A9` to `euckr`:

```
mysql> SELECT
           CONVERT('₩' USING euckr) AS euckr,
           HEX(CONVERT('₩' USING euckr)) AS hexeuckr;
+-------+----------+
| euckr | hexeuckr |
+-------+----------+
| ?     | 3F       |
+-------+----------+
```

A.11.8 Why do I get `Incorrect string value` error messages?

To see the problem, create a table with one Unicode (`ucs2`) column and one Chinese (`gb2312`) column.

```
mysql> CREATE TABLE ch
       (ucs2 CHAR(3) CHARACTER SET ucs2,
       gb2312 CHAR(3) CHARACTER SET gb2312);
```

In nonstrict SQL mode, try to place the rare character 狈 in both columns.

```
mysql> SET sql_mode = '';
mysql> INSERT INTO ch VALUES ('A狈B','A狈B');
Query OK, 1 row affected, 1 warning (0.00 sec)
```

The `INSERT` produces a warning. Use the following statement to see what it is:

```
mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
   Code: 1366
Message: Incorrect string value: '\xE6\xB1\x8CB' for column 'gb2312' at row 1
```

So it is a warning about the `gb2312` column only.

```
mysql> SELECT ucs2,HEX(ucs2),gb2312,HEX(gb2312) FROM ch;
+-------+--------------+--------+-------------+
| ucs2  | HEX(ucs2)    | gb2312 | HEX(gb2312) |
+-------+--------------+--------+-------------+
| A訷B | 00416C4C0042 | A?B    | 413F42      |
+-------+--------------+--------+-------------+
```

Several things need explanation here:

1. The 訷 character is not in the `gb2312` character set, as described earlier.
2. If you are using an old version of MySQL, you may see a different message.
3. A warning occurs rather than an error because MySQL is not set to use strict SQL mode. In nonstrict mode, MySQL tries to do what it can, to get the best fit, rather than give up. With strict SQL mode, the `Incorrect string value` message occurs as an error rather than a warning, and the `INSERT` fails.

A.11.9. Why does my GUI front end or browser display CJK characters incorrectly in my application using Access, PHP, or another API?

Obtain a direct connection to the server using the `mysql` client, and try the same query there. If `mysql` responds correctly, the trouble may be that your application interface requires initialization. Use `mysql` to tell you what character set or sets it uses with the statement `SHOW VARIABLES LIKE 'char%';`. If you are using Access, you are most likely connecting with Connector/ODBC. In this case, you should check Configuring Connector/ODBC. If, for example, you use `big5`, you would enter `SET NAMES 'big5'`. (In this case, no `;` character is required.) If you are using ASP, you might need to add `SET NAMES` in the code. Here is an example that has worked in the past:

```
<%
Session.CodePage=0
Dim strConnection
Dim Conn
strConnection="driver={MySQL ODBC 3.51 Driver};server=server;uid=username;" \
              & "pwd=password;database=database;stmt=SET NAMES 'big5';"
Set Conn = Server.CreateObject("ADODB.Connection")
Conn.Open strConnection
%>
```

In much the same way, if you are using any character set other than `latin1` with Connector/NET, you must specify the character set in the connection string. See Connector/NET Connections, for more information.

If you are using PHP, try this:

```
<?php
  $link = new mysqli($host, $usr, $pwd, $db);

  if( mysqli_connect_errno() )
  {
    printf("Connect failed: %s\n", mysqli_connect_error());
    exit();
  }

  $link->query("SET NAMES 'utf8'");
?>
```

In this case, we used `SET NAMES` to change `character_set_client`, `character_set_connection`, and `character_set_results`.

Another issue often encountered in PHP applications has to do with assumptions made by the browser. Sometimes adding or changing a `<meta>` tag suffices to correct the problem: for example, to insure that the user agent interprets page content as `UTF-8`, include `<meta http-equiv="Content-Type" content="text/html; charset=utf-8">` in the `<head>` section of the HTML page.

If you are using Connector/J, see Using Character Sets and Unicode.

A.11.10 ve upgraded to MySQL 5.6. How can I revert to behavior like that in MySQL 4.0 with regard to character sets?

In MySQL Version 4.0, there was a single “global” character set for both server and client, and the decision as to which character to use was made by the server administrator. This changed starting with MySQL Version 4.1. What happens now is a “handshake”, as described in Section 10.4, “Connection Character Sets and Collations”:

> When a client connects, it sends to the server the name of the character set that it wants to use. The server uses the name to set the `character_set_client`, `character_set_results`, and `character_set_connection` system variables. In effect, the server performs a `SET NAMES` operation using the character set name.

The effect of this is that you cannot control the client character set by starting `mysqld` with `--character-set-server=utf8`. However, some Asian customers prefer the MySQL 4.0 behavior. To make it possible to retain this behavior, we added a `mysqld` switch, `--character-set-client-handshake`, which can be turned off with `--skip-character-set-client-handshake`. If you start `mysqld` with `--skip-character-set-client-handshake`, then, when a client connects, it sends to the server the name of the character set that it wants to use. However, *the server ignores this request from the client*.

By way of example, suppose that your favorite server character set is `latin1`. Suppose further that the client uses `utf8` because this is what the client's operating system supports. Start the server with `latin1` as its default character set:

```
mysqld --character-set-server=latin1
```

And then start the client with the default character set `utf8`:

```
mysql --default-character-set=utf8
```

The resulting settings can be seen by viewing the output of `SHOW VARIABLES`:

```
mysql> SHOW VARIABLES LIKE 'char%';
+--------------------------+----------------------------------------+
| Variable_name            | Value                                  |
+--------------------------+----------------------------------------+
| character_set_client     | utf8                                   |
| character_set_connection | utf8                                   |
| character_set_database   | latin1                                 |
| character_set_filesystem | binary                                 |
| character_set_results    | utf8                                   |
| character_set_server     | latin1                                 |
| character_set_system     | utf8                                   |
| character_sets_dir       | /usr/local/mysql/share/mysql/charsets/ |
+--------------------------+----------------------------------------+
```

Now stop the client, and stop the server using `mysqladmin`. Then start the server again, but this time tell it to skip the handshake like so:

```
mysqld --character-set-server=utf8 --skip-character-set-client-handshake
```

Start the client with `utf8` once again as the default character set, then display the resulting settings:

```
mysql> SHOW VARIABLES LIKE 'char%';
+--------------------------+---------------------------------------+
| Variable_name            | Value                                 |
+--------------------------+---------------------------------------+
| character_set_client     | latin1                                |
| character_set_connection | latin1                                |
| character_set_database   | latin1                                |
| character_set_filesystem | binary                                |
| character_set_results    | latin1                                |
| character_set_server     | latin1                                |
| character_set_system     | utf8                                  |
| character_sets_dir       | /usr/local/mysql/share/mysql/charsets/ |
+--------------------------+---------------------------------------+
```

As you can see by comparing the differing results from `SHOW VARIABLES`, the server ignores the client's initial settings if the `--skip-character-set-client-handshake` option is used.

A.11.11. Why do some `LIKE` and `FULLTEXT` searches with CJK characters fail?

For `LIKE` searches, there is a very simple problem with binary string column types such as `BINARY` and `BLOB`: we must know where characters end. With multibyte character sets, different characters might have different octet lengths. For example, in `utf8`, `A` requires one byte but `ペ` requires three bytes, as shown here:

```
+-------------------------+--------------------------+
| OCTET_LENGTH(_utf8 'A') | OCTET_LENGTH(_utf8 'ペ') |
+-------------------------+--------------------------+
|                       1 |                        3 |
+-------------------------+--------------------------+
```

If we do not know where the first character in a string ends, we do not know where the second character begins, in which case even very simple searches such as `LIKE '_A%'` fail. The solution is to use a nonbinary string column type defined to have the proper CJK character set. For example: `mycol TEXT CHARACTER SET sjis`. Alternatively, convert to a CJK character set before comparing.

This is one reason why MySQL cannot permit encodings of nonexistent characters. If it is not strict about rejecting bad input, it has no way of knowing where characters end.

For `FULLTEXT` searches, we must know where words begin and end. With Western languages, this is rarely a problem because most (if not all) of these use an easy-to-identify word boundary: the space character. However, this is not usually the case with Asian writing. We could use arbitrary halfway measures, like assuming that all Han characters represent words, or (for Japanese) depending on changes from Katakana to Hiragana due to grammatical endings. However, the only sure solution requires a comprehensive word list, which means that we would have to include a dictionary in the server for each Asian language supported. This is simply not feasible.

A.11.12. How do I know whether character *X* is available in all character sets?

The majority of simplified Chinese and basic nonhalfwidth Japanese Kana characters appear in all CJK character sets. The following stored procedure accepts a `UCS-2` Unicode character, converts it to other character sets, and displays the results in hexadecimal.

```
DELIMITER //

CREATE PROCEDURE p_convert(ucs2_char CHAR(1) CHARACTER SET ucs2)
BEGIN

CREATE TABLE tj
             (ucs2 CHAR(1) character set ucs2,
              utf8 CHAR(1) character set utf8,
              big5 CHAR(1) character set big5,
              cp932 CHAR(1) character set cp932,
              eucjpms CHAR(1) character set eucjpms,
```

```
              euckr CHAR(1) character set euckr,
              gb2312 CHAR(1) character set gb2312,
              gbk CHAR(1) character set gbk,
              sjis CHAR(1) character set sjis,
              ujis CHAR(1) character set ujis);

INSERT INTO tj (ucs2) VALUES (ucs2_char);

UPDATE tj SET utf8=ucs2,
              big5=ucs2,
              cp932=ucs2,
              eucjpms=ucs2,
              euckr=ucs2,
              gb2312=ucs2,
              gbk=ucs2,
              sjis=ucs2,
              ujis=ucs2;

/* If there are conversion problems, UPDATE produces warnings. */

SELECT hex(ucs2) AS ucs2,
       hex(utf8) AS utf8,
       hex(big5) AS big5,
       hex(cp932) AS cp932,
       hex(eucjpms) AS eucjpms,
       hex(euckr) AS euckr,
       hex(gb2312) AS gb2312,
       hex(gbk) AS gbk,
       hex(sjis) AS sjis,
       hex(ujis) AS ujis
FROM tj;

DROP TABLE tj;

END//

DELIMITER ;
```

The input can be any single `ucs2` character, or it can be the code value (hexadecimal representation) of that character. For example, from Unicode's list of `ucs2` encodings and names (http://www.unicode.org/Public/UNIDATA/UnicodeData.txt), we know that the Katakana character *Pe* appears in all CJK character sets, and that its code value is `X'30DA'`. If we use this value as the argument to `p_convert()`, the result is as shown here:

```
mysql> CALL p_convert(X'30DA');
+------+--------+------+-------+---------+-------+--------+------+------+------+
| ucs2 | utf8   | big5 | cp932 | eucjpms | euckr | gb2312 | gbk  | sjis | ujis |
+------+--------+------+-------+---------+-------+--------+------+------+------+
| 30DA | E3839A | C772 | 8379  | A5DA    | ABDA  | A5DA   | A5DA | 8379 | A5DA |
+------+--------+------+-------+---------+-------+--------+------+------+------+
```

Since none of the column values is `3F` (that is, the question mark character, `?`), we know that every conversion worked.

A.11.13 Why do CJK strings sort incorrectly in Unicode? (I)

Note

The CJK sorting problems described here can occur for MySQL versions prior to MySQL 8.0. As of MySQL 8.0, they can be solved by using the `utf8mb4` character set and the `utf8mb4_ja_0900_as_cs` collation.

Sometimes people observe that the result of a `utf8_unicode_ci` or `ucs2_unicode_ci` search, or of an `ORDER BY` sort is not what they think a native would expect. Although we never rule out the possibility that there is a bug, we have found in the past that many people do not correctly read the standard table of weights for the Unicode Collation Algorithm. MySQL uses the tables found under http://www.unicode.org/Public/UCA/:

- UCA 4.0.0 table: http://www.unicode.org/Public/UCA/4.0.0/allkeys-4.0.0.txt

  This includes *xxx*_unicode_ci collations with no version number in the collation name.

- UCA 5.2.0 table: http://www.unicode.org/Public/UCA/5.2.0/allkeys.txt

  This includes collations with _520_ in the collation name.

- UCA 9.0.0 table: http://www.unicode.org/Public/UCA/9.0.0/allkeys.txt

  This includes collations with _0900_ in the collation name.

To handle newer UCA versions, we create new collations. We are very wary about changing ordering of existing collations because that affects indexes, which can bring about situations such as that reported in Bug #16526, illustrated as follows:

```
mysql> CREATE TABLE tj (s1 CHAR(1) CHARACTER SET utf8 COLLATE utf8_unicode_ci);
Query OK, 0 rows affected (0.05 sec)

mysql> INSERT INTO tj VALUES ('が'),('か');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql> SELECT * FROM tj WHERE s1 = 'か';
+------+
| s1   |
+------+
| が   |
| か   |
+------+
```

The character in the first result row is not the one that we searched for. Why did MySQL retrieve it? First we look for the Unicode code point value, which is possible by reading the hexadecimal number for the ucs2 version of the characters:

```
mysql> SELECT s1, HEX(CONVERT(s1 USING ucs2)) FROM tj;
+------+-----------------------------+
| s1   | HEX(CONVERT(s1 USING ucs2)) |
+------+-----------------------------+
| が   | 304C                        |
| か   | 304B                        |
+------+-----------------------------+
```

Now we search for 304B and 304C in the 4.0.0 allkeys table, and find these lines:

```
304B  ; [.1E57.0020.000E.304B] # HIRAGANA LETTER KA
304C  ; [.1E57.0020.000E.304B][.0000.0140.0002.3099] # HIRAGANA LETTER GA; QQCM
```

The official Unicode names (following the “#” mark) tell us the Japanese syllabary (Hiragana), the informal classification (letter, digit, or punctuation mark), and the Western identifier (KA or GA, which happen to be voiced and unvoiced components of the same letter pair). More importantly, the *primary weight* (the first hexadecimal number inside the square brackets) is 1E57 on both lines. For comparisons in both searching and sorting, MySQL pays attention to the primary weight only, ignoring all the other numbers. This means that we are sorting が and か correctly according to the Unicode specification. If we wanted to distinguish them, we'd have to use a non-UCA (Unicode Collation Algorithm) collation (utf8_bin or utf8_general_ci), or to compare the HEX() values, or use ORDER BY CONVERT(s1 USING sjis). Being correct “according to Unicode” is not enough, of course: the person who submitted the bug was equally correct. To solve this, we need another collation for Japanese according to the JIS X 4061 standard, in which voiced/unvoiced letter pairs like KA/GA are distinguishable for ordering purposes.

A.11.14 Why do CJK strings sort incorrectly in Unicode? (II)

Note

The CJK sorting problems described here can occur for MySQL versions prior to MySQL 8.0. As of MySQL 8.0, they can be solved by using the `utf8mb4` character set and the `utf8mb4_ja_0900_as_cs` collation.

If you are using Unicode (`ucs2` or `utf8`), and you know what the Unicode sort order is (see Section A.11, “MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”), but MySQL still seems to sort your table incorrectly, first verify the character set in the table definition:

```
mysql> SHOW CREATE TABLE t\G
******************** 1. row ******************
Table: t
Create Table: CREATE TABLE `t` (
`s1` char(1) CHARACTER SET ucs2 DEFAULT NULL
) ENGINE=MyISAM DEFAULT CHARSET=latin1
```

Since the character set for the column `s1` appears to be correct (`ucs2`), check what information the `INFORMATION_SCHEMA.COLUMNS` table can provide about this column:

```
mysql> SELECT COLUMN_NAME, CHARACTER_SET_NAME, COLLATION_NAME
       FROM INFORMATION_SCHEMA.COLUMNS
       WHERE COLUMN_NAME = 's1'
       AND TABLE_NAME = 't';
+-------------+--------------------+-----------------+
| COLUMN_NAME | CHARACTER_SET_NAME | COLLATION_NAME  |
+-------------+--------------------+-----------------+
| s1          | ucs2               | ucs2_general_ci |
+-------------+--------------------+-----------------+
```

(See Section 21.5, “The INFORMATION_SCHEMA COLUMNS Table”, for more information.)

You can see that the collation is `ucs2_general_ci` instead of `ucs2_unicode_ci`. The reason why this is so can be found using `SHOW CHARACTER SET`, as shown here:

```
mysql> SHOW CHARSET LIKE 'ucs2%';
+---------+--------------+-------------------+--------+
| Charset | Description   | Default collation | Maxlen |
+---------+--------------+-------------------+--------+
| ucs2    | UCS-2 Unicode | ucs2_general_ci   |      2 |
+---------+--------------+-------------------+--------+
```

For `ucs2` and `utf8`, the default collation is “general”. To specify a Unicode UCA collation, use `COLLATE ucs2_unicode_ci`, as shown in the preceding item.

A.11.15 Why are my supplementary characters rejected by MySQL?

Supplementary characters lie outside the Unicode *Basic Multilingual Plane / Plane 0*. BMP characters have code point values between `U+0000` and `U+FFFF`. Supplementary characters have code point values between `U+10000` and `U+10FFFF`.

To store supplementary characters, you must use a character set that permits them:

- The `utf8` and `ucs2` character sets support BMP characters only.

  The `utf8` character set permits only `UTF-8` characters that take up to three bytes. This has led to reports such as that found in Bug #12600, which we rejected as “not a bug”. With `utf8`, MySQL must truncate an input string when it encounters bytes that it does no understand. Otherwise, it is unknown how long the bad multibyte character is.

  One possible workaround is to use `ucs2` instead of `utf8`, in which case the “bad” characters are changed to question marks. However, no truncation takes place. You can also change the data type to `BLOB` or `BINARY`, which perform no validity checking.

- The `utf8mb4`, `utf16`, `utf16le`, and `utf32` character sets support BMP characters, as well as supplementary characters outside the BMP.

A.11.16. Should “CJK” be “CJKV”?

No. The term “CJKV” (*Chinese Japanese Korean Vietnamese*) refers to Vietnamese character sets which contain Han (originally Chinese) characters. MySQL supports the modern Vietnamese script with Western characters, but does not support the old Vietnamese script using Han characters.

As of MySQL 5.6, there are Vietnamese collations for Unicode character sets, as described in Section 10.10.1, “Unicode Character Sets”.

A.11.17. Does MySQL permit CJK characters to be used in database and table names?

Yes.

A.11.18. Where can I find translations of the MySQL Manual into Chinese, Japanese, and Korean?

The Japanese translation of the MySQL 5.6 manual can be downloaded from https://dev.mysql.com/doc/.

A.11.19. Where can I get help with CJK and related issues in MySQL?

The following resources are available:

- A listing of MySQL user groups can be found at https://wikis.oracle.com/display/mysql/List+of+MySQL+User+Groups.
- View feature requests relating to character set issues at http://tinyurl.com/y6xcuf.
- Visit the MySQL Character Sets, Collation, Unicode Forum. http://forums.mysql.com/ also provides foreign-language forums.

A.12 MySQL 5.6 FAQ: Connectors & APIs

For common questions, issues, and answers relating to the MySQL Connectors and other APIs, see the following areas of the Manual:

- C API Common Issues
- Common Problems with MySQL and PHP
- Connector/ODBC Notes and Tips
- Connector/NET Programming
- MySQL Connector/J 5.1 Developer Guide

A.13 MySQL 5.6 FAQ: C API, libmysql

Frequently asked questions about MySQL C API and libmysql.

A.13.1 What is “MySQL Native C API”? What are typical benefits and use cases? 3232
A.13.2 Which version of libmysql should I use? 3232
A.13.3 What if I want to use the “NoSQL” X DevAPI? 3232
A.13.4 How to I download libmysql? 3232
A.13.5 Where is the documentation? 3232
A.13.6 How do I report bugs? 3232

A.13.7 Is it possible to compile the library myself? 3232

A.13.1 What is “MySQL Native C API”? What are typical benefits and use cases?

libmysql is a C-based API that you can use in C applications to connect with the MySQL database server. It is also itself used as the foundation for drivers for standard database APIs like ODBC, Perl's DBI, and Python's DB API.

A.13.2 Which version of libmysql should I use?

For MySQL 8.0, 5.7, 5.6, and 5.5, we recommend libmysql 8.0.

A.13.3 What if I want to use the “NoSQL” X DevAPI?

For C-language and X DevApi Document Store for MySQL 8.0, we recommend MySQL Connector/C++. Connector/C++ 8.0 has compatible C headers. (This is not applicable to MySQL 5.7 or before.)

A.13.4 How to I download libmysql?

- Linux: The Client Utilities Package is available from the MySQL Community Server download page.
- Repos: The Client Utilities Package is available from the Yum, APT, SuSE repositories.
- Windows: The Client Utilities Package is available from Windows Installer.

A.13.5 Where is the documentation?

See MySQL 5.6 C API Developer Guide.

A.13.6 How do I report bugs?

Please report any bugs or inconsistencies you observe to our Bugs Database. Select the C API Client as shown.

A.13.7 Is it possible to compile the library myself?

Yes, you can download the libmysqlclient source code and compile it on your own. Here's an example:

```
$ git clone --depth 1 https://github.com/mysql/mysql-server
$ cd mysql-server
$ mkdir build
$ cd build
$ cmake .. -GNinja -DDOWNLOAD_BOOST=1 \
           -DWITH_BOOST=/tmp -DCMAKE_BUILD_TYPE=Release -DWITHOUT_SERVER=ON \
           -DWITH_SSL=system
$ ninja libmysqlclient.a
$ ls -la archive_output_directory/libmysqlclient.a
-rw-rw-r-- 1 kg kg 8,5M wrz 5 04:57 archive_output_directory/libmysqlclient.a
```

Note

This example uses https://ninja-build.org/ as a build system instead of make.

A.14 MySQL 5.6 FAQ: Replication

In the following section, we provide answers to questions that are most frequently asked about MySQL Replication.

A.14.1 Must the replica be connected to the source all the time? 3233

A.14.2 Must I enable networking on my source and replica to enable replication? 3233
A.14.3 How do I know how late a replica is compared to the source? In other words, how do I know the date of the last statement replicated by the replica? 3233
A.14.4 How do I force the source to block updates until the replica catches up? 3233
A.14.5 What issues should I be aware of when setting up two-way replication? 3234
A.14.6 How can I use replication to improve performance of my system? 3234
A.14.7 What should I do to prepare client code in my own applications to use performance-enhancing replication? 3234
A.14.8 When and how much can MySQL replication improve the performance of my system? 3234
A.14.9 How can I use replication to provide redundancy or high availability? 3235
A.14.10 How do I tell whether a replication source server is using statement-based or row-based binary logging format? 3236
A.14.11 How do I tell a replica to use row-based replication? 3236
A.14.12 How do I prevent `GRANT` and `REVOKE` statements from replicating to replica machines? .. 3236
A.14.13 Does replication work on mixed operating systems (for example, the source runs on Linux while replicas run on OS X and Windows)? 3236
A.14.14 Does replication work on mixed hardware architectures (for example, the source runs on a 64-bit machine while replicas run on 32-bit machines)? 3236

A.14.1 Must the replica be connected to the source all the time?

No, it does not. The replica can go down or stay disconnected for hours or even days, and then reconnect and catch up on updates. For example, you can set up a source/replica relationship over a dial-up link where the link is up only sporadically and for short periods of time. The implication of this is that, at any given time, the replica is not guaranteed to be in synchrony with the source unless you take some special measures.

To ensure that catchup can occur for a replica that has been disconnected, you must not remove binary log files from the source that contain information that has not yet been replicated to the replicas. Asynchronous replication can work only if the replica is able to continue reading the binary log from the point where it last read events.

A.14.2 Must I enable networking on my source and replica to enable replication?

Yes, networking must be enabled on the source and replica. If networking is not enabled, the replica cannot connect to the source and transfer the binary log. Verify that the `skip_networking` system variable has not been enabled in the configuration file for either server.

A.14.3 How do I know how late a replica is compared to the source? In other words, how do I know the date of the last statement replicated by the replica?

Check the `Seconds_Behind_Master` column in the output from `SHOW REPLICA | SLAVE STATUS`. See Section 17.1.5.1, “Checking Replication Status”.

When the replication SQL thread executes an event read from the source, it modifies its own time to the event timestamp. (This is why `TIMESTAMP` is well replicated.) In the `Time` column in the output of `SHOW PROCESSLIST`, the number of seconds displayed for the replication SQL thread is the number of seconds between the timestamp of the last replicated event and the real time of the replica machine. You can use this to determine the date of the last replicated event. Note that if your replica has been disconnected from the source for one hour, and then reconnects, you may immediately see large `Time` values such as 3600 for the replication SQL thread in `SHOW PROCESSLIST`. This is because the replica is executing statements that are one hour old. See Section 17.2.1, “Replication Threads”.

A.14.4 How do I force the source to block updates until the replica catches up?

Use the following procedure:

1. On the source, execute these statements:

```
mysql> FLUSH TABLES WITH READ LOCK;
mysql> SHOW MASTER STATUS;
```

Record the replication coordinates (the current binary log file name and position) from the output of the `SHOW` statement.

2. On the replica, issue the following statement, where the arguments to the `MASTER_POS_WAIT()` function are the replication coordinate values obtained in the previous step:

```
mysql> SELECT MASTER_POS_WAIT('log_name', log_pos);
```

The `SELECT` statement blocks until the replica reaches the specified log file and position. At that point, the replica is in synchrony with the source and the statement returns.

3. On the source, issue the following statement to enable the source to begin processing updates again:

```
mysql> UNLOCK TABLES;
```

A.14.5 What issues should I be aware of when setting up two-way replication?

MySQL replication currently does not support any locking protocol between source and replica to guarantee the atomicity of a distributed (cross-server) update. In other words, it is possible for client A to make an update to co-source 1, and in the meantime, before it propagates to co-source 2, client B could make an update to co-source 2 that makes the update of client A work differently than it did on co-source 1. Thus, when the update of client A makes it to co-source 2, it produces tables that are different from what you have on co-source 1, even after all the updates from co-source 2 have also propagated. This means that you should not chain two servers together in a two-way replication relationship unless you are sure that your updates can safely happen in any order, or unless you take care of mis-ordered updates somehow in the client code.

You should also realize that two-way replication actually does not improve performance very much (if at all) as far as updates are concerned. Each server must do the same number of updates, just as you would have a single server do. The only difference is that there is a little less lock contention because the updates originating on another server are serialized in one replication thread. Even this benefit might be offset by network delays.

A.14.6 How can I use replication to improve performance of my system?

Set up one server as the source and direct all writes to it. Then configure as many replicas as you have the budget and rackspace for, and distribute the reads among the source and the replicas. You can also start the replicas with the `--skip-innodb` option, enable the `low_priority_updates` system variable, and set the `delay_key_write` system variable to `ALL` to get speed improvements on the replica end. In this case, the replica uses nontransactional `MyISAM` tables instead of `InnoDB` tables to get more speed by eliminating transactional overhead.

A.14.7 What should I do to prepare client code in my own applications to use performance-enhancing replication?

See the guide to using replication as a scale-out solution, Section 17.3.4, “Using Replication for Scale-Out”.

A.14.8 When and how much can MySQL replication improve the performance of my system?

MySQL replication is most beneficial for a system that processes frequent reads and infrequent writes. In theory, by using a single-source/multiple-replica setup, you can scale the system by adding more replicas until you either run out of network bandwidth, or your update load grows to the point that the source cannot handle it.

To determine how many replicas you can use before the added benefits begin to level out, and how much you can improve performance of your site, you must know your query patterns, and determine empirically by benchmarking the relationship between the throughput for reads and writes on a typical source and a typical replica. The example here shows a rather simplified calculation of what you can get with replication for a hypothetical system. Let `reads` and `writes` denote the number of reads and writes per second, respectively.

Let's say that system load consists of 10% writes and 90% reads, and we have determined by benchmarking that `reads` is 1200 - 2 * `writes`. In other words, the system can do 1,200 reads per second with no writes, the average write is twice as slow as the average read, and the relationship is linear. Suppose that the source and each replica have the same capacity, and that we have one source and *N* replicas. Then we have for each server (source or replica):

`reads` = 1200 - 2 * `writes`

`reads` = 9 * `writes` / (*N* + 1) (reads are split, but writes replicated to all replicas)

9 * `writes` / (*N* + 1) + 2 * `writes` = 1200

`writes` = 1200 / (2 + 9/(*N* + 1))

The last equation indicates the maximum number of writes for *N* replicas, given a maximum possible read rate of 1,200 per second and a ratio of nine reads per write.

This analysis yields the following conclusions:

- If *N* = 0 (which means we have no replication), our system can handle about 1200/11 = 109 writes per second.
- If *N* = 1, we get up to 184 writes per second.
- If *N* = 8, we get up to 400 writes per second.
- If *N* = 17, we get up to 480 writes per second.
- Eventually, as *N* approaches infinity (and our budget negative infinity), we can get very close to 600 writes per second, increasing system throughput about 5.5 times. However, with only eight servers, we increase it nearly four times.

These computations assume infinite network bandwidth and neglect several other factors that could be significant on your system. In many cases, you may not be able to perform a computation similar to the one just shown that accurately predicts what happens on your system if you add *N* replicas. However, answering the following questions should help you decide whether and by how much replication may improve the performance of your system:

- What is the read/write ratio on your system?
- How much more write load can one server handle if you reduce the reads?
- For how many replicas do you have bandwidth available on your network?

A.14.9 How can I use replication to provide redundancy or high availability?

How you implement redundancy is entirely dependent on your application and circumstances. High-availability solutions (with automatic failover) require active monitoring and either custom scripts or third party tools to provide the failover support from the original MySQL server to the replica.

To handle the process manually, you should be able to switch from a failed source to a pre-configured replica by altering your application to talk to the new server or by adjusting the DNS for the MySQL server from the failed server to the new server.

For more information and some example solutions, see Section 17.3.7, “Switching Sources During Failover”.

A.14.10 How do I tell whether a replication source server is using statement-based or row-based binary logging format?

Check the value of the `binlog_format` system variable:

```
mysql> SHOW VARIABLES LIKE 'binlog_format';
```

The value shown is always one of `STATEMENT`, `ROW`, or `MIXED`. For `MIXED` mode, statement-based logging is used by default but replication switches automatically to row-based logging under certain conditions, such as unsafe statements. For information about when this may occur, see Section 5.4.4.3, “Mixed Binary Logging Format”.

A.14.11 How do I tell a replica to use row-based replication?

Replicas automatically know which format to use.

A.14.12 How do I prevent `GRANT` and `REVOKE` statements from replicating to replica machines?

Start the server with the `--replicate-wild-ignore-table=mysql.%` option to ignore replication for tables in the `mysql` database.

A.14.13 Does replication work on mixed operating systems (for example, the source runs on Linux while replicas run on OS X and Windows)?

Yes.

A.14.14 Does replication work on mixed hardware architectures (for example, the source runs on a 64-bit machine while replicas run on 32-bit machines)?

Yes.

A.15 MySQL 5.6 FAQ: MySQL Enterprise Thread Pool

A.15.1 What is the Thread Pool and what problem does it solve? 3236
A.15.2 How does the Thread Pool limit and manage concurrent sessions and transactions for optimal performance and throughput? 3237
A.15.3 How is the Thread Pool different from the client side Connection Pool? 3237
A.15.4 When should I use the Thread Pool? 3237
A.15.5 Are there recommended Thread Pool configurations? 3237

A.15.1 What is the Thread Pool and what problem does it solve?

The MySQL Thread Pool is a MySQL server plugin that extends the default connection-handling capabilities of the MySQL server to limit the number of concurrently executing statements/queries and transactions to ensure that each has sufficient CPU and memory resources to fulfill its task. For MySQL 5.6, the Thread Pool plugin is included in MySQL Enterprise Edition, a commercial product.

The default thread-handling model in MySQL Server executes statements using one thread per client connection. As more clients connect to the server and execute statements, overall performance degrades. The Thread Pool plugin provides an alternative thread-handling model designed to reduce overhead and improve performance. The Thread Pool plugin increases server performance by efficiently managing statement execution threads for large numbers of client connections, especially on modern multi-CPU/Core systems.

For more information, see Section 5.5.3, “MySQL Enterprise Thread Pool”.

A.15.2 How does the Thread Pool limit and manage concurrent sessions and transactions for optimal performance and throughput?

The Thread Pool uses a “divide and conquer” approach to limiting and balancing concurrency. Unlike the default connection handling of the MySQL Server, the Thread Pool separates connections and threads, so there is no fixed relationship between connections and the threads that execute statements received from those connections. The Thread Pool then manages client connections within configurable thread groups, where they are prioritized and queued based on the nature of the work they were submitted to accomplish.

For more information, see Section 5.5.3.3, “Thread Pool Operation”.

A.15.3 How is the Thread Pool different from the client side Connection Pool?

The MySQL Connection Pool operates on the client side to ensure that a MySQL client does not constantly connect to and disconnect from the MySQL server. It is designed to cache idle connections in the MySQL client for use by other users as they are needed. This minimizes the overhead and expense of establishing and tearing down connections as queries are submitted to the MySQL server. The MySQL Connection Pool has no visibility as to the query handling capabilities or load of the back-end MySQL server. By contrast, the Thread Pool operates on the MySQL server side and is designed to manage the execution of inbound concurrent connections and queries as they are received from the client connections accessing the back-end MySQL database. Because of the separation of duties, the MySQL Connection Pool and Thread Pool are orthogonal and can be used independent of each other.

MySQL Connection Pooling via the MySQL Connectors is covered in Chapter 23, *Connectors and APIs*.

A.15.4 When should I use the Thread Pool?

There are a few rules of thumb to consider for optimal Thread Pool use cases:

The MySQL `Threads_running` variable keeps track of the number of concurrent statements currently executing in the MySQL Server. If this variable consistently exceeds a region where the server won't operate optimally (usually going beyond 40 for InnoDB workloads), the Thread Pool should be beneficial, especially in extreme parallel overload situations.

If you are using the `innodb_thread_concurrency` to limit the number of concurrently executing statements, you should find that the Thread Pool solves the same problem, only better, by assigning connections to thread groups, then queuing executions based on transactional content, user defined designations, and so forth.

Lastly, if your workload comprises mainly short queries, the Thread Pool should be beneficial.

To learn more, see Section 5.5.3.4, “Thread Pool Tuning”.

A.15.5 Are there recommended Thread Pool configurations?

The Thread Pool has a number of user case driven configuration parameters that affect its performance. To learn about these and tips on tuning, see Section 5.5.3.4, “Thread Pool Tuning”.

A.16 MySQL 5.6 FAQ: InnoDB Change Buffer

A.16.1 What types of operations modify secondary indexes and result in change buffering? 3238
A.16.2 What is the benefit of the `InnoDB` change buffer? 3238
A.16.3 Does the change buffer support other types of indexes? 3238
A.16.4 How much space does `InnoDB` use for the change buffer? 3238
A.16.5 How do I determine the current size of the change buffer? 3238

A.16.6 When does change buffer merging occur? 3238
A.16.7 When is the change buffer flushed? 3239
A.16.8 When should the change buffer be used? 3239
A.16.9 When should the change buffer not be used? 3239
A.16.10 Where can I find additional information about the change buffer? 3239

A.16.1 What types of operations modify secondary indexes and result in change buffering?

`INSERT`, `UPDATE`, and `DELETE` operations can modify secondary indexes. If an affected index page is not in the buffer pool, the changes can be buffered in the change buffer.

A.16.2 What is the benefit of the `InnoDB` change buffer?

Buffering secondary index changes when secondary index pages are not in the buffer pool avoids expensive random access I/O operations that would be required to immediately read in affected index pages from disk. Buffered changes can be applied later, in batches, as pages are read into the buffer pool by other read operations.

A.16.3 Does the change buffer support other types of indexes?

No. The change buffer only supports secondary indexes. Clustered indexes, full-text indexes, and spatial indexes are not supported. Full-text indexes have their own caching mechanism.

A.16.4 How much space does `InnoDB` use for the change buffer?

Prior to the introduction of the `innodb_change_buffer_max_size` configuration option in MySQL 5.6, the maximum size of the on-disk change buffer in the system tablespace was 1/3 of the `InnoDB` buffer pool size.

In MySQL 5.6 and later, the `innodb_change_buffer_max_size` configuration option defines the maximum size of the change buffer as a percentage of the total buffer pool size. By default, `innodb_change_buffer_max_size` is set to 25. The maximum setting is 50.

`InnoDB` does not buffer an operation if it would cause the on-disk change buffer to exceed the defined limit.

Change buffer pages are not required to persist in the buffer pool and may be evicted by LRU operations.

A.16.5 How do I determine the current size of the change buffer?

The current size of the change buffer is reported by `SHOW ENGINE INNODB STATUS \G`, under the `INSERT BUFFER AND ADAPTIVE HASH INDEX` heading. For example:

```
-------------------------------------
INSERT BUFFER AND ADAPTIVE HASH INDEX
-------------------------------------
Ibuf: size 1, free list len 0, seg size 2, 0 merges
```

Relevant data points include:

- `size`: The number of pages used within the change buffer. Change buffer size is equal to `seg size - (1 + free list len)`. The `1 +` value represents the change buffer header page.
- `seg size`: The size of the change buffer, in pages.

For information about monitoring change buffer status, see Section 14.5.2, “Change Buffer”.

A.16.6 When does change buffer merging occur?

- When a page is read into the buffer pool, buffered changes are merged upon completion of the read, before the page is made available.

- Change buffer merging is performed as a background task. The `innodb_io_capacity` parameter sets an upper limit on the I/O activity performed by `InnoDB` background tasks such as merging data from the change buffer.
- A change buffer merge is performed during crash recovery. Changes are applied from the change buffer (in the system tablespace) to leaf pages of secondary indexes as index pages are read into the buffer pool.
- The change buffer is fully durable and can survive a system crash. Upon restart, change buffer merge operations resume as part of normal operations.
- A full merge of the change buffer can be forced as part of a slow server shutdown using `--innodb-fast-shutdown=0`.

A.16.7: When is the change buffer flushed?

Updated pages are flushed by the same flushing mechanism that flushes the other pages that occupy the buffer pool.

A.16.8: When should the change buffer be used?

The change buffer is a feature designed to reduce random I/O to secondary indexes as indexes grow larger and no longer fit in the `InnoDB` buffer pool. Generally, the change buffer should be used when the entire data set does not fit into the buffer pool, when there is substantial DML activity that modifies secondary index pages, or when there are lots of secondary indexes that are regularly changed by DML activity.

A.16.9: When should the change buffer not be used?

You might consider disabling the change buffer if the entire data set fits within the `InnoDB` buffer pool, if you have relatively few secondary indexes, or if you are using solid-state storage, where random reads are about as fast as sequential reads. Before making configuration changes, it is recommended that you run tests using a representative workload to determine if disabling the change buffer provides any benefit.

A.16.10: Where can I find additional information about the change buffer?

See Section 14.5.2, “Change Buffer”.

A.17 MySQL 5.6 FAQ: Virtualization Support

A.17.1 Is MySQL supported on virtualized environments such as Oracle VM, VMWare, Docker, Microsoft Hyper-V, or others? 3239

A.17.1: Is MySQL supported on virtualized environments such as Oracle VM, VMWare, Docker, Microsoft Hyper-V, or others?

MySQL is supported on virtualized environments, but is certified only for Oracle VM. Contact Oracle Support for more information.

Be aware of potential problems when using virtualization software. The usual ones are related to performance, performance degradations, slowness, or unpredictability of disk, I/O, network, and memory.

Appendix B Error Messages and Common Problems

Table of Contents

B.1 Error Message Sources and Elements 3241
B.2 Error Information Interfaces 3243
B.3 Problems and Common Errors 3244
B.3.1 How to Determine What Is Causing a Problem 3244
B.3.2 Common Errors When Using MySQL Programs 3245
B.3.3 Administration-Related Issues 3257
B.3.4 Query-Related Issues 3265
B.3.5 Optimizer-Related Issues 3271
B.3.6 Table Definition-Related Issues 3272
B.3.7 Known Issues in MySQL 3273

This appendix describes the types of error information MySQL provides and how to obtain information about them. The final section is for troubleshooting. It describes common problems and errors that may occur and potential resolutions.

Additional Resources

Other error-related documentation includes:

- Information about configuring where and how the server writes the error log: Section 5.4.2, “The Error Log”
- Information about the character set used for error messages: Section 10.6, “Error Message Character Set”
- Information about the language used for error messages: Section 10.12, “Setting the Error Message Language”
- Information about errors related to `InnoDB`: Section 14.21.4, “InnoDB Error Handling”
- Descriptions of the error messages that the MySQL server and client programs generate: MySQL 5.6 Error Message Reference

B.1 Error Message Sources and Elements

This section discusses how error messages originate within MySQL and the elements they contain.

- Error Message Sources
- Error Message Elements

Error Message Sources

Error messages can originate on the server side or the client side:

- On the server side, error messages may occur during the startup and shutdown processes, as a result of issues that occur during SQL statement execution, and so forth.
  - The MySQL server writes some error messages to its error log. These indicate issues of interest to database administrators or that require DBA action.
  - The server sends other error messages to client programs. These indicate issues pertaining only to a particular client. The MySQL client library takes errors received from the server and makes them available to the host client program.

- Client-side error messages are generated from within the MySQL client library, usually involving problems communicating with the server.

Example server-side error messages written to the error log:

- This message produced during the startup process provides a status or progress indicator:

```
2018-08-29 08:27:31 16664 [Note] Event Scheduler: scheduler
thread started with id 1
```

- This message indicates an issue that requires DBA action:

```
2018-10-02 03:20:39 0 [ERROR] Can't open the mysql.plugin
table. Please run mysql_upgrade to create it.
```

Example server-side error message sent to client programs, as displayed by the `mysql` client:

```
mysql> SELECT * FROM no_such_table;
ERROR 1146 (42S02): Table 'test.no_such_table' doesn't exist
```

Example client-side error message originating from within the client library, as displayed by the `mysql` client:

```
shell> mysql -h no-such-host
ERROR 2005 (HY000): Unknown MySQL server host 'no-such-host' (0)
```

Whether an error originates from within the client library or is received from the server, a MySQL client program may respond in varying ways. As just illustrated, the client may display the error message so the user can take corrective measures. The client may instead internally attempt to resolve or retry a failed operation, or take other action.

Error Message Elements

When an error occurs, error information includes several elements: an error code, SQLSTATE value, and message string. These elements have the following characteristics:

- Error code: This value is numeric. It is MySQL-specific and is not portable to other database systems.

  Each error number has a corresponding symbolic value. Examples:

  - The symbol for server error number `1146` is `ER_NO_SUCH_TABLE`.
  - The symbol for client error number `2005` is `CR_UNKNOWN_HOST`.

  Error codes are stable across General Availability (GA) releases of a given MySQL series. Before a series reaches GA status, new codes may still be under development and are subject to change.

- SQLSTATE value: This value is a five-character string (for example, `'42S02'`). SQLSTATE values are taken from ANSI SQL and ODBC and are more standardized than the numeric error codes. The first two characters of an SQLSTATE value indicate the error class:

  - Class = `'00'` indicates success.
  - Class = `'01'` indicates a warning.
  - Class = `'02'` indicates “not found.” This is relevant within the context of cursors and is used to control what happens when a cursor reaches the end of a data set. This condition also occurs for `SELECT ... INTO var_list` statements that retrieve no rows.
  - Class > `'02'` indicates an exception.

  For server-side errors, not all MySQL error numbers have corresponding SQLSTATE values. In these cases, `'HY000'` (general error) is used.

For client-side errors, the SQLSTATE value is always `'HY000'` (general error), so it is not meaningful for distinguishing one client error from another.

- Message string: This string provides a textual description of the error.

B.2 Error Information Interfaces

Error messages can originate on the server side or the client side, and each error message includes an error code, SQLSTATE value, and message string, as described in Section B.1, “Error Message Sources and Elements”. For lists of server-side, client-side, and global (shared between server and clients) errors, see MySQL 5.6 Error Message Reference.

For error checking from within programs, use error code numbers or symbols, not error message strings. Message strings do not change often, but it is possible. Also, if the database administrator changes the language setting, that affects the language of message strings; see Section 10.12, “Setting the Error Message Language”.

Error information in MySQL is available in the server error log, at the SQL level, from within client programs, and at the command line.

- Error Log
- SQL Error Message Interface
- Client Error Message Interface
- Command-Line Error Message Interface

Error Log

On the server side, some messages are intended for the error log. For information about configuring where and how the server writes the log, see Section 5.4.2, “The Error Log”.

Other server error messages are intended to be sent to client programs and are available as described in Client Error Message Interface.

SQL Error Message Interface

At the SQL level, there are several sources of error information in MySQL:

- SQL statement warning and error information is available through the `SHOW WARNINGS` and `SHOW ERRORS` statements. The `warning_count` system variable indicates the number of errors, warnings, and notes (with notes excluded if the `sql_notes` system variable is disabled). The `error_count` system variable indicates the number of errors. Its value excludes warnings and notes.
- The `GET DIAGNOSTICS` statement may be used to inspect the diagnostic information in the diagnostics area. See Section 13.6.7.3, “GET DIAGNOSTICS Statement”.
- `SHOW SLAVE STATUS` statement output includes information about replication errors occurring on replica servers.
- `SHOW ENGINE INNODB STATUS` statement output includes information about the most recent foreign key error if a `CREATE TABLE` statement for an `InnoDB` table fails.

Client Error Message Interface

Client programs receive errors from two sources:

- Errors that originate on the client side from within the MySQL client library.

- Errors that originate on the server side and are sent to the client by the server. These are received within the client library, which makes them available to the host client program.

Regardless of whether an error originates from within the client library or is received from the server, a MySQL client program obtains the error code, SQLSTATE value, message string, and other related information by calling C API functions in the client library:

- `mysql_errno()` returns the MySQL error code.
- `mysql_sqlstate()` returns the SQLSTATE value.
- `mysql_error()` returns the message string.
- `mysql_stmt_errno()`, `mysql_stmt_sqlstate()`, and `mysql_stmt_error()` are the corresponding error functions for prepared statements.
- `mysql_warning_count()` returns the number of errors, warnings, and notes for the most recent statement.

For descriptions of the client library error functions, see MySQL 5.6 C API Developer Guide.

A MySQL client program may respond to an error in varying ways. The client may display the error message so the user can take corrective measures, internally attempt to resolve or retry a failed operation, or take other action. For example, (using the `mysql` client), a failure to connect to the server might result in this message:

```
shell> mysql -h no-such-host
ERROR 2005 (HY000): Unknown MySQL server host 'no-such-host' (0)
```

Command-Line Error Message Interface

The `perror` program provides information from the command line about error numbers. See Section 4.8.1, “`perror` — Display MySQL Error Message Information”.

```
shell> perror 1231
MySQL error code 1231 (ER_WRONG_VALUE_FOR_VAR): Variable '%-.64s' can't
be set to the value of '%-.200s'
```

B.3 Problems and Common Errors

This section lists some common problems and error messages that you may encounter. It describes how to determine the causes of the problems and what to do to solve them.

B.3.1 How to Determine What Is Causing a Problem

When you run into a problem, the first thing you should do is to find out which program or piece of equipment is causing it:

- If you have one of the following symptoms, then it is probably a hardware problems (such as memory, motherboard, CPU, or hard disk) or kernel problem:
  - The keyboard does not work. This can normally be checked by pressing the Caps Lock key. If the Caps Lock light does not change, you have to replace your keyboard. (Before doing this, you should try to restart your computer and check all cables to the keyboard.)
  - The mouse pointer does not move.
  - The machine does not answer to a remote machine's pings.
  - Other programs that are not related to MySQL do not behave correctly.
  - Your system restarted unexpectedly. (A faulty user-level program should never be able to take down your system.)

In this case, you should start by checking all your cables and run some diagnostic tool to check your hardware! You should also check whether there are any patches, updates, or service packs for your operating system that could likely solve your problem. Check also that all your libraries (such as `glibc`) are up to date.

It is always good to use a machine with ECC memory to discover memory problems early.

- If your keyboard is locked up, you may be able to recover by logging in to your machine from another machine and executing `kbd_mode -a`.
- Please examine your system log file (`/var/log/messages` or similar) for reasons for your problem. If you think the problem is in MySQL, you should also examine MySQL's log files. See Section 5.4, “MySQL Server Logs”.
- If you do not think you have hardware problems, you should try to find out which program is causing problems. Try using `top`, `ps`, Task Manager, or some similar program, to check which program is taking all CPU or is locking the machine.
- Use `top`, `df`, or a similar program to check whether you are out of memory, disk space, file descriptors, or some other critical resource.
- If the problem is some runaway process, you can always try to kill it. If it does not want to die, there is probably a bug in the operating system.

If you have examined all other possibilities and concluded that the MySQL server or a MySQL client is causing the problem, it is time to create a bug report, see Section 1.6, “How to Report Bugs or Problems”. In the bug report, try to give a complete description of how the system is behaving and what you think is happening. Also state why you think that MySQL is causing the problem. Take into consideration all the situations described in this chapter. State any problems exactly how they appear when you examine your system. Use the “copy and paste” method for any output and error messages from programs and log files.

Try to describe in detail which program is not working and all symptoms you see. We have in the past received many bug reports that state only “the system does not work.” This provides us with no information about what could be the problem.

If a program fails, it is always useful to know the following information:

- Has the program in question made a segmentation fault (did it dump core)?
- Is the program taking up all available CPU time? Check with `top`. Let the program run for a while, it may simply be evaluating something computationally intensive.
- If the `mysqld` server is causing problems, can you get any response from it with `mysqladmin -u root ping` or `mysqladmin -u root processlist`?
- What does a client program say when you try to connect to the MySQL server? (Try with `mysql`, for example.) Does the client jam? Do you get any output from the program?

When sending a bug report, you should follow the outline described in Section 1.6, “How to Report Bugs or Problems”.

B.3.2 Common Errors When Using MySQL Programs

This section lists some errors that users frequently encounter when running MySQL programs. Although the problems show up when you try to run client programs, the solutions to many of the problems involves changing the configuration of the MySQL server.

B.3.2.1 Access denied

An `Access denied` error can have many causes. Often the problem is related to the MySQL accounts that the server permits client programs to use when connecting. See Section 6.2, “Access

Control and Account Management", and Section 6.2.14, "Troubleshooting Problems Connecting to MySQL".

B.3.2.2 Can't connect to [local] MySQL server

A MySQL client on Unix can connect to the `mysqld` server in two different ways: By using a Unix socket file to connect through a file in the file system (default `/tmp/mysql.sock`), or by using TCP/IP, which connects through a port number. A Unix socket file connection is faster than TCP/IP, but can be used only when connecting to a server on the same computer. A Unix socket file is used if you do not specify a host name or if you specify the special host name `localhost`.

If the MySQL server is running on Windows, you can connect using TCP/IP. If the server is started with the `named_pipe` system variable enabled, you can also connect with named pipes if you run the client on the host where the server is running. The name of the named pipe is `MySQL` by default. If you do not give a host name when connecting to `mysqld`, a MySQL client first tries to connect to the named pipe. If that does not work, it connects to the TCP/IP port. You can force the use of named pipes on Windows by using `.` as the host name.

The error (2002) `Can't connect to ...` normally means that there is no MySQL server running on the system or that you are using an incorrect Unix socket file name or TCP/IP port number when trying to connect to the server. You should also check that the TCP/IP port you are using has not been blocked by a firewall or port blocking service.

The error (2003) `Can't connect to MySQL server on '`*`server`*`' (10061)` indicates that the network connection has been refused. You should check that there is a MySQL server running, that it has network connections enabled, and that the network port you specified is the one configured on the server.

Start by checking whether there is a process named `mysqld` running on your server host. (Use `ps xa | grep mysqld` on Unix or the Task Manager on Windows.) If there is no such process, you should start the server. See Section 2.10.2, "Starting the Server".

If a `mysqld` process is running, you can check it by trying the following commands. The port number or Unix socket file name might be different in your setup. `host_ip` represents the IP address of the machine where the server is running.

```
shell> mysqladmin version
shell> mysqladmin variables
shell> mysqladmin -h `hostname` version variables
shell> mysqladmin -h `hostname` --port=3306 version
shell> mysqladmin -h host_ip version
shell> mysqladmin --protocol=SOCKET --socket=/tmp/mysql.sock version
```

Regarding the use of backticks rather than forward quotation marks with the `hostname` command; these cause the output of `hostname` (that is, the current host name) to be substituted into the `mysqladmin` command. If you have no `hostname` command or are running on Windows, you can manually type the host name of your machine (without backticks) following the `-h` option. You can also try `-h 127.0.0.1` to connect with TCP/IP to the local host.

Make sure that the server has not been configured to ignore network connections or (if you are attempting to connect remotely) that it has not been configured to listen only locally on its network interfaces. If the server was started with the `skip_networking` system variable enabled, it does accept TCP/IP connections at all. If the server was started with the `bind_address` system variable set to `127.0.0.1`, it listens for TCP/IP connections only locally on the loopback interface and does not accept remote connections.

Check to make sure that there is no firewall blocking access to MySQL. Your firewall may be configured on the basis of the application being executed, or the port number used by MySQL for communication (3306 by default). Under Linux or Unix, check your IP tables (or similar) configuration to ensure that the port has not been blocked. Under Windows, applications such as ZoneAlarm or Windows Firewall may need to be configured not to block the MySQL port.

Here are some reasons the `Can't connect to local MySQL server` error might occur:

- `mysqld` is not running on the local host. Check your operating system's process list to ensure the `mysqld` process is present.
- You're running a MySQL server on Windows with many TCP/IP connections to it. If you're experiencing that quite often your clients get that error, you can find a workaround here: Connection to MySQL Server Failing on Windows.
- Someone has removed the Unix socket file that `mysqld` uses (`/tmp/mysql.sock` by default). For example, you might have a `cron` job that removes old files from the `/tmp` directory. You can always run `mysqladmin version` to check whether the Unix socket file that `mysqladmin` is trying to use really exists. The fix in this case is to change the `cron` job to not remove `mysql.sock` or to place the socket file somewhere else. See Section B.3.3.6, “How to Protect or Change the MySQL Unix Socket File”.
- You have started the `mysqld` server with the `--socket=/path/to/socket` option, but forgotten to tell client programs the new name of the socket file. If you change the socket path name for the server, you must also notify the MySQL clients. You can do this by providing the same `--socket` option when you run client programs. You also need to ensure that clients have permission to access the `mysql.sock` file. To find out where the socket file is, you can do:

  ```
  shell> netstat -ln | grep mysql
  ```

  See Section B.3.3.6, “How to Protect or Change the MySQL Unix Socket File”.
- You are using Linux and one server thread has died (dumped core). In this case, you must kill the other `mysqld` threads (for example, with `kill`) before you can restart the MySQL server. See Section B.3.3.3, “What to Do If MySQL Keeps Crashing”.
- The server or client program might not have the proper access privileges for the directory that holds the Unix socket file or the socket file itself. In this case, you must either change the access privileges for the directory or socket file so that the server and clients can access them, or restart `mysqld` with a `--socket` option that specifies a socket file name in a directory where the server can create it and where client programs can access it.

If you get the error message `Can't connect to MySQL server on some_host`, you can try the following things to find out what the problem is:

- Check whether the server is running on that host by executing `telnet some_host 3306` and pressing the Enter key a couple of times. (3306 is the default MySQL port number. Change the value if your server is listening to a different port.) If there is a MySQL server running and listening to the port, you should get a response that includes the server's version number. If you get an error such as `telnet: Unable to connect to remote host: Connection refused`, then there is no server running on the given port.
- If the server is running on the local host, try using `mysqladmin -h localhost variables` to connect using the Unix socket file. Verify the TCP/IP port number that the server is configured to listen to (it is the value of the `port` variable.)
- If you are running under Linux and Security-Enhanced Linux (SELinux) is enabled, see Section 6.6, “SELinux”.

Connection to MySQL Server Failing on Windows

When you're running a MySQL server on Windows with many TCP/IP connections to it, and you're experiencing that quite often your clients get a `Can't connect to MySQL server` error, the reason might be that Windows does not allow for enough ephemeral (short-lived) ports to serve those connections.

The purpose of `TIME_WAIT` is to keep a connection accepting packets even after the connection has been closed. This is because Internet routing can cause a packet to take a slow route to its destination

and it may arrive after both sides have agreed to close. If the port is in use for a new connection, that packet from the old connection could break the protocol or compromise personal information from the original connection. The `TIME_WAIT` delay prevents this by ensuring that the port cannot be reused until after some time has been permitted for those delayed packets to arrive.

It is safe to reduce `TIME_WAIT` greatly on LAN connections because there is little chance of packets arriving at very long delays, as they could through the Internet with its comparatively large distances and latencies.

Windows permits ephemeral (short-lived) TCP ports to the user. After any port is closed it remains in a `TIME_WAIT` status for 120 seconds. The port is not available again until this time expires. The default range of port numbers depends on the version of Windows, with a more limited number of ports in older versions:

- Windows through Server 2003: Ports in range 1025–5000
- Windows Vista, Server 2008, and newer: Ports in range 49152–65535

With a small stack of available TCP ports (5000) and a high number of TCP ports being open and closed over a short period of time along with the `TIME_WAIT` status you have a good chance for running out of ports. There are two ways to address this problem:

- Reduce the number of TCP ports consumed quickly by investigating connection pooling or persistent connections where possible
- Tune some settings in the Windows registry (see below)

Important

The following procedure involves modifying the Windows registry. Before you modify the registry, make sure to back it up and make sure that you understand how to restore it if a problem occurs. For information about how to back up, restore, and edit the registry, view the following article in the Microsoft Knowledge Base: http://support.microsoft.com/kb/256986/EN-US/.

1. Start Registry Editor (`Regedt32.exe`).
2. Locate the following key in the registry:

   ```
   HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters
   ```

3. On the `Edit` menu, click `Add Value`, and then add the following registry value:

   ```
   Value Name: MaxUserPort
   Data Type: REG_DWORD
   Value: 65534
   ```

   This sets the number of ephemeral ports available to any user. The valid range is between 5000 and 65534 (decimal). The default value is 0x1388 (5000 decimal).

4. On the `Edit` menu, click `Add Value`, and then add the following registry value:

   ```
   Value Name: TcpTimedWaitDelay
   Data Type: REG_DWORD
   Value: 30
   ```

   This sets the number of seconds to hold a TCP port connection in `TIME_WAIT` state before closing. The valid range is between 30 and 300 decimal, although you may wish to check with Microsoft for the latest permitted values. The default value is 0x78 (120 decimal).

5. Quit Registry Editor.
6. Reboot the machine.

Note

Undoing the above should be as simple as deleting the registry entries you've created.

B.3.2.3 Lost connection to MySQL server

There are three likely causes for this error message.

Usually it indicates network connectivity trouble and you should check the condition of your network if this error occurs frequently. If the error message includes “during query,” this is probably the case you are experiencing.

Sometimes the “during query” form happens when millions of rows are being sent as part of one or more queries. If you know that this is happening, you should try increasing `net_read_timeout` from its default of 30 seconds to 60 seconds or longer, sufficient for the data transfer to complete.

More rarely, it can happen when the client is attempting the initial connection to the server. In this case, if your `connect_timeout` value is set to only a few seconds, you may be able to resolve the problem by increasing it to ten seconds, perhaps more if you have a very long distance or slow connection. You can determine whether you are experiencing this more uncommon cause by using `SHOW GLOBAL STATUS LIKE 'Aborted_connects'`. It increases by one for each initial connection attempt that the server aborts. You may see “reading authorization packet” as part of the error message; if so, that also suggests that this is the solution that you need.

If the cause is none of those just described, you may be experiencing a problem with `BLOB` values that are larger than `max_allowed_packet`, which can cause this error with some clients. Sometime you may see an `ER_NET_PACKET_TOO_LARGE` error, and that confirms that you need to increase `max_allowed_packet`.

B.3.2.4 Client does not support authentication protocol

The current implementation of the authentication protocol uses a password hashing algorithm that is incompatible with that used by older (pre-4.1) clients. Attempts to connect to a 4.1 or higher server with an older client may fail with the following message:

```
shell> mysql
Client does not support authentication protocol requested
by server; consider upgrading MySQL client
```

To deal with this problem, the preferred solution is to upgrade all client programs to use a 4.1.1 or higher client library. If that is not possible, use one of the following approaches:

- To connect to the server with a pre-4.1 client program, use an account that still has a pre-4.1-style password.
- Reset the password to pre-4.1 style for each user that needs to use a pre-4.1 client program. This can be done using the `SET PASSWORD` statement and the `OLD_PASSWORD()` function. As of MySQL 5.6.6, it is also necessary to first ensure that the authentication plugin for the account is `mysql_old_password`:

  ```
  mysql> UPDATE mysql.user SET plugin = 'mysql_old_password'
  mysql> WHERE User = 'some_user' AND Host = 'some_host';
  mysql> FLUSH PRIVILEGES;
  mysql> SET PASSWORD FOR
      -> 'some_user'@'some_host' = OLD_PASSWORD('new_password');
  ```

  Substitute the password you want to use for “*new_password*” in the preceding example. MySQL cannot tell you what the original password was, so you'll need to pick a new one.

- Tell the server to use the older password hashing algorithm by default:

  1. Start `mysqld` with the `old_passwords` system variable set to 1.

2. Assign an old-format password to each account that has had its password updated to the longer 4.1 format. You can identify these accounts with the following query:

```
mysql> SELECT Host, User, Password FROM mysql.user
    -> WHERE LENGTH(Password) > 16;
```

   For each account record displayed by the query, use the `Host` and `User` values and assign a password using one of the methods described previously.

The `Client does not support authentication protocol` error also can occur if multiple versions of MySQL are installed but client programs are dynamically linked and link to an older library. Make sure that clients use the most recent library version with which they are compatible. The procedure to do this depends on your system.

Note

The PHP `mysql` extension does not support the authentication protocol in MySQL 4.1.1 and higher. This is true regardless of the PHP version being used. If you wish to use the `mysql` extension with MySQL 4.1 or higher, you may need to follow one of the options discussed above for configuring MySQL to work with old clients. The `mysqli` extension (stands for "MySQL, Improved"; added in PHP 5) is compatible with the improved password hashing employed in MySQL 4.1 and higher, and no special configuration of MySQL need be done to use this MySQL client library. For more information about the `mysqli` extension, see http://php.net/mysqli.

For additional background on password hashing and authentication, see Section 6.1.2.4, “Password Hashing in MySQL”.

B.3.2.5 Password Fails When Entered Interactively

MySQL client programs prompt for a password when invoked with a `--password` or `-p` option that has no following password value:

```
shell> mysql -u user_name -p
Enter password:
```

On some systems, you may find that your password works when specified in an option file or on the command line, but not when you enter it interactively at the `Enter password:` prompt. This occurs when the library provided by the system to read passwords limits password values to a small number of characters (typically eight). That is a problem with the system library, not with MySQL. To work around it, change your MySQL password to a value that is eight or fewer characters long, or put your password in an option file.

B.3.2.6 Too many connections

If clients encounter `Too many connections` errors when attempting to connect to the `mysqld` server, all available connections are in use by other clients.

The permitted number of connections is controlled by the `max_connections` system variable. To support more connections, set `max_connections` to a larger value.

`mysqld` actually permits `max_connections` + 1 client connections. The extra connection is reserved for use by accounts that have the `SUPER` privilege. By granting the privilege to administrators and not to normal users (who should not need it), an administrator who also has the `PROCESS` privilege can connect to the server and use `SHOW PROCESSLIST` to diagnose problems even if the maximum number of unprivileged clients are connected. See Section 13.7.5.30, “SHOW PROCESSLIST Statement”.

For more information about how the server handles client connections, see Section 5.1.11.1, “Connection Interfaces”.

B.3.2.7 Out of memory

If you issue a query using the `mysql` client program and receive an error like the following one, it means that `mysql` does not have enough memory to store the entire query result:

```
mysql: Out of memory at line 42, 'malloc.c'
mysql: needed 8136 byte (8k), memory in use: 12481367 bytes (12189k)
ERROR 2008: MySQL client ran out of memory
```

To remedy the problem, first check whether your query is correct. Is it reasonable that it should return so many rows? If not, correct the query and try again. Otherwise, you can invoke `mysql` with the `--quick` option. This causes it to use the `mysql_use_result()` C API function to retrieve the result set, which places less of a load on the client (but more on the server).

B.3.2.8 MySQL server has gone away

This section also covers the related `Lost connection to server during query` error.

The most common reason for the `MySQL server has gone away` error is that the server timed out and closed the connection. In this case, you normally get one of the following error codes (which one you get is operating system-dependent).

Error Code	Description
`CR_SERVER_GONE_ERROR`	The client couldn't send a question to the server.
`CR_SERVER_LOST`	The client didn't get an error when writing to the server, but it didn't get a full answer (or any answer) to the question.

By default, the server closes the connection after eight hours if nothing has happened. You can change the time limit by setting the `wait_timeout` variable when you start `mysqld`. See Section 5.1.7, “Server System Variables”.

If you have a script, you just have to issue the query again for the client to do an automatic reconnection. This assumes that you have automatic reconnection in the client enabled (which is the default for the `mysql` command-line client).

Some other common reasons for the `MySQL server has gone away` error are:

- You (or the db administrator) has killed the running thread with a `KILL` statement or a `mysqladmin kill` command.
- You tried to run a query after closing the connection to the server. This indicates a logic error in the application that should be corrected.
- A client application running on a different host does not have the necessary privileges to connect to the MySQL server from that host.
- You got a timeout from the TCP/IP connection on the client side. This may happen if you have been using the commands: `mysql_options(..., MYSQL_OPT_READ_TIMEOUT,...)` or `mysql_options(..., MYSQL_OPT_WRITE_TIMEOUT,...)`. In this case increasing the timeout may help solve the problem.
- You have encountered a timeout on the server side and the automatic reconnection in the client is disabled (the `reconnect` flag in the `MYSQL` structure is equal to 0).
- You are using a Windows client and the server had dropped the connection (probably because `wait_timeout` expired) before the command was issued.

  The problem on Windows is that in some cases MySQL does not get an error from the OS when writing to the TCP/IP connection to the server, but instead gets the error when trying to read the answer from the connection.

The solution to this is to either do a `mysql_ping()` on the connection if there has been a long time since the last query (this is what Connector/ODBC does) or set `wait_timeout` on the `mysqld` server so high that it in practice never times out.

- You can also get these errors if you send a query to the server that is incorrect or too large. If `mysqld` receives a packet that is too large or out of order, it assumes that something has gone wrong with the client and closes the connection. If you need big queries (for example, if you are working with big `BLOB` columns), you can increase the query limit by setting the server's `max_allowed_packet` variable, which has a default value of 4MB (1MB before MySQL 5.6.6). You may also need to increase the maximum packet size on the client end. More information on setting the packet size is given in Section B.3.2.9, “Packet Too Large”.

  An `INSERT` or `REPLACE` statement that inserts a great many rows can also cause these sorts of errors. Either one of these statements sends a single request to the server irrespective of the number of rows to be inserted; thus, you can often avoid the error by reducing the number of rows sent per `INSERT` or `REPLACE`.

- It is also possible to see this error if host name lookups fail (for example, if the DNS server on which your server or network relies goes down). This is because MySQL is dependent on the host system for name resolution, but has no way of knowing whether it is working—from MySQL's point of view the problem is indistinguishable from any other network timeout.

  You may also see the `MySQL server has gone away` error if MySQL is started with the `skip_networking` system variable enabled.

  Another networking issue that can cause this error occurs if the MySQL port (default 3306) is blocked by your firewall, thus preventing any connections at all to the MySQL server.

- You can also encounter this error with applications that fork child processes, all of which try to use the same connection to the MySQL server. This can be avoided by using a separate connection for each child process.

- You have encountered a bug where the server died while executing the query.

You can check whether the MySQL server died and restarted by executing `mysqladmin version` and examining the server's uptime. If the client connection was broken because `mysqld` crashed and restarted, you should concentrate on finding the reason for the crash. Start by checking whether issuing the query again kills the server again. See Section B.3.3.3, “What to Do If MySQL Keeps Crashing”.

You can obtain more information about lost connections by starting `mysqld` with the `--log-warnings=2` option. This logs some of the disconnected errors in the `hostname.err` file. See Section 5.4.2, “The Error Log”.

If you want to create a bug report regarding this problem, be sure that you include the following information:

- Indicate whether the MySQL server died. You can find information about this in the server error log. See Section B.3.3.3, “What to Do If MySQL Keeps Crashing”.

- If a specific query kills `mysqld` and the tables involved were checked with `CHECK TABLE` before you ran the query, can you provide a reproducible test case? See Section 5.8, “Debugging MySQL”.

- What is the value of the `wait_timeout` system variable in the MySQL server? (`mysqladmin variables` gives you the value of this variable.)

- Have you tried to run `mysqld` with the general query log enabled to determine whether the problem query appears in the log? (See Section 5.4.3, “The General Query Log”.)

See also Section B.3.2.10, “Communication Errors and Aborted Connections”, and Section 1.6, “How to Report Bugs or Problems”.

B.3.2.9 Packet Too Large

A communication packet is a single SQL statement sent to the MySQL server, a single row that is sent to the client, or a binary log event sent from a source replication server to a replica.

The largest possible packet that can be transmitted to or from a MySQL 5.6 server or client is 1GB.

When a MySQL client or the `mysqld` server receives a packet bigger than `max_allowed_packet` bytes, it issues an `ER_NET_PACKET_TOO_LARGE` error and closes the connection. With some clients, you may also get a `Lost connection to MySQL server during query` error if the communication packet is too large.

Both the client and the server have their own `max_allowed_packet` variable, so if you want to handle big packets, you must increase this variable both in the client and in the server.

If you are using the `mysql` client program, its default `max_allowed_packet` variable is 16MB. To set a larger value, start `mysql` like this:

```
shell> mysql --max_allowed_packet=32M
```

That sets the packet size to 32MB.

The server's default `max_allowed_packet` value is 4MB (1MB before MySQL 5.6.6). You can increase this if the server needs to handle big queries (for example, if you are working with big `BLOB` columns). For example, to set the variable to 16MB, start the server like this:

```
shell> mysqld --max_allowed_packet=16M
```

You can also use an option file to set `max_allowed_packet`. For example, to set the size for the server to 16MB, add the following lines in an option file:

```
[mysqld]
max_allowed_packet=16M
```

It is safe to increase the value of this variable because the extra memory is allocated only when needed. For example, `mysqld` allocates more memory only when you issue a long query or when `mysqld` must return a large result row. The small default value of the variable is a precaution to catch incorrect packets between the client and server and also to ensure that you do not run out of memory by using large packets accidentally.

You can also get strange problems with large packets if you are using large `BLOB` values but have not given `mysqld` access to enough memory to handle the query. If you suspect this is the case, try adding `ulimit -d 256000` to the beginning of the `mysqld_safe` script and restarting `mysqld`.

B.3.2.10 Communication Errors and Aborted Connections

If connection problems occur such as communication errors or aborted connections, use these sources of information to diagnose problems:

- The error log. See Section 5.4.2, “The Error Log”.
- The general query log. See Section 5.4.3, “The General Query Log”.
- The `Aborted_xxx` and `Connection_errors_xxx` status variables. See Section 5.1.9, “Server Status Variables”.
- The host cache, which is accessible using the Performance Schema `host_cache` table. See Section 5.1.11.2, “DNS Lookups and the Host Cache”, and Section 22.12.10.1, “The host_cache Table”.

If you start the server with the `--log-warnings` option, you might find messages like this in your error log:

```
Aborted connection 854 to db: 'employees' user: 'josh'
```

If a client is unable even to connect, the server increments the `Aborted_connects` status variable. Unsuccessful connection attempts can occur for the following reasons:

- A client attempts to access a database but has no privileges for it.
- A client uses an incorrect password.
- A connection packet does not contain the right information.
- It takes more than `connect_timeout` seconds to obtain a connect packet. See Section 5.1.7, “Server System Variables”.

If these kinds of things happen, it might indicate that someone is trying to break into your server! If the general query log is enabled, messages for these types of problems are logged to it.

If a client successfully connects but later disconnects improperly or is terminated, the server increments the `Aborted_clients` status variable, and logs an `Aborted connection` message to the error log. The cause can be any of the following:

- The client program did not call `mysql_close()` before exiting.
- The client had been sleeping more than `wait_timeout` or `interactive_timeout` seconds without issuing any requests to the server. See Section 5.1.7, “Server System Variables”.
- The client program ended abruptly in the middle of a data transfer.

Other reasons for problems with aborted connections or aborted clients:

- The `max_allowed_packet` variable value is too small or queries require more memory than you have allocated for `mysqld`. See Section B.3.2.9, “Packet Too Large”.
- Use of Ethernet protocol with Linux, both half and full duplex. Some Linux Ethernet drivers have this bug. You should test for this bug by transferring a huge file using FTP between the client and server machines. If a transfer goes in burst-pause-burst-pause mode, you are experiencing a Linux duplex syndrome. Switch the duplex mode for both your network card and hub/switch to either full duplex or to half duplex and test the results to determine the best setting.
- A problem with the thread library that causes interrupts on reads.
- Badly configured TCP/IP.
- Faulty Ethernets, hubs, switches, cables, and so forth. This can be diagnosed properly only by replacing hardware.

See also Section B.3.2.8, “MySQL server has gone away”.

B.3.2.11 The table is full

If a table-full error occurs, it may be that the disk is full or that the table has reached its maximum size. The effective maximum table size for MySQL databases is usually determined by operating system constraints on file sizes, not by MySQL internal limits. See Section 8.4.6, “Limits on Table Size”.

B.3.2.12 Can't create/write to file

If you get an error of the following type for some queries, it means that MySQL cannot create a temporary file for the result set in the temporary directory:

```
Can't create/write to file '\\sqla3fe_0.ism'.
```

The preceding error is a typical message for Windows; the Unix message is similar.

One fix is to start `mysqld` with the `--tmpdir` option or to add the option to the `[mysqld]` section of your option file. For example, to specify a directory of `C:\temp`, use these lines:

```
[mysqld]
tmpdir=C:/temp
```

The `C:\temp` directory must exist and have sufficient space for the MySQL server to write to. See Section 4.2.2.2, “Using Option Files”.

Another cause of this error can be permissions issues. Make sure that the MySQL server can write to the `tmpdir` directory.

Check also the error code that you get with `perror`. One reason the server cannot write to a table is that the file system is full:

```
shell> perror 28
OS error code  28:  No space left on device
```

If you get an error of the following type during startup, it indicates that the file system or directory used for storing data files is write protected. Provided that the write error is to a test file, the error is not serious and can be safely ignored.

```
Can't create test file /usr/local/mysql/data/master.lower-test
```

B.3.2.13 Commands out of sync

If you get `Commands out of sync; you can't run this command now` in your client code, you are calling client functions in the wrong order.

This can happen, for example, if you are using `mysql_use_result()` and try to execute a new query before you have called `mysql_free_result()`. It can also happen if you try to execute two queries that return data without calling `mysql_use_result()` or `mysql_store_result()` in between.

B.3.2.14 Ignoring user

If you get the following error, it means that when `mysqld` was started or when it reloaded the grant tables, it found an account in the `user` table that had an invalid password.

`Found wrong password for user '`*`some_user`*`'@'`*`some_host`*`'; ignoring user`

As a result, the account is simply ignored by the permission system.

The following list indicates possible causes of and fixes for this problem:

- You may be running a new version of `mysqld` with an old `user` table. Check whether the `Password` column of that table is shorter than 16 characters. If so, correct this condition by running `mysql_upgrade`.
- The account has an old password (eight characters long). Update the account in the `user` table to have a new password.
- You have specified a password in the `user` table without using the `PASSWORD()` function. Use `mysql` to update the account in the `user` table with a new password, making sure to use the `PASSWORD()` function:

  ```
  mysql> UPDATE user SET Password=PASSWORD('new_password')
      -> WHERE User='some_user' AND Host='some_host';
  ```

B.3.2.15 Table 'tbl_name' doesn't exist

If you get either of the following errors, it usually means that no table exists in the default database with the given name:

```
Table 'tbl_name' doesn't exist
Can't find file: 'tbl_name' (errno: 2)
```

In some cases, it may be that the table does exist but that you are referring to it incorrectly:

- Because MySQL uses directories and files to store databases and tables, database and table names are case-sensitive if they are located on a file system that has case-sensitive file names.
- Even for file systems that are not case-sensitive, such as on Windows, all references to a given table within a query must use the same lettercase.

You can check which tables are in the default database with `SHOW TABLES`. See Section 13.7.5, "SHOW Statements".

B.3.2.16 Can't initialize character set

You might see an error like this if you have character set problems:

```
MySQL Connection Failed: Can't initialize character set charset_name
```

This error can have any of the following causes:

- The character set is a multibyte character set and you have no support for the character set in the client. In this case, you need to recompile the client by running `CMake` with the `-DDEFAULT_CHARSET=charset_name` or `-DWITH_EXTRA_CHARSETS=charset_name` option. See Section 2.9.7, "MySQL Source-Configuration Options".

  All standard MySQL binaries are compiled with `-DWITH_EXTRA_CHARSETS=complex`, which enables support for all multibyte character sets. See Section 2.9.7, "MySQL Source-Configuration Options".

- The character set is a simple character set that is not compiled into `mysqld`, and the character set definition files are not in the place where the client expects to find them.

  In this case, you need to use one of the following methods to solve the problem:

  - Recompile the client with support for the character set. See Section 2.9.7, "MySQL Source-Configuration Options".
  - Specify to the client the directory where the character set definition files are located. For many clients, you can do this with the `--character-sets-dir` option.
  - Copy the character definition files to the path where the client expects them to be.

B.3.2.17 File Not Found and Similar Errors

If you get `ERROR 'file_name' not found (errno: 23)`, `Can't open file: file_name (errno: 24)`, or any other error with `errno 23` or `errno 24` from MySQL, it means that you have not allocated enough file descriptors for the MySQL server. You can use the `perror` utility to get a description of what the error number means:

```
shell> perror 23
OS error code  23:  File table overflow
shell> perror 24
OS error code  24:  Too many open files
shell> perror 11
OS error code  11:  Resource temporarily unavailable
```

The problem here is that `mysqld` is trying to keep open too many files simultaneously. You can either tell `mysqld` not to open so many files at once or increase the number of file descriptors available to `mysqld`.

To tell `mysqld` to keep open fewer files at a time, you can make the table cache smaller by reducing the value of the `table_open_cache` system variable (the default value is 64). This may not entirely prevent running out of file descriptors because in some circumstances the server may attempt to extend the cache size temporarily, as described in Section 8.4.3.1, "How MySQL Opens and Closes Tables". Reducing the value of `max_connections` also reduces the number of open files (the default value is 100).

To change the number of file descriptors available to `mysqld`, you can use the `--open-files-limit` option to `mysqld_safe` or set the `open_files_limit` system variable. See Section 5.1.7, “Server System Variables”. The easiest way to set these values is to add an option to your option file. See Section 4.2.2.2, “Using Option Files”. If you have an old version of `mysqld` that does not support setting the open files limit, you can edit the `mysqld_safe` script. There is a commented-out line `ulimit -n 256` in the script. You can remove the `#` character to uncomment this line, and change the number `256` to set the number of file descriptors to be made available to `mysqld`.

`--open-files-limit` and `ulimit` can increase the number of file descriptors, but only up to the limit imposed by the operating system. There is also a “hard” limit that can be overridden only if you start `mysqld_safe` or `mysqld` as `root` (just remember that you also need to start the server with the `--user` option in this case so that it does not continue to run as `root` after it starts up). If you need to increase the operating system limit on the number of file descriptors available to each process, consult the documentation for your system.

Note

If you run the `tcsh` shell, `ulimit` does not work! `tcsh` also reports incorrect values when you ask for the current limits. In this case, you should start `mysqld_safe` using `sh`.

B.3.2.18 Table-Corruption Issues

If you have started `mysqld` with the `myisam_recover_options` system variable set, MySQL automatically checks and tries to repair `MyISAM` tables if they are marked as 'not closed properly' or 'crashed'. If this happens, MySQL writes an entry in the `hostname.err` file `'Warning: Checking table ...'` which is followed by `Warning: Repairing table` if the table needs to be repaired. If you get a lot of these errors, without `mysqld` having died unexpectedly just before, then something is wrong and needs to be investigated further.

When the server detects `MyISAM` table corruption, it writes additional information to the error log, such as the name and line number of the source file, and the list of threads accessing the table. Example: `Got an error from thread_id=1, mi_dynrec.c:368`. This is useful information to include in bug reports.

See also Section 5.1.6, “Server Command Options”, and Section 5.8.1.7, “Making a Test Case If You Experience Table Corruption”.

B.3.3 Administration-Related Issues

B.3.3.1 Problems with File Permissions

If you have problems with file permissions, the `UMASK` or `UMASK_DIR` environment variable might be set incorrectly when `mysqld` starts. For example, `mysqld` might issue the following error message when you create a table:

```
ERROR: Can't find file: 'path/with/filename.frm' (Errcode: 13)
```

The default `UMASK` and `UMASK_DIR` values are `0660` and `0700`, respectively. `mysqld` assumes that the value for `UMASK` or `UMASK_DIR` is in octal if it starts with a zero. For example, setting `UMASK=0600` is equivalent to `UMASK=384` because 0600 octal is 384 decimal.

Assuming that you start `mysqld` using `mysqld_safe`, change the default `UMASK` value as follows:

```
UMASK=384  # = 600 in octal
export UMASK
mysqld_safe &
```

Note

An exception applies for the error log file if you start `mysqld` using `mysqld_safe`, which does not respect `UMASK`: `mysqld_safe` may create the

> error log file if it does not exist prior to starting `mysqld`, and `mysqld_safe` uses a umask set to a strict value of `0137`. If this is unsuitable, create the error file manually with the desired access mode prior to executing `mysqld_safe`.

By default, `mysqld` creates database directories with an access permission value of `0700`. To modify this behavior, set the `UMASK_DIR` variable. If you set its value, new directories are created with the combined `UMASK` and `UMASK_DIR` values. For example, to give group access to all new directories, start `mysqld_safe` as follows:

```
UMASK_DIR=504  # = 770 in octal
export UMASK_DIR
mysqld_safe &
```

For additional details, see Section 4.9, “Environment Variables”.

B.3.3.2 How to Reset the Root Password

If you have never assigned a `root` password for MySQL, the server does not require a password at all for connecting as `root`. However, this is insecure. For instructions on assigning passwords, see Section 2.10.4, “Securing the Initial MySQL Accounts”.

If you know the `root` password and want to change it, see Section 13.7.1.7, “SET PASSWORD Statement”.

If you assigned a `root` password previously but have forgotten it, you can assign a new password. The following sections provide instructions for Windows and Unix and Unix-like systems, as well as generic instructions that apply to any system.

Resetting the Root Password: Windows Systems

On Windows, use the following procedure to reset the password for the MySQL `'root'@'localhost'` account. To change the password for a `root` account with a different host name part, modify the instructions to use that host name.

1. Log on to your system as Administrator.
2. Stop the MySQL server if it is running. For a server that is running as a Windows service, go to the Services manager: From the **Start** menu, select **Control Panel**, then **Administrative Tools**, then **Services**. Find the MySQL service in the list and stop it.

   If your server is not running as a service, you may need to use the Task Manager to force it to stop.
3. Create a text file containing the following statement on a single line. Replace the password with the password that you want to use.

   ```
   SET PASSWORD FOR 'root'@'localhost' = PASSWORD('MyNewPass');
   ```
4. Save the file. This example assumes that you name the file `C:\mysql-init.txt`.
5. Open a console window to get to the command prompt: From the **Start** menu, select **Run**, then enter `cmd` as the command to be run.
6. Start the MySQL server with the `init_file` system variable set to name the file (notice that the backslash in the option value is doubled):

   ```
   C:\> cd "C:\Program Files\MySQL\MySQL Server 5.6\bin"
   C:\> mysqld --init-file=C:\\mysql-init.txt
   ```

   If you installed MySQL to a different location, adjust the `cd` command accordingly.

   The server executes the contents of the file named by the `init_file` system variable at startup, changing the `'root'@'localhost'` account password.

To have server output to appear in the console window rather than in a log file, add the `--console` option to the `mysqld` command.

If you installed MySQL using the MySQL Installation Wizard, you may need to specify a `--defaults-file` option. For example:

```
C:\> mysqld
        --defaults-file="C:\\ProgramData\\MySQL\\MySQL Server 5.6\\my.ini"
        --init-file=C:\\mysql-init.txt
```

The appropriate `--defaults-file` setting can be found using the Services Manager: From the **Start** menu, select **Control Panel**, then **Administrative Tools**, then **Services**. Find the MySQL service in the list, right-click it, and choose the `Properties` option. The `Path to executable` field contains the `--defaults-file` setting.

7. After the server has started successfully, delete `C:\mysql-init.txt`.

You should now be able to connect to the MySQL server as `root` using the new password. Stop the MySQL server and restart it normally. If you run the server as a service, start it from the Windows Services window. If you start the server manually, use whatever command you normally use.

Resetting the Root Password: Unix and Unix-Like Systems

On Unix, use the following procedure to reset the password for the MySQL `'root'@'localhost'` account. To change the password for a `root` account with a different host name part, modify the instructions to use that host name.

The instructions assume that you start the MySQL server from the Unix login account that you normally use for running it. For example, if you run the server using the `mysql` login account, you should log in as `mysql` before using the instructions. Alternatively, you can log in as `root`, but in this case you *must* start `mysqld` with the `--user=mysql` option. If you start the server as `root` without using `--user=mysql`, the server may create `root`-owned files in the data directory, such as log files, and these may cause permission-related problems for future server startups. If that happens, you must either change the ownership of the files to `mysql` or remove them.

1. Log on to your system as the Unix user that the MySQL server runs as (for example, `mysql`).
2. Stop the MySQL server if it is running. Locate the `.pid` file that contains the server's process ID. The exact location and name of this file depend on your distribution, host name, and configuration. Common locations are `/var/lib/mysql/`, `/var/run/mysqld/`, and `/usr/local/mysql/data/`. Generally, the file name has an extension of `.pid` and begins with either `mysqld` or your system's host name.

   Stop the MySQL server by sending a normal `kill` (not `kill -9`) to the `mysqld` process. Use the actual path name of the `.pid` file in the following command:

   ```
   shell> kill `cat /mysql-data-directory/host_name.pid`
   ```

   Use backticks (not forward quotation marks) with the `cat` command. These cause the output of `cat` to be substituted into the `kill` command.
3. Create a text file containing the following statement on a single line. Replace the password with the password that you want to use.

   ```
   SET PASSWORD FOR 'root'@'localhost' = PASSWORD('MyNewPass');
   ```
4. Save the file. This example assumes that you name the file `/home/me/mysql-init`. The file contains the password, so do not save it where it can be read by other users. If you are not logged in as `mysql` (the user the server runs as), make sure that the file has permissions that permit `mysql` to read it.
5. Start the MySQL server with the `init_file` system variable set to name the file:

```
shell> mysqld --init-file=/home/me/mysql-init &
```

The server executes the contents of the file named by the `init_file` system variable at startup, changing the `'root'@'localhost'` account password.

Other options may be necessary as well, depending on how you normally start your server. For example, `--defaults-file` may be needed before the `init_file` argument.

6. After the server has started successfully, delete `/home/me/mysql-init`.

You should now be able to connect to the MySQL server as `root` using the new password. Stop the server and restart it normally.

Resetting the Root Password: Generic Instructions

The preceding sections provide password-resetting instructions specifically for Windows and Unix and Unix-like systems. Alternatively, on any platform, you can reset the password using the `mysql` client (but this approach is less secure):

1. Stop the MySQL server if necessary, then restart it with the `--skip-grant-tables` option. This enables anyone to connect without a password and with all privileges, and disables account-management statements such as `SET PASSWORD`. Because this is insecure, you might want to use `--skip-grant-tables` in conjunction with enabling the `skip_networking` system variable to prevent remote clients from connecting.

2. Connect to the MySQL server using the `mysql` client; no password is necessary because the server was started with `--skip-grant-tables`:

   ```
   shell> mysql
   ```

3. In the `mysql` client, tell the server to reload the grant tables so that account-management statements work:

   ```
   mysql> FLUSH PRIVILEGES;
   ```

   Then change the `'root'@'localhost'` account password. Replace the password with the password that you want to use. To change the password for a `root` account with a different host name part, modify the instructions to use that host name.

   ```
   mysql> SET PASSWORD FOR 'root'@'localhost' = PASSWORD('MyNewPass');
   ```

You should now be able to connect to the MySQL server as `root` using the new password. Stop the server and restart it normally (without the `--skip-grant-tables` option and without enabling the `skip_networking` system variable).

B.3.3.3 What to Do If MySQL Keeps Crashing

Each MySQL version is tested on many platforms before it is released. This does not mean that there are no bugs in MySQL, but if there are bugs, they should be very few and can be hard to find. If you have a problem, it always helps if you try to find out exactly what crashes your system, because you have a much better chance of getting the problem fixed quickly.

First, you should try to find out whether the problem is that the `mysqld` server dies or whether your problem has to do with your client. You can check how long your `mysqld` server has been up by executing `mysqladmin version`. If `mysqld` has died and restarted, you may find the reason by looking in the server's error log. See Section 5.4.2, “The Error Log”.

On some systems, you can find in the error log a stack trace of where `mysqld` died that you can resolve with the `resolve_stack_dump` program. See Section 5.8, “Debugging MySQL”. Note that the variable values written in the error log may not always be 100% correct.

Many unexpected server exits are caused by corrupted data files or index files. MySQL updates the files on disk with the `write()` system call after every SQL statement and before the client is notified about the result. (This is not true if you are running with the `delay_key_write` system variable enabled, in which case data files are written but not index files.) This means that data file contents are safe even if `mysqld` crashes, because the operating system ensures that the unflushed data is written to disk. You can force MySQL to flush everything to disk after every SQL statement by starting `mysqld` with the `--flush` option.

The preceding means that normally you should not get corrupted tables unless one of the following happens:

- The MySQL server or the server host was killed in the middle of an update.
- You have found a bug in `mysqld` that caused it to die in the middle of an update.
- Some external program is manipulating data files or index files at the same time as `mysqld` without locking the table properly.
- You are running many `mysqld` servers using the same data directory on a system that does not support good file system locks (normally handled by the `lockd` lock manager), or you are running multiple servers with external locking disabled.
- You have a crashed data file or index file that contains very corrupt data that confused `mysqld`.
- You have found a bug in the data storage code. This isn't likely, but it is at least possible. In this case, you can try to change the storage engine to another engine by using `ALTER TABLE` on a repaired copy of the table.

Because it is very difficult to know why something is crashing, first try to check whether things that work for others result in an unexpected exit for you. Try the following things:

- Stop the `mysqld` server with `mysqladmin shutdown`, run `myisamchk --silent --force */*.MYI` from the data directory to check all `MyISAM` tables, and restart `mysqld`. This ensures that you are running from a clean state. See Chapter 5, *MySQL Server Administration*.
- Start `mysqld` with the general query log enabled (see Section 5.4.3, “The General Query Log”). Then try to determine from the information written to the log whether some specific query kills the server. About 95% of all bugs are related to a particular query. Normally, this is one of the last queries in the log file just before the server restarts. See Section 5.4.3, “The General Query Log”. If you can repeatedly kill MySQL with a specific query, even when you have checked all tables just before issuing it, then you have isolated the bug and should submit a bug report for it. See Section 1.6, “How to Report Bugs or Problems”.
- Try to make a test case that we can use to repeat the problem. See Section 5.8, “Debugging MySQL”.
- Try the `fork_big.pl` script. (It is located in the `tests` directory of source distributions.)
- Configuring MySQL for debugging makes it much easier to gather information about possible errors if something goes wrong. Reconfigure MySQL with the `-DWITH_DEBUG=1` option to `CMake` and then recompile. See Section 5.8, “Debugging MySQL”.
- Make sure that you have applied the latest patches for your operating system.
- Use the `--skip-external-locking` option to `mysqld`. On some systems, the `lockd` lock manager does not work properly; the `--skip-external-locking` option tells `mysqld` not to use external locking. (This means that you cannot run two `mysqld` servers on the same data directory and that you must be careful if you use `myisamchk`. Nevertheless, it may be instructive to try the option as a test.)
- If `mysqld` appears to be running but not responding, try `mysqladmin -u root processlist`. Sometimes `mysqld` is not hung even though it seems unresponsive. The problem may be that

all connections are in use, or there may be some internal lock problem. `mysqladmin -u root processlist` usually is able to make a connection even in these cases, and can provide useful information about the current number of connections and their status.

- Run the command `mysqladmin -i 5 status` or `mysqladmin -i 5 -r status` in a separate window to produce statistics while running other queries.
- Try the following:

  1. Start `mysqld` from `gdb` (or another debugger). See Section 5.8, “Debugging MySQL”.
  2. Run your test scripts.
  3. Print the backtrace and the local variables at the three lowest levels. In `gdb`, you can do this with the following commands when `mysqld` has crashed inside `gdb`:

     ```
     backtrace
     info local
     up
     info local
     up
     info local
     ```

     With `gdb`, you can also examine which threads exist with `info threads` and switch to a specific thread with `thread N`, where `N` is the thread ID.

- Try to simulate your application with a Perl script to force MySQL to exit or misbehave.
- Send a normal bug report. See Section 1.6, “How to Report Bugs or Problems”. Be even more detailed than usual. Because MySQL works for many people, the crash might result from something that exists only on your computer (for example, an error that is related to your particular system libraries).
- If you have a problem with tables containing dynamic-length rows and you are using only `VARCHAR` columns (not `BLOB` or `TEXT` columns), you can try to change all `VARCHAR` to `CHAR` with `ALTER TABLE`. This forces MySQL to use fixed-size rows. Fixed-size rows take a little extra space, but are much more tolerant to corruption.

  The current dynamic row code has been in use for several years with very few problems, but dynamic-length rows are by nature more prone to errors, so it may be a good idea to try this strategy to see whether it helps.
- Consider the possibility of hardware faults when diagnosing problems. Defective hardware can be the cause of data corruption. Pay particular attention to your memory and disk subsystems when troubleshooting hardware.

B.3.3.4 How MySQL Handles a Full Disk

This section describes how MySQL responds to disk-full errors (such as “no space left on device”), and to quota-exceeded errors (such as “write failed” or “user block limit reached”).

This section is relevant for writes to `MyISAM` tables. It also applies for writes to binary log files and binary log index file, except that references to “row” and “record” should be understood to mean “event.”

When a disk-full condition occurs, MySQL does the following:

- It checks once every minute to see whether there is enough space to write the current row. If there is enough space, it continues as if nothing had happened.
- Every 10 minutes it writes an entry to the log file, warning about the disk-full condition.

To alleviate the problem, take the following actions:

- To continue, you only have to free enough disk space to insert all records.
- Alternatively, to abort the thread, use `mysqladmin kill`. The thread is aborted the next time it checks the disk (in one minute).
- Other threads might be waiting for the table that caused the disk-full condition. If you have several “locked” threads, killing the one thread that is waiting on the disk-full condition enables the other threads to continue.

Exceptions to the preceding behavior are when you use `REPAIR TABLE` or `OPTIMIZE TABLE` or when the indexes are created in a batch after `LOAD DATA` or after an `ALTER TABLE` statement. All of these statements may create large temporary files that, if left to themselves, would cause big problems for the rest of the system. If the disk becomes full while MySQL is doing any of these operations, it removes the big temporary files and mark the table as crashed. The exception is that for `ALTER TABLE`, the old table is left unchanged.

B.3.3.5 Where MySQL Stores Temporary Files

On Unix, MySQL uses the value of the `TMPDIR` environment variable as the path name of the directory in which to store temporary files. If `TMPDIR` is not set, MySQL uses the system default, which is usually `/tmp`, `/var/tmp`, or `/usr/tmp`.

On Windows, MySQL checks in order the values of the `TMPDIR`, `TEMP`, and `TMP` environment variables. For the first one found to be set, MySQL uses it and does not check those remaining. If none of `TMPDIR`, `TEMP`, or `TMP` are set, MySQL uses the Windows system default, which is usually `C:\windows\temp\`.

If the file system containing your temporary file directory is too small, you can use the `mysqld --tmpdir` option to specify a directory in a file system where you have enough space.

The `--tmpdir` option can be set to a list of several paths that are used in round-robin fashion. Paths should be separated by colon characters (`:`) on Unix and semicolon characters (`;`) on Windows.

Note

To spread the load effectively, these paths should be located on different *physical* disks, not different partitions of the same disk.

If the MySQL server is acting as a replica, you can set the `slave_load_tmpdir` system variable to specify a separate directory for holding temporary files when replicating `LOAD DATA` statements. This directory should be in a disk-based file system (not a memory-based file system) so that the temporary files used to replicate LOAD DATA can survive machine restarts. The directory also should not be one that is cleared by the operating system during the system startup process. However, replication can now continue after a restart if the temporary files have been removed.

MySQL arranges that temporary files are removed if `mysqld` is terminated. On platforms that support it (such as Unix), this is done by unlinking the file after opening it. The disadvantage of this is that the name does not appear in directory listings and you do not see a big temporary file that fills up the file system in which the temporary file directory is located. (In such cases, `lsof +L1` may be helpful in identifying large files associated with `mysqld`.)

When sorting (`ORDER BY` or `GROUP BY`), MySQL normally uses one or two temporary files. The maximum disk space required is determined by the following expression:

```
(length of what is sorted + sizeof(row pointer))
* number of matched rows
* 2
```

The row pointer size is usually four bytes, but may grow in the future for really big tables.

For some statements, MySQL creates temporary SQL tables that are not hidden and have names that begin with `#sql`.

Some SELECT queries creates temporary SQL tables to hold intermediate results.

DDL operations that rebuild the table and are not performed online using the ALGORITHM=INPLACE technique create a temporary copy of the original table in the same directory as the original table.

Online DDL operations may use temporary log files for recording concurrent DML, temporary sort files when creating an index, and temporary intermediate tables files when rebuilding the table. For more information, see Section 14.13.3, “Online DDL Space Requirements”.

See also Orphan Temporary Tables.

B.3.3.6 How to Protect or Change the MySQL Unix Socket File

The default location for the Unix socket file that the server uses for communication with local clients is /tmp/mysql.sock. (For some distribution formats, the directory might be different, such as /var/lib/mysql for RPMs.)

On some versions of Unix, anyone can delete files in the /tmp directory or other similar directories used for temporary files. If the socket file is located in such a directory on your system, this might cause problems.

On most versions of Unix, you can protect your /tmp directory so that files can be deleted only by their owners or the superuser (root). To do this, set the sticky bit on the /tmp directory by logging in as root and using the following command:

```
shell> chmod +t /tmp
```

You can check whether the sticky bit is set by executing ls -ld /tmp. If the last permission character is t, the bit is set.

Another approach is to change the place where the server creates the Unix socket file. If you do this, you should also let client programs know the new location of the file. You can specify the file location in several ways:

- Specify the path in a global or local option file. For example, put the following lines in /etc/my.cnf:

  ```
  [mysqld]
  socket=/path/to/socket

  [client]
  socket=/path/to/socket
  ```

  See Section 4.2.2.2, “Using Option Files”.

- Specify a --socket option on the command line to mysqld_safe and when you run client programs.

- Set the MYSQL_UNIX_PORT environment variable to the path of the Unix socket file.

- Recompile MySQL from source to use a different default Unix socket file location. Define the path to the file with the MYSQL_UNIX_ADDR option when you run CMake. See Section 2.9.7, “MySQL Source-Configuration Options”.

You can test whether the new socket location works by attempting to connect to the server with this command:

```
shell> mysqladmin --socket=/path/to/socket version
```

B.3.3.7 Time Zone Problems

If you have a problem with SELECT NOW() returning values in UTC and not your local time, you have to tell the server your current time zone. The same applies if UNIX_TIMESTAMP() returns

the wrong value. This should be done for the environment in which the server runs (for example, in `mysqld_safe` or `mysql.server`). See Section 4.9, “Environment Variables”.

You can set the time zone for the server with the `--timezone=timezone_name` option to `mysqld_safe`. You can also set it by setting the `TZ` environment variable before you start `mysqld`.

The permissible values for `--timezone` or `TZ` are system dependent. Consult your operating system documentation to see what values are acceptable.

B.3.4 Query-Related Issues

B.3.4.1 Case Sensitivity in String Searches

For nonbinary strings (`CHAR`, `VARCHAR`, `TEXT`), string searches use the collation of the comparison operands. For binary strings (`BINARY`, `VARBINARY`, `BLOB`), comparisons use the numeric values of the bytes in the operands; this means that for alphabetic characters, comparisons are case-sensitive.

A comparison between a nonbinary string and binary string is treated as a comparison of binary strings.

Simple comparison operations (`>=, >, =, <, <=`, sorting, and grouping) are based on each character's “sort value.” Characters with the same sort value are treated as the same character. For example, if `e` and `é` have the same sort value in a given collation, they compare as equal.

The default character set and collation are `latin1` and `latin1_swedish_ci`, so nonbinary string comparisons are case-insensitive by default. This means that if you search with *`col_name`* `LIKE 'a%'`, you get all column values that start with `A` or `a`. To make this search case-sensitive, make sure that one of the operands has a case-sensitive or binary collation. For example, if you are comparing a column and a string that both have the `latin1` character set, you can use the `COLLATE` operator to cause either operand to have the `latin1_general_cs` or `latin1_bin` collation:

```
col_name COLLATE latin1_general_cs LIKE 'a%'
col_name LIKE 'a%' COLLATE latin1_general_cs
col_name COLLATE latin1_bin LIKE 'a%'
col_name LIKE 'a%' COLLATE latin1_bin
```

If you want a column always to be treated in case-sensitive fashion, declare it with a case-sensitive or binary collation. See Section 13.1.17, “CREATE TABLE Statement”.

To cause a case-sensitive comparison of nonbinary strings to be case-insensitive, use `COLLATE` to name a case-insensitive collation. The strings in the following example normally are case-sensitive, but `COLLATE` changes the comparison to be case-insensitive:

```
mysql> SET @s1 = 'MySQL' COLLATE latin1_bin,
    ->     @s2 = 'mysql' COLLATE latin1_bin;
mysql> SELECT @s1 = @s2;
+-----------+
| @s1 = @s2 |
+-----------+
|         0 |
+-----------+
mysql> SELECT @s1 COLLATE latin1_swedish_ci = @s2;
+-------------------------------------+
| @s1 COLLATE latin1_swedish_ci = @s2 |
+-------------------------------------+
|                                   1 |
+-------------------------------------+
```

A binary string is case-sensitive in comparisons. To compare the string as case-insensitive, convert it to a nonbinary string and use `COLLATE` to name a case-insensitive collation:

```
mysql> SET @s = BINARY 'MySQL';
mysql> SELECT @s = 'mysql';
+--------------+
```

```
| @s = 'mysql' |
+--------------+
|            0 |
+--------------+
mysql> SELECT CONVERT(@s USING latin1) COLLATE latin1_swedish_ci = 'mysql';
+-------------------------------------------------------------+
| CONVERT(@s USING latin1) COLLATE latin1_swedish_ci = 'mysql' |
+-------------------------------------------------------------+
|                                                           1 |
+-------------------------------------------------------------+
```

To determine whether a value compares as a nonbinary or binary string, use the `COLLATION()` function. This example shows that `VERSION()` returns a string that has a case-insensitive collation, so comparisons are case-insensitive:

```
mysql> SELECT COLLATION(VERSION());
+----------------------+
| COLLATION(VERSION()) |
+----------------------+
| utf8_general_ci      |
+----------------------+
```

For binary strings, the collation value is `binary`, so comparisons are case-sensitive. One context in which you may see `binary` is for compression functions, which return binary strings as a general rule: string:

```
mysql> SELECT COLLATION(COMPRESS('x'));
+--------------------------+
| COLLATION(COMPRESS('x')) |
+--------------------------+
| binary                   |
+--------------------------+
```

To check the sort value of a string, the `WEIGHT_STRING()` may be helpful. See Section 12.8, “String Functions and Operators”.

B.3.4.2 Problems Using DATE Columns

The format of a `DATE` value is `'YYYY-MM-DD'`. According to standard SQL, no other format is permitted. You should use this format in `UPDATE` expressions and in the `WHERE` clause of `SELECT` statements. For example:

```
SELECT * FROM t1 WHERE date >= '2003-05-05';
```

As a convenience, MySQL automatically converts a date to a number if the date is used in numeric context and vice versa. MySQL also permits a “relaxed” string format when updating and in a `WHERE` clause that compares a date to a `DATE`, `DATETIME`, or `TIMESTAMP` column. “Relaxed” format means that any punctuation character may be used as the separator between parts. For example, `'2004-08-15'` and `'2004#08#15'` are equivalent. MySQL can also convert a string containing no separators (such as `'20040815'`), provided it makes sense as a date.

When you compare a `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP` to a constant string with the `<`, `<=`, `=`, `>=`, `>`, or `BETWEEN` operators, MySQL normally converts the string to an internal long integer for faster comparison (and also for a bit more “relaxed” string checking). However, this conversion is subject to the following exceptions:

- When you compare two columns
- When you compare a `DATE`, `TIME`, `DATETIME`, or `TIMESTAMP` column to an expression
- When you use any comparison method other than those just listed, such as `IN` or `STRCMP()`.

For those exceptions, the comparison is done by converting the objects to strings and performing a string comparison.

To be on the safe side, assume that strings are compared as strings and use the appropriate string functions if you want to compare a temporal value to a string.

The special “zero” date `'0000-00-00'` can be stored and retrieved as `'0000-00-00'`. When a `'0000-00-00'` date is used through Connector/ODBC, it is automatically converted to `NULL` because ODBC cannot handle that kind of date.

Because MySQL performs the conversions just described, the following statements work (assume that `idate` is a `DATE` column):

```
INSERT INTO t1 (idate) VALUES (19970505);
INSERT INTO t1 (idate) VALUES ('19970505');
INSERT INTO t1 (idate) VALUES ('97-05-05');
INSERT INTO t1 (idate) VALUES ('1997.05.05');
INSERT INTO t1 (idate) VALUES ('1997 05 05');
INSERT INTO t1 (idate) VALUES ('0000-00-00');

SELECT idate FROM t1 WHERE idate >= '1997-05-05';
SELECT idate FROM t1 WHERE idate >= 19970505;
SELECT MOD(idate,100) FROM t1 WHERE idate >= 19970505;
SELECT idate FROM t1 WHERE idate >= '19970505';
```

However, the following statement does not work:

```
SELECT idate FROM t1 WHERE STRCMP(idate,'20030505')=0;
```

`STRCMP()` is a string function, so it converts `idate` to a string in `'YYYY-MM-DD'` format and performs a string comparison. It does not convert `'20030505'` to the date `'2003-05-05'` and perform a date comparison.

If you enable the `ALLOW_INVALID_DATES` SQL mode, MySQL permits you to store dates that are given only limited checking: MySQL requires only that the day is in the range from 1 to 31 and the month is in the range from 1 to 12. This makes MySQL very convenient for Web applications where you obtain year, month, and day in three different fields and you want to store exactly what the user inserted (without date validation).

MySQL permits you to store dates where the day or month and day are zero. This is convenient if you want to store a birthdate in a `DATE` column and you know only part of the date. To disallow zero month or day parts in dates, enable the `NO_ZERO_IN_DATE` SQL mode.

MySQL permits you to store a “zero” value of `'0000-00-00'` as a “dummy date.” This is in some cases more convenient than using `NULL` values. If a date to be stored in a `DATE` column cannot be converted to any reasonable value, MySQL stores `'0000-00-00'`. To disallow `'0000-00-00'`, enable the `NO_ZERO_DATE` SQL mode.

To have MySQL check all dates and accept only legal dates (unless overridden by `IGNORE`), set the `sql_mode` system variable to `"NO_ZERO_IN_DATE,NO_ZERO_DATE"`.

B.3.4.3 Problems with NULL Values

The concept of the `NULL` value is a common source of confusion for newcomers to SQL, who often think that `NULL` is the same thing as an empty string `''`. This is not the case. For example, the following statements are completely different:

```
mysql> INSERT INTO my_table (phone) VALUES (NULL);
mysql> INSERT INTO my_table (phone) VALUES ('');
```

Both statements insert a value into the `phone` column, but the first inserts a `NULL` value and the second inserts an empty string. The meaning of the first can be regarded as “phone number is not known” and the meaning of the second can be regarded as “the person is known to have no phone, and thus no phone number.”

To help with `NULL` handling, you can use the `IS NULL` and `IS NOT NULL` operators and the `IFNULL()` function.

In SQL, the NULL value is never true in comparison to any other value, even NULL. An expression that contains NULL always produces a NULL value unless otherwise indicated in the documentation for the operators and functions involved in the expression. All columns in the following example return NULL:

```
mysql> SELECT NULL, 1+NULL, CONCAT('Invisible',NULL);
```

To search for column values that are NULL, you cannot use an expr = NULL test. The following statement returns no rows, because expr = NULL is never true for any expression:

```
mysql> SELECT * FROM my_table WHERE phone = NULL;
```

To look for NULL values, you must use the IS NULL test. The following statements show how to find the NULL phone number and the empty phone number:

```
mysql> SELECT * FROM my_table WHERE phone IS NULL;
mysql> SELECT * FROM my_table WHERE phone = '';
```

See Section 3.3.4.6, “Working with NULL Values”, for additional information and examples.

You can add an index on a column that can have NULL values if you are using the MyISAM, InnoDB, or MEMORY storage engine. Otherwise, you must declare an indexed column NOT NULL, and you cannot insert NULL into the column.

When reading data with LOAD DATA, empty or missing columns are updated with ''. To load a NULL value into a column, use \N in the data file. The literal word NULL may also be used under some circumstances. See Section 13.2.6, “LOAD DATA Statement”.

When using DISTINCT, GROUP BY, or ORDER BY, all NULL values are regarded as equal.

When using ORDER BY, NULL values are presented first, or last if you specify DESC to sort in descending order.

Aggregate (group) functions such as COUNT(), MIN(), and SUM() ignore NULL values. The exception to this is COUNT(*), which counts rows and not individual column values. For example, the following statement produces two counts. The first is a count of the number of rows in the table, and the second is a count of the number of non-NULL values in the age column:

```
mysql> SELECT COUNT(*), COUNT(age) FROM person;
```

For some data types, MySQL handles NULL values specially. If you insert NULL into a TIMESTAMP column, the current date and time is inserted. If you insert NULL into an integer or floating-point column that has the AUTO_INCREMENT attribute, the next number in the sequence is inserted.

B.3.4.4 Problems with Column Aliases

An alias can be used in a query select list to give a column a different name. You can use the alias in GROUP BY, ORDER BY, or HAVING clauses to refer to the column:

```
SELECT SQRT(a*b) AS root FROM tbl_name
  GROUP BY root HAVING root > 0;
SELECT id, COUNT(*) AS cnt FROM tbl_name
  GROUP BY id HAVING cnt > 0;
SELECT id AS 'Customer identity' FROM tbl_name;
```

Standard SQL disallows references to column aliases in a WHERE clause. This restriction is imposed because when the WHERE clause is evaluated, the column value may not yet have been determined. For example, the following query is illegal:

```
SELECT id, COUNT(*) AS cnt FROM tbl_name
  WHERE cnt > 0 GROUP BY id;
```

The WHERE clause determines which rows should be included in the GROUP BY clause, but it refers to the alias of a column value that is not known until after the rows have been selected, and grouped by the GROUP BY.

In the select list of a query, a quoted column alias can be specified using identifier or string quoting characters:

```
SELECT 1 AS `one`, 2 AS 'two';
```

Elsewhere in the statement, quoted references to the alias must use identifier quoting or the reference is treated as a string literal. For example, this statement groups by the values in column `id`, referenced using the alias `` `a` ``:

```
SELECT id AS 'a', COUNT(*) AS cnt FROM tbl_name
  GROUP BY `a`;
```

But this statement groups by the literal string `'a'` and does not work as expected:

```
SELECT id AS 'a', COUNT(*) AS cnt FROM tbl_name
  GROUP BY 'a';
```

B.3.4.5 Rollback Failure for Nontransactional Tables

If you receive the following message when trying to perform a `ROLLBACK`, it means that one or more of the tables you used in the transaction do not support transactions:

```
Warning: Some non-transactional changed tables couldn't be rolled back
```

These nontransactional tables are not affected by the `ROLLBACK` statement.

If you were not deliberately mixing transactional and nontransactional tables within the transaction, the most likely cause for this message is that a table you thought was transactional actually is not. This can happen if you try to create a table using a transactional storage engine that is not supported by your `mysqld` server (or that was disabled with a startup option). If `mysqld` does not support a storage engine, it instead creates the table as a `MyISAM` table, which is nontransactional.

You can check the storage engine for a table by using either of these statements:

```
SHOW TABLE STATUS LIKE 'tbl_name';
SHOW CREATE TABLE tbl_name;
```

See Section 13.7.5.37, “SHOW TABLE STATUS Statement”, and Section 13.7.5.12, “SHOW CREATE TABLE Statement”.

To check which storage engines your `mysqld` server supports, use this statement:

```
SHOW ENGINES;
```

See Section 13.7.5.17, “SHOW ENGINES Statement” for full details.

B.3.4.6 Deleting Rows from Related Tables

If the total length of the `DELETE` statement for `related_table` is more than 1MB (the default value of the `max_allowed_packet` system variable), you should split it into smaller parts and execute multiple `DELETE` statements. You probably get the fastest `DELETE` by specifying only 100 to 1,000 `related_column` values per statement if the `related_column` is indexed. If the `related_column` isn't indexed, the speed is independent of the number of arguments in the `IN` clause.

B.3.4.7 Solving Problems with No Matching Rows

If you have a complicated query that uses many tables but that returns no rows, you should use the following procedure to find out what is wrong:

1. Test the query with `EXPLAIN` to check whether you can find something that is obviously wrong. See Section 13.8.2, “EXPLAIN Statement”.

2. Select only those columns that are used in the `WHERE` clause.

3. Remove one table at a time from the query until it returns some rows. If the tables are large, it is a good idea to use `LIMIT 10` with the query.

4. Issue a `SELECT` for the column that should have matched a row against the table that was last removed from the query.

5. If you are comparing `FLOAT` or `DOUBLE` columns with numbers that have decimals, you cannot use equality (`=`) comparisons. This problem is common in most computer languages because not all floating-point values can be stored with exact precision. In some cases, changing the `FLOAT` to a `DOUBLE` fixes this. See Section B.3.4.8, “Problems with Floating-Point Values”.

6. If you still cannot figure out what is wrong, create a minimal test that can be run with `mysql test < query.sql` that shows your problems. You can create a test file by dumping the tables with `mysqldump --quick db_name tbl_name_1 ... tbl_name_n > query.sql`. Open the file in an editor, remove some insert lines (if there are more than needed to demonstrate the problem), and add your `SELECT` statement at the end of the file.

   Verify that the test file demonstrates the problem by executing these commands:

   ```
   shell> mysqladmin create test2
   shell> mysql test2 < query.sql
   ```

   Attach the test file to a bug report, which you can file using the instructions in Section 1.6, “How to Report Bugs or Problems”.

B.3.4.8 Problems with Floating-Point Values

Floating-point numbers sometimes cause confusion because they are approximate and not stored as exact values. A floating-point value as written in an SQL statement may not be the same as the value represented internally. Attempts to treat floating-point values as exact in comparisons may lead to problems. They are also subject to platform or implementation dependencies. The `FLOAT` and `DOUBLE` data types are subject to these issues. For `DECIMAL` columns, MySQL performs operations with a precision of 65 decimal digits, which should solve most common inaccuracy problems.

The following example uses `DOUBLE` to demonstrate how calculations that are done using floating-point operations are subject to floating-point error.

```
mysql> CREATE TABLE t1 (i INT, d1 DOUBLE, d2 DOUBLE);
mysql> INSERT INTO t1 VALUES (1, 101.40, 21.40), (1, -80.00, 0.00),
    -> (2, 0.00, 0.00), (2, -13.20, 0.00), (2, 59.60, 46.40),
    -> (2, 30.40, 30.40), (3, 37.00, 7.40), (3, -29.60, 0.00),
    -> (4, 60.00, 15.40), (4, -10.60, 0.00), (4, -34.00, 0.00),
    -> (5, 33.00, 0.00), (5, -25.80, 0.00), (5, 0.00, 7.20),
    -> (6, 0.00, 0.00), (6, -51.40, 0.00);

mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b
    -> FROM t1 GROUP BY i HAVING a <> b;

+------+-------+------+
| i    | a     | b    |
+------+-------+------+
|    1 |  21.4 | 21.4 |
|    2 |  76.8 | 76.8 |
|    3 |   7.4 |  7.4 |
|    4 |  15.4 | 15.4 |
|    5 |   7.2 |  7.2 |
|    6 | -51.4 |    0 |
+------+-------+------+
```

The result is correct. Although the first five records look like they should not satisfy the comparison (the values of `a` and `b` do not appear to be different), they may do so because the difference between the numbers shows up around the tenth decimal or so, depending on factors such as computer architecture or the compiler version or optimization level. For example, different CPUs may evaluate floating-point numbers differently.

If columns `d1` and `d2` had been defined as `DECIMAL` rather than `DOUBLE`, the result of the `SELECT` query would have contained only one row—the last one shown above.

The correct way to do floating-point number comparison is to first decide on an acceptable tolerance for differences between the numbers and then do the comparison against the tolerance value. For example, if we agree that floating-point numbers should be regarded the same if they are same within a precision of one in ten thousand (0.0001), the comparison should be written to find differences larger than the tolerance value:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
    -> GROUP BY i HAVING ABS(a - b) > 0.0001;
+------+-------+------+
| i    | a     | b    |
+------+-------+------+
|    6 | -51.4 |    0 |
+------+-------+------+
1 row in set (0.00 sec)
```

Conversely, to get rows where the numbers are the same, the test should find differences within the tolerance value:

```
mysql> SELECT i, SUM(d1) AS a, SUM(d2) AS b FROM t1
    -> GROUP BY i HAVING ABS(a - b) <= 0.0001;
+------+------+------+
| i    | a    | b    |
+------+------+------+
|    1 | 21.4 | 21.4 |
|    2 | 76.8 | 76.8 |
|    3 |  7.4 |  7.4 |
|    4 | 15.4 | 15.4 |
|    5 |  7.2 |  7.2 |
+------+------+------+
5 rows in set (0.03 sec)
```

Floating-point values are subject to platform or implementation dependencies. Suppose that you execute the following statements:

```
CREATE TABLE t1(c1 FLOAT(53,0), c2 FLOAT(53,0));
INSERT INTO t1 VALUES('1e+52','-1e+52');
SELECT * FROM t1;
```

On some platforms, the `SELECT` statement returns `inf` and `-inf`. On others, it returns `0` and `-0`.

An implication of the preceding issues is that if you attempt to create a replica by dumping table contents with `mysqldump` on the source and reloading the dump file into the replica, tables containing floating-point columns might differ between the two hosts.

B.3.5 Optimizer-Related Issues

MySQL uses a cost-based optimizer to determine the best way to resolve a query. In many cases, MySQL can calculate the best possible query plan, but sometimes MySQL does not have enough information about the data at hand and has to make “educated” guesses about the data.

For the cases when MySQL does not do the "right" thing, tools that you have available to help MySQL are:

- Use the `EXPLAIN` statement to get information about how MySQL processes a query. To use it, just add the keyword `EXPLAIN` to the front of your `SELECT` statement:

  ```
  mysql> EXPLAIN SELECT * FROM t1, t2 WHERE t1.i = t2.i;
  ```

  `EXPLAIN` is discussed in more detail in Section 13.8.2, “EXPLAIN Statement”.

- Use `ANALYZE TABLE` *`tbl_name`* to update the key distributions for the scanned table. See Section 13.7.2.1, “ANALYZE TABLE Statement”.

- Use `FORCE INDEX` for the scanned table to tell MySQL that table scans are very expensive compared to using the given index:

```
SELECT * FROM t1, t2 FORCE INDEX (index_for_column)
WHERE t1.col_name=t2.col_name;
```

  `USE INDEX` and `IGNORE INDEX` may also be useful. See Section 8.9.3, “Index Hints”.

- Global and table-level `STRAIGHT_JOIN`. See Section 13.2.9, “SELECT Statement”.

- You can tune global or thread-specific system variables. For example, start `mysqld` with the `--max-seeks-for-key=1000` option or use `SET max_seeks_for_key=1000` to tell the optimizer to assume that no key scan causes more than 1,000 key seeks. See Section 5.1.7, “Server System Variables”.

B.3.6 Table Definition-Related Issues

B.3.6.1 Problems with ALTER TABLE

If you get a duplicate-key error when using `ALTER TABLE` to change the character set or collation of a character column, the cause is either that the new column collation maps two keys to the same value or that the table is corrupted. In the latter case, you should run `REPAIR TABLE` on the table. `REPAIR TABLE` works for `MyISAM`, `ARCHIVE`, and `CSV` tables.

If `ALTER TABLE` dies with the following error, the problem may be that MySQL crashed during an earlier `ALTER TABLE` operation and there is an old table named `A-xxx` or `B-xxx` lying around:

```
Error on rename of './database/name.frm'
to './database/B-xxx.frm' (Errcode: 17)
```

In this case, go to the MySQL data directory and delete all files that have names starting with `A-` or `B-`. (You may want to move them elsewhere instead of deleting them.)

`ALTER TABLE` works in the following way:

- Create a new table named `A-xxx` with the requested structural changes.
- Copy all rows from the original table to `A-xxx`.
- Rename the original table to `B-xxx`.
- Rename `A-xxx` to your original table name.
- Delete `B-xxx`.

If something goes wrong with the renaming operation, MySQL tries to undo the changes. If something goes seriously wrong (although this shouldn't happen), MySQL may leave the old table as `B-xxx`. A simple rename of the table files at the system level should get your data back.

If you use `ALTER TABLE` on a transactional table or if you are using Windows, `ALTER TABLE` unlocks the table if you had done a `LOCK TABLE` on it. This is done because `InnoDB` and these operating systems cannot drop a table that is in use.

B.3.6.2 TEMPORARY Table Problems

Temporary tables created with `CREATE TEMPORARY TABLE` have the following limitations:

- `TEMPORARY` tables are supported only by the `InnoDB`, `MEMORY`, `MyISAM`, and `MERGE` storage engines.
- Temporary tables are not supported for NDB Cluster.

- The `SHOW TABLES` statement does not list `TEMPORARY` tables.
- To rename `TEMPORARY` tables, `RENAME TABLE` does not work. Use `ALTER TABLE` instead:

```
ALTER TABLE old_name RENAME new_name;
```

- You cannot refer to a `TEMPORARY` table more than once in the same query. For example, the following does not work:

```
SELECT * FROM temp_table JOIN temp_table AS t2;
```

  The statement produces this error:

```
ERROR 1137: Can't reopen table: 'temp_table'
```

- The `Can't reopen table` error also occurs if you refer to a temporary table multiple times in a stored function under different aliases, even if the references occur in different statements within the function. It may occur for temporary tables created outside stored functions and referred to across multiple calling and callee functions.
- If a `TEMPORARY` is created with the same name as an existing non-`TEMPORARY` table, the non-`TEMPORARY` table is hidden until the `TEMPORARY` table is dropped, even if the tables use different storage engines.
- There are known issues in using temporary tables with replication. See Section 17.4.1.29, “Replication and Temporary Tables”, for more information.

B.3.7 Known Issues in MySQL

This section lists known issues in recent versions of MySQL.

For information about platform-specific issues, see the installation and debugging instructions in Section 2.1, “General Installation Guidance”, and Section 5.8, “Debugging MySQL”.

The following problems are known:

- Subquery optimization for `IN` is not as effective as for `=`.
- Even if you use `lower_case_table_names=2` (which enables MySQL to remember the case used for databases and table names), MySQL does not remember the case used for database names for the function `DATABASE()` or within the various logs (on case-insensitive systems).
- Dropping a `FOREIGN KEY` constraint does not work in replication because the constraint may have another name on the replica.
- `REPLACE` (and `LOAD DATA` with the `REPLACE` option) does not trigger `ON DELETE CASCADE`.
- `DISTINCT` with `ORDER BY` does not work inside `GROUP_CONCAT()` if you do not use all and only those columns that are in the `DISTINCT` list.
- When inserting a big integer value (between 2^{63} and $2^{64}-1$) into a decimal or string column, it is inserted as a negative value because the number is evaluated in signed integer context.
- `ANALYZE TABLE`, `OPTIMIZE TABLE`, and `REPAIR TABLE` may cause problems on nontransactional tables for which you are using `INSERT DELAYED`.
- With statement-based binary logging, the source server writes the executed queries to the binary log. This is a very fast, compact, and efficient logging method that works perfectly in most cases. However, it is possible for the data on the source and replica to become different if a query is designed in such a way that the data modification is nondeterministic (generally not a recommended practice, even outside of replication).

For example:

- `CREATE TABLE ... SELECT` or `INSERT ... SELECT` statements that insert zero or `NULL` values into an `AUTO_INCREMENT` column.
- `DELETE` if you are deleting rows from a table that has foreign keys with `ON DELETE CASCADE` properties.
- `REPLACE ... SELECT`, `INSERT IGNORE ... SELECT` if you have duplicate key values in the inserted data.

If and only if the preceding queries have no `ORDER BY` clause guaranteeing a deterministic order.

For example, for `INSERT ... SELECT` with no `ORDER BY`, the `SELECT` may return rows in a different order (which results in a row having different ranks, hence getting a different number in the `AUTO_INCREMENT` column), depending on the choices made by the optimizers on the source and replica.

A query is optimized differently on the source and replica only if:

- The table is stored using a different storage engine on the source than on the replica. (It is possible to use different storage engines on the source and replica. For example, you can use `InnoDB` on the source, but `MyISAM` on the replica if the replica has less available disk space.)
- MySQL buffer sizes (`key_buffer_size`, and so on) are different on the source and replica.
- The source and replica run different MySQL versions, and the optimizer code differs between these versions.

This problem may also affect database restoration using `mysqlbinlog|mysql`.

The easiest way to avoid this problem is to add an `ORDER BY` clause to the aforementioned nondeterministic queries to ensure that the rows are always stored or modified in the same order. Using row-based or mixed logging format also avoids the problem.

- Log file names are based on the server host name if you do not specify a file name with the startup option. To retain the same log file names if you change your host name to something else, you must explicitly use options such as `--log-bin=`*`old_host_name`*`-bin`. See Section 5.1.6, “Server Command Options”. Alternatively, rename the old files to reflect your host name change. If these are binary logs, you must edit the binary log index file and fix the binary log file names there as well. (The same is true for the relay logs on a replica.)
- `mysqlbinlog` does not delete temporary files left after a `LOAD DATA` statement. See Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”.
- `RENAME` does not work with `TEMPORARY` tables or tables used in a `MERGE` table.
- When using `SET CHARACTER SET`, you cannot use translated characters in database, table, and column names.
- You cannot use `_` or `%` with `ESCAPE` in `LIKE ... ESCAPE`.
- The server uses only the first `max_sort_length` bytes when comparing data values. This means that values cannot reliably be used in `GROUP BY`, `ORDER BY`, or `DISTINCT` if they differ only after the first `max_sort_length` bytes. To work around this, increase the variable value. The default value of `max_sort_length` is 1024 and can be changed at server startup time or at runtime.
- Numeric calculations are done with `BIGINT` or `DOUBLE` (both are normally 64 bits long). Which precision you get depends on the function. The general rule is that bit functions are performed with `BIGINT` precision, `IF()` and `ELT()` with `BIGINT` or `DOUBLE` precision, and the rest with `DOUBLE`

precision. You should try to avoid using unsigned long long values if they resolve to be larger than 63 bits (9223372036854775807) for anything other than bit fields.

- You can have up to 255 `ENUM` and `SET` columns in one table.
- In `MIN()`, `MAX()`, and other aggregate functions, MySQL currently compares `ENUM` and `SET` columns by their string value rather than by the string's relative position in the set.
- In an `UPDATE` statement, columns are updated from left to right. If you refer to an updated column, you get the updated value instead of the original value. For example, the following statement increments `KEY` by `2`, **not** `1`:

```
mysql> UPDATE tbl_name SET KEY=KEY+1,KEY=KEY+1;
```

- You can refer to multiple temporary tables in the same query, but you cannot refer to any given temporary table more than once. For example, the following does not work:

```
mysql> SELECT * FROM temp_table, temp_table AS t2;
ERROR 1137: Can't reopen table: 'temp_table'
```

- The optimizer may handle `DISTINCT` differently when you are using “hidden” columns in a join than when you are not. In a join, hidden columns are counted as part of the result (even if they are not shown), whereas in normal queries, hidden columns do not participate in the `DISTINCT` comparison.

  An example of this is:

```
SELECT DISTINCT mp3id FROM band_downloads
       WHERE userid = 9 ORDER BY id DESC;
```

  and

```
SELECT DISTINCT band_downloads.mp3id
       FROM band_downloads,band_mp3
       WHERE band_downloads.userid = 9
       AND band_mp3.id = band_downloads.mp3id
       ORDER BY band_downloads.id DESC;
```

  In the second case, you may get two identical rows in the result set (because the values in the hidden `id` column may differ).

  This happens only for queries that do not have the `ORDER BY` columns in the result.

- If you execute a `PROCEDURE` on a query that returns an empty set, in some cases the `PROCEDURE` does not transform the columns.
- Creation of a table of type `MERGE` does not check whether the underlying tables are compatible types.
- If you use `ALTER TABLE` to add a `UNIQUE` index to a table used in a `MERGE` table and then add a normal index on the `MERGE` table, the key order is different for the tables if there was an old, non-`UNIQUE` key in the table. This is because `ALTER TABLE` puts `UNIQUE` indexes before normal indexes to be able to detect duplicate keys as early as possible.

Appendix C Indexes

Table of Contents

General Index 3277
C Function Index 3472
Command Index 3475
Function Index 3517
INFORMATION_SCHEMA Index 3550
Join Types Index 3562
Operator Index 3564
Option Index 3572
Privileges Index 3658
SQL Modes Index 3667
Statement/Syntax Index 3671
Status Variable Index 3744
System Variable Index 3766
Transaction Isolation Level Index 3826

General Index

Symbols

! (logical NOT), 1332
!= (not equal), 1327
", 1132
#mysql50 identifier prefix, 1133, 1139
%, 1339
% (modulo), 1343
% (wildcard character), 1124
& (bitwise AND), 1428
&& (logical AND), 1332
() (parentheses), 1325
(Control+Z) \Z, 1124, 1609
* (multiplication), 1339
+ (addition), 1338
- (subtraction), 1338
- (unary minus), 1338
--init-rpl-role
    removed features, 20
--innodb-read-only
    new features, 12
--language
    deprecated features, 19
--log
    removed features, 20
--one-thread
    removed features, 20
--password option, 768
--rpl-recovery-rank
    removed features, 20
--safe-mode
    removed features, 20
--skip-innodb
    deprecated features, 19
--skip-thread-priority

removed features, 20
--table-cache
removed features, 20
-? option (NDB Cluster programs), 2697
-c option (NDB Cluster programs), 2697
-c option (ndb_mgmd) (OBSOLETE), 2621
-d option
ndb_index_stat, 2649
ndb_mgmd, 2622
-e option
ndb_mgm, 2627
-f option
ndb_mgmd, 2621
-l option
ndbinfo_select_all, 2617
-n option
ndbd, 2614
ndbmtd, 2614
-p option, 768
-P option
ndb_mgmd, 2625
-v option
ndb_mgmd, 2626
-V option (NDB Cluster programs), 2698
.frm file, 1560
.ibd file, 1560
.my.cnf option file, 259, 261, 278, 736, 768, 823
.MYD file, 1560
.MYI file, 1560
.mylogin.cnf option file, 259, 414
.mysql_history file, 325, 769
.mysql_secret file, 296
.pid (process ID) file, 959
/ (division), 1339
/etc/passwd, 776, 1629
3306 port, 174, 537
:= (assignment operator), 1333
:= (assignment), 1165
< (less than), 1328
<< (left shift), 246, 1429
<= (less than or equal), 1327
<=> (equal to), 1327
<> (not equal), 1327
= (assignment operator), 1334
= (assignment), 1165
= (equal), 1327
> (greater than), 1328
>= (greater than or equal), 1328
>> (right shift), 1429
[api] (NDB Cluster), 2458
[computer] (NDB Cluster), 2460
[mgm] (NDB Cluster), 2458
[mysqld] (NDB Cluster), 2458
[ndbd default] (NDB Cluster), 2452
[ndbd] (NDB Cluster), 2452
[ndb_mgmd] (NDB Cluster), 2458
[shm] (NDB Cluster), 2460
[tcp] (NDB Cluster), 2460

\" (double quote), 1124
\' (single quote), 1124
\. (mysql client command), 242, 327
\0 (ASCII NUL), 1124, 1609
\b (backspace), 1124, 1609
\n (linefeed), 1124, 1609
\n (newline), 1124, 1609
\N (NULL), 1609
\r (carriage return), 1124, 1609
\t (tab), 1124, 1609
\Z (Control+Z) ASCII 26, 1124, 1609
\\ (escape), 1124
^ (bitwise XOR), 1429
_ (wildcard character), 1125
_ai collation sufffix, 1182
_as collation sufffix, 1182
_bin collation sufffix, 1182, 1203
_ci collation sufffix, 1182
_cs collation sufffix, 1182
_rowid
  SELECT statements, 1528, 1545, 1546
`, 1132
| (bitwise OR), 1428
|| (logical OR), 1333
~ (invert bits), 1429

A

abort-on-error option
  ndb_move_data, 2652
abort-slave-event-count option
  mysqld, 2261
aborted clients, 3253
aborted connection, 3253
Aborted_clients status variable, 651
Aborted_connects status variable, 651
ABS(), 1340
access control, 781, 798
access denied errors, 3245
access privileges, 781
account management, 781
account names, 796
accounts
  adding privileges, 803
  anonymous user, 189
  creating, 803
  default, 189
  deleting, 805
  root, 189
accounts table
  performance_schema, 3132
ACID, 1820, 1825, 3829
ACLs, 781
ACOS(), 1340
activating plugins, 717
ActiveState Perl, 216
adaptive flushing, 3829
adaptive hash index, 1837, 3829
add-drop-database option

mysqldump, 356
add-drop-table option
mysqldump, 356
add-drop-trigger option
mysqldump, 356
add-locks option
mysqldump, 365
ADDDATE(), 1350
adding
character sets, 1229
new account privileges, 803
new user privileges, 803
addition (+), 1338
ADDTIME(), 1350
addtodest option
mysqlhotcopy, 445
administration
server, 331
administration of NDB Cluster, 2626
administrative programs, 254
ADO.NET, 3830
AES_DECRYPT(), 1431
AES_ENCRYPT(), 1431
After create
thread state, 1110
age
calculating, 231
aggregate functions, 1471
AIO, 3830
alias names
case sensitivity, 1136
aliases
for expressions, 1480
for tables, 1624
in GROUP BY clauses, 1480
names, 1132
on expressions, 1624
ALL, 1640
SELECT modifier, 1627
ALL join type
optimizer, 1060
ALL privilege, 785
ALL PRIVILEGES privilege, 785
all-databases option
mysqlcheck, 343
mysqldump, 363
all-in-1 option
mysqlcheck, 343
all-tablespaces option
mysqldump, 356
allocating local table
thread state, 1116
allow-keywords option
mysqldump, 357
allow-suspicious-udfs option
mysqld, 524
allowold option
mysqlhotcopy, 445

ALLOW_INVALID_DATES SQL mode, 670
ALTER COLUMN, 1509
ALTER DATABASE, 1496
ALTER EVENT, 1497
  and replication, 2360
ALTER FUNCTION, 1499
ALTER IGNORE TABLE
  deprecated features, 19
ALTER LOGFILE GROUP, 1499
  (see also NDB Cluster Disk Data)
ALTER privilege, 785
ALTER PROCEDURE, 1500
ALTER ROUTINE privilege, 785
ALTER SCHEMA, 1496
ALTER SERVER, 1501
ALTER TABLE, 1501, 1509, 3272
  and replication metadata repositories, 2316
  ROW_FORMAT, 1942
ALTER TABLESPACE, 1520
  (see also NDB Cluster Disk Data)
ALTER USER statement, 1720
ALTER VIEW, 1521
altering
  database, 1496
  schema, 1496
altering table
  thread state, 1110
altering user accounts, 1720
ANALYSE()
  PROCEDURE, 1032
analyze option
  myisamchk, 399
  mysqlcheck, 343
ANALYZE TABLE
  and partitioning, 2911
ANALYZE TABLE statement, 1739
Analyzing
  thread state, 1110
AND
  bitwise, 1428
  logical, 1332
anonymous user, 189, 189, 798, 801
ANSI, 3830
ANSI mode
  running, 40
ansi option
  mysqld, 524
ANSI SQL mode, 669, 674
ANSI_QUOTES SQL mode, 670
Antelope, 3830
Antelope file format, 1933, 1985
ANY, 1639
Apache, 249
API, 3830
API node (NDB Cluster)
  defined, 2382
API nodes (see SQL nodes)
APIs, 3177

list of, 54
Perl, 3185
append option
ndb_restore, 2663
application programming interface (API), 3830
apply, 3830
apply-slave-statements option
mysqldump, 359
apply_status table (OBSOLETE), 2835
(see also NDB Cluster replication)
approximate-value literals, 1486
approximate-value numeric literals, 1126, 1487
Arbitration, 2519
ArbitrationDelay, 2480, 2552
ArbitrationRank, 2480, 2551
ArbitrationTimeout, 2518
arbitrator_validity_detail
ndbinfo table, 2776
arbitrator_validity_summary
ndbinfo table, 2777
ARCHIVE storage engine, 2123, 2141
Area(), 1462
arithmetic expressions, 1338
arithmetic functions, 1428
arithmetic operators, 1428
.ARM file, 3829
.ARZ file, 3829
AS, 1624, 1630
AsBinary(), 1457
ASCII(), 1369
ASIN(), 1340
ASP.net, 3830
assembly, 3830
assignment operator
:=, 1333
=, 1334
assignment operators, 1333
AsText(), 1457
asymmetric_decrypt(), 927
asymmetric_derive(), 927
asymmetric_encrypt(), 928
asymmetric_sign(), 928
asymmetric_verify(), 929
asynchronous I/O, 1904, 3831
asynchronous replication (see NDB Cluster replication)
ATAN(), 1341
ATAN2(), 1341
atomic, 3831
atomic DDL, 3831
atomic instruction, 3831
attackers
security against, 775
attribute demotion
replication, 2353
attribute promotion
replication, 2353
audit-log option
mysqld, 901

audit_log plugin, 883
  installing, 885
  startup failure, 896
audit_log_buffer_size system variable, 902
audit_log_connection_policy system variable, 902
audit_log_current_session system variable, 903
Audit_log_current_size status variable, 907
Audit_log_events status variable, 907
Audit_log_events_filtered status variable, 907
Audit_log_events_lost status variable, 907
Audit_log_events_written status variable, 907
Audit_log_event_max_drop_size status variable, 907
audit_log_exclude_accounts system variable, 903
audit_log_file system variable, 903
audit_log_flush system variable, 903
audit_log_format system variable, 904
audit_log_include_accounts system variable, 904
audit_log_policy system variable, 905
audit_log_rotate_on_size system variable, 905
audit_log_statement_policy system variable, 906
audit_log_strategy system variable, 906
Audit_log_total_size status variable, 907
Audit_log_write_waits status variable, 907
authentication
  for the InnoDB memcached interface, 2089
authentication plugin
  authentication_pam, 853
  authentication_windows, 863
  authentication_windows_client, 863
  auth_socket, 867
  auth_test_plugin, 869
  mysql_clear_password, 852
  mysql_native_password, 842
  mysql_old_password, 843
  sha256_password, 848
  test_plugin_server, 869
authentication_pam authentication plugin, 853
AUTHENTICATION_PAM_LOG environment variable, 457, 862
authentication_windows authentication plugin, 863
authentication_windows_client authentication plugin, 863
authentication_windows_log_level system variable, 546
authentication_windows_use_principal_name system variable, 546
auth_socket authentication plugin, 867
auth_test_plugin authentication plugin, 869
auto-generate-sql option
  mysqlslap, 383
auto-generate-sql-add-autoincrement option
  mysqlslap, 383
auto-generate-sql-execute-number option
  mysqlslap, 383
auto-generate-sql-guid-primary option
  mysqlslap, 384
auto-generate-sql-load-type option
  mysqlslap, 384
auto-generate-sql-secondary-indexes option
  mysqlslap, 384
auto-generate-sql-unique-query-number option
  mysqlslap, 384

auto-generate-sql-unique-write-number option
  mysqlslap, 384
auto-generate-sql-write-number option
  mysqlslap, 384
auto-inc lock, 1872
auto-increment, 1851, 1851, 1856, 1857, 3831
auto-increment locking, 3831
auto-rehash option
  mysql, 310
auto-repair option
  mysqlcheck, 343
auto-vertical-output option
  mysql, 310
auto.cnf file, 2249
  and SHOW SLAVE HOSTS statement, 1788
autocommit, 3832
autocommit mode, 1879
autocommit system variable, 547
automatic_sp_privileges system variable, 547
AutoReconnect
  API and SQL nodes, 2554
AUTO_INCREMENT, 247, 1257
  and NULL values, 3268
  and replication, 2349
auto_increment_increment system variable, 2256
auto_increment_offset system variable, 2259
availability, 3832
AVG(), 1472
AVG(DISTINCT), 1472
avoid_temporal_upgrade system variable, 548

B

B-tree, 3832
B-tree indexes, 1023, 1858
background threads, 1905
  read, 1904
  write, 1904
backslash
  escape character, 1123
backspace (\b), 1124, 1609
backticks, 3832
backup, 3832
BACKUP Events (NDB Cluster), 2725
backup identifiers
  native backup and restore, 2745
backup option
  myisamchk, 398
  myisampack, 409
backup-path option
  ndb_restore, 2663
BackupDataBufferSize, 2525, 2746
BackupDataDir, 2487
BackupDiskWriteSpeedPct, 2526
backupid option
  ndb_restore, 2664
BackupLogBufferSize, 2526, 2746
BackupMaxWriteSize, 2528, 2747
BackupMemory, 2527, 2746

BackupReportFrequency, 2527
backups, 937, 3190
  databases and tables, 348, 444
  in NDB Cluster, 2659, 2743, 2743, 2743, 2746
  in NDB Cluster replication, 2842
  InnoDB, 2074
  with mysqldump, 946
backups, troubleshooting
  in NDB Cluster, 2747
BackupWriteSize, 2527, 2746
back_log system variable, 548
Barracuda, 3833
Barracuda file format, 1922, 1933, 1985
base64-output option
  mysqlbinlog, 426
basedir option
  mysql.server, 288
  mysqld, 524
  mysqld_safe, 283
  mysql_install_db, 295
  mysql_plugin, 298
  mysql_upgrade, 303
basedir system variable, 549
batch mode, 241
batch option
  mysql, 310
batch SQL files, 307
BatchByteSize, 2552
Batched Key Access
  optimization, 987, 988
batched updates (NDB Cluster Replication), 2839
BatchSize, 2552
BatchSizePerLocalScan, 2496
BEGIN, 1653, 1687
  labels, 1687
  XA transactions, 1667
BENCHMARK(), 1440
benchmarks, 1105, 1106
beta, 3833
BETWEEN ... AND, 1328
bidirectional replication
  in NDB Cluster, 2848, 2851
big5, 3219
BIGINT data type, 1254
big_tables system variable, 549
BIN(), 1369
BINARY, 1415
binary collation, 1203
BINARY data type, 1276, 1279
binary distributions
  installing, 74
binary log, 701, 3833
  event groups, 1678
binary logging
  and NDB Cluster, 2409
binary-as-hex option
  mysql, 310
binary-mode option

mysql, 311
bind-address option
mysql, 311
mysqladmin, 336
mysqlbinlog, 426
mysqlcheck, 343
mysqldump, 353
mysqlimport, 370
mysqlshow, 377
ndb_mgmd, 2620
bind_address system variable, 549
binlog, 3833
Binlog Dump
thread command, 1108
BINLOG statement, 1804
mysqlbinlog output, 436
binlog-checksum option
mysqld, 2292
binlog-do-db option
mysqld, 2289
binlog-ignore-db option
mysqld, 2291
binlog-row-event-max-size option
mysqlbinlog, 426
mysqld, 2288
binlogging_impossible_mode system variable, 2296
Binlog_cache_disk_use status variable, 651
binlog_cache_size system variable, 2292
Binlog_cache_use status variable, 651
binlog_checksum system variable, 2293
binlog_direct_non_transactional_updates system variable, 2293
binlog_error_action system variable, 2294
binlog_format
BLACKHOLE, 2350
binlog_format system variable, 2295
binlog_gtid_simple_recovery, 2306
binlog_index table (OBSOLETE) (see NDB Cluster replication)
binlog_max_flush_queue_time system variable, 2297
binlog_order_commits system variable, 2297
binlog_rows_query_log_events system variable, 2299
binlog_row_image system variable, 2298
Binlog_stmt_cache_disk_use status variable, 651
binlog_stmt_cache_size system variable, 2299
Binlog_stmt_cache_use status variable, 651
BIT data type, 1252
bit functions, 1428
example, 246
bit operators, 1428
bit-value literal introducer, 1130
bit-value literals, 1130
BIT_AND(), 1473
BIT_COUNT, 246
BIT_COUNT(), 1429
BIT_LENGTH(), 1370
BIT_OR, 246
BIT_OR(), 1473
BIT_XOR(), 1473
BLACKHOLE

binlog_format, 2350
replication, 2350
BLACKHOLE storage engine, 2123, 2143
blind query expansion, 1402, 3833
BLOB, 3833
BLOB columns
default values, 1281
indexing, 1019, 1543
inserting binary data, 1125
size, 1306
BLOB data type, 1277, 1280
blob-info option
ndb_desc, 2643
Block Nested-Loop
optimization, 986, 988
Block Nested-Loop join algorithm, 976
block-search option
myisamchk, 400
blocked hosts
unblocking, 681
blocks
ndbinfo table, 2777
block_encryption_mode system variable, 550
BOOL data type, 1253
BOOLEAN data type, 1253
boolean literals, 1131
boolean options, 266
Boolean search, 1397
bootstrap option
mysqld, 524
bottleneck, 3833
bounce, 3834
brackets
square, 1252
brief option
mysqlaccess, 421
browser-start-page option
ndb_setup.py, 2685
buddy allocator, 2029, 3834
buffer, 3834
buffer pool, 1075, 1896, 1897, 1898, 1899, 1900, 3834
and compressed tables, 1930
monitoring, 1831
buffer pool instance, 3834
buffer sizes, 1075
client, 3177
Buffer(), 1465
bugs
known, 3273
NDB Cluster
reporting, 2645
reporting, 2, 35
bugs database, 35
bugs.mysql.com, 35
builddir option
mysql_install_db, 295
BuildIndexThreads, 2529
BUILD_CONFIG option

CMake, 169
built-in, 3834
bulk loading
for InnoDB tables, 1042
for MyISAM tables, 1050
bulk_insert_buffer_size system variable, 551, 2130
business rules, 3834
Bytes_received status variable, 651
Bytes_sent status variable, 651

C

C, 3835
C API, 3177, 3185, 3835
FAQ, 3231
C#, 3835
C++, 3179, 3835
C:\my.cnf option file, 736
ca-certs-file option
ndb_setup.py, 2685
cache, 3835
CACHE INDEX
and partitioning, 2927
CACHE INDEX statement, 1804
caches
clearing, 1806
cache_policies table, 2107
calculating
aggregate value for a set of rows, 1471
cardinality, 1777
dates, 231
calendar, 1392
CALL, 1587
can't create/write to file, 3254
Can't reopen table
error message, 3273
cardinality, 1000, 3835
carriage return (\r), 1124, 1609
CASE, 1335, 1690
case sensitivity
access checking, 795
account names, 797
in identifiers, 1136
in names, 1136
in searches, 3265
in string comparisons, 1383
of database names, 41
of replication filtering options, 2323
of table names, 41
CAST, 1416
cast functions, 1412
cast operators, 1412
casts, 1321, 1326, 1412
CC environment variable, 180, 457
CEIL(), 1341
CEILING(), 1341
Centroid(), 1462
cert-file option
ndb_setup.py, 2685

.cfg file, 3835
cflags option
  mysql_config, 452
change buffer, 1834, 3836
  monitoring, 1836
change buffering, 3836
  disabling, 1835
CHANGE MASTER TO, 1672
  in NDB Cluster, 2837
Change user
  thread command, 1108
changes to privileges, 806
changing
  column, 1507
  field, 1507
  socket location, 288, 3264
  table, 1501, 1509, 3272
Changing master
  thread state, 1120
CHAR data type, 1274, 1275
CHAR VARYING data type, 1276
CHAR(), 1370
CHARACTER data type, 1275
character set introducer, 1188
character set repertoire, 1210
character sets, 1176
  adding, 1229
  and replication, 2350
  Asian, 1224
  Baltic, 1223
  binary, 1227
  Central European, 1221
  Cyrillic, 1223
  Middle East, 1222
  repertoire, 1178
  restrictions, 1228
  South European, 1222
  Unicode, 1215
  West European, 1220
CHARACTER VARYING data type, 1276
character-set-client-handshake option
  mysqld, 524
character-sets-dir option
  myisamchk, 398
  myisampack, 409
  mysql, 311
  mysqladmin, 336
  mysqlbinlog, 427
  mysqlcheck, 343
  mysqldump, 358
  mysqlimport, 370
  mysqlshow, 377
  mysql_upgrade, 303
  ndb_move_data, 2653
character-sets-dir option (NDB Cluster programs), 2695, 2695
characters
  multibyte, 1232
CHARACTER_LENGTH(), 1370

CHARACTER_SETS
  INFORMATION_SCHEMA table, 2979
character_sets_dir system variable, 553
character_set_client system variable, 551
character_set_connection system variable, 551
character_set_database system variable, 552
character_set_filesystem system variable, 552
character_set_results system variable, 552
character_set_server system variable, 552
character_set_system system variable, 553
charset command
  mysql, 320
charset option
  comp_err, 293
CHARSET(), 1440
CHAR_LENGTH(), 1370
check option
  myisamchk, 397
  mysqlcheck, 343
check options
  myisamchk, 397
CHECK TABLE
  and partitioning, 2911
CHECK TABLE statement, 1740
check-only-changed option
  myisamchk, 397
  mysqlcheck, 343
check-orphans option
  ndb_blob_tool, 2629
check-upgrade option
  mysqlcheck, 343
checking
  tables for errors, 955
Checking master version
  thread state, 1118
checking permissions
  thread state, 1110
checking privileges on cached query
  thread state, 1117
checking query cache for query
  thread state, 1117
Checking table
  thread state, 1110
checkpoint, 3836
CHECKPOINT Events (NDB Cluster), 2721
checkpoint option
  mysqlhotcopy, 445
Checksum, 2599
checksum, 3836
Checksum (NDB Cluster), 2604
checksum errors, 153
CHECKSUM TABLE
  and replication, 2350
CHECKSUM TABLE statement, 1744
child table, 3837
Chinese, Japanese, Korean character sets
  frequently asked questions, 3219
choosing

a MySQL version, 60
data types, 1307
chroot option
mysqld, 525
mysqlhotcopy, 445
circular replication
in NDB Cluster, 2828, 2848, 2851
CJK (Chinese, Japanese, Korean)
Access, PHP, etc., 3219
availability of specific characters, 3219
big5, 3219
character sets available, 3219
characters displayed as question marks, 3219
CJKV, 3219
collations, 3219, 3219
conversion problems with Japanese character sets, 3219
data truncation, 3219
Database and table names, 3219
documentation in Chinese, 3219
documentation in Japanese, 3219
documentation in Korean, 3219
FAQ, 3219
gb2312, gbk, 3219
Japanese character sets, 3219
Korean character set, 3219
LIKE and FULLTEXT, 3219
MySQL 4.0 behavior, 3219
ORDER BY treatment, 3219, 3219
problems with Access, PHP, etc., 3219
problems with Big5 character sets (Chinese), 3219
problems with data truncation, 3219
problems with euckr character set (Korean), 3219
problems with GB character sets (Chinese), 3219
problems with LIKE and FULLTEXT, 3219
problems with Yen sign (Japanese), 3219
rejected characters, 3219
sort order problems, 3219, 3219
sorting problems, 3219, 3219
testing availability of characters, 3219
Unicode collations, 3219
Vietnamese, 3219
Yen sign, 3219
clean page, 3837
clean shutdown, 690, 732, 2363, 3837
cleaning up
thread state, 1110
clear command
mysql, 321
Clearing
thread state, 1121
clearing
caches, 1806
client, 3837
signal handling, 460
client connections, 676
client libraries, 3837
client programs, 253
client tools, 3177

client-side prepared statement, 3837
clients
  debugging, 743
CLOB, 3837
cloning tables, 1562
CLOSE, 1695
Close stmt
  thread command, 1108
closing
  tables, 1032
closing tables
  thread state, 1111
cluster database (OBSOLETE) (see NDB Cluster replication)
cluster logs, 2717, 2719
cluster.binlog_index table (OBSOLETE) (see NDB Cluster replication)
clustered index, 3837
  InnoDB, 1857
Clustering (see NDB Cluster)
CLUSTERLOG commands (NDB Cluster), 2719
CLUSTERLOG STATISTICS command (NDB Cluster), 2725
cluster_operations
  ndbinfo table, 2778
cluster_replication database (OBSOLETE) (see NDB Cluster replication)
cluster_transactions
  ndbinfo table, 2779
CMake
  BUILD_CONFIG option, 169
  CMAKE_BUILD_TYPE option, 169
  CMAKE_CXX_FLAGS option, 177
  CMAKE_C_FLAGS option, 177
  CMAKE_INSTALL_PREFIX option, 170
  COMPILATION_COMMENT option, 172
  CPACK_MONOLITHIC_INSTALL option, 169
  DEFAULT_CHARSET option, 172
  DEFAULT_COLLATION option, 172
  ENABLED_LOCAL_INFILE option, 173, 779
  ENABLED_PROFILING option, 173
  ENABLE_DEBUG_SYNC option, 173
  ENABLE_DOWNLOADS option, 173
  ENABLE_DTRACE option, 173
  ENABLE_GCOV option, 173
  ENABLE_GPROF option, 173
  IGNORE_AIO_CHECK option, 174
  INNODB_PAGE_ATOMIC_REF_COUNT option, 174
  INSTALL_BINDIR option, 170
  INSTALL_DOCDIR option, 170
  INSTALL_DOCREADMEDIR option, 170
  INSTALL_INCLUDEDIR option, 170
  INSTALL_INFODIR option, 170
  INSTALL_LAYOUT option, 170
  INSTALL_LIBDIR option, 170
  INSTALL_MANDIR option, 170
  INSTALL_MYSQLSHAREDIR option, 170
  INSTALL_MYSQLTESTDIR option, 170
  INSTALL_PLUGINDIR option, 170
  INSTALL_SBINDIR option, 171
  INSTALL_SCRIPTDIR option, 171
  INSTALL_SECURE_FILE_PRIVDIR option, 171

INSTALL_SECURE_FILE_PRIV_EMBEDDEDDIR option, 171
INSTALL_SHAREDIR option, 171
INSTALL_SQLBENCHDIR option, 171
INSTALL_SUPPORTFILESDIR option, 171
MEMCACHED_HOME option, 178
MYSQL_DATADIR option, 171
MYSQL_MAINTAINER_MODE option, 174
MYSQL_PROJECT_NAME option, 174
MYSQL_TCP_PORT option, 174
MYSQL_UNIX_ADDR option, 174
ODBC_INCLUDES option, 171
ODBC_LIB_DIR option, 171
OPTIMIZER_TRACE option, 174
options, 165
REPRODUCIBLE_BUILD option, 174
running after prior invocation, 161, 179
SUNPRO_CXX_LIBRARY option, 177
SYSCONFDIR option, 171
TMPDIR option, 171
VERSION file, 180
WITH_ASAN option, 174
WITH_BUNDLED_LIBEVENT option, 178
WITH_BUNDLED_MEMCACHED option, 178
WITH_CLASSPATH option, 178
WITH_DEBUG option, 175
WITH_DEFAULT_COMPILER_OPTIONS option, 177
WITH_DEFAULT_FEATURE_SET option, 175
WITH_EDITLINE option, 175
WITH_EMBEDDED_SERVER option, 175
WITH_EMBEDDED_SHARED_LIBRARY option, 175
WITH_ERROR_INSERT option, 178
WITH_EXTRA_CHARSETS option, 175
WITH_GMOCK option, 175
WITH_INNODB_MEMCACHED option, 175
WITH_LIBEDIT option, 176
WITH_LIBEVENT option, 175
WITH_LIBWRAP option, 176
WITH_NDBCLUSTER option, 179
WITH_NDBCLUSTER_STORAGE_ENGINE option, 178
WITH_NDBMTD option, 179
WITH_NDB_BINLOG option, 179
WITH_NDB_DEBUG option, 179
WITH_NDB_JAVA option, 179
WITH_NDB_PORT option, 179
WITH_NDB_TEST option, 179
WITH_NUMA option, 176
WITH_SSL option, 176
WITH_SYMVER16 option, 176
WITH_UNIT_TESTS option, 177
WITH_UNIXODBC option, 177
WITH_VALGRIND option, 177
WITH_ZLIB option, 177
CMakeCache.txt file, 179
CMAKE_BUILD_TYPE option
CMake, 169
CMAKE_CXX_FLAGS option
CMake, 177
CMAKE_C_FLAGS option

CMake, 177
CMAKE_INSTALL_PREFIX option
CMake, 170
COALESCE(), 1328
coercibility
collation, 1201
COERCIBILITY(), 1441
cold backup, 3838
collating
strings, 1232
collation
adding, 1232
coercibility, 1201
INFORMATION_SCHEMA, 1206
modifying, 1233
COLLATION(), 1441
collations, 1176
Asian, 1224
Baltic, 1223
binary, 1203, 1227
Central European, 1221
Cyrillic, 1223
Middle East, 1222
naming conventions, 1181
PAD SPACE, 1204, 1279
South European, 1222
Unicode, 1215
West European, 1220
_ai suffix, 1182
_as suffix, 1182
_bin suffix, 1182, 1203
_ci suffix, 1182
_ss suffix, 1182
COLLATIONS
INFORMATION_SCHEMA table, 2979
COLLATION_CHARACTER_SET_APPLICABILITY
INFORMATION_SCHEMA table, 2980
collation_connection system variable, 553
collation_database system variable, 553
collation_server system variable, 554
column, 3838
changing, 1507
types, 1251
column alias
problems, 3268
quoting, 1133, 3268
column comments, 1544
column format, 1544
column index, 3838
column names
case sensitivity, 1136
column prefix, 3838
column storage, 1544
column-names option
mysql, 311
column-type-info option
mysql, 311
columns

displaying, 375
indexes, 1019
names, 1132
other types, 1308
selecting, 229
storage requirements, 1303
COLUMNS
INFORMATION_SCHEMA table, 2980
columns option
mysqlimport, 370
columns partitioning, 2879
columns per table
maximum, 1037
columns_priv table
system table, 693, 791
COLUMN_PRIVILEGES
INFORMATION_SCHEMA table, 2983
comma-separated values data, reading, 1608, 1630
command interceptor, 3838
command option precedence, 257
command options
mysql, 307
mysqladmin, 334
mysqld, 522
command options (NDB Cluster)
mysqld, 2557
ndbd, 2609
ndbinfo_select_all, 2617
ndb_mgm, 2627
ndb_mgmd, 2619
command syntax, 4
command-line history
mysql, 325
command-line options (NDB Cluster), 2694
command-line tool, 100, 307
commands
for binary distribution, 75
commands out of sync, 3255
comment syntax, 1172
comments
adding, 1172
starting, 45
comments option
mysql, 311
mysqldump, 357
COMMIT, 1653
XA transactions, 1667
commit, 3838
commit option
mysqlaccess, 421
mysqlslap, 384
committing alter table to storage engine
thread state, 1111
Committing events to binlog
thread state, 1120
compact option
mysqldump, 360
compact row format, 1940, 3838

comparison operators, 1326
comparisons
  access checking, 795
  account names, 797
compatibility
  between MySQL versions, 195
  with mSQL, 1386
  with ODBC, 616, 1135, 1255, 1322, 1330, 1544, 1632
  with Oracle, 42, 1475, 1508, 1815
  with PostgreSQL, 43
  with standard SQL, 39
compatible option
  mysqldump, 361
COMPILATION_COMMENT option
  CMake, 172
compiling
  optimizing, 1096
compiling MySQL server
  problems, 179
complete-insert option
  mysqldump, 361
completion_type system variable, 554
composite index, 3839
composite partitioning, 2892
compound statements, 1687
compress option
  mysql, 312
  mysqladmin, 336
  mysqlcheck, 343
  mysqldump, 353
  mysqlimport, 370
  mysqlshow, 377
  mysqlslap, 384
  mysql_upgrade, 303
COMPRESS(), 1432
compressed backup, 3839
compressed row format, 1942, 3839
compressed table, 3839
compressed tables, 408, 2134
CompressedBackup, 2528
CompressedLCP, 2507
compression, 1921, 3839
  algorithms, 1927
  application and schema design, 1925
  BLOBs, VARCHAR and TEXT, 1929
  buffer pool considerations, 1930
  compressed page size, 1926
  configuration characteristics, 1926
  connection, 280
  data and indexes, 1928
  data characteristics, 1923
  enabling for a table, 1922
  implementation, 1927
  information schema, 2028
  KEY_BLOCK_SIZE, 1926
  log file format, 1930
  modification log, 1928
  monitoring, 1926

overflow pages, 1929
overview, 1921
tuning, 1923
workload characteristics, 1925
compression failure, 3840
Compression status variable, 652
comp_err, 253, 292
charset option, 293
debug option, 293
debug-info option, 293
header-file option, 293
help option, 293
in-file option, 293
name-file option, 293
out-dir option, 293
out-file option, 293
state-file option, 293
version option, 293
CONCAT(), 1370
concatenation
string, 1123, 1370
CONCAT_WS(), 1371
concurrency, 1820, 3840
of commits, 1981
of threads, 2022
tickets, 1982
concurrency option
mysqlslap, 384
concurrent inserts, 1089, 1091
concurrent_insert system variable, 554
condition handling
INOUT parameters, 1719
new features, 17
OUT parameters, 1719
Conditions, 1696
conditions, 1772, 1802
cond_instances table
performance_schema, 3110
config-cache option
ndb_mgmd, 2620
config-file option
my_print_defaults, 453
ndb_config, 2633
ndb_mgmd, 2621
config.ini (NDB Cluster), 2428, 2468, 2469, 2626
configdir option
ndb_mgmd, 2621
configinfo option
ndb_config, 2632
configuration
NDB Cluster, 2451
server, 462
configuration file, 3840
configuration files, 823
configure option
MySQLInstallerConsole, 101
configuring backups
in NDB Cluster, 2746

configuring NDB Cluster, 2411, 2449, 2626, 2748
Configuring NDB Cluster (concepts), 2382
config_from_node option
  ndb_config, 2633
config_options table, 2107
config_params
  ndbinfo table, 2780
conflict detection status variables
  NDB Cluster Replication, 2857
conflict resolution
  and ndb_replication system table, 2853
  enabling, 2853
  in NDB Cluster Replication, 2851
  mysqld startup options, 2852
Connect
  thread command, 1108
connect command
  mysql, 321
CONNECT command (NDB Cluster), 2700
connect option
  ndb_restore, 2664
Connect Out
  thread command, 1108
connect-delay option
  ndbd, 2611
  ndbmtd, 2611
connect-expired-password option
  mysql, 312
connect-retries option
  ndbd, 2611
  ndbmtd, 2611
  ndb_mgm, 2627
connect-retries option (NDB Cluster programs), 2695
connect-retry-delay option
  ndbd, 2611
  ndbmtd, 2611
connect-retry-delay option (NDB Cluster programs), 2695
connect-string option (NDB Cluster programs), 2697
connect-timeout option
  mysql, 312
  mysqladmin, 336
ConnectBackoffMaxTime, 2555
ConnectCheckIntervalDelay, 2511
connecting
  remotely with SSH, 841
  to the server, 219, 276
  verification, 798
Connecting to master
  thread state, 1118
connection, 3840
  aborted, 3253
connection compression, 280
CONNECTION Events (NDB Cluster), 2720
connection management, 676
connection pool, 3840
connection string, 3840 (see NDB Cluster)
connection-server-id option
  mysqlbinlog, 427

connection-timeout option
  ndb_error_reporter, 2646
ConnectionMap, 2550
connections option
  ndb_config, 2633
Connections status variable, 653
CONNECTION_CONTROL plugin
  installing, 871
  status variables, 876
  system variables, 875
Connection_control_delay_generated status variable, 877
connection_control_failed_connections_threshold system variable, 875
CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS
  INFORMATION_SCHEMA table, 3059
CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS plugin
  installing, 871
connection_control_max_connection_delay system variable, 876
connection_control_min_connection_delay system variable, 876
Connection_errors_accept status variable, 652
Connection_errors_internal status variable, 652
Connection_errors_max_connections status variable, 652
Connection_errors_peer_address status variable, 652
Connection_errors_select status variable, 653
Connection_errors_tcpwrap status variable, 653
CONNECTION_ID(), 1441
connector, 3841
Connector/C++, 3177, 3179, 3841
Connector/J, 3177, 3179, 3841
Connector/NET, 3177, 3180, 3841
Connector/ODBC, 3177, 3180, 3841
Connector/PHP, 3841
Connector/Python, 3177, 3180
Connectors, 3177
connect_timeout system variable, 555
consistent read, 3841
consistent reads, 1880
console option
  mysqld, 525
const table
  optimizer, 1058, 1627
constant table, 966
constraint, 3842
constraints, 45
  foreign keys, 1565
containers table, 2107
Contains(), 1469
contributing companies
  list of, 55
contributors
  list of, 48
Control+C
  statement termination, 307, 318
CONV(), 1341
conventions
  syntax, 2
  typographical, 2
CONVERT, 1416
CONVERT TO, 1510

converting HEAP to MyISAM
  thread state, 1111
CONVERT_TZ(), 1351
copy option
  mysqlaccess, 422
copy to tmp table
  thread state, 1111
copying databases, 210
copying tables, 1562
Copying to group table
  thread state, 1111
Copying to tmp table
  thread state, 1111
Copying to tmp table on disk
  thread state, 1111
core-file option
  mysqld, 525
core-file option (NDB Cluster programs), 2696
core-file-size option
  mysqld_safe, 283
core_file system variable, 556
correct-checksum option
  myisamchk, 398
correlated subqueries, 1642
corruption, 2113
  InnoDB, 2075
COS(), 1342
COT(), 1342
count option
  myisam_ftdump, 390
  mysqladmin, 336
  mysqlshow, 377
COUNT(), 1473
COUNT(DISTINCT), 1474
counter, 3842
counters
  ndbinfo table, 2780
counting
  table rows, 236
covering index, 3842
CPACK_MONOLITHIC_INSTALL option
  CMake, 169
CPU-bound, 3842
crash, 736, 3842
  recovery, 955
  repeated, 3260
  replication, 2363
crash recovery, 3842
  InnoDB, 2076
crash-me, 1105
crash-me program, 1105
crash-safe binary log, 15
crash-safe replication, 15, 2274, 2332
CrashOnCorruptedTuple, 2506
CRC32(), 1342
CREATE ... IF NOT EXISTS
  and replication, 2350
CREATE DATABASE, 1521

Create DB
  thread command, 1108
CREATE EVENT, 1522
  and replication, 2360
CREATE FUNCTION, 1533
CREATE FUNCTION statement, 1749
CREATE INDEX, 1526
CREATE LOGFILE GROUP, 1531
  (see also NDB Cluster Disk Data)
CREATE NODEGROUP command (NDB Cluster), 2702
create option
  mysqlslap, 384
CREATE privilege, 786
CREATE PROCEDURE, 1533
CREATE ROUTINE privilege, 786
CREATE SCHEMA, 1521
CREATE SERVER, 1538
CREATE TABLE, 1539
  DIRECTORY options
    and replication, 2356
  KEY_BLOCK_SIZE, 1926
  options for table compression, 1922
  ROW_FORMAT, 1942
CREATE TABLE ... SELECT
  and replication, 2351
CREATE TABLESPACE, 1572
  (see also NDB Cluster Disk Data)
CREATE TABLESPACE privilege, 786
CREATE TEMPORARY TABLES privilege, 786
CREATE TRIGGER, 1574
CREATE USER privilege, 786
CREATE USER statement, 803, 806, 1721
CREATE VIEW, 1576
CREATE VIEW privilege, 786
create-options option
  mysqldump, 361
create-schema option
  mysqlslap, 384
Created_tmp_disk_tables status variable, 653
Created_tmp_files status variable, 653
Created_tmp_tables status variable, 653
create_asymmetric_priv_key(), 929
create_asymmetric_pub_key(), 930
create_dh_parameters(), 930
create_digest(), 930
creating
  bug reports, 35
  database, 1521
  databases, 223
  default startup options, 259
  function, 1749
  schema, 1521
  tables, 225
Creating delayed handler
  thread state, 1116
Creating index
  thread state, 1111
Creating sort index

thread state, 1111
creating table
thread state, 1111
Creating tmp table
thread state, 1111
creating user accounts, 1721
CROSS JOIN, 1630
cross-bootstrap option
mysql_install_db, 295
Crosses(), 1467
CRUD, 3842
CR_SERVER_GONE_ERROR, 3251
CR_SERVER_LOST_ERROR, 3251
CSV data, reading, 1608, 1630
csv option
mysqlslap, 384
CSV storage engine, 2123, 2140
CURDATE(), 1351
CURRENT_DATE, 1351
CURRENT_TIME, 1351
CURRENT_TIMESTAMP, 1351
CURRENT_USER(), 1441
cursor, 3843
Cursors, 1694
CURTIME(), 1351
CXX environment variable, 180, 457
cxxflags option
mysql_config, 452

D

Daemon
thread command, 1109
daemon option
ndb_mgmd, 2622
daemon_memcached_enable_binlog system variable, 1967
daemon_memcached_engine_lib_name system variable, 1967
daemon_memcached_engine_lib_path system variable, 1967
daemon_memcached_option system variable, 1968
daemon_memcached_r_batch_size system variable, 1968
daemon_memcached_w_batch_size system variable, 1969
data
importing, 327, 368
loading into tables, 226
retrieving, 227
size, 1029
data dictionary, 3843
data directory, 3843
mysql_upgrade_info file, 302
DATA DIRECTORY
and replication, 2356
data files, 3843
data node (NDB Cluster)
defined, 2382
data nodes
memory allocation, 2530
data nodes (NDB Cluster), 2609, 2618
Data on disk (NDB Cluster)
and INFORMATION_SCHEMA.FILES table, 3048

Data truncation with CJK characters, 3219
data type
BIGINT, 1254
BINARY, 1276, 1279
BIT, 1252
BLOB, 1277, 1280
BOOL, 1253, 1308
BOOLEAN, 1253, 1308
CHAR, 1274, 1275
CHAR VARYING, 1276
CHARACTER, 1275
CHARACTER VARYING, 1276
DATE, 1261, 1263
DATETIME, 1261, 1263
DEC, 1254
DECIMAL, 1254, 1486
DOUBLE, 1255
DOUBLE PRECISION, 1255
ENUM, 1277, 1282
FIXED, 1254
FLOAT, 1255, 1255, 1255
GEOMETRY, 1289
GEOMETRYCOLLECTION, 1289
INT, 1254
INTEGER, 1254
LINESTRING, 1289
LONG, 1280
LONGBLOB, 1277
LONGTEXT, 1277
MEDIUMBLOB, 1277
MEDIUMINT, 1253
MEDIUMTEXT, 1277
MULTILINESTRING, 1289
MULTIPOINT, 1289
MULTIPOLYGON, 1289
NATIONAL CHAR, 1275
NATIONAL VARCHAR, 1276
NCHAR, 1275
NUMERIC, 1254
NVARCHAR, 1276
POINT, 1289
POLYGON, 1289
REAL, 1255
SET, 1277, 1285
SMALLINT, 1253
TEXT, 1277, 1280
TIME, 1262, 1264
TIMESTAMP, 1261, 1263
TINYBLOB, 1276
TINYINT, 1253
TINYTEXT, 1276
VARBINARY, 1276, 1279
VARCHAR, 1274, 1276
VARCHARACTER, 1276
YEAR, 1262, 1265
data types, 1251
date and time, 1259
new features, 18

numeric, 1252
string, 1274
data warehouse, 3843
data-file-length option
myisamchk, 398
database, 3843
altering, 1496
creating, 1521
deleting, 1580
renaming, 1585
Database information
obtaining, 1756
database metadata, 2976
database names
case sensitivity, 41, 1136
database objects
metadata, 1180
database option
mysql, 312
mysqlbinlog, 427
ndb_blob_tool, 2629
ndb_desc, 2643
ndb_index_stat, 2649
ndb_move_data, 2653
ndb_show_tables, 2687
DATABASE(), 1442
databases
backups, 937
copying, 210
creating, 223, 1521
defined, 4
displaying, 375
dumping, 348, 444
information about, 240
names, 1132
replicating, 2221
selecting, 224, 1818
symbolic links, 1098
using, 223
databases option
mysqlcheck, 343
mysqldump, 363
DataDir, 2480, 2486
datadir option
mysql.server, 288
mysqld, 525
mysqld_safe, 283
mysql_install_db, 295
mysql_plugin, 298
mysql_upgrade, 303
datadir system variable, 556
DataMemory, 2487
DATE, 3266
date and time data types, 1259
date and time functions, 1348
date calculations, 231
DATE columns
problems, 3266

DATE data type, 1261, 1263
date data types
  storage requirements, 1305
date literals, 1126
date values
  problems, 1264
DATE(), 1351
DATEDIFF(), 1351
dates
  used with partitioning, 2872
  used with partitioning (examples), 2875, 2887, 2892, 2915
DATETIME data type, 1261, 1263
datetime_format
  deprecated features, 19
datetime_format system variable, 556
DATE_ADD(), 1352
date_format
  deprecated features, 19
date_format system variable, 556
DATE_FORMAT(), 1353
DATE_SUB(), 1352, 1354
DAY(), 1354
Daylight Saving Time, 688, 1027, 1365
DAYNAME(), 1354
DAYOFMONTH(), 1354
DAYOFWEEK(), 1354
DAYOFYEAR(), 1354
db option
  mysqlaccess, 422
db table
  sorting, 801
  system table, 189, 693, 791
DB2 SQL mode, 674
DBI interface, 3185
DBI->quote, 1125
DBI->trace, 739
DBI/DBD interface, 3185
DBI_TRACE environment variable, 457, 739
DBI_USER environment variable, 457
DBUG package, 743
DCL, 1725, 1736, 3844
DDEX provider, 3844
DDL, 1496, 3844
DDL log, 714
deadlock, 1087, 1660, 1883, 1887, 1888, 1888, 1888, 2011, 2366, 3172, 3844
deadlock detection, 3844
DEALLOCATE PREPARE, 1682, 1686
deb file
  MySQL APT Repository, 137
  MySQL SLES Repository, 137
Debug
  thread command, 1109
debug option
  comp_err, 293
  myisamchk, 394
  myisampack, 409
  mysql, 312
  mysqlaccess, 422

mysqladmin, 336
mysqlbinlog, 428
mysqlcheck, 343
mysqld, 526
mysqldump, 357
mysqldumpslow, 443
mysqlhotcopy, 445
mysqlimport, 371
mysqlshow, 377
mysqlslap, 384
mysql_config_editor, 417
mysql_upgrade, 304
my_print_defaults, 453
debug option (NDB Cluster programs), 2696
debug system variable, 556
debug-check option
mysql, 312
mysqladmin, 336
mysqlbinlog, 428
mysqlcheck, 344
mysqldump, 357
mysqlimport, 371
mysqlshow, 377
mysqlslap, 385
mysql_upgrade, 304
debug-info option
comp_err, 293
mysql, 312
mysqladmin, 336
mysqlbinlog, 428
mysqlcheck, 344
mysqldump, 357
mysqlimport, 371
mysqlshow, 377
mysqlslap, 385
mysql_upgrade, 304
debug-level option
ndb_setup.py, 2685
debug-sync-timeout option
mysqld, 526
debugging
client, 743
MySQL, 736
server, 736
debugging support, 165
debug_sync system variable, 557
DEC data type, 1254
decimal arithmetic, 1486
DECIMAL data type, 1254, 1486
decimal point, 1252
DECLARE, 1688
DECODE(), 1433
decode_bits myisamchk variable, 395
DEFAULT
constraint, 47
removed features, 20
default
privileges, 189

default accounts, 189
default host name, 276
default installation location, 74
default options, 259
default proxy user, 817
DEFAULT value clause, 1302, 1543
default values, 1302, 1543, 1595
  BLOB and TEXT columns, 1281
  explicit, 1302
  implicit, 1302
  suppression, 47
DEFAULT(), 1481
default-auth option, 271
  mysql, 312
  mysqladmin, 336
  mysqlbinlog, 428
  mysqlcheck, 345
  mysqldump, 354
  mysqlimport, 371
  mysqlshow, 377
  mysqlslap, 385
  mysql_upgrade, 304
default-authentication-plugin option
  mysqld, 526
default-character-set option
  mysql, 312
  mysqladmin, 336
  mysqlcheck, 344
  mysqldump, 358
  mysqlimport, 371
  mysqlshow, 377
  mysql_upgrade, 304
default-time-zone option
  mysqld, 527
DefaultHashMapSize, 2497, 2554
DefaultOperationRedoProblemAction
  API and SQL nodes, 2554
defaults
  embedded, 3182
defaults-extra-file option, 264
  myisamchk, 394
  mysql, 313
  mysqladmin, 337
  mysqlbinlog, 428
  mysqlcheck, 344
  mysqld, 527
  mysqldump, 355
  mysqld_multi, 289
  mysqld_safe, 283
  mysqlimport, 371
  mysqlshow, 378
  mysqlslap, 385
  mysql_install_db, 295
  mysql_upgrade, 304
  my_print_defaults, 453
  NDB client programs, 2696
defaults-file option, 265
  myisamchk, 395

mysql, 313
mysqladmin, 337
mysqlbinlog, 428
mysqlcheck, 344
mysqld, 528
mysqldump, 355
mysqld_multi, 289
mysqld_safe, 283
mysqlimport, 371
mysqlshow, 378
mysqlslap, 385
mysql_install_db, 296
mysql_upgrade, 304
my_print_defaults, 453
NDB client programs, 2696
defaults-group-suffix option, 265
myisamchk, 395
mysql, 313
mysqladmin, 337
mysqlbinlog, 429
mysqlcheck, 344
mysqld, 528
mysqldump, 356
mysqlimport, 371
mysqlshow, 378
mysqlslap, 385
mysql_upgrade, 304
my_print_defaults, 453
NDB client programs, 2697
DEFAULT_CHARSET option
CMake, 172
DEFAULT_COLLATION option
CMake, 172
default_storage_engine system variable, 557
default_tmp_storage_engine system variable, 558
default_week_format system variable, 558
DEFINER privileges, 1775, 2959
DEGREES(), 1342
delay option
ndbinfo_select_all, 2617
DELAYED, 1601
INSERT modifier, 1597
when ignored, 1597
Delayed insert
thread command, 1109
delayed inserts
thread states, 1116
delayed replication, 2347
delayed-insert option
mysqldump, 364
Delayed_errors status variable, 653
delayed_insert_limit, 1603
delayed_insert_limit system variable, 559
Delayed_insert_threads status variable, 653
delayed_insert_timeout system variable, 560
delayed_queue_size system variable, 560
Delayed_writes status variable, 653
delay_key_write system variable, 558, 2130

DELETE, 1588
  and NDB Cluster, 2403
delete, 3844
delete buffering, 3845
delete option
  mysqlimport, 372
  ndb_index_stat, 2649
DELETE privilege, 786
delete-master-logs option
  mysqldump, 359
delete-orphans option
  ndb_blob_tool, 2629
deleting
  accounts, 805
  database, 1580
  foreign key, 1510, 1569
  function, 1750
  index, 1509, 1581
  primary key, 1509
  rows, 3269
  schema, 1580
  table, 1583
  user, 1724
  users, 1724
deleting from main table
  thread state, 1111
deleting from reference tables
  thread state, 1111
deletion
  mysql.sock, 3264
delimiter command
  mysql, 321
delimiter option
  mysql, 313
  mysqlslap, 385
  ndb_select_all, 2682
demo_test table, 2085
denormalized, 3845
deprecated features, 18
  --language, 19
  --skip-innodb, 19
  ALTER IGNORE TABLE, 19
  datetime_format, 19
  date_format, 19
  ERROR_FOR_DIVISION_BY_ZERO, 19
  GROUP BY sorting, 19
  have_profiling, 19
  innodb_additional_mem_pool_size, 19
  innodb_locks_unsafe_for_binlog, 19
  innodb_use_sys_malloc, 19
  msql2mysql, 19
  mysqlaccess, 19
  mysqlbug, 19
  mysqlhotcopy, 19
  mysql_convert_table_format, 19
  mysql_find_rows, 19
  mysql_fix_extensions, 19
  mysql_old_password, 19

mysql_setpermission, 19
mysql_waitpid, 19
mysql_zap, 19
NO_ZERO_DATE, 19
NO_ZERO_IN_DATE, 19
OLD_PASSWORD(), 19
old_passwords, 19
pre-4.1 passwords, 19
profiling, 19
profiling_history_size, 19
secure_auth, 19
SQL mode, 19
timed_mutexes, 19
time_format, 19
derived tables, 1643
optimization, 1001, 1009
des-key-file option
mysqld, 528
DESC, 1814
descending index, 3845
descending option
ndb_select_all, 2681
DESCRIBE, 240, 1814
description option
myisamchk, 400
design
issues, 3273
DES_DECRYPT(), 1433
DES_ENCRYPT(), 1433
detach option
mysqlslap, 386
development of NDB Cluster, 2388
development source tree, 162
dictionary collation, German, 1178, 1221, 1221
dict_obj_types
ndbinfo table, 2782
diff-default option
ndb_config, 2633
digits, 1252
Dimension(), 1458
directory structure
default, 74
dirty page, 1969, 3845
dirty read, 3845
disable named command
mysql, 313
--disable option prefix, 266
disable-indexes option
ndb_restore, 2664
disable-keys option
mysqldump, 365
disable-log-bin option
mysqlbinlog, 429
DISCARD TABLESPACE, 1511, 1846
discard_or_import_tablespace
thread state, 1111
disconnect-slave-event-count option
mysqld, 2261

disconnecting
  from the server, 219
disconnect_on_expired_password system variable, 560
Disjoint(), 1469
Disk Data tables (NDB Cluster) (see NDB Cluster Disk Data)
disk failure
  InnoDB, 2075
disk full, 3262
disk I/O, 1044
disk option
  ndb_select_all, 2682
disk performance, 1096
disk-based, 3845
disk-bound, 3845
DiskCheckpointSpeed, 2516
DiskCheckpointSpeedInRestart, 2516
DiskIOThreadPool, 2542, 2545
Diskless, 2506
diskpagebuffer
  ndbinfo table, 2786
DiskPageBufferEntries, 2540
DiskPageBufferMemory, 2541, 2545
disks
  splitting data across, 1099
DiskSyncSize, 2515
disk_write_speed_aggregate
  ndbinfo table, 2783
disk_write_speed_aggregate_node
  ndbinfo table, 2785
disk_write_speed_base
  ndbinfo table, 2782
display size, 1251
display triggers, 1799
display width, 1251
displaying
  database information, 375
  information
    Cardinality, 1777
    Collation, 1777
    SHOW, 1756, 1760, 1798
    SHOW statement, 1776, 1778
  table status, 1795
DISTINCT, 229, 996
  AVG(), 1472
  COUNT(), 1474
  MAX(), 1474
  MIN(), 1475
  SELECT modifier, 1627
  SUM(), 1475
DISTINCTROW
  SELECT modifier, 1627
distributed privileges (NDB Cluster), 2759
  and NDB API applications, 2762
DIV, 1339
division (/), 1339
div_precision_increment system variable, 561
DML, 1587, 3845
  DELETE statement, 1588

INSERT statement, 1594
UPDATE statement, 1649
DNS, 678
DO, 1592
DocBook XML
documentation source format, 4
Docker images
on Windows, 153
document id, 3846
Documentation
in Chinese, 3219
in Japanese, 3219
in Korean, 3219
Documenters
list of, 53
dont-ignore-systab-0 option
ndb_restore, 2664
DOUBLE data type, 1255
DOUBLE PRECISION data type, 1255
double quote (\"), 1124
doublewrite buffer, 657, 1943, 1984, 3846
downgrades
NDB Cluster, 2435, 2729
downgrading, 211
downloading, 61
drop, 3846
DROP ... IF EXISTS
and replication, 2356
DROP DATABASE, 1580
Drop DB
thread command, 1109
DROP EVENT, 1581
DROP FOREIGN KEY, 1510, 1569
DROP FUNCTION, 1582
DROP FUNCTION statement, 1750
DROP INDEX, 1509, 1581
DROP LOGFILE GROUP, 1582
(see also NDB Cluster Disk Data)
DROP NODEGROUP command (NDB Cluster), 2702
DROP PREPARE, 1686
DROP PRIMARY KEY, 1509
DROP privilege, 786
DROP PROCEDURE, 1582
DROP SCHEMA, 1580
DROP SERVER, 1583
DROP TABLE, 1583
and NDB Cluster, 2403
DROP TABLESPACE, 1584
(see also NDB Cluster Disk Data)
DROP TRIGGER, 1584
DROP USER statement, 803, 1724
DROP VIEW, 1584
drop-source option
ndb_move_data, 2653
dropping
accounts, 805
user, 1724
dry-scp option

ndb_error_reporter, 2646
dryrun option
mysqlhotcopy, 445
DSN, 3846
DTrace, 746
and memcached, 2174
DUAL, 1623
dump option
myisam_ftdump, 391
ndb_index_stat, 2649
ndb_redo_log_reader, 2657
dump-date option
mysqldump, 357
dump-file option
ndb_blob_tool, 2629
dump-slave option
mysqldump, 359
DUMPFILE, 1630
dumping
databases and tables, 348, 444
Duplicate Weedout
semijoin strategy, 1003
duplicate-key error, 1857
dynamic cursor, 3846
dynamic row format, 1941, 3846
dynamic SQL, 3847
dynamic statement, 3847
dynamic table characteristics, 2133

E

early adopter, 3847
edit command
mysql, 321
ego command
mysql, 321
Eiffel, 3847
Eiffel Wrapper, 3187
ELT(), 1371
embedded, 3847
embedded MySQL server library, 3180
embedded option
mysql_config, 452
embedded-libs option
mysql_config, 452
--enable option prefix, 266
enable-cleartext-plugin option
mysql, 313
mysqladmin, 337
mysqlcheck, 345
mysqldump, 354
mysqlimport, 372
mysqlshow, 378
mysqlslap, 386
ENABLED_LOCAL_INFILE option
CMake, 173, 779
ENABLED_PROFILING option
CMake, 173
ENABLE_DEBUG_SYNC option

CMake, 173
ENABLE_DOWNLOADS option
CMake, 173
ENABLE_DTRACE option
CMake, 173
ENABLE_GCOV option
CMake, 173
ENABLE_GPROF option
CMake, 173
ENCODE(), 1434
ENCRYPT(), 1434
encrypted connections, 828
command options, 272
encryption, 775, 828
encryption functions, 1429
end
thread state, 1112
END, 1687
EndPoint(), 1460
end_markers_in_json system variable, 561
enforce_gtid_consistency system variable, 2306
engine condition pushdown, 973
engine option
mysqlslap, 386
ENGINES
INFORMATION_SCHEMA table, 2983
engine_condition_pushdown
removed features, 20
ENTER SINGLE USER MODE command (NDB Cluster), 2702
entering
queries, 220
enterprise components
MySQL Enterprise Audit, 3191
MySQL Enterprise Backup, 3190
MySQL Enterprise Data Masking and De-Identification, 3192
MySQL Enterprise Encryption, 3191
MySQL Enterprise Firewall, 3192
MySQL Enterprise Monitor, 3189
MySQL Enterprise Security, 3191
MySQL Enterprise Thread Pool, 3192
enterprise extensions
MySQL Enterprise Audit, 883
MySQL Enterprise Encryption, 923
MySQL Enterprise Firewall, 908
MySQL Enterprise Security, 853, 863
MySQL Enterprise Thread Pool, 722
ENUM
size, 1307
ENUM data type, 1277, 1282
Envelope(), 1458
environment variable
AUTHENTICATION_PAM_LOG, 457, 862
CC, 180, 457
CXX, 180, 457
DBI_TRACE, 457, 739
DBI_USER, 457
HOME, 325, 457
LD_LIBRARY_PATH, 217

LD_RUN_PATH, 217, 457
LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN, 457
LIBMYSQL_PLUGINS, 457
LIBMYSQL_PLUGIN_DIR, 457
MYSQL_DEBUG, 256, 457, 743
MYSQL_GROUP_SUFFIX, 457
MYSQL_HISTFILE, 325, 457
MYSQL_HISTIGNORE, 325, 457
MYSQL_HOME, 457
MYSQL_HOST, 278, 457
MYSQL_OPENSSL_UDF_DH_BITS_THRESHOLD, 457, 926
MYSQL_OPENSSL_UDF_DSA_BITS_THRESHOLD, 457, 926
MYSQL_OPENSSL_UDF_RSA_BITS_THRESHOLD, 457, 926
MYSQL_PS1, 457
MYSQL_PWD, 256, 278, 457
MYSQL_TCP_PORT, 256, 457, 735, 736
MYSQL_TEST_LOGIN_FILE, 265, 414, 457
MYSQL_UNIX_PORT, 184, 256, 457, 735, 736
PATH, 108, 114, 187, 257, 457
TMPDIR, 184, 256, 457, 3263
TZ, 457, 685, 3264
UMASK, 457, 3257
UMASK_DIR, 457, 3258
USER, 278, 457
environment variables, 256, 280, 823
list of, 457
epoch, 1261
equal (=), 1327
Equals(), 1469
eq_ref join type
optimizer, 1058
Errcode, 455
errno, 455
Error
thread command, 1109
ERROR Events (NDB Cluster), 2724
error log, 3847
error logs (NDB Cluster), 2615
error messages
can't find file, 3257
Can't reopen table, 3273
displaying, 455
languages, 1228, 1228
The used command is not allowed with this MySQL version, 780
error-insert option
ndb_move_data, 2653
errors
access denied, 3245
and replication, 2364
checking tables for, 955
common, 3244
directory checksum, 153
in subqueries, 1645
known, 3273
list of, 3245
lost connection, 3249
reporting, 35, 35
sources of information, 3243

error_count system variable, 562
ERROR_FOR_DIVISION_BY_ZERO
  deprecated features, 19
ERROR_FOR_DIVISION_BY_ZERO SQL mode, 670
escape (\\), 1124
escape sequences
  option files, 262
  strings, 1123
establishing encrypted connections, 829
estimating
  query performance, 1068
event groups, 1678
event log format (NDB Cluster), 2720
event logs (NDB Cluster), 2717, 2719, 2719
EVENT privilege, 786
event scheduler, 2939
  thread states, 1121
Event Scheduler, 2948
  altering events, 1497
  and MySQL privileges, 2953
  and mysqladmin debug, 2952
  and replication, 2359, 2360
  and SHOW PROCESSLIST, 2949
  concepts, 2948
  creating events, 1522
  dropping events, 1581
  enabling and disabling, 2949
  event metadata, 2951
  obtaining status information, 2952
  SQL statements, 2951
  starting and stopping, 2949
  time representation, 2951
event severity levels (NDB Cluster), 2719
event table
  system table, 693
event types (NDB Cluster), 2718, 2720
EventLogBufferSize, 2521
events, 2939, 2948
  altering, 1497
  creating, 1522
  dropping, 1581
  metadata, 2951
  restrictions, 2968
  status variables, 2955
EVENTS
  INFORMATION_SCHEMA table, 2953, 2984
events option
  mysqldump, 363
events_stages_current table
  performance_schema, 3121
events_stages_history table
  performance_schema, 3122
events_stages_history_long table
  performance_schema, 3123
events_stages_summary_by_account_by_event_name table
  performance_schema, 3140
events_stages_summary_by_host_by_event_name table
  performance_schema, 3140

events_stages_summary_by_thread_by_event_name table
  performance_schema, 3140
events_stages_summary_by_user_by_event_name table
  performance_schema, 3140
events_stages_summary_global_by_event_name table
  performance_schema, 3140
events_statements_current table
  performance_schema, 3127
events_statements_history table
  performance_schema, 3130
events_statements_history_long table
  performance_schema, 3130
events_statements_summary_by_account_by_event_name table
  performance_schema, 3141
events_statements_summary_by_digest table
  performance_schema, 3141
events_statements_summary_by_host_by_event_name table
  performance_schema, 3141
events_statements_summary_by_thread_by_event_name table
  performance_schema, 3141
events_statements_summary_by_user_by_event_name table
  performance_schema, 3141
events_statements_summary_global_by_event_name table
  performance_schema, 3141
events_waits_current table
  performance_schema, 3116
events_waits_history table
  performance_schema, 3118
events_waits_history_long table
  performance_schema, 3119
events_waits_summary_by_account_by_event_name table
  performance_schema, 3138
events_waits_summary_by_host_by_event_name table
  performance_schema, 3138
events_waits_summary_by_instance table
  performance_schema, 3138
events_waits_summary_by_thread_by_event_name table
  performance_schema, 3138
events_waits_summary_by_user_by_event_name table
  performance_schema, 3138
events_waits_summary_global_by_event_name table
  performance_schema, 3138
event_scheduler system variable, 562
eviction, 3847
exact-value literals, 1486
exact-value numeric literals, 1126, 1487
example option
  mysqld_multi, 290
EXAMPLE storage engine, 2123, 2155
examples
  compressed tables, 410
  myisamchk output, 400
  queries, 242
exception interceptor, 3847
exceptions table
  NDB Cluster Replication, 2859
exclude-databases option
  ndb_restore, 2665

exclude-gtids option
  mysqlbinlog, 429
exclude-intermediate-sql-tables option
  ndb_restore, 2665
exclude-missing-columns option
  ndb_move_data, 2653
  ndb_restore, 2665
exclude-missing-tables option
  ndb_restore, 2665
exclude-tables option
  ndb_restore, 2665
exclusive lock, 1872, 3847
Execute
  thread command, 1109
EXECUTE, 1682, 1686
execute option
  mysql, 313
  ndb_mgm, 2627
EXECUTE privilege, 786
ExecuteOnComputer, 2478, 2484, 2551
executing
  thread state, 1112
executing SQL statements from text files, 241, 327
Execution of init_command
  thread state, 1112
execution threads (NDB Cluster), 2532
EXISTS
  with subqueries, 1641
exit command
  mysql, 321
EXIT command (NDB Cluster), 2702
EXIT SINGLE USER MODE command (NDB Cluster), 2702
exit-info option
  mysqld, 528
EXP(), 1342
expired passwords, 808
expire_logs_days system variable, 2300
EXPLAIN, 1053, 1814
EXPLAIN PARTITIONS, 2912, 2913
EXPLAIN used with partitioned tables, 2912
explicit default values, 1302
explicit_defaults_for_timestamp system variable, 563
EXPORT_SET(), 1371
expression aliases, 1480, 1624
expression syntax, 1168
expressions
  extended, 234
extend-check option
  myisamchk, 397, 398
extended option
  mysqlcheck, 344
extended-insert option
  mysqldump, 365
extensions
  to standard SQL, 39
extent, 3848
ExteriorRing(), 1462
external locking, 528, 613, 955, 1095, 1114

external-locking option
  mysqld, 528
external_user system variable, 565
extra-file option
  my_print_defaults, 453
extra-node-info option
  ndb_desc, 2643
extra-partition-info option
  ndb_desc, 2643
EXTRACT(), 1355
extracting
  dates, 231
ExtractValue(), 1419
ExtraSendBufferMemory
  API nodes, 2552
  data nodes, 2546
  management nodes, 2481

F

failover, 3848
  in NDB Cluster replication, 2840
  Java clients, 2383
FALSE, 1126, 1131
  testing for, 1330, 1330
FAQs
  C API, 3231
  Connectors and APIs, 3231
  libmysql, 3231
  NDB Cluster, 3206
  replication, 3232
  Virtualization Support, 3239
Fast Index Creation, 1946, 3848
fast option
  myisamchk, 397
  mysqlcheck, 345
fast shutdown, 3849
features of MySQL, 5
features, deprecated (see deprecated features)
features, new (see new features)
features, removed (see removed features)
FEDERATED storage engine, 2123, 2150
Fetch
  thread command, 1109
FETCH, 1695
field
  changing, 1507
Field List
  thread command, 1109
FIELD(), 1372
fields option
  ndb_config, 2633
fields-enclosed-by option
  mysqldump, 361, 372
  ndb_restore, 2666
fields-escaped-by option
  mysqldump, 361, 372
fields-optionally-enclosed-by option
  mysqldump, 361, 372

ndb_restore, 2667
fields-terminated-by option
mysqldump, 361, 372
ndb_restore, 2667
FILE, 1374
file format, 1933, 3849
Antelope, 1929
Barracuda, 1922
identifying, 1937
modifying, 1938
FILE privilege, 786
file-per-table, 1841, 3849
files
binary log, 701
created by CREATE TABLE, 1560
DDL log, 714
error messages, 1228
general query log, 699
log, 715
metadata log, 714
not found message, 3257
permissions, 3257
repairing, 398
script, 241
size limits, 1035
slow query log, 712
text, 327, 368
tmp, 184
FILES
INFORMATION_SCHEMA table, 3048
filesort optimization, 992
FileSystemPath, 2486
FileSystemPathDataFiles, 2543
FileSystemPathDD, 2542
FileSystemPathUndoFiles, 2543
file_instances table
performance_schema, 3110
file_summary_by_event_name table
performance_schema, 3143
file_summary_by_instance table
performance_schema, 3143
fill factor, 1858, 3849
fill_help_tables.sql, 690
FIND_IN_SET(), 1372
Finished reading one binlog; switching to next binlog
thread state, 1118
firewalls (software)
and NDB Cluster, 2819, 2820
Firewall_access_denied status variable, 923
Firewall_access_granted status variable, 923
Firewall_access_suspicious status variable, 923
Firewall_cached_entries status variable, 923
firewall_users MySQL Enterprise Firewall table, 919
firewall_users table
system table, 694
firewall_whitelist MySQL Enterprise Firewall table, 919
firewall_whitelist table
system table, 694

FirstMatch
  semijoin strategy, 1003
fix-db-names option
  mysqlcheck, 345
fix-table-names option
  mysqlcheck, 345
FIXED data type, 1254
fixed row format, 3849
fixed-point arithmetic, 1486
FLOAT data type, 1255, 1255, 1255
floating-point number, 1255
floating-point values
  and replication, 2356
floats, 1126
FLOOR(), 1342
flow control functions, 1334
FLUSH
  and replication, 2356
flush, 3849
flush list, 3850
FLUSH MASTER
  removed features, 20
flush option
  mysqld, 528
FLUSH SLAVE
  removed features, 20
FLUSH statement, 1806
flush system variable, 565
flush tables, 334
flush-logs option
  mysqldump, 365
flush-privileges option
  mysqldump, 366
flushing, 1899
flushlog option
  mysqlhotcopy, 445
Flush_commands status variable, 654
flush_time system variable, 565
FOR UPDATE, 1627
FORCE
  plugin activation option, 720
FORCE INDEX, 1072, 3272
FORCE KEY, 1072
force option
  myisamchk, 397, 398
  myisampack, 409
  mysql, 313
  mysqladmin, 337
  mysqlcheck, 345
  mysqldump, 358
  mysqlimport, 372
  mysql_convert_table_format, 447
  mysql_install_db, 296
  mysql_upgrade, 304
force-if-open option
  mysqlbinlog, 429
force-read option
  mysqlbinlog, 429

FORCE_PLUS_PERMANENT
  plugin activation option, 720
foreign key, 3850
  constraint, 46, 46
  deleting, 1510, 1569
FOREIGN KEY constraint, 3850
foreign key constraints, 1565
FOREIGN KEY constraints
  and online DDL, 1960
foreign keys, 44, 244, 1509
foreign_key_checks system variable, 565
FORMAT(), 1372
forums, 35
FOUND_ROWS(), 1443
fractional seconds
  and replication, 2357
fractional seconds precision, 1252, 1261
fragment replicas (NDB Cluster), 2384
FragmentLogFileSize, 2498
FreeBSD troubleshooting, 180
freeing items
  thread state, 1112
.frm file, 3848
FROM, 1624
FROM_BASE64(), 1372
FROM_DAYS(), 1355
FROM_UNIXTIME(), 1355
fs option
  ndb_error_reporter, 2646
FTS, 3850
ft_boolean_syntax system variable, 566
ft_max_word_len myisamchk variable, 395
ft_max_word_len system variable, 567
ft_min_word_len myisamchk variable, 395
ft_min_word_len system variable, 567
ft_query_expansion_limit system variable, 568
ft_stopword_file myisamchk variable, 395
ft_stopword_file system variable, 568
full backup, 3850
full disk, 3262
full table scan, 3850
full table scans
  avoiding, 1000
full-text queries
  optimization, 1020
full-text search, 1393, 3850
FULLTEXT, 1393
fulltext
  stopword list, 1406
FULLTEXT index, 3851
  InnoDB, 1858
  monitoring, 1863
FULLTEXT initialization
  thread state, 1112
fulltext join type
  optimizer, 1059
func table
  system table, 693, 729

function
creating, 1749
deleting, 1750
function names
parsing, 1140
resolving ambiguity, 1140
functions, 1310
aggregate, 1471
and replication, 2357
arithmetic, 1428
bit, 1428
cast, 1412
date and time, 1348
encryption, 1429
flow control, 1334
for SELECT and WHERE clauses, 1310
GROUP BY, 1471
grouping, 1325
GTIDs, 1470
information, 1439
locking, 1438
mathematical, 1340
miscellaneous, 1480
stored, 2941
string, 1368
string comparison, 1383
user-defined, 1749, 1750
fuzzy checkpointing, 3851

G

GA, 3851
MySQL releases, 60
GAC, 3851
gap, 3851
gap event, 2828
gap lock, 1872, 3851
InnoDB, 1886, 2000
gb2312, gbk, 3219
gci option
ndb_select_all, 2682
gci64 option
ndb_select_all, 2682
GCP Stop errors (NDB Cluster), 2545
gdb
using, 738
gdb option
mysqld, 529
general information, 1
General Public License, 5
general query log, 699, 3851
general tablespace, 3852
general_log system variable, 568
general_log table
system table, 693
general_log_file system variable, 568
geographic feature, 1288
GeomCollFromText(), 1453
GeomCollFromWKB(), 1455

geometry, 1288
GEOMETRY data type, 1289
geometry values
  internal storage format, 1297
  WKB format, 1296
  WKT format, 1295
GEOMETRYCOLLECTION data type, 1289
GeometryCollection(), 1456
GeometryCollectionFromText(), 1453
GeometryCollectionFromWKB(), 1455
GeometryFromText(), 1453
GeometryFromWKB(), 1455
GeometryN(), 1464
GeometryType(), 1458
GeomFromText(), 1453
GeomFromWKB(), 1455
geospatial feature, 1288
German dictionary collation, 1178, 1221, 1221
German phone book collation, 1178, 1221, 1221
GET DIAGNOSTICS, 1701
getting MySQL, 61
GET_FORMAT(), 1356
GET_LOCK(), 1438
GIS, 1287
GIS data types
  storage requirements, 1307
Git tree, 162
Glassfish, 3852
GLength(), 1460
GLOBAL
  SET statement, 1752
global privileges, 1725, 1736
global transaction, 3852
GLOBAL_STATUS
  INFORMATION_SCHEMA table, 2988
GLOBAL_VARIABLES
  INFORMATION_SCHEMA table, 2988
go command
  mysql, 321
Google Test, 173
got handler lock
  thread state, 1116
got old table
  thread state, 1116
GRANT OPTION privilege, 787
GRANT statement, 803, 1725
grant table distribution (NDB Cluster), 2759
grant tables
  columns_priv table, 693, 791
  db table, 189, 693, 791
  host table, 693
  procs_priv table, 693, 791
  proxies_priv, 816
  proxies_priv table, 189, 693, 791
  re-creating, 184
  sorting, 800, 801
  structure, 790
  tables_priv table, 693, 791

user table, 189, 693, 791
granting
privileges, 1725
grants
display, 1775
greater than (>), 1328
greater than or equal (>=), 1328
greatest timestamp wins (conflict resolution), 2855
greatest timestamp, delete wins (conflict resolution), 2855
GREATEST(), 1329
Group (NDB Cluster), 2604
GROUP BY
aliases in, 1480
extensions to standard SQL, 1479, 1625
implicit sorting, 992
maximum sort length, 1625
WITH ROLLUP, 1476
GROUP BY functions, 1471
GROUP BY optimizing, 994
GROUP BY sorting
deprecated features, 19
group commit, 1871, 3852
grouping
expressions, 1325
GROUP_CONCAT(), 1474
group_concat_max_len system variable, 569
GTID functions, 1470
gtid_executed system variable, 2307
gtid_mode system variable, 2308
gtid_next system variable, 2308
gtid_owned system variable, 2309
gtid_purged system variable, 2309
GTID_SUBSET(), 1470
GTID_SUBTRACT(), 1470
GUID, 3852

H

HANDLER, 1593
Handlers, 1697
Handler_commit status variable, 654
Handler_delete status variable, 654
Handler_discover status variable, 2584
Handler_external_lock status variable, 654
Handler_mrr_init status variable, 654
Handler_prepare status variable, 654
Handler_read_first status variable, 654
Handler_read_key status variable, 654
Handler_read_last status variable, 654
Handler_read_next status variable, 654
Handler_read_prev status variable, 654
Handler_read_rnd status variable, 655
Handler_read_rnd_next status variable, 655
Handler_rollback status variable, 655
Handler_savepoint status variable, 655
Handler_savepoint_rollback status variable, 655
Handler_update status variable, 655
Handler_write status variable, 655
hash index, 3852

hash indexes, 1023
hash partitioning, 2887
hash partitions
  managing, 2905
  splitting and merging, 2905
have_compress system variable, 569
have_crypt system variable, 569
have_csv
  removed features, 20
have_dynamic_loading system variable, 569
have_geometry system variable, 569
have_innodb
  removed features, 20
have_ndbcluster
  removed features, 20
have_openssl system variable, 569
have_partitioning
  removed features, 20
have_profiling
  deprecated features, 19
have_profiling system variable, 569
have_query_cache system variable, 569
have_rtree_keys system variable, 570
have_ssl system variable, 570
have_symlink system variable, 570
HAVING, 1625
HDD, 3852
header option
  ndb_select_all, 2681
header-file option
  comp_err, 293
HEAP storage engine, 2123, 2136
heartbeat, 3852
HeartbeatIntervalDbApi, 2510
HeartbeatIntervalDbDb, 2509
HeartbeatIntervalMgmdMgmd
  management nodes, 2482
HeartbeatOrder, 2510
HeartbeatThreadPriority, 2481, 2553
help command
  mysql, 320
HELP command (NDB Cluster), 2700
help option
  comp_err, 293
  myisamchk, 394
  myisampack, 409
  myisam_ftdump, 390
  mysql, 310
  mysqlaccess, 421
  mysqladmin, 336
  mysqlbinlog, 426
  mysqlcheck, 343
  mysqld, 524
  mysqldump, 358
  mysqldumpslow, 443
  mysqld_multi, 290
  mysqld_safe, 283
  mysqlhotcopy, 445

mysqlimport, 370
MySQLInstallerConsole, 101
mysqlshow, 377
mysqlslap, 383
mysql_config_editor, 417
mysql_convert_table_format, 447
mysql_find_rows, 448
mysql_install_db, 295
mysql_plugin, 298
mysql_setpermission, 449
mysql_upgrade, 303
mysql_waitpid, 450
my_print_defaults, 453
ndb_setup.py, 2686
perror, 455
resolveip, 456
resolve_stack_dump, 454
HELP option
myisamchk, 394
help option (NDB Cluster programs), 2697
HELP statement, 1816
help tables
system tables, 694
help_category table
system table, 694
help_keyword table
system table, 694
help_relation table
system table, 694
help_topic table
system table, 694
hex option
ndb_restore, 2667
HEX(), 1342, 1373
hex-blob option
mysqldump, 361
hexadecimal literal introducer, 1129
hexadecimal literals, 1128
hexdump option
mysqlbinlog, 429
high-water mark, 3853
HIGH_NOT_PRECEDENCE SQL mode, 670
HIGH_PRIORITY
INSERT modifier, 1598
SELECT modifier, 1627
hints, 41
index, 1072, 1624
histignore option
mysql, 313
history list, 3853
history of MySQL, 8
hole punching, 3853
HOME environment variable, 325, 457
host, 3853
host cache, 678
new features, 18
host name
default, 276

host name caching, 678
host name resolution, 678
host names, 276
  in account names, 796
  in default accounts, 189
host option, 271
  mysql, 314
  mysqlaccess, 422
  mysqladmin, 337
  mysqlbinlog, 429
  mysqlcheck, 345
  mysqldump, 354
  mysqlhotcopy, 446
  mysqlimport, 372
  mysqlshow, 378
  mysqlslap, 386
  mysql_convert_table_format, 447
  mysql_setpermission, 449
  mysql_upgrade, 304
  ndb_config, 2634
host table
  system table, 693
HostName, 2479, 2484, 2551
HostName (NDB Cluster), 2818
hostname system variable, 571
HostName1, 2600, 2604
HostName2, 2600, 2605
hosts table
  performance_schema, 3133
host_cache table
  performance_schema, 678, 3149
hot, 3853
hot backup, 3853
HOUR(), 1356
howto option
  mysqlaccess, 422
html option
  mysql, 314

I

i-am-a-dummy option
  mysql, 317
ib-file set, 1934, 3854
ibbackup_logfile, 3854
.ibd file, 3853
ibdata file, 1560, 3854
ibtmp file, 3854
.ibz file, 3854
ib_logfile, 3855
icc
  MySQL builds, 74
Id, 2477, 2483, 2550
identifiers, 1132
  case sensitivity, 1136
  quoting, 1132
identity system variable, 571
IF, 1691
IF(), 1335

IFNULL(), 1336
IGNORE
  DELETE modifier, 1590, 1606
  INSERT modifier, 1598
  UPDATE modifier, 1650
  with partitioned tables, 1598
IGNORE INDEX, 1072
IGNORE KEY, 1072
ignore option
  mysqlimport, 372
ignore-db-dir option
  mysqld, 529
ignore-lines option
  mysqlimport, 372
ignore-spaces option
  mysql, 314
ignore-table option
  mysqldump, 363
IGNORE_AIO_CHECK option
  CMake, 174
ignore_builtin_innodb system variable, 1969
ignore_db_dirs system variable, 571
IGNORE_SPACE SQL mode, 670
ilist, 3855
implicit default values, 1302
implicit GROUP BY sorting, 992
implicit row lock, 3855
IMPORT TABLESPACE, 1511, 1846
importing
  data, 327, 368
IN, 1639
IN(), 1329
in-file option
  comp_err, 293
in-memory database, 3855
include option
  mysql_config, 452
include-databases option
  ndb_restore, 2667
include-gtids option
  mysqlbinlog, 429
include-master-host-port option
  mysqldump, 359
include-tables option
  ndb_restore, 2667
increasing with replication
  speed, 2221
incremental backup, 3855
incremental recovery, 951
  using NDB Cluster replication, 2847
index, 3855
  deleting, 1509, 1581
  rebuilding, 208
index cache, 3856
index condition pushdown, 3856
INDEX DIRECTORY
  and replication, 2356
index dives

range optimization, 970
index dives (for statistics estimation), 1915
index extensions, 1024
index hint, 3856
index hints, 1072, 1624
index join type
optimizer, 1060
index prefix, 3856
index prefixes
partitioning, 2928
INDEX privilege, 787
index statistics
NDB, 2547
index-record lock
InnoDB, 1886, 2000
indexed temporary table
semijoin strategy, 1003
indexes, 1526
and BLOB columns, 1019, 1543
and IS NULL, 1024
and LIKE, 1023
and ndb_restore, 2672
and NULL values, 1543
and TEXT columns, 1019, 1543
assigning to key cache, 1804
BLOB columns, 1527
block size, 573
column prefixes, 1019
columns, 1019
leftmost prefix of, 1018, 1021
multi-column, 1020
multiple-part, 1526
names, 1132
TEXT columns, 1527
TIMESTAMP lookups, 1027
use of, 1017
IndexMemory, 2489
IndexStatAutoCreate
data nodes, 2547
IndexStatAutoUpdate
data nodes, 2548
IndexStatSaveScale
data nodes, 2548
IndexStatSaveSize
data nodes, 2548
IndexStatTriggerPct
data nodes, 2549
IndexStatTriggerScale
data nodes, 2549
IndexStatUpdateDelay
data nodes, 2549
index_merge join type
optimizer, 1059
index_subquery join type
optimizer, 1059
INET6_ATON(), 1481
INET6_NTOA(), 1482
INET_ATON(), 1481

INET_NTOA(), 1481
infimum record, 3856
INFO Events (NDB Cluster), 2724
information functions, 1439
information option
  myisamchk, 397
INFORMATION SCHEMA
  InnoDB tables, 2028
INFORMATION_SCHEMA, 2976, 3856
  and security issues, 2823
  collation and searching, 1206
  connection-control tables, 3059
  InnoDB tables, 3013
  INNODB_CMP table, 2029
  INNODB_CMPMEM table, 2029
  INNODB_CMPMEM_RESET table, 2029
  INNODB_CMP_RESET table, 2029
  INNODB_LOCKS table, 2030
  INNODB_LOCK_WAITS table, 2030
  INNODB_TRX table, 2030
  MySQL Enterprise Firewall tables, 3059
  NDB Cluster tables, 3048
  Thread pool tables, 3054
INFORMATION_SCHEMA queries
  optimization, 1010
INFORMATION_SCHEMA.ENGINES table
  and NDB Cluster, 2812
INFORMATION_SCHEMA.GLOBAL_STATUS table
  and NDB Cluster, 2817
INFORMATION_SCHEMA.GLOBAL_VARIABLES table
  and NDB Cluster, 2814
INFO_BIN file
  binary distribution configuration options, 37, 156
init
  thread state, 1112
Init DB
  thread command, 1109
init-command option
  mysql, 314
InitFragmentLogFiles, 2499
initial option
  ndbd, 2612
  ndbmtd, 2612
  ndb_mgmd, 2622
initial-start option
  ndbd, 2613
  ndbmtd, 2613
Initialized
  thread state, 1121
InitialLogFileGroup, 2543
InitialNoOfOpenFiles, 2499
InitialTablespace, 2544
init_connect system variable, 571
init_file system variable, 571
init_slave system variable, 2270
INNER JOIN, 1630
innochecksum, 254, 389
InnoDB, 1820, 3856

.frm files, 1839
adaptive hash index, 1837
and application feature requirements, 2400
application performance, 1850
applications supported, 2399
architecture, 1828
asynchronous I/O, 1904
auto-inc lock, 1872
auto-increment columns, 1851
autocommit mode, 1879, 1879
availability, 2398
backups, 2074
change buffer, 1834
checkpoints, 1945
clustered index, 1857
COMPACT row format, 1940
compared to NDB Cluster, 2397, 2398, 2399, 2400
configuration parameters, 1961
consistent reads, 1880
corruption, 2075
crash recovery, 2076, 2076
creating tables, 1838
data files, 1863
deadlock detection, 1888
deadlock example, 1888
deadlocks, 1848, 1887, 1888
disk failure, 2075
disk I/O, 1943
disk I/O optimization, 1044
DYNAMIC row format, 1941, 1942
exclusive lock, 1872
file space management, 1944
file-per-table tablespace, 1866
files, 1851
FULLTEXT indexes, 1858
gap lock, 1872, 1886, 2000
in-memory structures, 1829
index-record lock, 1886, 2000
insert-intention lock, 1872
intention lock, 1872
limitations, 2120
limits, 2119
Linux, 1904
lock modes, 1872
locking, 1872, 1872, 1883
locking reads, 1882
log files, 1870
memory usage, 1847
migrating tables, 1845
Monitors, 1944, 2075, 2112
multi-versioning, 1827
new features, 10
next-key lock, 1872, 1886, 2000
NFS, 1891
on-disk structures, 1838
online DDL, 1946
page size, 1858
physical index structure, 1858

point-in-time recovery, 2075
primary keys, 1839, 1850
raw partitions, 1865
record-level locks, 1886, 2000
recovery, 2075
redo log, 1870
REDUNDANT row format, 1939
replication, 2077
restrictions, 2120
row format, 1839, 1942
secondary index, 1857
semi-consistent read, 2000
shared lock, 1872
Solaris issues, 153
storage, 1849
storage layout, 1848
system variables, 1961
table properties, 1839
tables, 1838
converting from other storage engines, 1846
transaction model, 1872, 1876
transactions, 1847
transferring data, 1849
troubleshooting, 2112
CREATE TABLE failure, 2115
data dictionary problems, 2115
deadlocks, 1887, 1888
defragmenting tables, 1945
I/O problems, 2113
online DDL, 1960
open file error, 2115
orphan intermediate tables, 2115
orphan temporary tables, 2116
performance problems, 1039
recovery problems, 2113
restoring orphan ibd files, 2117
SQL errors, 2118
tablespace does not exist, 2117
InnoDB buffer pool, 1075, 1831, 1896, 1897, 1898, 1900
InnoDB Monitors, 2060
enabling, 2061
output, 2064
innodb option
mysqld, 1966
InnoDB parameters, new
innodb_file_format_check, 1935
innodb_large_prefix, 1998
InnoDB storage engine, 1820, 2123
InnoDB tables
storage requirements, 1304
innodb-status-file option
mysqld, 1966
innodb_adaptive_flushing system variable, 1969
innodb_adaptive_flushing_lwm system variable, 1969
innodb_adaptive_hash_index
and innodb_thread_concurrency, 1903
innodb_adaptive_hash_index system variable, 1970
innodb_adaptive_max_sleep_delay system variable, 1970

innodb_additional_mem_pool_size
  deprecated features, 19
innodb_additional_mem_pool_size system variable, 1971
  and innodb_use_sys_malloc, 1902
innodb_api_bk_commit_interval system variable, 1971
innodb_api_disable_rowlock system variable, 1971
innodb_api_enable_binlog system variable, 1972
innodb_api_enable_mdl system variable, 1972
innodb_api_trx_level system variable, 1972
innodb_autoextend_increment system variable, 1973
innodb_autoinc_lock_mode, 3857
innodb_autoinc_lock_mode system variable, 1973
Innodb_available_undo_logs status variable, 655
INNODB_BUFFER_PAGE
  INFORMATION_SCHEMA table, 3013
INNODB_BUFFER_PAGE_LRU
  INFORMATION_SCHEMA table, 3016
Innodb_buffer_pool_bytes_data status variable, 655
Innodb_buffer_pool_bytes_dirty status variable, 656
innodb_buffer_pool_dump_at_shutdown system variable, 1973
innodb_buffer_pool_dump_now system variable, 1974
Innodb_buffer_pool_dump_status status variable, 655
innodb_buffer_pool_filename system variable, 1974
innodb_buffer_pool_instances system variable, 1974
innodb_buffer_pool_load_abort system variable, 1975
innodb_buffer_pool_load_at_startup system variable, 1976
innodb_buffer_pool_load_now system variable, 1976
Innodb_buffer_pool_load_status status variable, 655
Innodb_buffer_pool_pages_data status variable, 656
Innodb_buffer_pool_pages_dirty status variable, 656
Innodb_buffer_pool_pages_flushed status variable, 656
Innodb_buffer_pool_pages_free status variable, 656
Innodb_buffer_pool_pages_latched status variable, 656
Innodb_buffer_pool_pages_misc status variable, 656
Innodb_buffer_pool_pages_total status variable, 656
Innodb_buffer_pool_reads status variable, 657
Innodb_buffer_pool_read_ahead status variable, 656
Innodb_buffer_pool_read_ahead_evicted status variable, 656
Innodb_buffer_pool_read_ahead_rnd status variable, 656
Innodb_buffer_pool_read_requests status variable, 656
innodb_buffer_pool_size system variable, 1976
INNODB_BUFFER_POOL_STATS
  INFORMATION_SCHEMA table, 3019
Innodb_buffer_pool_wait_free status variable, 657
Innodb_buffer_pool_write_requests status variable, 657
innodb_change_buffering, 1835
innodb_change_buffering system variable, 1977
innodb_change_buffering_debug, 1978
innodb_change_buffer_max_size system variable, 1977
innodb_checksums system variable, 1980
innodb_checksum_algorithm system variable, 1978
INNODB_CMP
  INFORMATION_SCHEMA table, 3022
INNODB_CMPMEM
  INFORMATION_SCHEMA table, 3023
INNODB_CMPMEM_RESET
  INFORMATION_SCHEMA table, 3023
INNODB_CMP_PER_INDEX

INFORMATION_SCHEMA table, 3025
innodb_cmp_per_index_enabled system variable, 1981
INNODB_CMP_PER_INDEX_RESET
INFORMATION_SCHEMA table, 3025
INNODB_CMP_RESET
INFORMATION_SCHEMA table, 3022
innodb_commit_concurrency system variable, 1981
innodb_compression_failure_threshold_pct
new features, 12
innodb_compression_failure_threshold_pct system variable, 1981
innodb_compression_level
new features, 12
innodb_compression_level system variable, 1982
innodb_compression_pad_pct_max
new features, 12
innodb_compression_pad_pct_max system variable, 1982
innodb_concurrency_tickets, 1903
innodb_concurrency_tickets system variable, 1982
innodb_data_file_path system variable, 1983
Innodb_data_fsyncs status variable, 657
innodb_data_home_dir system variable, 1984
Innodb_data_pending_fsyncs status variable, 657
Innodb_data_pending_reads status variable, 657
Innodb_data_pending_writes status variable, 657
Innodb_data_read status variable, 657
Innodb_data_reads status variable, 657
Innodb_data_writes status variable, 657
Innodb_data_written status variable, 657
Innodb_dblwr_pages_written status variable, 657
Innodb_dblwr_writes status variable, 657
innodb_disable_sort_file_cache system variable, 1984
innodb_doublewrite system variable, 1984
innodb_fast_shutdown system variable, 1985
innodb_file_format, 1933, 3857
Antelope, 1929
Barracuda, 1922
identifying, 1937
innodb_file_format system variable, 1985
innodb_file_format_check, 1935
innodb_file_format_check system variable, 1986
innodb_file_format_max system variable, 1986
innodb_file_per_table, 1922, 3857
innodb_file_per_table system variable, 1987
innodb_fil_make_page_dirty_debug, 1985
innodb_flushing_avg_loops system variable, 1991
innodb_flush_log_at_timeout system variable, 1987
innodb_flush_log_at_trx_commit system variable, 1988
innodb_flush_method system variable, 1989
innodb_flush_neighbors system variable, 1990
innodb_force_load_corrupted system variable, 1991
innodb_force_recovery system variable, 1991
DROP TABLE, 1584
innodb_ft_aux_table system variable, 1992
INNODB_FT_BEING_DELETED
INFORMATION_SCHEMA table, 3026
innodb_ft_cache_size system variable, 1992
INNODB_FT_CONFIG
INFORMATION_SCHEMA table, 3027

INNODB_FT_DEFAULT_STOPWORD
  INFORMATION_SCHEMA table, 3028
INNODB_FT_DELETED
  INFORMATION_SCHEMA table, 3029
innodb_ft_enable_diag_print system variable, 1993
innodb_ft_enable_stopword system variable, 1993
INNODB_FT_INDEX_CACHE
  INFORMATION_SCHEMA table, 3030
INNODB_FT_INDEX_TABLE
  INFORMATION_SCHEMA table, 3031
innodb_ft_max_token_size system variable, 1994
innodb_ft_min_token_size system variable, 1994
innodb_ft_num_word_optimize system variable, 1994
innodb_ft_result_cache_limit system variable, 1995
innodb_ft_server_stopword_table system variable, 1995
innodb_ft_sort_pll_degree system variable, 1996
innodb_ft_total_cache_size system variable, 1996
innodb_ft_user_stopword_table system variable, 1997
Innodb_have_atomic_builtins status variable, 657
innodb_index_stats table
  system table, 694, 1909
innodb_io_capacity, 1905
innodb_io_capacity system variable, 1997
innodb_io_capacity_max system variable, 1997
innodb_large_prefix system variable, 1998
innodb_limit_optimistic_insert_debug, 1998
INNODB_LOCKS
  INFORMATION_SCHEMA table, 3033
innodb_locks_unsafe_for_binlog
  deprecated features, 19
innodb_locks_unsafe_for_binlog system variable, 1999
INNODB_LOCK_WAITS
  INFORMATION_SCHEMA table, 3034
innodb_lock_wait_timeout, 3857
innodb_lock_wait_timeout system variable, 1999
innodb_log_buffer_size system variable, 2002
innodb_log_checkpoint_now system variable, 2002
innodb_log_compressed_pages
  new features, 12
innodb_log_compressed_pages system variable, 2002
innodb_log_files_in_group system variable, 2003
innodb_log_file_size system variable, 2003
innodb_log_group_home_dir system variable, 2004
Innodb_log_waits status variable, 658
Innodb_log_writes status variable, 658
Innodb_log_write_requests status variable, 658
innodb_lru_scan_depth system variable, 2004
innodb_max_dirty_pages_pct system variable, 2005
innodb_max_dirty_pages_pct_lwm system variable, 2005
innodb_max_purge_lag system variable, 2006
innodb_max_purge_lag_delay system variable, 2006
innodb_memcache database, 2085, 2107
innodb_memcached_config.sql script, 2085
INNODB_METRICS
  INFORMATION_SCHEMA table, 3035
innodb_mirrored_log_groups system variable, 2006
innodb_monitor_disable system variable, 2006
innodb_monitor_enable system variable, 2007

innodb_monitor_reset system variable, 2007
innodb_monitor_reset_all system variable, 2007
innodb_numa_interleave variable, 2007
Innodb_num_open_files status variable, 658
innodb_old_blocks_pct, 1897
innodb_old_blocks_pct system variable, 2008
innodb_old_blocks_time, 1897
innodb_old_blocks_time system variable, 2008
innodb_online_alter_log_max_size system variable, 2009
innodb_open_files system variable, 2009
innodb_optimize_fulltext_only system variable, 2009
Innodb_os_log_fsyncs status variable, 658
Innodb_os_log_pending_fsyncs status variable, 658
Innodb_os_log_pending_writes status variable, 658
Innodb_os_log_written status variable, 658
Innodb_pages_created status variable, 658
Innodb_pages_read status variable, 658
Innodb_pages_written status variable, 658
INNODB_PAGE_ATOMIC_REF_COUNT option
  CMake, 174
Innodb_page_size status variable, 658
innodb_page_size system variable, 2010
innodb_print_all_deadlocks system variable, 2010
  innodb_print_all_deadlocks, 2010
innodb_purge_batch_size system variable, 2011
innodb_purge_threads system variable, 2011
innodb_random_read_ahead system variable, 2012
innodb_read_ahead_threshold, 1898
innodb_read_ahead_threshold system variable, 2012
innodb_read_io_threads, 1904
innodb_read_io_threads system variable, 2013
innodb_read_only system variable, 2013
innodb_replication_delay system variable, 2013
innodb_rollback_on_timeout system variable, 2014
innodb_rollback_segments system variable, 2014
Innodb_rows_deleted status variable, 659
Innodb_rows_inserted status variable, 659
Innodb_rows_read status variable, 659
Innodb_rows_updated status variable, 659
Innodb_row_lock_current_waits status variable, 658
Innodb_row_lock_time status variable, 658
Innodb_row_lock_time_avg status variable, 658
Innodb_row_lock_time_max status variable, 658
Innodb_row_lock_waits status variable, 659
innodb_saved_page_number_debug, 2014
innodb_sort_buffer_size system variable, 2015
innodb_spin_wait_delay, 1906
innodb_spin_wait_delay system variable, 2016
innodb_stats_auto_recalc system variable, 2016
innodb_stats_include_delete_marked system variable, 1911, 2016
innodb_stats_method system variable, 2017
innodb_stats_on_metadata system variable, 2017
innodb_stats_persistent system variable
  innodb_stats_persistent, 2018
innodb_stats_persistent_sample_pages system variable, 2018
innodb_stats_sample_pages system variable, 2019
innodb_stats_transient_sample_pages, 1915
innodb_stats_transient_sample_pages system variable, 2019

innodb_status_output
  new features, 13
innodb_status_output system variable, 2019
innodb_status_output_locks
  new features, 13
innodb_status_output_locks system variable, 2020
innodb_stat_persistent system variable, 2018
innodb_strict_mode, 1037, 3857
innodb_strict_mode system variable, 2020
innodb_support_xa system variable, 2021
innodb_sync_array_size system variable, 2021
innodb_sync_spin_loops system variable, 2022
INNODB_SYS_COLUMNS
  INFORMATION_SCHEMA table, 3037
INNODB_SYS_DATAFILES
  INFORMATION_SCHEMA table, 3038
INNODB_SYS_FIELDS
  INFORMATION_SCHEMA table, 3038
INNODB_SYS_FOREIGN
  INFORMATION_SCHEMA table, 3039
INNODB_SYS_FOREIGN_COLS
  INFORMATION_SCHEMA table, 3040
INNODB_SYS_INDEXES
  INFORMATION_SCHEMA table, 3040
INNODB_SYS_TABLES
  INFORMATION_SCHEMA table, 3042
INNODB_SYS_TABLESPACES
  INFORMATION_SCHEMA table, 3043
INNODB_SYS_TABLESTATS
  INFORMATION_SCHEMA table, 3044
innodb_table_locks system variable, 2022
innodb_table_stats table
  system table, 694, 1909
innodb_thread_concurrency, 1903
innodb_thread_concurrency system variable, 2022
innodb_thread_sleep_delay, 1903
innodb_thread_sleep_delay system variable, 2023
innodb_tmpdir system variable, 2024
Innodb_truncated_status_writes status variable, 659
INNODB_TRX
  INFORMATION_SCHEMA table, 3045
innodb_trx_purge_view_update_only_debug, 2025
innodb_trx_rseg_n_slots_debug, 2025
innodb_undo_directory system variable, 2025
innodb_undo_logs system variable, 2026
innodb_undo_tablespaces system variable, 2026
innodb_use_native_aio, 1904
innodb_use_native_aio system variable, 2027
innodb_use_sys_malloc
  and innodb_thread_concurrency, 1903
  deprecated features, 19
innodb_use_sys_malloc system variable, 1902, 2027
innodb_version system variable, 2028
innodb_write_io_threads, 1904
innodb_write_io_threads system variable, 2028
INOUT parameter
  condition handling, 1719
INSERT, 1015, 1594

insert, 3858
  thread state, 1117
INSERT ... SELECT, 1598
insert buffer, 3858
insert buffering, 3858
  disabling, 1835
INSERT DELAYED, 1601, 1601
insert intention lock, 3858
INSERT privilege, 787
INSERT(), 1373
insert-ignore option
  mysqldump, 365
insert-intention lock, 1872
insertable views
  insertable, 2957
inserting
  speed of, 1015
inserts
  concurrent, 1089, 1091
insert_id system variable, 572
install option
  mysqld, 529
  MySQLInstallerConsole, 101
  ndbd, 2613
  ndbmtd, 2613
  ndb_mgmd, 2622
INSTALL PLUGIN statement, 1750
install-manual option
  mysqld, 530
Installation, 100
installation layouts, 74
installation overview, 156
installing
  binary distribution, 74
  Linux RPM packages, 137
  OS X DMG packages, 116
  overview, 58
  Perl, 215
  Perl on Windows, 216
  Solaris PKG packages, 153
  source distribution, 156
installing NDB Cluster, 2411
  Debian Linux, 2418
  Linux, 2413
  Linux binary release, 2414
  Linux RPM, 2416
  Linux source release, 2418
  Ubuntu Linux, 2418
  Windows, 2419
  Windows binary release, 2419
  Windows source, 2423
installing plugins, 717, 1750
installing UDFs, 728
INSTALL_BINDIR option
  CMake, 170
INSTALL_DOCDIR option
  CMake, 170
INSTALL_DOCREADMEDIR option

CMake, 170
INSTALL_INCLUDEDIR option
CMake, 170
INSTALL_INFODIR option
CMake, 170
INSTALL_LAYOUT option
CMake, 170
INSTALL_LIBDIR option
CMake, 170
INSTALL_MANDIR option
CMake, 170
INSTALL_MYSQLSHAREDIR option
CMake, 170
INSTALL_MYSQLTESTDIR option
CMake, 170
INSTALL_PLUGINDIR option
CMake, 170
INSTALL_SBINDIR option
CMake, 171
INSTALL_SCRIPTDIR option
CMake, 171
INSTALL_SECURE_FILE_PRIVDIR option
CMake, 171
INSTALL_SECURE_FILE_PRIV_EMBEDDEDDIR option
CMake, 171
INSTALL_SHAREDIR option
CMake, 171
INSTALL_SQLBENCHDIR option
CMake, 171
INSTALL_SUPPORTFILESDIR option
CMake, 171
instance, 3858
INSTR(), 1373
instrumentation, 3858
INT data type, 1254
integer arithmetic, 1486
INTEGER data type, 1254
integers, 1126
intention lock, 1872, 3858
interactive option
ndb_mgmd, 2623
interactive_timeout system variable, 572
interceptor, 3859
InteriorRingN(), 1462
internal locking, 1087
internal memory allocator
disabling, 1902
internal storage format
geometry values, 1297
Intersects(), 1469
INTERVAL
temporal interval syntax, 1170
interval syntax, 1170
INTERVAL(), 1330
INTO
SELECT, 1628
introducer
binary character set, 1228

bit-value literal, 1130
character set, 1188
hexadecimal literal, 1129
string literal, 1124, 1186
invalid data
constraint, 47
invalidating query cache entries
thread state, 1117
inverted index, 3859
invisible index, 1526
INVOKER privileges, 1775, 2959
IOPS, 3859
IP addresses
in account names, 796
in default accounts, 189
IPv6 addresses
in account names, 796
in default accounts, 189
IPv6 connections, 189, 613
IS boolean_value, 1330
IS NOT boolean_value, 1330
IS NOT DISTINCT FROM operator, 1327
IS NOT NULL, 1331
IS NULL, 989, 1330
and indexes, 1024
IsClosed(), 1460
IsEmpty(), 1458
.isl file, 3854
ISNULL(), 1331
ISOLATION LEVEL, 1663
isolation level, 1876, 3859
IsSimple(), 1458
IS_FREE_LOCK(), 1439
IS_IPV4(), 1482
IS_IPV4_COMPAT(), 1483
IS_IPV4_MAPPED(), 1483
IS_IPV6(), 1483
IS_USED_LOCK(), 1439
ITERATE, 1692
iterations option
mysqlslap, 386

J

J2EE, 3859
Japanese character sets
conversion, 3219
Japanese, Korean, Chinese character sets
frequently asked questions, 3219
Java, 3179, 3859
JBoss, 3859
JDBC, 3177, 3859
jdbc:mysql:loadbalance://, 2383
JNDI, 3860
join, 3860
nested-loop algorithm, 979
JOIN, 1630
join algorithm
Block Nested-Loop, 976

Nested-Loop, 976
join option
myisampack, 409
join type
ALL, 1060
const, 1058
eq_ref, 1058
fulltext, 1059
index, 1060
index_merge, 1059
index_subquery, 1059
range, 1059
ref, 1058
ref_or_null, 1059
system, 1058
unique_subquery, 1059
joins
USING versus ON, 1634
join_buffer_size system variable, 572

K

keep-my-cnf option
mysql_install_db, 296
keepold option
mysqlhotcopy, 446
keep_files_on_create system variable, 573
Key cache
MyISAM, 1075
key cache
assigning indexes to, 1804
key partitioning, 2890
key partitions
managing, 2905
splitting and merging, 2905
key space
MyISAM, 2131
key-file option
ndb_setup.py, 2686
key-value store, 1024
keys, 1019
foreign, 44, 245
multi-column, 1020
searching on two, 246
keys option
mysqlshow, 378
keys-used option
myisamchk, 398
keystore, 3860
keywords, 1143
Key_blocks_not_flushed status variable, 659
Key_blocks_unused status variable, 659
Key_blocks_used status variable, 659
KEY_BLOCK_SIZE, 1922, 1926, 3860
key_buffer_size myisamchk variable, 395
key_buffer_size system variable, 573
key_cache_age_threshold system variable, 574
key_cache_block_size system variable, 575
key_cache_division_limit system variable, 575

KEY_COLUMN_USAGE
  INFORMATION_SCHEMA table, 2988
Key_reads status variable, 659
Key_read_requests status variable, 659
Key_writes status variable, 659
Key_write_requests status variable, 659
Kill
  thread command, 1109
KILL statement, 1811
Killed
  thread state, 1112
Killing slave
  thread state, 1120
known errors, 3273
Korean, 3219

L

labels
  stored program block, 1687
language option
  mysqld, 530
language support
  error messages, 1228
lap option
  ndb_redo_log_reader, 2658
large page support, 1103
large tables
  NDB Cluster, 1551
large-pages option
  mysqld, 530
large_files_support system variable, 575
large_pages system variable, 575
large_page_size system variable, 576
LAST_DAY(), 1356
last_insert_id system variable, 576
LAST_INSERT_ID(), 1444, 1597
  and replication, 2349
  and stored routines, 2943
  and triggers, 2943
Last_query_cost status variable, 659
Last_query_partial_plans status variable, 660
latch, 3860
LateAlloc, 2504
layout of installation, 74
lc-messages option
  mysqld, 531
lc-messages-dir option
  mysqld, 531
LCASE(), 1373
LcpScanProgressTimeout, 2500
lc_messages system variable, 576
lc_messages_dir system variable, 576
lc_time_names system variable, 577
ldata option
  mysql_install_db, 296
LDML syntax, 1240
LD_LIBRARY_PATH environment variable, 217
LD_RUN_PATH environment variable, 217, 457

LEAST(), 1331
LEAVE, 1692
ledir option
  mysqld_safe, 283
LEFT JOIN, 982, 1630
LEFT OUTER JOIN, 1630
LEFT(), 1373
leftmost prefix of indexes, 1018, 1021
legal names, 1132
length option
  myisam_ftdump, 391
LENGTH(), 1373
less than (<), 1328
less than or equal (<=), 1327
libaio, 75, 141, 174
libmysql, 3860
  FAQ, 3231
libmysqlclient, 3860
libmysqlclient library, 3177
libmysqld, 3180, 3860
  options, 3182
libmysqld library, 3177
libmysqld-libs option
  mysql_config, 452
LIBMYSQL_ENABLE_CLEARTEXT_PLUGIN environment variable, 457
LIBMYSQL_PLUGINS environment variable, 457
LIBMYSQL_PLUGIN_DIR environment variable, 457
library
  libmysqlclient, 3177
  libmysqld, 3177
libs option
  mysql_config, 452
libs_r option
  mysql_config, 452
license system variable, 577
lifecycle interceptor, 3860
LIKE, 1383
  and indexes, 1023
  and wildcards, 1023
LIMIT, 1443, 1626
  and replication, 2361
  optimizations, 997
limitations
  InnoDB, 2120
  replication, 2348
limitations of NDB Cluster, 2400
limits
  file-size, 1035
  InnoDB, 2119
  maximum columns per table, 1037
  maximum number of databases, 1035, 1522
  maximum number of tables, 1035, 1541
  maximum row size, 1037
  maximum tables per join, 1632
  maximum tables per view, 2972
  table size, 1035
line-numbers option
  mysql, 314

linear hash partitioning, 2889
linear key partitioning, 2891
linefeed (\n), 1124, 1609
LineFromText(), 1453
LineFromWKB(), 1455
lines-terminated-by option
  mysqldump, 361, 372
  ndb_restore, 2668
LINESTRING data type, 1289
LineString(), 1456
LineStringFromText(), 1453
LineStringFromWKB(), 1455
links
  symbolic, 1098
list, 3861
list option
  MySQLInstallerConsole, 103
list partitioning, 2877, 2879
list partitions
  adding and dropping, 2900
  managing, 2900
literals, 1123
  bit value, 1130
  boolean, 1131
  date, 1126
  hexadecimal, 1128
  numeric, 1126
  string, 1123
  time, 1126
LN(), 1343
load balancing, 3861
LOAD DATA, 1603, 3268
  and replication, 2361
  LOCAL loading, 778
load emulation, 380
LOAD INDEX INTO CACHE
  and partitioning, 2927
LOAD XML, 1613
loading
  tables, 226
LOAD_FILE(), 1374
local option
  mysqlimport, 372, 779
local-infile option
  mysql, 314, 779
local-load option
  mysqlbinlog, 429
local-service option
  mysqld, 531
localhost, 3861
  special treatment of, 277
LOCALTIME, 1357
LOCALTIMESTAMP, 1357
local_infile system variable, 577, 779
LOCATE(), 1374
lock, 3861
lock escalation, 3861
LOCK IN SHARE MODE, 1627

lock mode, 3861
Lock Monitor, 2060, 2064
lock option
  ndb_select_all, 2681
LOCK TABLES, 1657
LOCK TABLES privilege, 787
lock-all-tables option
  mysqldump, 366
lock-tables option
  mysqldump, 366
  mysqlimport, 372
locked_in_memory system variable, 578
LockExecuteThreadToCPU, 2529
locking, 1096, 1872, 3861
  external, 528, 613, 955, 1095, 1114
  information schema, 2030
  InnoDB, 1872
  internal, 1087
  metadata, 1092
  row-level, 1087
  table-level, 1087
locking functions, 1438
locking methods, 1087
locking read, 3861
LockMaintThreadsToCPU, 2529
LockPagesInMainMemory, 2504
lock_wait_timeout system variable, 577
log, 3862
  removed features, 20
log buffer, 3862
log file, 3862
log files
  maintaining, 715
log files (NDB Cluster), 2615
  ndbmtd, 2618
log group, 3862
log option
  mysqld_multi, 290
LOG(), 1343
log-bin option
  mysqld, 2288
log-bin-index option
  mysqld, 2289
log-error option
  mysqld, 531
  mysqldump, 358
  mysqld_safe, 283
log-isam option
  mysqld, 532
log-name option
  ndb_mgmd, 2623
log-raw option
  mysqld, 532
log-short-format option
  mysqld, 532
log-slow-admin-statements option
  mysqld, 532
log-slow-slave-statements option

mysqld, 2261
log-tc option
mysqld, 533
log-tc-size option
mysqld, 533
log-warnings option
mysqld, 533, 2262
LOG10(), 1343
LOG2(), 1343
logbuffers
ndbinfo table, 2788
LogDestination, 2479
logging
passwords, 769
logging commands (NDB Cluster), 2719
logging slow query
thread state, 1112
logical, 3862
logical backup, 3862
logical operators, 1332
login
thread state, 1112
login-path option, 265
mysql, 314
mysqladmin, 337
mysqlbinlog, 430
mysqlcheck, 345
mysqldump, 354
mysqlimport, 372
mysqlshow, 378
mysqlslap, 386
mysql_upgrade, 304
my_print_defaults, 454
NDB client programs, 2697
LogLevelCheckpoint, 2523
LogLevelCongestion, 2523
LogLevelConnection, 2523
LogLevelError, 2523
LogLevelInfo, 2524
LogLevelNodeRestart, 2523
LogLevelShutdown, 2522
LogLevelStartup, 2522
LogLevelStatistic, 2522
logs
flushing, 695
server, 694
logspaces
ndbinfo table, 2788
log_bin system variable, 2300
log_bin_basename system variable, 2300
log_bin_index system variable, 2301
log_bin_trust_function_creators system variable, 2301
log_bin_use_v1_row_events system variable, 2301
log_error system variable, 578
log_output system variable, 578
log_queries_not_using_indexes system variable, 579
log_slave_updates system variable, 2302
log_slow_admin_statements system variable

mysqld, 579
log_slow_queries
removed features, 20
log_slow_slave_statements system variable, 2271
log_throttle_queries_not_using_indexes system variable, 579
log_warnings system variable, 580
Long Data
thread command, 1109
LONG data type, 1280
LONGBLOB data type, 1277
LongMessageBuffer, 2496
LONGTEXT data type, 1277
long_query_time system variable, 580
LOOP, 1692
labels, 1687
loops option
ndbinfo_select_all, 2617
ndb_index_stat, 2651
ndb_show_tables, 2688
Loose Index Scan
GROUP BY optimizing, 994
--loose option prefix, 266
LooseScan
semijoin strategy, 1003
loose_, 3862
lossy-conversions option
ndb_move_data, 2653
ndb_restore, 2668
lost connection errors, 3249
lost+found directory, 529
low-priority option
mysqlimport, 373
low-water mark, 3862
LOWER(), 1374
lower_case_file_system system variable, 581
GRANT, 1730
lower_case_table_names system variable, 581
LOW_PRIORITY
DELETE modifier, 1590
INSERT modifier, 1598
UPDATE modifier, 1650
low_priority_updates system variable, 581
LPAD(), 1375
LRU, 3863
LRU page replacement, 1897
LSN, 3863
LTRIM(), 1375

M

main features of MySQL, 5
maintaining
log files, 715
tables, 959
maintenance
tables, 340
MAKEDATE(), 1357
MAKETIME(), 1357
MAKE_SET(), 1375

Making temporary file (append) before replaying LOAD DATA INFILE
thread state, 1119
Making temporary file (create) before replaying LOAD DATA INFILE
thread state, 1119
malicious SQL statements
and NDB Cluster, 2822
manage keys
thread state, 1112
management client (NDB Cluster), 2626
(see also mgm)
management node (NDB Cluster)
defined, 2382
management nodes (NDB Cluster), 2619
(see also mgmd)
managing NDB Cluster, 2699
managing NDB Cluster processes, 2609
manual
available formats, 2
online location, 2
syntax conventions, 2
typographical conventions, 2
Master has sent all binlog to slave; waiting for binlog to be updated
thread state, 1118
master thread, 3863
master-data option
mysqldump, 359
master-info-file option
mysqld, 2262
master-retry-count option
mysqld, 2262
master_info_repository system variable, 2271, 2316
MASTER_POS_WAIT(), 1484, 1677
master_verify_checksum system variable, 2302
MATCH ... AGAINST(), 1393
matching
patterns, 234
materialization
subqueries, 1003
math, 1486
mathematical functions, 1340
MAX(), 1474
MAX(DISTINCT), 1474
max-allowed-packet option
mysql, 314
mysqldump, 365
max-binlog-dump-events option
mysqld, 2292
max-join-size option
mysql, 314
max-record-length option
myisamchk, 399
max-relay-log-size option
mysqld, 2263
MaxAllocate, 2497
MaxBufferedEpochBytes, 2514
MaxBufferedEpochs, 2514
MAXDB SQL mode, 674
MaxDiskWriteSpeed, 2517

MaxDiskWriteSpeedOtherNodeRestart, 2517
MaxDiskWriteSpeedOwnRestart, 2517
MaxDMLOperationsPerTransaction, 2493
--maximum option prefix, 266
maximums
  maximum columns per table, 1036
  maximum number of databases, 1035, 1522
  maximum number of tables, 1035, 1541
  maximum row size, 1037
  maximum tables per join, 1632
  maximum tables per view, 2972
  table size, 1035
MaxLCPStartDelay, 2500
MaxNoOfAttributes, 2501
MaxNoOfConcurrentIndexOperations, 2494
MaxNoOfConcurrentOperations, 2492
MaxNoOfConcurrentScans, 2495
MaxNoOfConcurrentSubOperations, 2504
MaxNoOfConcurrentTransactions, 2491
MaxNoOfExecutionThreads
  ndbmtd, 2532
MaxNoOfFiredTriggers, 2494
MaxNoOfIndexes, 2503
MaxNoOfLocalOperations, 2493
MaxNoOfLocalScans, 2495
MaxNoOfOpenFiles, 2499
MaxNoOfOrderedIndexes, 2502
MaxNoOfSavedMessages, 2499
MaxNoOfSubscribers, 2503
MaxNoOfSubscriptions, 2503
MaxNoOfTables, 2501
MaxNoOfTriggers, 2502
MaxNoOfUniqueHashIndexes, 2502
MaxParallelCopyInstances, 2496
MaxParallelScansPerFragment, 2497
MaxScanBatchSize, 2553
MaxStartFailRetries, 2547
max_allowed_packet
  and replication, 2361
max_allowed_packet system variable, 582
max_binlog_cache_size system variable, 2303
max_binlog_size system variable, 2303
max_binlog_stmt_cache_size system variable, 2304
max_connections system variable, 583
MAX_CONNECTIONS_PER_HOUR, 820
max_connect_errors system variable, 582
max_delayed_threads system variable, 583
max_digest_length system variable, 583
max_error_count system variable, 584
max_heap_table_size system variable, 584
max_insert_delayed_threads system variable, 585
max_join_size system variable, 329, 585
max_length_for_sort_data system variable, 586
max_long_data_size
  removed features, 20
max_prepared_stmt_count system variable, 586
MAX_QUERIES_PER_HOUR, 820
max_relay_log_size system variable, 2272

MAX_ROWS
  and DataMemory (NDB Cluster), 2488
  and NDB Cluster, 2871
  NDB Cluster, 1551
max_seeks_for_key system variable, 586
max_sort_length system variable, 587
max_sp_recursion_depth system variable, 587
max_tmp_tables system variable, 588
MAX_UPDATES_PER_HOUR, 820
Max_used_connections status variable, 660
MAX_USER_CONNECTIONS, 820
max_user_connections system variable, 588
max_write_lock_count system variable, 588
MBR, 1468
MBRContains(), 1468
MBRDisjoint(), 1468
MBREqual(), 1468
MBRIntersects(), 1468
MBROverlaps(), 1468
MBRTouches(), 1469
MBRWithin(), 1469
MD5(), 1435
MDL, 3863
medium trust, 3863
medium-check option
  myisamchk, 398
  mysqlcheck, 345
MEDIUMBLOB data type, 1277
MEDIUMINT data type, 1253
MEDIUMTEXT data type, 1277
membership
  ndbinfo table, 2789
memcached, 2079, 3864
MEMCACHED_HOME option
  CMake, 178
MEMCACHED_SASL_PWDB environment variable, 2089
memcapable command, 2080
memlock option
  mysqld, 533
memory allocation library, 284
memory allocator
  innodb_use_sys_malloc, 1902
MEMORY storage engine, 2123, 2136
  and replication, 2361
  optimization, 1020
memory usage
  myisamchk, 406
memory use, 1101
  in NDB Cluster, 2403
  Performance Schema, 3073
memoryusage
  ndbinfo table, 2791
memory_per_fragment
  ndbinfo table, 2792
MemReportFrequency, 2524
merge, 3864
MERGE storage engine, 2123, 2145
MERGE tables

defined, 2145
metadata
database, 2976
database object, 1180
InnoDB, 3013
stored routines, 2943
triggers, 2947
views, 2959
metadata lock, 3864
metadata locking, 1092
metadata log, 714
metadata_locks_cache_size system variable, 589
metadata_locks_hash_instances system variable, 589
method option
mysqlhotcopy, 446
methods
locking, 1087
metrics counter, 3864
mgmd (NDB Cluster)
defined, 2382
(see also management node (NDB Cluster))
MICROSECOND(), 1357
MID(), 1375
midpoint insertion, 1897
midpoint insertion strategy, 3864
milestone
MySQL releases, 60
MIN(), 1475
MIN(DISTINCT), 1475
MinDiskWriteSpeed, 2518
MinFreePct, 2488, 2491
mini-transaction, 3864
minimum bounding rectangle, 1468
minus
unary (-), 1338
MINUTE(), 1357
min_examined_row_limit system variable, 589
mirror sites, 61
miscellaneous functions, 1480
mixed statements (Replication), 2367
mixed-mode insert, 3864
MLineFromText(), 1453
MLineFromWKB(), 1455
MM.MySQL, 3865
MOD (modulo), 1343
MOD(), 1343
modes
batch, 241
modify option
MySQLInstallerConsole, 103
modulo (%), 1343
modulo (MOD), 1343
monitor
terminal, 219
monitoring, 1831, 1836, 1863, 1926, 2057, 3189
threads, 1106
Monitors, 2060
enabling, 2061

InnoDB, 1944, 2075, 2112
output, 2064
Mono, 3865
MONTH(), 1357
MONTHNAME(), 1357
MPointFromText(), 1453
MPointFromWKB(), 1455
MPolyFromText(), 1453
MPolyFromWKB(), 1455
.MRG file, 3863
mSQL compatibility, 1386
msql2mysql, 451
deprecated features, 19
MSSQL SQL mode, 675
multi mysqld, 289
multi-column indexes, 1020
multi-core, 3865
Multi-Range Read
optimization, 985
multi-source replication
in NDB Cluster, 2851
multibyte character sets, 3256
multibyte characters, 1232
MULTILINESTRING data type, 1289
MultiLineString(), 1456
MultiLineStringFromText(), 1453
MultiLineStringFromWKB(), 1455
multiple buffer pools, 1896
multiple servers, 729
multiple-part index, 1526
multiplication (*), 1339
MULTIPOINT data type, 1289
MultiPoint(), 1456
MultiPointFromText(), 1453
MultiPointFromWKB(), 1455
MULTIPOLYGON data type, 1289
MultiPolygon(), 1456
MultiPolygonFromText(), 1453
MultiPolygonFromWKB(), 1455
mutex, 3865
mutex wait
monitoring, 2057
mutex_instances table
performance_schema, 3111
MVCC, 3865
MVCC (multi-version concurrency control), 1827
My
derivation, 8
my-print-defaults option
mysql_plugin, 299
my.cnf, 3865
and NDB Cluster, 2428, 2468, 2469
in NDB Cluster, 2748
my.cnf option file, 2348
my.ini, 3865
mycnf option
ndb_config, 2634
ndb_mgmd, 2623

.MYD file, 3863
.MYI file, 3863
MyISAM
  compressed tables, 408, 2134
  converting tables to InnoDB, 1846
MyISAM key cache, 1075
MyISAM storage engine, 2123, 2127
myisam-block-size option
  mysqld, 534
myisamchk, 254, 391
  analyze option, 399
  backup option, 398
  block-search option, 400
  character-sets-dir option, 398
  check option, 397
  check-only-changed option, 397
  correct-checksum option, 398
  data-file-length option, 398
  debug option, 394
  defaults-extra-file option, 394
  defaults-file option, 395
  defaults-group-suffix option, 395
  description option, 400
  example output, 400
  extend-check option, 397, 398
  fast option, 397
  force option, 397, 398
  help option, 394
  HELP option, 394
  information option, 397
  keys-used option, 398
  max-record-length option, 399
  medium-check option, 398
  no-defaults option, 395
  no-symlinks option, 398
  options, 394
  parallel-recover option, 399
  print-defaults option, 395
  quick option, 399
  read-only option, 398
  recover option, 399
  safe-recover option, 399
  set-auto-increment[option, 400
  set-collation option, 399
  silent option, 395
  sort-index option, 400
  sort-records option, 400
  sort-recover option, 399
  tmpdir option, 399
  unpack option, 399
  update-state option, 398
  verbose option, 395
  version option, 395
  wait option, 395
myisamlog, 254, 407
myisampack, 254, 408, 1572, 2134
  backup option, 409
  character-sets-dir option, 409

debug option, 409
force option, 409
help option, 409
join option, 409
silent option, 409
test option, 409
tmpdir option, 409
verbose option, 409
version option, 409
wait option, 410
myisam_block_size myisamchk variable, 395
myisam_data_pointer_size system variable, 590
myisam_ftdump, 254, 390
count option, 390
dump option, 391
help option, 390
length option, 391
stats option, 391
verbose option, 391
myisam_max_sort_file_size system variable, 590, 2131
myisam_mmap_size system variable, 591
myisam_recover_options system variable, 591, 2131
myisam_repair_threads system variable, 592
myisam_sort_buffer_size myisamchk variable, 395
myisam_sort_buffer_size system variable, 592, 2131
myisam_stats_method system variable, 593
myisam_use_mmap system variable, 593
MyODBC drivers, 3865
MySQL
debugging, 736
defined, 4
forums, 35
introduction, 4
pronunciation, 5
upgrading, 300
websites, 35
mysql, 253, 307, 3865
auto-rehash option, 310
auto-vertical-output option, 310
batch option, 310
binary-as-hex option, 310
binary-mode option, 311
bind-address option, 311
character-sets-dir option, 311
charset command, 320
clear command, 321
column-names option, 311
column-type-info option, 311
comments option, 311
compress option, 312
connect command, 321
connect-expired-password option, 312
connect-timeout option, 312
database option, 312
debug option, 312
debug-check option, 312
debug-info option, 312
default-auth option, 312

default-character-set option, 312
defaults-extra-file option, 313
defaults-file option, 313
defaults-group-suffix option, 313
delimiter command, 321
delimiter option, 313
disable named commands, 313
edit command, 321
ego command, 321
enable-cleartext-plugin option, 313
execute option, 313
exit command, 321
force option, 313
go command, 321
help command, 320
help option, 310
histignore option, 313
host option, 314
html option, 314
i-am-a-dummy option, 317
ignore-spaces option, 314
init-command option, 314
line-numbers option, 314
local-infile option, 314, 779
login-path option, 314
max-allowed-packet option, 314
max-join-size option, 314
named-commands option, 314
net-buffer-length option, 314
no-auto-rehash option, 314
no-beep option, 315
no-defaults option, 315
nopager command, 321
notee command, 321
nowarning command, 322
one-database option, 315
pager command, 322
pager option, 315
password option, 315
pipe option, 316
plugin-dir option, 316
port option, 316
print command, 322
print-defaults option, 316
prompt command, 322
prompt option, 316
protocol option, 316
quick option, 316
quit command, 322
raw option, 316
reconnect option, 317
rehash command, 322
safe-updates option, 317
secure-auth option, 317
select-limit option, 317
server-public-key-path option, 317
shared-memory-base-name option, 318
show-warnings option, 318

sigint-ignore option, 318
silent option, 318
skip-column-names option, 318
skip-line-numbers option, 318
socket option, 318
source command, 322
SSL options, 318
status command, 322
system command, 322
table option, 319
tee command, 323
tee option, 319
unbuffered option, 319
use command, 323
user option, 319
verbose option, 319
version option, 319
vertical option, 319
wait option, 319
warnings command, 323
xml option, 319
MySQL APT Repository, 137, 206
MySQL binary distribution, 60
MySQL C API, 3185
mysql client parser
versus mysqld parser, 331
MySQL Cluster Manager
and ndb_mgm, 2699
mysql command options, 307
mysql commands
list of, 320
MySQL Dolphin name, 8
MySQL Enterprise Audit, 883, 3191
new features, 10
MySQL Enterprise Backup, 3190, 3866
MySQL Enterprise Data Masking and De-Identification, 3192
MySQL Enterprise Encryption, 923, 3191
new features, 10
MySQL Enterprise Firewall, 908, 3192
installing, 909
new features, 10
using, 912
MySQL Enterprise Firewall stored procedures
sp_reload_firewall_rules(), 919
sp_set_firewall_mode(), 920
MySQL Enterprise Firewall tables
firewall_users, 919
firewall_whitelist, 919
MySQL Enterprise Firewall UDFs
mysql_firewall_flush_status(), 921
normalize_statement(), 921
read_firewall_users(), 921
read_firewall_whitelist(), 921
set_firewall_mode(), 921
MySQL Enterprise Monitor, 3189
MySQL Enterprise Security, 853, 863, 3191
MySQL Enterprise Thread Pool, 722, 3192
elements, 722

installing, 723
MySQL history, 8
mysql history file, 325
MySQL Installer, 81
MySQL name, 8
MySQL privileges
and NDB Cluster, 2822
mysql prompt command, 323
MySQL server
mysqld, 281, 462
MySQL SLES Repository, 137, 206
mysql source (command for reading from text files), 242, 327
MySQL source distribution, 60
MySQL storage engines, 2123
MySQL system tables
and NDB Cluster, 2822
and replication, 2362
MySQL version, 61
MySQL Yum Repository, 131, 134, 204
mysql \. (command for reading from text files), 242, 327
mysql.event table, 2954
mysql.ndb_binlog_index table, 2834
(see also NDB Cluster replication)
mysql.server, 252, 287
basedir option, 288
datadir option, 288
pid-file option, 288
service-startup-timeout option, 289
mysql.slave_master_info table, 2316
mysql.slave_relay_log_info table, 2316
mysql.sock
protection, 3264
MYSQL323 SQL mode, 675
MYSQL40 SQL mode, 675
mysqlaccess, 254, 420
brief option, 421
commit option, 421
copy option, 422
db option, 422
debug option, 422
deprecated features, 19
help option, 421
host option, 422
howto option, 422
old_server option, 422
password option, 422
plan option, 422
preview option, 422
relnotes option, 422
rhost option, 422
rollback option, 422
spassword option, 422
superuser option, 423
table option, 423
user option, 423
version option, 423
mysqladmin, 253, 331, 1522, 1581, 1794, 1800, 1806, 1811
bind-address option, 336

character-sets-dir option, 336
compress option, 336
connect-timeout option, 336
count option, 336
debug option, 336
debug-check option, 336
debug-info option, 336
default-auth option, 336
default-character-set option, 336
defaults-extra-file option, 337
defaults-file option, 337
defaults-group-suffix option, 337
enable-cleartext-plugin option, 337
force option, 337
help option, 336
host option, 337
login-path option, 337
no-beep option, 337
no-defaults option, 337
password option, 338
pipe option, 338
plugin-dir option, 338
port option, 338
print-defaults option, 338
protocol option, 338
relative option, 338
secure-auth option, 339
shared-memory-base-name option, 339
shutdown-timeout option, 339
silent option, 339
sleep option, 339
socket option, 339
SSL options, 339
user option, 339
verbose option, 340
version option, 340
vertical option, 340
wait option, 340
mysqladmin command options, 334
mysqladmin option
mysqld_multi, 290
mysqlbackup command, 3866
mysqlbinlog, 254, 423
base64-output option, 426
bind-address option, 426
binlog-row-event-max-size option, 426
character-sets-dir option, 427
connection-server-id option, 427
database option, 427
debug option, 428
debug-check option, 428
debug-info option, 428
default-auth option, 428
defaults-extra-file option, 428
defaults-file option, 428
defaults-group-suffix option, 429
disable-log-bin option, 429
exclude-gtids option, 429

force-if-open option, 429
force-read option, 429
help option, 426
hexdump option, 429
host option, 429
include-gtids option, 429
local-load option, 429
login-path option, 430
no-defaults option, 430
offset option, 430
open-files-limit option, 430
password option, 430
plugin-dir option, 430
port option, 430
print-defaults option, 430
protocol option, 430
raw option, 431
read-from-remote-master option, 431
read-from-remote-server option, 431
result-file option, 431
secure-auth option, 431
server-id option, 432
server-id-bits option, 432
set-charset option, 432
shared-memory-base-name option, 432
short-form option, 432
skip-gtids option, 432
socket option, 432
SSL options, 433
start-datetime option, 433
start-position option, 433
stop-datetime option, 433
stop-never option, 433
stop-never-slave-server-id option, 433
stop-position option, 433
to-last-log option, 434
user option, 434
verbose option, 434
verify-binlog-checksum option, 434
version option, 434
mysqlbug, 293
deprecated features, 19
mysqlcheck, 253, 340
all-databases option, 343
all-in-1 option, 343
analyze option, 343
auto-repair option, 343
bind-address option, 343
character-sets-dir option, 343
check option, 343
check-only-changed option, 343
check-upgrade option, 343
compress option, 343
databases option, 343
debug option, 343
debug-check option, 344
debug-info option, 344
default-auth option, 345

default-character-set option, 344
defaults-extra-file option, 344
defaults-file option, 344
defaults-group-suffix option, 344
enable-cleartext-plugin option, 345
extended option, 344
fast option, 345
fix-db-names option, 345
fix-table-names option, 345
force option, 345
help option, 343
host option, 345
login-path option, 345
medium-check option, 345
no-defaults option, 345
optimize option, 345
password option, 346
pipe option, 346
plugin-dir option, 346
port option, 346
print-defaults option, 346
protocol option, 346
quick option, 346
repair option, 346
secure-auth option, 347
shared-memory-base-name option, 347
silent option, 347
skip-database option, 347
socket option, 347
SSL options, 347
tables option, 347
use-frm option, 347
user option, 348
verbose option, 348
version option, 348
write-binlog option, 348
mysqlclient, 3866
mysqld, 252, 3866
abort-slave-event-count option, 2261
allow-suspicious-udfs option, 524
ansi option, 524
as NDB Cluster process, 2557, 2747
audit-log option, 901
basedir option, 524
binlog-checksum option, 2292
binlog-do-db option, 2289
binlog-ignore-db option, 2291
binlog-row-event-max-size option, 2288
bootstrap option, 524
character-set-client-handshake option, 524
chroot option, 525
command options, 522
console option, 525
core-file option, 525
datadir option, 525
debug option, 526
debug-sync-timeout option, 526
default-authentication-plugin option, 526

default-time-zone option, 527
defaults-extra-file option, 527
defaults-file option, 528
defaults-group-suffix option, 528
des-key-file option, 528
disconnect-slave-event-count option, 2261
exit-info option, 528
external-locking option, 528
flush option, 528
gdb option, 529
help option, 524
ignore-db-dir option, 529
innodb option, 1966
innodb-status-file option, 1966
install option, 529
install-manual option, 530
language option, 530
large-pages option, 530
lc-messages option, 531
lc-messages-dir option, 531
local-service option, 531
log-bin option, 2288
log-bin-index option, 2289
log-error option, 531
log-isam option, 532
log-raw option, 532
log-short-format option, 532
log-slow-admin-statements option, 532
log-slow-slave-statements option, 2261
log-tc option, 533
log-tc-size option, 533
log-warnings option, 533, 2262
log_slow_admin_statements system variable, 579
master-info-file option, 2262
master-retry-count option, 2262
max-binlog-dump-events option, 2292
max-relay-log-size option, 2263
memlock option, 533
myisam-block-size option, 534
MySQL server, 281, 462
ndb-batch-size option, 2558
ndb-blob-read-batch-bytes option, 2558
ndb-blob-write-batch-bytes option, 2559
ndb-cluster-connection-pool option, 2558
ndb-connectstring option, 2559
ndb-log-apply-status, 2560
ndb-log-empty-epochs, 2561
ndb-log-empty-update, 2561
ndb-log-exclusive-reads, 2561
ndb-log-fail-terminate, 2562
ndb-log-orig, 2562
ndb-log-transaction-id, 2562
ndb-nodeid, 2563
ndb-optimization-delay option, 2564
ndb-transid-mysql-connection-map option, 2564
ndb-wait-connected option, 2564
ndb-wait-setup option, 2565
ndbcluster option, 2557

no-defaults option, 534
old-style-user-limits option, 534
partition option, 535
performance-schema-consumer-events-stages-current option, 3159
performance-schema-consumer-events-stages-history option, 3159
performance-schema-consumer-events-stages-history-long option, 3159
performance-schema-consumer-events-statements-current option, 3159
performance-schema-consumer-events-statements-history option, 3159
performance-schema-consumer-events-statements-history-long option, 3159
performance-schema-consumer-events-waits-current option, 3159
performance-schema-consumer-events-waits-history option, 3159
performance-schema-consumer-events-waits-history-long option, 3159
performance-schema-consumer-global-instrumentation option, 3159
performance-schema-consumer-statements-digest option, 3159
performance-schema-consumer-thread-instrumentation option, 3159
performance-schema-consumer-xxx option, 3158
performance-schema-instrument option, 3158
plugin option prefix, 536
plugin-load option, 535
plugin-load-add option, 536
port option, 537
port-open-timeout option, 537
print-defaults option, 537
relay-log-purge option, 2263
relay-log-space-limit option, 2263
remove option, 537
replicate-do-db option, 2264
replicate-do-table option, 2266
replicate-ignore-db option, 2265
replicate-ignore-table option, 2266
replicate-rewrite-db option, 2267
replicate-same-server-id option, 2267
replicate-wild-do-table option, 2267
replicate-wild-ignore-table option, 2268
role in NDB Cluster (see SQL Node (NDB Cluster))
safe-user-create option, 537
server-id-bits option, 2565
server_uuid variable, 2249
show-slave-auth-info option, 2256
skip-grant-tables option, 538
skip-host-cache option, 539
skip-innodb option, 539, 1967
skip-ndbcluster option, 2566
skip-new option, 539
skip-partition option, 539
skip-show-database option, 539
skip-slave-start option, 2268
skip-ssl option, 541
skip-stack-trace option, 540
skip-symbolic-links option, 542
slave-skip-errors option, 2269
slave-sql-verify-checksum option, 2269
slow-start-timeout option, 540
socket option, 540
sporadic-binlog-dump-fail option, 2292
sql-mode option, 540
ssl option, 541
standalone option, 542

starting, 778
super-large-pages option, 542
symbolic-links option, 542
sysdate-is-now option, 542
tc-heuristic-recover option, 543
temp-pool option, 543
tmpdir option, 544
transaction-isolation option, 543
transaction-read-only option, 544
user option, 544
validate-password option, 880
verbose option, 545
version option, 545
mysqld (NDB Cluster), 2609
mysqld option
malloc-lib, 284
mysqld_multi, 290
mysqld_safe, 284
mysql_plugin, 299
mysqld options, 462
mysqld options and variables
NDB Cluster, 2557
mysqld parser
versus mysql client parser, 331
mysqld system variables, 462
mysqld-version option
mysqld_safe, 284
MySQLdb, 3866
mysqldump, 210, 253, 348, 3866
add-drop-database option, 356
add-drop-table option, 356
add-drop-trigger option, 356
add-locks option, 365
all-databases option, 363
all-tablespaces option, 356
allow-keywords option, 357
apply-slave-statements option, 359
bind-address option, 353
character-sets-dir option, 358
comments option, 357
compact option, 360
compatible option, 361
complete-insert option, 361
compress option, 353
create-options option, 361
databases option, 363
debug option, 357
debug-check option, 357
debug-info option, 357
default-auth option, 354
default-character-set option, 358
defaults-extra-file option, 355
defaults-file option, 355
defaults-group-suffix option, 356
delayed-insert option, 364
delete-master-logs option, 359
disable-keys option, 365
dump-date option, 357

dump-slave option, 359
enable-cleartext-plugin option, 354
events option, 363
extended-insert option, 365
fields-enclosed-by option, 361, 372
fields-escaped-by option, 361, 372
fields-optionally-enclosed-by option, 361, 372
fields-terminated-by option, 361, 372
flush-logs option, 365
flush-privileges option, 366
force option, 358
help option, 358
hex-blob option, 361
host option, 354
ignore-table option, 363
include-master-host-port option, 359
insert-ignore option, 365
lines-terminated-by option, 361, 372
lock-all-tables option, 366
lock-tables option, 366
log-error option, 358
login-path option, 354
master-data option, 359
max-allowed-packet option, 365
net-buffer-length option, 365
no-autocommit option, 366
no-create-db option, 357
no-create-info option, 357
no-data option, 364
no-defaults option, 356
no-set-names option, 358
no-tablespaces option, 357
opt option, 365
order-by-primary option, 366
password option, 354
pipe option, 354
plugin-dir option, 354
port option, 354
print-defaults option, 356
problems, 368, 2973
protocol option, 354
quick option, 365
quote-names option, 361
replace option, 357
result-file option, 361
routines option, 364
secure-auth option, 355
set-charset option, 358
set-gtid-purged option, 360
shared-memory-base-name option, 366
single-transaction option, 366
skip-comments option, 358
skip-opt option, 365
socket option, 355
SSL options, 355
tab option, 361
tables option, 364
triggers option, 364

tz-utc option, 362
user option, 355
using for backups, 946
verbose option, 358
version option, 358
views, 368, 2973
where option, 364
workarounds, 368, 2973
xml option, 362
mysqldumpslow, 254, 442
debug option, 443
help option, 443
verbose option, 444
mysqld_multi, 252, 289
defaults-extra-file option, 289
defaults-file option, 289
example option, 290
help option, 290
log option, 290
mysqladmin option, 290
mysqld option, 290
no-defaults option, 289
no-log option, 290
password option, 290
silent option, 290
tcp-ip option, 290
user option, 291
verbose option, 291
version option, 291
mysqld_safe, 252, 282
basedir option, 283
core-file-size option, 283
datadir option, 283
defaults-extra-file option, 283
defaults-file option, 283
help option, 283
ledir option, 283
log-error option, 283
malloc-lib option, 284
mysqld option, 284
mysqld-version option, 284
nice option, 285
no-defaults option, 285
open-files-limit option, 285
pid-file option, 285
plugin-dir option, 285
port option, 285
skip-kill-mysqld option, 285
skip-syslog option, 285
socket option, 285
syslog option, 285
syslog-tag option, 285
timezone option, 285
user option, 286
mysqlhotcopy, 255, 444
addtodest option, 445
allowold option, 445
checkpoint option, 445

chroot option, 445
debug option, 445
deprecated features, 19
dryrun option, 445
flushlog option, 445
help option, 445
host option, 446
keepold option, 446
method option, 446
noindices option, 446
old_server option, 446
password option, 446
port option, 446
quiet option, 446
record_log_pos option, 446
regexp option, 446
resetmaster option, 446
resetslave option, 446
socket option, 446
suffix option, 446
tmpdir option, 447
user option, 447
mysqlimport, 210, 253, 368, 1604
bind-address option, 370
character-sets-dir option, 370
columns option, 370
compress option, 370
debug option, 371
debug-check option, 371
debug-info option, 371
default-auth option, 371
default-character-set option, 371
defaults-extra-file option, 371
defaults-file option, 371
defaults-group-suffix option, 371
delete option, 372
enable-cleartext-plugin option, 372
force option, 372
help option, 370
host option, 372
ignore option, 372
ignore-lines option, 372
local option, 372, 779
lock-tables option, 372
login-path option, 372
low-priority option, 373
no-defaults option, 373
password option, 373
pipe option, 373
plugin-dir option, 373
port option, 373
print-defaults option, 373
protocol option, 373
replace option, 374
secure-auth option, 374
shared-memory-base-name option, 374
silent option, 374
socket option, 374

SSL options, 374
use-threads option, 374
user option, 374
verbose option, 375
version option, 375
MySQLInstallerConsole, 100
configure option, 101
help option, 101
install option, 101
list option, 103
modify option, 103
remove option, 103
status option, 103
update option, 103
upgrade option, 103
mysqlshow, 254, 375
bind-address option, 377
character-sets-dir option, 377
compress option, 377
count option, 377
debug option, 377
debug-check option, 377
debug-info option, 377
default-auth option, 377
default-character-set option, 377
defaults-extra-file option, 378
defaults-file option, 378
defaults-group-suffix option, 378
enable-cleartext-plugin option, 378
help option, 377
host option, 378
keys option, 378
login-path option, 378
no-defaults option, 378
password option, 379
pipe option, 379
plugin-dir option, 379
port option, 379
print-defaults option, 379
protocol option, 379
secure-auth option, 379
shared-memory-base-name option, 380
show-table-type option, 380
socket option, 380
SSL options, 380
status option, 380
user option, 380
verbose option, 380
version option, 380
mysqlslap, 254, 380
auto-generate-sql option, 383
auto-generate-sql-add-autoincrement option, 383
auto-generate-sql-execute-number option, 383
auto-generate-sql-guid-primary option, 384
auto-generate-sql-load-type option, 384
auto-generate-sql-secondary-indexes option, 384
auto-generate-sql-unique-query-number option, 384
auto-generate-sql-unique-write-number option, 384

auto-generate-sql-write-number option, 384
commit option, 384
compress option, 384
concurrency option, 384
create option, 384
create-schema option, 384
csv option, 384
debug option, 384
debug-check option, 385
debug-info option, 385
default-auth option, 385
defaults-extra-file option, 385
defaults-file option, 385
defaults-group-suffix option, 385
delimiter option, 385
detach option, 386
enable-cleartext-plugin option, 386
engine option, 386
help option, 383
host option, 386
iterations option, 386
login-path option, 386
no-defaults option, 386
no-drop option, 386
number-char-cols option, 386
number-int-cols option, 386
number-of-queries option, 386
only-print option, 387
password option, 387
pipe option, 387
plugin-dir option, 387
port option, 387
post-query option, 387
post-system option, 387
pre-query option, 387
pre-system option, 387
print-defaults option, 388
protocol option, 388
query option, 388
secure-auth option, 388
shared-memory-base-name option, 388
silent option, 388
socket option, 388
SSL options, 389
user option, 389
verbose option, 389
version option, 389
mysql_clear_password authentication plugin, 852
mysql_cluster_backup_privileges, 2760
mysql_cluster_move_grant_tables, 2760
mysql_cluster_move_privileges, 2760
mysql_cluster_privileges_are_distributed, 2760
mysql_cluster_restore_local_privileges, 2762
mysql_cluster_restore_privileges, 2761
mysql_cluster_restore_privileges_from_local, 2762
mysql_config, 451
cflags option, 452
cxxflags option, 452

embedded option, 452
embedded-libs option, 452
include option, 452
libmysqld-libs option, 452
libs option, 452
libs_r option, 452
plugindir option, 452
port option, 452
socket option, 452
variable option, 452
version option, 452
mysql_config_editor, 254, 414
debug option, 417
help option, 417
verbose option, 417
version option, 417
mysql_convert_table_format, 255, 447
deprecated features, 19
force option, 447
help option, 447
host option, 447
password option, 447
port option, 447
socket option, 447
type option, 448
user option, 448
verbose option, 448
version option, 448
MYSQL_DATADIR option
CMake, 171
MYSQL_DEBUG environment variable, 256, 457, 743
mysql_find_rows, 255, 448
deprecated features, 19
help option, 448
regexp option, 448
rows option, 448
skip-use-db option, 448
start_row option, 448
mysql_firewall_flush_status() MySQL Enterprise Firewall UDF, 921
mysql_firewall_max_query_size system variable, 922
mysql_firewall_mode system variable, 922
mysql_firewall_trace system variable, 922
MYSQL_FIREWALL_USERS
INFORMATION_SCHEMA table, 3060
MYSQL_FIREWALL_WHITELIST
INFORMATION_SCHEMA table, 3060
mysql_fix_extensions, 255, 448
deprecated features, 19
MYSQL_GROUP_SUFFIX environment variable, 457
MYSQL_HISTFILE environment variable, 325, 457
MYSQL_HISTIGNORE environment variable, 325, 457
MYSQL_HOME environment variable, 457
MYSQL_HOST environment variable, 278, 457
mysql_info(), 1503, 1596, 1612, 1651
mysql_insert_id(), 1597
mysql_install_db, 183, 253, 293
basedir option, 295
builddir option, 295

cross-bootstrap option, 295
datadir option, 295
defaults-extra-file option, 295
defaults-file option, 296
force option, 296
help option, 295
keep-my-cnf option, 296
ldata option, 296
no-defaults option, 296
random-passwords option, 296
rpm option, 297
skip-name-resolve option, 297
srcdir option, 297
user option, 297
verbose option, 297
windows option, 297
MYSQL_MAINTAINER_MODE option
CMake, 174
mysql_native_password authentication plugin, 842
mysql_old_password
deprecated features, 19
mysql_old_password authentication plugin, 843
MYSQL_OPENSSL_UDF_DH_BITS_THRESHOLD environment variable, 457, 926
MYSQL_OPENSSL_UDF_DSA_BITS_THRESHOLD environment variable, 457, 926
MYSQL_OPENSSL_UDF_RSA_BITS_THRESHOLD environment variable, 457, 926
mysql_options()
MYSQL_OPT_LOCAL_INFILE, 779
mysql_plugin, 253, 297
basedir option, 298
datadir option, 298
help option, 298
my-print-defaults option, 299
mysqld option, 299
no-defaults option, 299
plugin-dir option, 299
plugin-ini option, 299
print-defaults option, 299
verbose option, 299
version option, 299
MYSQL_PROJECT_NAME option
CMake, 174
MYSQL_PS1 environment variable, 457
MYSQL_PWD environment variable, 256, 278, 457
mysql_real_escape_string(), 1125, 1376
mysql_secure_installation, 253, 299
mysql_setpermission, 255, 449
deprecated features, 19
help option, 449
host option, 449
password option, 449
port option, 449
socket option, 450
user option, 450
MYSQL_TCP_PORT environment variable, 256, 457, 735, 736
MYSQL_TCP_PORT option
CMake, 174
MYSQL_TEST_LOGIN_FILE environment variable, 265, 414, 457
mysql_tzinfo_to_sql, 253, 300

MYSQL_UNIX_ADDR option
  CMake, 174
MYSQL_UNIX_PORT environment variable, 184, 256, 457, 735, 736
mysql_upgrade, 253, 300
  basedir option, 303
  character-sets-dir option, 303
  compress option, 303
  datadir option, 303
  debug option, 304
  debug-check option, 304
  debug-info option, 304
  default-auth option, 304
  default-character-set option, 304
  defaults-extra-file option, 304
  defaults-file option, 304
  defaults-group-suffix option, 304
  force option, 304
  help option, 303
  host option, 304
  login-path option, 304
  mysql_upgrade_info file, 302
  no-defaults option, 305
  password option, 305
  pipe option, 305
  plugin-dir option, 305
  port option, 305
  print-defaults option, 305
  protocol option, 305
  shared-memory-base-name option, 306
  socket option, 306
  SSL options, 306
  tmpdir option, 306
  upgrade-system-tables option, 306
  user option, 306
  verbose option, 306
  version-check option, 306
  write-binlog option, 306
mysql_upgrade_info file
  mysql_upgrade, 302
mysql_waitpid, 255, 450
  deprecated features, 19
  help option, 450
  verbose option, 450
  version option, 450
mysql_zap, 255, 450
  deprecated features, 19
my_print_defaults, 255, 453
  config-file option, 453
  debug option, 453
  defaults-extra-file option, 453
  defaults-file option, 453
  defaults-group-suffix option, 453
  extra-file option, 453
  help option, 453
  login-path option, 454
  no-defaults option, 454
  show option, 454
  verbose option, 454

version option, 454

N

name-file option
comp_err, 293
named pipes, 106, 111
named time zone support
Unknown or incorrect time zone, 686
named-commands option
mysql, 314
named_pipe system variable, 593
named_pipe_full_access_group system variable, 594
names, 1132
case sensitivity, 1136
variables, 1165
NAME_CONST(), 1484, 2967
naming
releases of MySQL, 60
NATIONAL CHAR data type, 1275
NATIONAL VARCHAR data type, 1276
native backup and restore
backup identifiers, 2745
native C API, 3866
NATURAL JOIN, 1630
natural key, 3866
NATURAL LEFT JOIN, 1630
NATURAL LEFT OUTER JOIN, 1630
NATURAL RIGHT JOIN, 1630
NATURAL RIGHT OUTER JOIN, 1630
NCHAR data type, 1275
NDB API
and distributed grant tables, 2762
and distributed privileges, 2762
NDB API counters (NDB Cluster), 2762
scope, 2765
status variables associated with, 2767
types, 2765
NDB API database objects
and NDB Cluster replication, 2827
NDB API replica status variables
and NDB Cluster Replication, 2825
NDB client programs
defaults-extra-file option, 2696
defaults-file option, 2696
defaults-group-suffix option, 2697
login-path option, 2697
no-defaults option, 2698
print-defaults option, 2698
NDB Cluster, 2378
"quick" configuration, 2449
administration, 2557, 2609, 2619, 2626, 2627, 2694, 2699, 2725
and application feature requirements, 2400
and DNS, 2412
and INFORMATION_SCHEMA, 2823
and IP addressing, 2412
and MySQL privileges, 2821
and MySQL root user, 2822, 2824
and networking, 2387

and transactions, 2488
API node, 2382, 2549
applications supported, 2399
availability, 2398
available platforms, 2379
BACKUP Events, 2725
backups, 2659, 2743, 2743, 2743, 2746, 2747
CHECKPOINT Events, 2721
cluster logs, 2717, 2719
CLUSTERLOG commands, 2719
CLUSTERLOG STATISTICS command, 2725
commands, 2557, 2609, 2619, 2627, 2699
compared to InnoDB, 2397, 2398, 2399, 2400
compared to standalone MySQL Server, 2397, 2398, 2399, 2400
concepts, 2382
configuration, 2411, 2449, 2449, 2476, 2477, 2483, 2549, 2626, 2748
configuration (example), 2469
configuration changes, 2730
configuration files, 2428, 2468
configuration parameters, 2451, 2452, 2458, 2458, 2460
configuring, 2746
CONNECT command, 2700
CONNECTION Events, 2720
connection string, 2475
CREATE NODEGROUP command, 2702
data node, 2382, 2483
data nodes, 2609, 2618
defining node hosts, 2476
deployment with Auto-Installer (no longer supported), 2438
direct connections between nodes, 2603
Disk Data tables (see NDB Cluster Disk Data)
DROP NODEGROUP command, 2702
ENTER SINGLE USER MODE command, 2702
ERROR Events, 2724
error logs, 2615
event log format, 2720
event logging thresholds, 2719
event logs, 2717, 2718
event severity levels, 2719
event types, 2718, 2720
execution threads, 2532
EXIT command, 2702
EXIT SINGLE USER MODE command, 2702
FAQ, 3206
fragment replicas, 2384
GCP Stop errors, 2545
general description, 2380
HELP command, 2700
HostName parameter
  and security, 2818
INFO Events, 2724
information sources, 2380
insecurity of communication protocols, 2817
installation, 2411
installation (Linux), 2413
installation (Windows), 2419
installing .deb file (Linux), 2418
installing binary (Windows), 2419

installing binary release (Linux), 2414
installing from source (Linux), 2418
installing from source (Windows), 2423
installing RPM (Linux), 2416
interconnects, 2608
Java clients, 2383
large tables, 1551
log files, 2615, 2618
logging commands, 2719
management client (ndb_mgm), 2626
management commands, 2725
management node, 2382, 2477
management nodes, 2619
managing, 2699
MAX_ROWS, 1551
memory usage and recovery, 2403, 2730
mgm, 2694
mgm client, 2699
mgm management client, 2725
mgm process, 2627
mgmd, 2694
mgmd process, 2619
monitoring, 2762
multiple management servers, 2731
mysqld options and variables for, 2557
mysqld process, 2557, 2747
ndbd, 2609, 2694
ndbd process, 2609, 2727
ndbinfo_select_all, 2616
ndbmtd, 2618
ndb_mgm, 2430, 2626
ndb_mgmd process, 2619
network configuration
  and security, 2818
network transporters, 2608
networking, 2603, 2604
new features, 14
node failure (single user mode), 2732
node identifiers, 2605, 2605
node logs, 2717
NODERESTART Events, 2722
nodes and node groups, 2384
nodes and types, 2382
partitioning support, 2402
partitions, 2384
performing queries, 2431
preparing for replication, 2836
process management, 2609
QUIT command, 2702
replication, 2824
  (see also NDB Cluster replication)
REPORT command, 2701
requirements, 2387
resetting, 2730
RESTART command, 2701
restarting, 2434
restoring backups, 2659
rolling restarts (multiple management servers), 2731

runtime statistics, 2725
SCHEMA Events, 2724
security, 2817
and firewalls, 2819, 2820
and HostName parameter, 2818
and network configuration, 2818
and network ports, 2821
and remote administration, 2821
networking, 2817
security procedures, 2823
shared memory transport, 2604
SHOW command, 2700
SHUTDOWN command, 2702
shutting down, 2434
single user mode, 2702, 2731
SINGLEUSER Events, 2724
SQL node, 2382, 2549
SQL nodes, 2747
SQL statements for monitoring, 2812
START BACKUP command, 2842
START command, 2700
start phases (summary), 2728
starting, 2449
starting nodes, 2423, 2430
starting or restarting, 2728
STARTUP Events, 2721
STATISTICS Events, 2723
STATUS command, 2701
status variables, 2584
STOP command, 2700
storage requirements, 1304
thread states, 1120
trace files, 2615
transaction handling, 2406
transaction isolation level, 2404
transporters
shared memory (SHM), 2604
TCP/IP, 2603
troubleshooting backups, 2747
upgrades and downgrades, 2435, 2729
USING HASH, 1529
using tables and data, 2431
NDB Cluster 7.3, 2388
NDB Cluster 7.4, 2388
NDB Cluster Auto-Installer
setup program, 2684
setup program (Windows), 2684
NDB Cluster Auto-Installer (no longer supported), 2438
adding and removing hosts in, 2443
adding processes, 2445
and ndb_setup.py, 2440
and Python, 2438
and setup.bat (Windows), 2440
architecture, 2438
authentication with remote hosts, 2439
Define Attributes screen, 2445
Define Cluster screen, 2441
Define Hosts screen, 2442

Define Processes screen, 2444
Deploy Cluster screen, 2447
remote vs local hosts, 2439, 2439
removing processes, 2445
requirements, 2438
security issues, 2439
software requirements, 2438
starting, 2440
supported platforms, 2438
supported web browsers, 2438
using, 2439
Welcome screen, 2441
NDB Cluster Disk Data, 2748
creating log file groups, 2750
creating tables, 2749, 2751
creating tablespaces, 2750
dropping Disk Data objects, 2753
storage requirements, 2755
NDB Cluster How-To, 2411
NDB Cluster limitations, 2400
and differences from standard MySQL limits, 2403
binary logging, 2409
database objects, 2407
Disk Data storage, 2410
error handling and reporting, 2407
geometry data types, 2402
implementation, 2409
imposed by configuration, 2403
memory usage and transaction handling, 2406
multiple management servers, 2410
multiple MySQL servers, 2410
partitioning, 2402
performance, 2408
replication, 2403
resolved in current version from previous versions, 2411
syntax, 2401
transactions, 2404
unsupported features, 2408
NDB Cluster processes, 2609
NDB Cluster programs, 2609
NDB Cluster replication, 2824
and --initial option, 2831
and circular replication, 2828
and NDB API database objects, 2827
and primary key, 2830
and single point of failure, 2839
and unique keys, 2830
backups, 2842
bidirectional replication, 2848
circular replication, 2848
concepts, 2826, 2826
conflict resolution, 2851
failover, 2839, 2840
gap event, 2828
known issues, 2827
loss of connection, 2827
point-in-time recovery, 2847
preparing, 2836

read conflict detection and resolution, 2862
requirements, 2826
reset-replica.pl backup automation script, 2844
restoring from backup, 2842
starting, 2838
storage engines other than NDB on replica, 2832
synchronization of source and replica, 2844
system tables used, 2834
NDB Cluster Replication
and NDB API replica status variables, 2825
NDB Cluster replication conflict resolution
exceptions table, 2859
ndb option
perror, 455
NDB statistics variables
and NDB API counters, 2767
NDB statistics variables (NDB Cluster), 2762
scope, 2765
types, 2765
NDB storage engine (see NDB Cluster)
FAQ, 3206
NDB tables
and MySQL root user, 2822
NDB utilities
security issues, 2824
NDB$CFT_CAUSE, 2860
NDB$EPOCH(), 2855
limitations, 2857
NDB$EPOCH2(), 2858
NDB$EPOCH2_TRANS(), 2858
NDB$EPOCH_TRANS(), 2855, 2857
NDB$MAX(), 2855
NDB$MAX_DELETE_WIN(), 2855
NDB$OLD, 2855
NDB$OP_TYPE, 2860
NDB$ORIG_TRANSID, 2860
ndb-batch-size option
mysqld, 2558
ndb-blob-read-batch-bytes option
mysqld, 2558
ndb-blob-write-batch-bytes option
mysqld, 2559
ndb-cluster-connection-pool option
mysqld, 2558
ndb-connectstring option
mysqld, 2559
ndb_config, 2635
ndb-connectstring option (NDB Cluster programs), 2697
ndb-deferred-constraints option (NDB Cluster), 2560
ndb-distribution option (NDB Cluster), 2560
ndb-log-apply-status option
mysqld, 2560
ndb-log-empty-epochs option
mysqld, 2561
ndb-log-empty-update option
mysqld, 2561
ndb-log-exclusive-reads option
mysqld, 2561

ndb-log-fail-terminate option
  mysqld, 2562
ndb-log-orig option
  mysqld, 2562
ndb-log-transaction-id option
  mysqld, 2562
ndb-log-update-as-write (mysqld option), 2852
ndb-log-update-minimal option (NDB Cluster), 2563
ndb-mgmd-host option (NDB Cluster programs), 2697
ndb-mgmd-host option (NDB Cluster), 2563
ndb-nodegroup-map option
  ndb_restore, 2669
ndb-nodeid option
  mysqld, 2563
ndb-nodeid option (NDB Cluster programs), 2698
ndb-optimization-delay option
  mysqld, 2564
ndb-optimized-node-selection option (NDB Cluster), 2698
ndb-transid-mysql-connection-map option
  mysqld, 2564
ndb-wait-connected option
  mysqld, 2564
ndb-wait-setup option
  mysqld, 2565
ndbcluster option
  mysqld, 2557
NDBCLUSTER storage engine (see NDB Cluster)
ndbd, 2609, 2609
  -n option, 2614
  connect-delay option, 2611
  connect-retries option, 2611
  connect-retry-delay option, 2611
  initial option, 2612
  initial-start option, 2613
  install option, 2613
  nostart option, 2614
  nowait-nodes option, 2614
  remove option, 2615
  verbose option, 2615
ndbd (NDB Cluster)
  defined, 2382
    (see also data node (NDB Cluster))
ndbinfo database, 2772
  and query cache, 2774
  basic usage, 2775
  determining support for, 2772
ndbinfo_database system variable, 2582
ndbinfo_max_bytes system variable, 2582
ndbinfo_max_rows system variable, 2583
ndbinfo_offline system variable, 2583
ndbinfo_select_all, 2609, 2616
  -l option, 2617
  delay option, 2617
  loops option, 2617
ndbinfo_show_hidden system variable, 2583
ndbinfo_table_prefix system variable, 2583
ndbinfo_version system variable, 2584, 2584
ndbmtd, 2609, 2618

-n option, 2614
configuration, 2537, 2538
connect-delay option, 2611
connect-retries option, 2611
connect-retry-delay option, 2611
initial option, 2612
initial-start option, 2613
install option, 2613
MaxNoOfExecutionThreads, 2532
nostart option, 2614
nowait-nodes option, 2614
remove option, 2615
trace files, 2618, 2618
verbose option, 2615
Ndb_api_adaptive_send_deferred_count status variable, 2584
Ndb_api_adaptive_send_deferred_count_session status variable, 2584
Ndb_api_adaptive_send_deferred_count_slave status variable, 2584
Ndb_api_adaptive_send_forced_count status variable, 2584
Ndb_api_adaptive_send_forced_count_session status variable, 2584
Ndb_api_adaptive_send_forced_count_slave status variable, 2584
Ndb_api_adaptive_send_unforced_count status variable, 2584
Ndb_api_adaptive_send_unforced_count_slave session variable, 2585
Ndb_api_adaptive_send_unforced_count_slave status variable, 2585
Ndb_api_bytes_received_count status variable, 2586
Ndb_api_bytes_received_count_session status variable, 2585
Ndb_api_bytes_received_count_slave status variable, 2585
Ndb_api_bytes_sent_count status variable, 2585
Ndb_api_bytes_sent_count_session status variable, 2585
Ndb_api_bytes_sent_count_slave status variable, 2585
Ndb_api_event_bytes_count status variable, 2586
Ndb_api_event_bytes_count_injector status variable, 2586
Ndb_api_event_data_count status variable, 2586
Ndb_api_event_data_count_injector status variable, 2586
Ndb_api_event_nondata_count status variable, 2586
Ndb_api_event_nondata_count_injector status variable, 2586
Ndb_api_pk_op_count status variable, 2587
Ndb_api_pk_op_count_session status variable, 2586
Ndb_api_pk_op_count_slave status variable, 2587
Ndb_api_pruned_scan_count status variable, 2587
Ndb_api_pruned_scan_count_session status variable, 2587
Ndb_api_pruned_scan_count_slave status variable, 2587
Ndb_api_range_scan_count status variable, 2588
Ndb_api_range_scan_count_session status variable, 2588
Ndb_api_range_scan_count_slave status variable, 2588
Ndb_api_read_row_count status variable, 2588
Ndb_api_read_row_count_session status variable, 2588
Ndb_api_read_row_count_slave status variable, 2588
Ndb_api_scan_batch_count status variable, 2589
Ndb_api_scan_batch_count_session status variable, 2589
Ndb_api_scan_batch_count_slave status variable, 2589
Ndb_api_table_scan_count status variable, 2589
Ndb_api_table_scan_count_session status variable, 2589
Ndb_api_table_scan_count_slave status variable, 2589
Ndb_api_trans_abort_count status variable, 2590
Ndb_api_trans_abort_count_session status variable, 2590
Ndb_api_trans_abort_count_slave status variable, 2590
Ndb_api_trans_close_count status variable, 2591
Ndb_api_trans_close_count_session status variable, 2590

Ndb_api_trans_close_count_slave status variable, 2590
Ndb_api_trans_commit_count status variable, 2591
Ndb_api_trans_commit_count_session status variable, 2591
Ndb_api_trans_commit_count_slave status variable, 2591
Ndb_api_trans_local_read_row_count status variable, 2592
Ndb_api_trans_local_read_row_count_session status variable, 2591
Ndb_api_trans_local_read_row_count_slave status variable, 2591
Ndb_api_trans_start_count status variable, 2592
Ndb_api_trans_start_count_session status variable, 2592
Ndb_api_trans_start_count_slave status variable, 2592
Ndb_api_uk_op_count status variable, 2593
Ndb_api_uk_op_count_session status variable, 2592
Ndb_api_uk_op_count_slave status variable, 2592
Ndb_api_wait_exec_complete_count status variable, 2593
Ndb_api_wait_exec_complete_count_session status variable, 2593
Ndb_api_wait_exec_complete_count_slave status variable, 2593
Ndb_api_wait_meta_request_count status variable, 2594
Ndb_api_wait_meta_request_count_session status variable, 2593
Ndb_api_wait_meta_request_count_slave status variable, 2594
Ndb_api_wait_nanos_count status variable, 2594
Ndb_api_wait_nanos_count_session status variable, 2594
Ndb_api_wait_nanos_count_slave status variable, 2594
Ndb_api_wait_scan_result_count status variable, 2595
Ndb_api_wait_scan_result_count_session status variable, 2594
Ndb_api_wait_scan_result_count_slave status variable, 2595
ndb_apply_status table (NDB Cluster replication), 2835, 2836, 2840
  (see also NDB Cluster replication)
ndb_autoincrement_prefetch_sz system variable, 2567
ndb_binlog_index table
  system table, 694, 2834
ndb_binlog_index table (NDB Cluster replication), 2834, 2841
  (see also NDB Cluster replication)
ndb_blob_tool, 2609, 2628
  check-orphans option, 2629
  database option, 2629
  delete-orphans option, 2629
  dump-file option, 2629
  verbose option, 2629
ndb_cache_check_time system variable, 2567
ndb_clear_apply_status system variable, 2567
Ndb_cluster_node_id status variable, 2595
ndb_config, 2609, 2630
  config-file option, 2633
  configinfo option, 2632
  config_from_node option, 2633
  connections option, 2633
  diff-default option, 2633
  fields option, 2633
  host option, 2634
  mycnf option, 2634
  ndb-connectstring option, 2635
  nodeid option, 2634
  nodes option, 2634
  query option, 2635, 2635
  query-all option, 2635
  rows option, 2635
  system option, 2636
  type option, 2636

usage option, 2636
version option, 2636
xml option, 2636
Ndb_config_from_host status variable, 2595
Ndb_config_from_port status variable, 2595
Ndb_conflict_fn_epoch status variable, 2596
Ndb_conflict_fn_epoch2 status variable, 2596
Ndb_conflict_fn_epoch2_trans status variable, 2596
Ndb_conflict_fn_epoch_trans status variable, 2596
Ndb_conflict_fn_max status variable, 2595
Ndb_conflict_fn_max_del_win status variable, 2595
Ndb_conflict_fn_old status variable, 2595
Ndb_conflict_last_conflict_epoch status variable, 2596
Ndb_conflict_last_stable_epoch status variable, 2597
Ndb_conflict_reflected_op_discard_count status variable, 2596
Ndb_conflict_reflected_op_prepare_count status variable, 2596
Ndb_conflict_refresh_op_count status variable, 2597
Ndb_conflict_trans_conflict_commit_count status variable, 2597
Ndb_conflict_trans_detect_iter_count status variable, 2597
Ndb_conflict_trans_reject_count status variable, 2597
Ndb_conflict_trans_row_conflict_count status variable, 2597
Ndb_conflict_trans_row_reject_count status variable, 2597
ndb_cpcd, 2609, 2639
ndb_deferred_constraints system variable, 2568
ndb_delete_all, 2609, 2639
transactional option, 2639
ndb_desc, 2609, 2639
blob-info option, 2643
database option, 2643
extra-node-info option, 2643
extra-partition-info option, 2643
retries option, 2643
table option, 2644
unqualified option, 2644
ndb_distribution system variable, 2568
ndb_dist_priv.sql, 2759
ndb_drop_index, 2609, 2644
ndb_drop_table, 2609, 2645
Ndb_epoch_delete_delete_count status variable, 2598
ndb_error_reporter, 2609, 2645
connection-timeout option, 2646
dry-scp option, 2646
fs option, 2646
options, 2646
skip-nodegroup option, 2646
ndb_eventbuffer_free_percent system variable, 2568
ndb_eventbuffer_max_alloc system variable, 2569
Ndb_execute_count status variable, 2598
ndb_extra_logging system variable, 2569
ndb_force_send system variable, 2569
ndb_index_stat, 2609, 2647
-d option, 2649
database option, 2649
delete option, 2649
dump option, 2649
example, 2647
interpreting output, 2647
loops option, 2651

query option, 2649
sys-check option, 2651
sys-create option, 2650
sys-create-if-not-exist option, 2650
sys-create-if-not-valid option, 2650
sys-drop option, 2650
sys-skip-events option, 2651
sys-skip-tables option, 2651
update option, 2649
verbose option, 2651
ndb_index_stat_enable system variable, 2570
ndb_index_stat_option system variable, 2570
ndb_join_pushdown system variable, 2572
Ndb_last_commit_epoch_server status variable, 2598
Ndb_last_commit_epoch_session status variable, 2598
ndb_log_apply_status system variable, 2574
ndb_log_apply_status variable (NDB Cluster replication), 2840
ndb_log_bin system variable, 2574
ndb_log_binlog_index system variable, 2574
ndb_log_empty_epochs system variable, 2575
ndb_log_empty_update system variable, 2575
ndb_log_exclusive_reads (system variable), 2864
ndb_log_exclusive_reads system variable, 2575
ndb_log_orig system variable, 2575
ndb_log_transaction_id system variable, 2576
ndb_mgm, 2609, 2626 (see mgm)
-e option, 2627
connect-retries option, 2627
execute option, 2627
using with MySQL Cluster Manager, 2699
ndb_mgm (NDB Cluster management node client), 2430
ndb_mgmd, 2609 (see mgmd)
-d option, 2622
-f option, 2621
-P option, 2625
-v option, 2626
bind-address option, 2620
config-cache option, 2620
config-file option, 2621
configdir option, 2621
daemon option, 2622
initial option, 2622
install option, 2622
interactive option, 2623
log-name option, 2623
mycnf option, 2623
no-nodeid-checks option, 2623
nodaemon option, 2623
nowait-nodes option, 2623
print-full-config option, 2625
reload option, 2625
remove option, 2625
verbose option, 2626
ndb_mgmd (NDB Cluster process), 2619
ndb_mgmd (NDB Cluster)
defined, 2382
(see also management node (NDB Cluster))
ndb_move_data, 2609, 2652

abort-on-error option, 2652
character-sets-dir option, 2653
database option, 2653
drop-source option, 2653
error-insert option, 2653
exclude-missing-columns option, 2653
lossy-conversions option, 2653
promote-attributes option, 2653
staging-tries option, 2654
verbose option, 2654
Ndb_number_of_data_nodes status variable, 2598
ndb_optimized_node_selection system variable, 2576
ndb_print_backup_file, 2609, 2654
ndb_print_file, 2609, 2654
ndb_print_frag_file, 2609, 2655
ndb_print_schema_file, 2609, 2656
ndb_print_sys_file, 2609, 2656
Ndb_pruned_scan_count status variable, 2598
Ndb_pushed_queries_defined status variable, 2598
Ndb_pushed_queries_dropped status variable, 2598
Ndb_pushed_queries_executed status variable, 2598
Ndb_pushed_reads status variable, 2598
ndb_recv_thread_activation_threshold system variable, 2577
ndb_recv_thread_cpu_mask system variable, 2577
ndb_redo_log_reader, 2656
dump option, 2657
lap option, 2658
twiddle option, 2659
ndb_replication system table, 2853
ndb_report_thresh_binlog_epoch_slip system variable, 2577
ndb_report_thresh_binlog_mem_usage system variable, 2578
ndb_restore, 2659
and circular replication, 2849
and distributed privileges, 2761
append option, 2663
backup-path option, 2663
backupid option, 2664
connect option, 2664
disable-indexes option, 2664
dont-ignore-systab-0 option, 2664
errors, 2675
exclude-databases option, 2665
exclude-intermediate-sql-tables option, 2665
exclude-missing-columns option, 2665
exclude-missing-tables option, 2665
exclude-tables option, 2665
fields-enclosed-by option, 2666
fields-optionally-enclosed-by option, 2667
fields-terminated-by option, 2667
hex option, 2667
include-databases option, 2667
include-tables option, 2667
lines-terminated-by option, 2668
lossy-conversions option, 2668
ndb-nodegroup-map option, 2669
no-binlog option, 2669
no-restore-disk-objects option, 2669
no-upgrade option, 2669

nodeid option, 2669
parallelism option, 2669
preserve-trailing-spaces option, 2670
print option, 2670
print-data option, 2670
print-log option, 2671
print-meta option, 2671
progress-frequency option, 2671
promote-attributes option, 2671
rebuild-indexes option, 2672
restore-data option, 2672
restore-epoch option, 2673
restore-meta option, 2673
restore-privilege-tables option, 2673
rewrite-database option, 2673
skip-broken-objects option, 2674
skip-table-check option, 2674
skip-unknown-objects option, 2675
tab option, 2675
typical and required options, 2662
verbose option, 2675
Ndb_scan_count status variable, 2599
ndb_select_all, 2609, 2680
database option, 2681
delimiter option, 2682
descending option, 2681
disk option, 2682
gci option, 2682
gci64 option, 2682
header option, 2681
lock option, 2681
nodata option, 2682
order option, 2681
parallelism option, 2681
rowid option, 2682
tupscan option, 2682
useHexFormat option, 2681
ndb_select_count, 2609, 2683
ndb_setup.py, 2609, 2684
browser-start-page option, 2685
ca-certs-file option, 2685
cert-file option, 2685
debug-level option, 2685
help option, 2686
key-file option, 2686
no-browser option, 2686
port option, 2686
server-log-file option, 2686
server-name option, 2687
use-https option, 2687
ndb_show_foreign_key_mock_tables system variable, 2578
ndb_show_tables, 2609, 2687
database option, 2687
loops option, 2688
parsable option, 2688
show-temp-status option, 2688
type option, 2688
unqualified option, 2688

ndb_size.pl, 2609, 2688
ndb_size.pl script, 1304
ndb_slave_conflict_role system variable, 2578
ndb_table_no_logging system variable, 2579
ndb_table_temporary system variable, 2580
ndb_transid_mysql_connection_map
  INFORMATION_SCHEMA table, 3053
ndb_use_copying_alter_table system variable, 2580
ndb_use_exact_count system variable, 2580
ndb_use_transactions system variable, 2580
ndb_version system variable, 2581
ndb_version_string system variable, 2581
ndb_waiter, 2609, 2691
  no-contact option, 2692
  not-started option, 2692
  nowait-nodes option, 2692
  single-user option, 2692
  timeout option, 2692
  wait-nodes option, 2692
negative values, 1126
neighbor page, 3867
nested queries, 1637
Nested-Loop join algorithm, 976
nested-loop join algorithm, 979
.NET, 3866
net-buffer-length option
  mysql, 314
  mysqldump, 365
netmask notation
  in account names, 797
network ports
  and NDB Cluster, 2821
net_buffer_length system variable, 594
net_read_timeout system variable, 595
net_retry_count system variable, 595
net_write_timeout system variable, 595
new features, 9
  --innodb-read-only, 12
  condition handling, 17
  data types, 18
  host cache, 18
  InnoDB, 10
  innodb_compression_failure_threshold_pct, 12
  innodb_compression_level, 12
  innodb_compression_pad_pct_max, 12
  innodb_log_compressed_pages, 12
  innodb_status_output, 13
  innodb_status_output_locks, 13
  MySQL Enterprise Audit, 10
  MySQL Enterprise Encryption, 10
  MySQL Enterprise Firewall, 10
  NDB Cluster, 14
  OpenGIS, 18
  optimizer, 16
  partitioning, 13
  Performance Schema, 14
  replication and logging, 14
  security, 9

server defaults, 10
new features in NDB Cluster, 2388
new system variable, 596
newline (\n), 1124, 1609
next-key lock, 1872, 3867
InnoDB, 1886, 2000
NFS
InnoDB, 1891
nice option
mysqld_safe, 285
no matching rows, 3269
no-auto-rehash option
mysql, 314
no-autocommit option
mysqldump, 366
no-beep option
mysql, 315
mysqladmin, 337
no-binlog option
ndb_restore, 2669
no-browser option
ndb_setup.py, 2686
no-contact option
ndb_waiter, 2692
no-create-db option
mysqldump, 357
no-create-info option
mysqldump, 357
no-data option
mysqldump, 364
no-defaults option, 265
myisamchk, 395
mysql, 315
mysqladmin, 337
mysqlbinlog, 430
mysqlcheck, 345
mysqld, 534
mysqldump, 356
mysqld_multi, 289
mysqld_safe, 285
mysqlimport, 373
mysqlshow, 378
mysqlslap, 386
mysql_install_db, 296
mysql_plugin, 299
mysql_upgrade, 305
my_print_defaults, 454
NDB client programs, 2698
no-drop option
mysqlslap, 386
no-log option
mysqld_multi, 290
no-nodeid-checks option
ndb_mgmd, 2623
no-restore-disk-objects option
ndb_restore, 2669
no-set-names option
mysqldump, 358

no-symlinks option
  myisamchk, 398
no-tablespaces option
  mysqldump, 357
no-upgrade option
  ndb_restore, 2669
nodaemon option
  ndb_mgmd, 2623
nodata option
  ndb_select_all, 2682
node groups (NDB Cluster), 2384
node logs (NDB Cluster), 2717
NodeGroup, 2485
NodeId, 2478, 2484, 2550
nodeid option
  ndb_config, 2634
  ndb_restore, 2669
NodeId1, 2600, 2605
NodeId2, 2600, 2605
NodeIdServer, 2605
NODERESTART Events (NDB Cluster), 2722
nodes
  ndbinfo table, 2794
nodes option
  ndb_config, 2634
noindices option
  mysqlhotcopy, 446
non-locking read, 3867
non-repeatable read, 3867
nonblocking I/O, 3867
nondelimited strings, 1128
nondeterministic functions
  optimization, 998
  replication, 998
Nontransactional tables, 3269
NoOfDiskPagesToDiskAfterRestartACC (DEPRECATED), 2518
NoOfDiskPagesToDiskAfterRestartTUP (DEPRECATED), 2517
NoOfDiskPagesToDiskDuringRestartACC, 2518
NoOfDiskPagesToDiskDuringRestartTUP, 2518
NoOfFragmentLogFiles, 2498
NoOfFragmentLogParts, 2537
NoOfReplicas, 2485
nopager command
  mysql, 321
normalized, 3867
normalize_statement() MySQL Enterprise Firewall UDF, 921
NoSQL, 3868
nostart option
  ndbd, 2614
  ndbmtd, 2614
NOT
  logical, 1332
NOT BETWEEN, 1328
not equal (!=), 1327
not equal (<>), 1327
NOT EXISTS
  with subqueries, 1641
NOT IN, 1330

NOT LIKE, 1385
NOT NULL
  constraint, 47
NOT NULL constraint, 3868
NOT REGEXP, 1386
not-started option
  ndb_waiter, 2692
notee command
  mysql, 321
Not_flushed_delayed_rows status variable, 660
NOW(), 1357
NOWAIT (START BACKUP command), 2744
nowait-nodes option
  ndbd, 2614
  ndbmtd, 2614
  ndb_mgmd, 2623
  ndb_waiter, 2692
nowarning command
  mysql, 322
NO_AUTO_CREATE_USER SQL mode, 671
NO_AUTO_VALUE_ON_ZERO SQL mode, 671
NO_BACKSLASH_ESCAPES SQL mode, 671
NO_DIR_IN_CREATE SQL mode, 671
NO_ENGINE_SUBSTITUTION SQL mode, 671
NO_FIELD_OPTIONS SQL mode, 672
NO_KEY_OPTIONS SQL mode, 672
NO_TABLE_OPTIONS SQL mode, 672
NO_UNSIGNED_SUBTRACTION SQL mode, 672
NO_ZERO_DATE
  deprecated features, 19
NO_ZERO_DATE SQL mode, 673
NO_ZERO_IN_DATE
  deprecated features, 19
NO_ZERO_IN_DATE SQL mode, 673
NUL, 1124, 1609
NULL, 233, 3267, 3868
  ORDER BY, 992, 1625
  testing for null, 1327, 1328, 1330, 1331, 1336
NULL value, 233, 1131
  ORDER BY, 1132
NULL values
  and AUTO_INCREMENT columns, 3268
  and indexes, 1543
  and TIMESTAMP columns, 3268
  vs. empty values, 3267
NULL-complemented row, 980, 984
null-rejected condition, 984
NULLIF(), 1336
Numa, 2530
number-char-cols option
  mysqlslap, 386
number-int-cols option
  mysqlslap, 386
number-of-queries option
  mysqlslap, 386
numbers, 1126
NUMERIC data type, 1254
numeric data types, 1252

storage requirements, 1304
numeric literals
approximate-value, 1126, 1487
exact-value, 1126, 1487
numeric precision, 1251
numeric scale, 1252
numeric-dump-file option
resolve_stack_dump, 454
NumInteriorRings(), 1462
NumPoints(), 1460
NVARCHAR data type, 1276

O

objects
stored, 2939
objects_summary_global_by_type table
performance_schema, 3143
obtaining information about partitions, 2912
OCT(), 1375
OCTET_LENGTH(), 1375
ODBC, 3868
ODBC compatibility, 616, 1135, 1255, 1322, 1330, 1544, 1632
ODBC_INCLUDES= option
CMake, 171
ODBC_LIB_DIR option
CMake, 171
ODirect, 2506
OFF
plugin activation option, 720
off-page column, 3868
offset option
mysqlbinlog, 430
OGC (see Open Geospatial Consortium)
OLAP, 1476
old system variable, 596
old-style-user-limits option
mysqld, 534
old_alter_table system variable, 596
OLD_PASSWORD(), 1435
deprecated features, 19
old_passwords
deprecated features, 19
old_passwords system variable, 597
old_server option
mysqlaccess, 422
mysqlhotcopy, 446
OLTP, 3869
ON
plugin activation option, 720
ON DUPLICATE KEY
INSERT modifier, 1598
ON DUPLICATE KEY UPDATE, 1594
ON versus USING
joins, 1634
one-database option
mysql, 315
ONE_SHOT
removed features, 20

online, 3869
online DDL, 1946, 1947, 3869
  concurrency, 1955
  limitations, 1960
online location of manual, 2
online upgrades and downgrades (NDB Cluster), 2729
only-print option
  mysqlslap, 387
ONLY_FULL_GROUP_BY
  SQL mode, 1479
ONLY_FULL_GROUP_BY SQL mode, 673
OPEN, 1695
Open Geospatial Consortium, 1287
Open Source
  defined, 5
open tables, 334, 1032
open-files-limit option
  mysqlbinlog, 430
  mysqld_safe, 285
Opened_files status variable, 660
Opened_tables status variable, 660
Opened_table_definitions status variable, 660
OpenGIS, 1287
  new features, 18
opening
  tables, 1032
Opening master dump table
  thread state, 1120
Opening mysql.ndb_apply_status
  thread state, 1120
Opening table
  thread state, 1112
Opening tables
  thread state, 1112
opens, 334
OpenSSL, 164, 828
  compared to yaSSL, 840
OpenSSL versus yaSSL
  detecting, 841
Open_files status variable, 660
open_files_limit system variable, 598
Open_streams status variable, 660
Open_tables status variable, 660
Open_table_definitions status variable, 660
operating systems
  file-size limits, 1035
  supported, 60
operations
  arithmetic, 1338
operations_per_fragment
  ndbinfo table, 2796
operators, 1310
  arithmetic, 1428
  assignment, 1165, 1333
  bit, 1428
  cast, 1337, 1412
  logical, 1332
  precedence, 1325

string, 1368
string comparison, 1383
.OPT file, 3868
opt option
mysqldump, 365
optimistic, 3869
optimization, 962, 1044
Batched Key Access, 987, 988
benchmarking, 1104
BLOB types, 1031
Block Nested-Loop, 987, 988
character and string types, 1031
data change statements, 1015
data size, 1029
DELETE statements, 1016
derived tables, 1001
disk I/O, 1096
foreign keys, 1019
full table scans, 1000
full-text queries, 1020
indexes, 1017
INFORMATION_SCHEMA queries, 1010
InnoDB tables, 1039
INSERT statements, 1015
many tables, 1032
MEMORY storage engine, 1020
MEMORY tables, 1053
memory usage, 1101
Multi-Range Read, 985
MyISAM tables, 1048
nondeterministic functions, 998
numeric types, 1031
Performance Schema queries, 1012
PERFORMANCE_SCHEMA, 1106
primary keys, 1018
REPAIR TABLE statements, 1051
SELECT statements, 964
spatial queries, 1020
SQL statements, 964
subqueries, 1001
subquery, 1005
subquery materialization, 1003
tips, 1016
UPDATE statements, 1016
WHERE clauses, 965
optimization (NDB), 973, 2572
optimizations, 971
LIMIT clause, 997
row constructors, 999
optimize option
mysqlcheck, 345
OPTIMIZE TABLE
and partitioning, 2911
OPTIMIZE TABLE statement, 1744
optimizer, 3869
and replication, 2362
controlling, 1069
new features, 16

query plan evaluation, 1069
switchable optimizations, 1069
optimizer statistics
for InnoDB tables, 1909
Optimizer Statistics, 1915
optimizer_prune_level system variable, 598
optimizer_search_depth system variable, 599
optimizer_switch system variable, 599, 1069
OPTIMIZER_TRACE
INFORMATION_SCHEMA table, 2990
OPTIMIZER_TRACE option
CMake, 174
optimizer_trace system variable, 600
optimizer_trace_features system variable, 600
optimizer_trace_limit system variable, 601
optimizer_trace_max_mem_size system variable, 601
optimizer_trace_offset system variable, 601
optimizing
DISTINCT, 996
filesort, 992
GROUP BY, 994
LEFT JOIN, 982
ORDER BY, 990
outer joins, 982
RIGHT JOIN, 982
tables, 958
thread state, 1113
OPTION
removed features, 20
option, 3869
option file, 3870
option files, 259, 823
.my.cnf, 259, 261, 278, 736, 768, 823
.mylogin.cnf, 259, 414
C:\my.cnf, 736
escape sequences, 262
my.cnf, 2348
option prefix
--disable, 266
--enable, 266
--loose, 266
--maximum, 266
--skip, 266
options
boolean, 266
CMake, 165
command-line
mysql, 307
mysqladmin, 334
embedded server, 3182
libmysqld, 3182
myisamchk, 394
mysqld, 462
provided by MySQL, 219
replication, 2348
OR, 246, 971
bitwise, 1428
logical, 1333

OR Index Merge optimization, 971
Oracle compatibility, 42, 1475, 1508, 1815
ORACLE SQL mode, 675
ORD(), 1375
ORDER BY, 230, 1512, 1624
  maximum sort length, 1625
  NULL, 992, 1625
  NULL value, 1132
ORDER BY optimization, 990
order option
  ndb_select_all, 2681
order-by-primary option
  mysqldump, 366
orphan stored objects, 2961
OS X
  installation, 116
Out of resources error
  and partitioned tables, 2926
OUT parameter
  condition handling, 1719
out-dir option
  comp_err, 293
out-file option
  comp_err, 293
out-of-range handling, 1258
outer joins
  optimizing, 982
OUTFILE, 1629
overflow handling, 1258
overflow page, 3870
Overlaps(), 1469
OverloadLimit, 2601, 2606
overview, 1

P

packages
  list of, 54
PAD SPACE collations, 1204, 1279
PAD_CHAR_TO_FULL_LENGTH SQL mode, 673
page, 3870
page cleaner, 3870
page size, 3870
  InnoDB, 1858
pager command
  mysql, 322
pager option
  mysql, 315
PAM
  pluggable authentication, 853
.par file, 3870
parallel-recover option
  myisamchk, 399
parallelism option
  ndb_restore, 2669
parameters
  server, 462
PARAMETERS
  INFORMATION_SCHEMA table, 2990

parent table, 3871
parentheses (and), 1325
parsable option
  ndb_show_tables, 2688
partial backup, 3871
partial index, 3871
partial trust, 3871
partial updates
  and replication, 2364
PARTITION, 2867
PARTITION BY LIST COLUMNS, 2879
PARTITION BY RANGE COLUMNS, 2879
partition management, 2899
partition option
  mysqld, 535
partition pruning, 2914
partitioning, 2867
  advantages, 2871
  and dates, 2872
  and foreign keys, 2927
  and FULLTEXT indexes, 2927
  and key cache, 2927
  and query cache, 2926
  and replication, 2363, 2365
  and SQL mode, 2365, 2923
  and subqueries, 2928
  and temporary tables, 2927, 2930
  by hash, 2887
  by key, 2890
  by linear hash, 2889
  by linear key, 2891
  by list, 2877
  by range, 2873
  COLUMNS, 2879
  concepts, 2869
  data type of partitioning key, 2927
  enabling, 2867
  functions allowed in partitioning expressions, 2934
  index prefixes, 2928
  keys, 2871
  limitations, 2923
  new features, 13
  operators not permitted in partitioning expressions, 2923
  operators supported in partitioning expressions, 2923
  optimization, 2913, 2914
  partitioning expression, 2871
  resources, 2868
  storage engines (limitations), 2933
  subpartitioning, 2929
  support, 2867
  support in NDB Cluster, 2402
  tables, 2867
  types, 2871
Partitioning
  maximum number of partitions, 2926
partitioning information statements, 2912
partitioning keys and primary keys, 2930
partitioning keys and unique keys, 2930

partitions
  adding and dropping, 2899
  analyzing, 2911
  checking, 2911
  managing, 2899
  modifying, 2899
  optimizing, 2911
  repairing, 2911
  splitting and merging, 2899
  truncating, 2899
PARTITIONS
  INFORMATION_SCHEMA table, 2991
partitions (NDB Cluster), 2384
password
  root user, 189
password encryption
  reversibility of, 1436
password option, 271
  mysql, 315
  mysqlaccess, 422
  mysqladmin, 338
  mysqlbinlog, 430
  mysqlcheck, 346
  mysqldump, 354
  mysqld_multi, 290
  mysqlhotcopy, 446
  mysqlimport, 373
  mysqlshow, 379
  mysqlslap, 387
  mysql_convert_table_format, 447
  mysql_setpermission, 449
  mysql_upgrade, 305
password policy, 877
password validation, 877
PASSWORD(), 799, 806, 1435, 3255
passwords
  administrator guidelines, 769
  expiration, 808
  for the InnoDB memcached interface, 2089
  for users, 782
  forgotten, 3258
  hashing, 770
  logging, 769
  lost, 3258
  resetting, 808, 3258
  security, 767, 781
  setting, 806, 1728, 1737
  user guidelines, 768
PATH environment variable, 108, 114, 187, 257, 457
path name separators
  Windows, 262
pattern matching, 234, 1386
performance, 962
  benchmarks, 1106
  disk I/O, 1096
  estimating, 1068
Performance Schema, 2055, 3063, 3871
  event filtering, 3077

memory use, 3073
new features, 14
Performance Schema queries
optimization, 1012
performance-schema-consumer-events-stages-current option
mysqld, 3159
performance-schema-consumer-events-stages-history option
mysqld, 3159
performance-schema-consumer-events-stages-history-long option
mysqld, 3159
performance-schema-consumer-events-statements-current option
mysqld, 3159
performance-schema-consumer-events-statements-history option
mysqld, 3159
performance-schema-consumer-events-statements-history-long option
mysqld, 3159
performance-schema-consumer-events-waits-current option
mysqld, 3159
performance-schema-consumer-events-waits-history option
mysqld, 3159
performance-schema-consumer-events-waits-history-long option
mysqld, 3159
performance-schema-consumer-global-instrumentation option
mysqld, 3159
performance-schema-consumer-statements-digest option
mysqld, 3159
performance-schema-consumer-thread-instrumentation option
mysqld, 3159
performance-schema-consumer-xxx option
mysqld, 3158
performance-schema-instrument option
mysqld, 3158
performance_schema
accounts table, 3132
cond_instances table, 3110
events_stages_current table, 3121
events_stages_history table, 3122
events_stages_history_long table, 3123
events_stages_summary_by_account_by_event_name table, 3140
events_stages_summary_by_host_by_event_name table, 3140
events_stages_summary_by_thread_by_event_name table, 3140
events_stages_summary_by_user_by_event_name table, 3140
events_stages_summary_global_by_event_name table, 3140
events_statements_current table, 3127
events_statements_history table, 3130
events_statements_history_long table, 3130
events_statements_summary_by_account_by_event_name table, 3141
events_statements_summary_by_digest table, 3141
events_statements_summary_by_host_by_event_name table, 3141
events_statements_summary_by_thread_by_event_name table, 3141
events_statements_summary_by_user_by_event_name table, 3141
events_statements_summary_global_by_event_name table, 3141
events_waits_current table, 3116
events_waits_history table, 3118
events_waits_history_long table, 3119
events_waits_summary_by_account_by_event_name table, 3138
events_waits_summary_by_host_by_event_name table, 3138
events_waits_summary_by_instance table, 3138

events_waits_summary_by_thread_by_event_name table, 3138
events_waits_summary_by_user_by_event_name table, 3138
events_waits_summary_global_by_event_name table, 3138
file_instances table, 3110
file_summary_by_event_name table, 3143
file_summary_by_instance table, 3143
hosts table, 3133
host_cache table, 678, 3149
mutex_instances table, 3111
objects_summary_global_by_type table, 3143
performance_timers table, 3152
rwlock_instances table, 3112
session_account_connect_attrs table, 3136
session_connect_attrs table, 3136
setup_actors table, 3105
setup_consumers table, 3106
setup_instruments table, 3107
setup_objects table, 3108
setup_timers table, 3109
socket_instances table, 3113
socket_summary_by_event_name table, 3148
socket_summary_by_instance table, 3148
table_io_waits_summary_by_index_usage table, 3146
table_io_waits_summary_by_table table, 3145
table_lock_waits_summary_by_table table, 3146
thread table, 3152
users table, 3133
performance_schema database, 3063
restrictions, 3174
TRUNCATE TABLE, 3102, 3174
PERFORMANCE_SCHEMA storage engine, 3063
performance_schema system variable, 3160
Performance_schema_accounts_lost status variable, 3170
performance_schema_accounts_size system variable, 3160
Performance_schema_cond_classes_lost status variable, 3170
Performance_schema_cond_instances_lost status variable, 3170
performance_schema_digests_size system variable, 3160
Performance_schema_digest_lost status variable, 3170
performance_schema_events_stages_history_long_size system variable, 3161
performance_schema_events_stages_history_size system variable, 3161
performance_schema_events_statements_history_long_size system variable, 3161
performance_schema_events_statements_history_size system variable, 3162
performance_schema_events_waits_history_long_size system variable, 3162
performance_schema_events_waits_history_size system variable, 3162
Performance_schema_file_classes_lost status variable, 3170
Performance_schema_file_handles_lost status variable, 3170
Performance_schema_file_instances_lost status variable, 3170
Performance_schema_hosts_lost status variable, 3170
performance_schema_hosts_size system variable, 3162
Performance_schema_locker_lost status variable, 3170
performance_schema_max_cond_classes system variable, 3163
performance_schema_max_cond_instances system variable, 3163
performance_schema_max_digest_length system variable, 3163
performance_schema_max_file_classes system variable, 3164
performance_schema_max_file_handles system variable, 3164
performance_schema_max_file_instances system variable, 3164
performance_schema_max_mutex_classes system variable, 3164
performance_schema_max_mutex_instances system variable, 3165

performance_schema_max_rwlock_classes system variable, 3165
performance_schema_max_rwlock_instances system variable, 3165
performance_schema_max_socket_classes system variable, 3165
performance_schema_max_socket_instances system variable, 3166
performance_schema_max_stage_classes system variable, 3166
performance_schema_max_statement_classes system variable, 3166
performance_schema_max_table_handles system variable, 3167
performance_schema_max_table_instances system variable, 3167
performance_schema_max_thread_classes system variable, 3167
performance_schema_max_thread_instances system variable, 3168
Performance_schema_mutex_classes_lost status variable, 3170
Performance_schema_mutex_instances_lost status variable, 3170
Performance_schema_rwlock_classes_lost status variable, 3171
Performance_schema_rwlock_instances_lost status variable, 3171
Performance_schema_session_connect_attrs_lost status variable, 3171
performance_schema_session_connect_attrs_size system variable, 3168
performance_schema_setup_actors_size system variable, 3168
performance_schema_setup_objects_size system variable, 3169
Performance_schema_socket_classes_lost status variable, 3171
Performance_schema_socket_instances_lost status variable, 3171
Performance_schema_stage_classes_lost status variable, 3171
Performance_schema_statement_classes_lost status variable, 3171
Performance_schema_table_handles_lost status variable, 3171
Performance_schema_table_instances_lost status variable, 3171
Performance_schema_thread_classes_lost status variable, 3171
Performance_schema_thread_instances_lost status variable, 3171
Performance_schema_users_lost status variable, 3171
performance_schema_users_size system variable, 3169
performance_timers table
  performance_schema, 3152
PERIOD_ADD(), 1358
PERIOD_DIFF(), 1358
Perl, 3871
  installing, 215
  installing on Windows, 216
Perl API, 3185, 3871
Perl DBI/DBD
  installation problems, 217
permission checks
  effect on speed, 1016
perror, 256, 455
  help option, 455
  ndb option, 455
  silent option, 455
  verbose option, 455
  version option, 455
persistent statistics, 3871
pessimistic, 3871
phantom, 3872
phantom rows, 1886
phone book collation, German, 1178, 1221, 1221
PHP, 3872
PHP API, 3872
physical, 3872
physical backup, 3872
PI(), 1344
pid-file option
  mysql.server, 288

mysqld_safe, 285
pid_file system variable, 601
Ping
thread command, 1109
pipe option, 271
mysql, 316, 346
mysqladmin, 338
mysqldump, 354
mysqlimport, 373
mysqlshow, 379
mysqlslap, 387
mysql_upgrade, 305
PIPES_AS_CONCAT SQL mode, 674
PITR, 3872
plan option
mysqlaccess, 422
plan stability, 3872
platforms
supported, 60
pluggable authentication
PAM, 853
restrictions, 813
Windows, 863
plugin
audit_log, 883
plugin activation options
FORCE, 720
FORCE_PLUS_PERMANENT, 720
OFF, 720
ON, 720
plugin API, 716
plugin installing
audit_log, 885
CONNECTION_CONTROL, 871
CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS, 871
MySQL Enterprise Firewall plugins, 909
MySQL Enterprise Thread Pool, 723
validate_password, 878
plugin option prefix
mysqld, 536
plugin services, 728
plugin table
system table, 693
plugin-dir option, 271
mysql, 316
mysqladmin, 338
mysqlbinlog, 430
mysqlcheck, 346
mysqldump, 354
mysqld_safe, 285
mysqlimport, 373
mysqlshow, 379
mysqlslap, 387
mysql_plugin, 299
mysql_upgrade, 305
plugin-ini option
mysql_plugin, 299
plugin-load option

mysqld, 535
plugin-load-add option
mysqld, 536
plugindir option
mysql_config, 452
plugins
activating, 717
installing, 717, 1750
security, 842
server, 716
uninstalling, 717, 1751
PLUGINS
INFORMATION_SCHEMA table, 2995
plugin_dir system variable, 602
POINT data type, 1289
Point(), 1456
point-in-time recovery, 951, 3872
InnoDB, 2075
using NDB Cluster replication, 2847
PointFromText(), 1453
PointFromWKB(), 1455
PointN(), 1461
PolyFromText(), 1453
PolyFromWKB(), 1455
POLYGON data type, 1289
Polygon(), 1457
PolygonFromText(), 1453
PolygonFromWKB(), 1455
port, 3873
port option, 271
mysql, 316
mysqladmin, 338
mysqlbinlog, 430
mysqlcheck, 346
mysqld, 537
mysqldump, 354
mysqld_safe, 285
mysqlhotcopy, 446
mysqlimport, 373
mysqlshow, 379
mysqlslap, 387
mysql_config, 452
mysql_convert_table_format, 447
mysql_setpermission, 449
mysql_upgrade, 305
ndb_setup.py, 2686
port system variable, 602
port-open-timeout option
mysqld, 537
portability, 964
types, 1308
PortNumber, 2478
PortNumber (OBSOLETE / REMOVED), 2601
PortNumberStats, 2481
ports, 174, 185, 276, 430, 457, 735, 766, 822, 841, 3113, 3246
POSITION(), 1376
post-filtering
Performance Schema, 3077

post-query option
  mysqlslap, 387
post-system option
  mysqlslap, 387
PostgreSQL compatibility, 43
POSTGRESQL SQL mode, 675
postinstall
  multiple servers, 729
postinstallation
  setup and testing, 181
POW(), 1344
POWER(), 1344
pre-4.1 passwords
  deprecated features, 19
pre-filtering
  Performance Schema, 3077
pre-query option
  mysqlslap, 387
pre-system option
  mysqlslap, 387
precedence
  command options, 257
  operator, 1325
precision
  arithmetic, 1486
  fractional seconds, 1252, 1261
  numeric, 1251
precision math, 1486
preload_buffer_size system variable, 602
Prepare
  thread command, 1109
PREPARE, 1682, 1685
  XA transactions, 1667
prepared backup, 3873
prepared statement, 3873
prepared statements, 1682, 1685, 1686, 1686
  repreparation, 1086
Prepared_stmt_count status variable, 660
preparing
  thread state, 1113
preparing for alter table
  thread state, 1113
preserve-trailing-spaces option
  ndb_restore, 2670
preview option
  mysqlaccess, 422
primary key, 3873
  constraint, 46
  deleting, 1509
PRIMARY KEY, 1509, 1545
primary keys
  and partitioning keys, 2930
print command
  mysql, 322
print option
  ndb_restore, 2670
print-data option
  ndb_restore, 2670

print-defaults option, 265
  myisamchk, 395
  mysql, 316
  mysqladmin, 338
  mysqlbinlog, 430
  mysqlcheck, 346
  mysqld, 537
  mysqldump, 356
  mysqlimport, 373
  mysqlshow, 379
  mysqlslap, 388
  mysql_plugin, 299
  mysql_upgrade, 305
  NDB client programs, 2698
print-full-config option
  ndb_mgmd, 2625
print-log option
  ndb_restore, 2671
print-meta option
  ndb_restore, 2671
privilege
  changes, 806
privilege checks
  effect on speed, 1016
privilege information
  location, 790
privilege system, 781
privileges
  access, 781
  adding, 803
  ALL, 785
  ALL PRIVILEGES, 785
  ALTER, 785
  ALTER ROUTINE, 785
  and replication, 2362
  checking, 805
  CREATE, 786
  CREATE ROUTINE, 786
  CREATE TABLESPACE, 786
  CREATE TEMPORARY TABLES, 786
  CREATE USER, 786
  CREATE VIEW, 786
  default, 189
  DEFINER, 1775, 2959
  DELETE, 786
  deleting, 1724
  display, 1775
  DROP, 786
  dropping, 1724
  EVENT, 786
  EXECUTE, 786
  FILE, 786
  GRANT OPTION, 787
  granting, 1725
  INDEX, 787
  INSERT, 787
  INVOKER, 1775, 2959
  LOCK TABLES, 787

PROCESS, 787
PROXY, 787
REFERENCES, 787
RELOAD, 787
REPLICATION CLIENT, 788
REPLICATION SLAVE, 788
revoking, 805, 1736
SELECT, 788
SHOW DATABASES, 788
SHOW VIEW, 788
SHUTDOWN, 789
SQL SECURITY, 2959
stored objects, 2959
SUPER, 789
TEMPORARY tables, 786, 1562, 1731
TRIGGER, 790
UPDATE, 790
USAGE, 790
problems
access denied errors, 3245
common errors, 3244
compiling MySQL server, 179
DATE columns, 3266
date values, 1264
installing on Solaris, 153
installing Perl, 217
lost connection errors, 3249
reporting, 2, 35
starting the server, 185
table locking, 1090
time zone, 3264
proc table
system table, 693
PROCEDURE, 1627
PROCEDURE ANALYSE(), 1032
procedures
stored, 2941
process, 3873
process management (NDB Cluster), 2609
PROCESS privilege, 787
processes
display, 1782
monitoring, 1106
Processing events
thread state, 1120
Processing events from schema table
thread state, 1120
Processlist
thread command, 1109
PROCESSLIST, 1782
INFORMATION_SCHEMA table, 2996
possible inconsistency with INFORMATION_SCHEMA tables, 2035
procs_priv table
system table, 693, 791
profiling
deprecated features, 19
PROFILING
INFORMATION_SCHEMA table, 2997

profiling system variable, 603
profiling_history_size
  deprecated features, 19
profiling_history_size system variable, 603
program options (NDB Cluster), 2694
program variables
  setting, 266
program-development utilities, 255
programs
  administrative, 254
  client, 253
  stored, 1687, 2939
  utility, 254
progress-frequency option
  ndb_restore, 2671
promote-attributes option
  ndb_move_data, 2653
  ndb_restore, 2671
prompt command
  mysql, 322
prompt option
  mysql, 316
prompts
  meanings, 222
pronunciation
  MySQL, 5
protocol option, 271
  mysql, 316
  mysqladmin, 338
  mysqlbinlog, 430
  mysqlcheck, 346
  mysqldump, 354
  mysqlimport, 373
  mysqlshow, 379
  mysqlslap, 388
  mysql_upgrade, 305
protocol_version system variable, 603
proxies_priv
  grant table, 816
proxies_priv table
  system table, 189, 693, 791
proximity search, 1397
PROXY privilege, 787
proxy users, 814
  conflict with anonymous users, 818
  default proxy user, 817
  PAM authentication, 859
  PROXY privilege, 816
  system variables, 820
  Windows authentication, 866
proxy_user system variable, 603
pseudo-record, 3873
pseudo_slave_mode system variable, 603
pseudo_thread_id system variable, 603
Pthreads, 3873
purge, 1907, 3874
PURGE BINARY LOGS, 1670
purge buffering, 3874

purge configuration, 1907
purge lag, 3874
PURGE MASTER LOGS, 1670
purge scheduling, 1907
purge thread, 3874
Purging old relay logs
  thread state, 1113
pushdown joins (NDB), 2572
Python, 3180, 3874
  third-party driver, 3186
Python API, 3874

Q

Qcache_free_blocks status variable, 661
Qcache_free_memory status variable, 661
Qcache_hits status variable, 661
Qcache_inserts status variable, 661
Qcache_lowmem_prunes status variable, 661
Qcache_not_cached status variable, 661
Qcache_queries_in_cache status variable, 661
Qcache_total_blocks status variable, 661
QUARTER(), 1358
queries
  entering, 220
  estimating performance, 1068
  examples, 242
  speed of, 964
Queries status variable, 661
Query
  thread command, 1109
query, 3874
Query Cache, 1079
query cache
  and ndbinfo database tables, 2774
  and partitioned tables, 2926
  thread states, 1117
query end
  thread state, 1113
query execution plan, 3874
query expansion, 1402
query option
  mysqlslap, 388
  ndb_config, 2635, 2635
  ndb_index_stat, 2649
query-all option
  ndb_config, 2635
query_alloc_block_size system variable, 604
query_cache_limit system variable, 604
query_cache_min_res_unit system variable, 604
query_cache_size system variable, 605
query_cache_type system variable, 605
query_cache_wlock_invalidate system variable, 606
query_prealloc_size system variable, 606
questions, 334
Questions status variable, 661
Queueing master event to the relay log
  thread state, 1118
QUICK

DELETE modifier, 1590
quick option
myisamchk, 399
mysql, 316
mysqlcheck, 346
mysqldump, 365
quiesce, 3874
quiet option
mysqlhotcopy, 446
Quit
thread command, 1109
quit command
mysql, 322
QUIT command (NDB Cluster), 2702
quotation marks
in strings, 1125
QUOTE(), 1125, 1376
quote-names option
mysqldump, 361
quoting, 1125
account names, 796
column alias, 1133, 3268
host names in account names, 796
schema objects, 1727
user names in account names, 796
quoting binary data, 1125
quoting of identifiers, 1132

R

R-tree, 3875
RADIANS(), 1344
RAID, 3875
RAND(), 1345
random dive, 3875
random-passwords option
mysql_install_db, 296
RANDOM_BYTES(), 1436
rand_seed1 system variable, 607
rand_seed2 system variable, 607
range join type
optimizer, 1059
range partitioning, 2873, 2879
range partitions
adding and dropping, 2900
managing, 2900
range_alloc_block_size system variable, 607
raw backup, 3875
raw option
mysql, 316
mysqlbinlog, 431
raw partitions, 1865
RC
MySQL releases, 60
re-creating
grant tables, 184
READ COMMITTED, 3875
implementation in NDB Cluster, 2404
transaction isolation level, 1877

read conflict detection and resolution
  in NDB Cluster Replication, 2862
read phenomena, 3875
READ UNCOMMITTED, 3875
  transaction isolation level, 1879
read view, 3876
read-ahead, 3876
  linear, 1898
  random, 1898
read-from-remote-master option
  mysqlbinlog, 431
read-from-remote-server option
  mysqlbinlog, 431
read-only option
  myisamchk, 398
read-only transaction, 3876
Reading event from the relay log
  thread state, 1119
Reading from net
  thread state, 1113
Reading master dump table data
  thread state, 1120
read_buffer_size myisamchk variable, 395
read_buffer_size system variable, 607
read_firewall_users() MySQL Enterprise Firewall UDF, 921
read_firewall_whitelist() MySQL Enterprise Firewall UDF, 921
read_only system variable, 608
read_rnd_buffer_size system variable, 609
REAL data type, 1255
RealtimeScheduler, 2530
REAL_AS_FLOAT SQL mode, 674
rebuild-indexes option
  ndb_restore, 2672
Rebuilding the index on master dump table
  thread state, 1120
ReceiveBufferMemory, 2601
reconfiguring, 179
reconnect option
  mysql, 317
Reconnecting after a failed binlog dump request
  thread state, 1118
Reconnecting after a failed master event read
  thread state, 1118
reconnection
  automatic, 3153
record lock, 3876
record-level locks
  InnoDB, 1886, 2000
record_log_pos option
  mysqlhotcopy, 446
RECOVER
  XA transactions, 1667
recover option
  myisamchk, 399
recovery
  from crash, 955
  incremental, 951
  InnoDB, 2075

point in time, 951
redo, 3876
redo log, 1870, 1870, 3876
redo log archiving, 3876
RedoBuffer, 2521
RedoOverCommitCounter
data nodes, 2546
RedoOverCommitLimit
data nodes, 2546
reducing
data size, 1029
redundant row format, 1939, 3877
ref join type
optimizer, 1058
references, 1509
REFERENCES privilege, 787
referential integrity, 1820, 3877
REFERENTIAL_CONSTRAINTS
INFORMATION_SCHEMA table, 2998
Refresh
thread command, 1109
ref_or_null, 989
ref_or_null join type
optimizer, 1059
REGEXP, 1386
REGEXP operator, 1386
regexp option
mysqlhotcopy, 446
mysql_find_rows, 448
Register Slave
thread command, 1110
Registering slave on master
thread state, 1118
regular expression syntax, 1386
rehash command
mysql, 322
relational, 3877
relational databases
defined, 5
relative option
mysqladmin, 338
relay log (replication), 2316
relay-log-purge option
mysqld, 2263
relay-log-space-limit option
mysqld, 2263
relay_log system variable, 2272
relay_log_basename system variable, 2273
relay_log_index system variable, 2273
relay_log_info_file system variable, 2273
relay_log_info_repository system variable, 2274, 2316
relay_log_purge system variable, 2274
relay_log_recovery system variable, 2274
relay_log_space_limit system variable, 2275
release numbers, 60
RELEASE SAVEPOINT, 1657
releases
GA, 60

milestone, 60
naming scheme, 60
RC, 60
RELEASE_LOCK(), 1439
relevance, 3877
relnotes option
mysqlaccess, 422
reload option
ndb_mgmd, 2625
RELOAD privilege, 787
remote administration (NDB Cluster)
and security issues, 2821
remove option
mysqld, 537
MySQLInstallerConsole, 103
ndbd, 2615
ndbmtd, 2615
ndb_mgmd, 2625
removed features, 19
--init-rpl-role, 20
--log, 20
--one-thread, 20
--rpl-recovery-rank, 20
--safe-mode, 20
--skip-thread-priority, 20
--table-cache, 20
DEFAULT, 20
engine_condition_pushdown, 20
FLUSH MASTER, 20
FLUSH SLAVE, 20
have_csv, 20
have_innodb, 20
have_ndbcluster, 20
have_partitioning, 20
log, 20
log_slow_queries, 20
max_long_data_size, 20
ONE_SHOT, 20
OPTION, 20
rpl_recovery_rank, 20
Rpl_status, 20
SHOW AUTHORS, 20
SHOW CONTRIBUTORS, 20
SHOW ENGINE INNODB MUTEX, 20
SLAVE START, 20
SLAVE STOP, 20
sql_big_tables, 20
sql_low_priority_updates, 20
sql_max_join_size, 20
Removing duplicates
thread state, 1113
removing tmp table
thread state, 1113
rename
thread state, 1113
rename database, 1585
rename result table
thread state, 1113

RENAME TABLE, 1585
RENAME USER statement, 1735
renaming user accounts, 1735
Reopen tables
  thread state, 1113
repair
  tables, 340
Repair by sorting
  thread state, 1113
Repair done
  thread state, 1113
repair option
  mysqlcheck, 346
repair options
  myisamchk, 398
REPAIR TABLE
  and partitioning, 2911
  and replication, 2363
REPAIR TABLE statement, 1747
  and replication, 1747
  options, 1748
  output, 1749
  partitioning support, 1748
  storage engine support, 1748
Repair with keycache
  thread state, 1113
repairing
  tables, 956
REPEAT, 1693
  labels, 1687
REPEAT(), 1376
REPEATABLE READ, 3877
  transaction isolation level, 1877
repertoire, 3877
  character set, 1178, 1210
  string, 1178
replace, 256
REPLACE, 1620
replace option
  mysqldump, 357
  mysqlimport, 374
replace utility, 455
REPLACE(), 1376
replica, 3878
replicas
  statements, 1672
replicate-do-db option
  mysqld, 2264
replicate-do-table option
  mysqld, 2266
replicate-ignore-db option
  mysqld, 2265
replicate-ignore-table option
  mysqld, 2266
replicate-rewrite-db option
  mysqld, 2267
replicate-same-server-id option
  mysqld, 2267

replicate-wild-do-table option
mysqld, 2267
replicate-wild-ignore-table option
mysqld, 2268
replication, 2221, 3878
and AUTO_INCREMENT, 2349
and character sets, 2350
and CHECKSUM TABLE statement, 2350
and CREATE ... IF NOT EXISTS, 2350
and CREATE TABLE ... SELECT, 2351
and DATA DIRECTORY, 2356
and DROP ... IF EXISTS, 2356
and errors on replica, 2364
and floating-point values, 2356
and FLUSH, 2356
and fractional seconds, 2357
and functions, 2357
and INDEX DIRECTORY, 2356
and invoked features, 2359
and LAST_INSERT_ID(), 2349
and LIMIT, 2361
and LOAD DATA, 2361
and max_allowed_packet, 2361
and MEMORY tables, 2361
and mysql (system) database, 2362
and partial updates, 2364
and partitioned tables, 2363
and partitioning, 2365
and privileges, 2362
and query optimizer, 2362
and REPAIR TABLE statement, 1747, 2363
and reserved words, 2363
and scheduled events, 2359, 2360
and SQL mode, 2365
and stored routines, 2359
and temporary tables, 2365
and time zones, 2366
and TIMESTAMP, 2349
and transactions, 2366, 2366
and triggers, 2359, 2368
and TRUNCATE TABLE, 2368
and variables, 2368
and views, 2370
attribute demotion, 2353
attribute promotion, 2353
BLACKHOLE, 2350
circular, 2828
crashes, 2363
delayed, 2347
in NDB Cluster, 2824
(see also NDB Cluster replication)
nondeterministic functions, 998
relay log, 2316
replication metadata repositories, 2316
row-based vs statement-based, 2234
safe and unsafe statements, 2238
semisynchronous, 2343
shutdown and restart, 2363, 2365

statements incompatible with STATEMENT format, 2234
timeouts, 2366
unexpected halt, 2332
with differing tables on source and replica, 2352
with ZFS, 2161
replication and logging
new features, 14
REPLICATION CLIENT privilege, 788
replication filtering options
and case sensitivity, 2323
replication formats
compared, 2234
replication implementation, 2313
replication limitations, 2348
replication metadata repositories, 2316
replication options, 2348
replication replica
thread states, 1118, 1119, 1120
REPLICATION SLAVE privilege, 788
replication source
thread states, 1118
replication source servers
statements, 1670
replication, asynchronous (see NDB Cluster replication)
REPORT command (NDB Cluster), 2701
reporting
bugs, 2, 35
errors, 35
problems, 2
report_host system variable, 2275
report_password system variable, 2275
report_port system variable, 2276
report_user system variable, 2276
REPRODUCIBLE_BUILD option
CMake, 174
Requesting binlog dump
thread state, 1118
REQUIRE option
GRANT statement, 1732
reschedule
thread state, 1117
reserved words, 1143
and replication, 2363
ReservedSendBufferMemory, 2546
RESET MASTER, 1671
RESET MASTER statement, 1813
RESET SLAVE, 1677
RESET SLAVE ALL, 1677
RESET SLAVE statement, 1813
Reset stmt
thread command, 1110
reset-replica.pl
NDB Cluster replication, 2844
resetmaster option
mysqlhotcopy, 446
resetslave option
mysqlhotcopy, 446
RESIGNAL, 1704

resolveip, 256, 456
help option, 456
silent option, 456
version option, 457
resolve_stack_dump, 255, 454
help option, 454
numeric-dump-file option, 454
symbols-file option, 454
version option, 454
resource limits
user accounts, 588, 820, 1734
resources
ndbinfo table, 2799
RESTART command (NDB Cluster), 2701
restarting
the server, 188
RestartOnErrorInsert, 2506
RestartSubscriberConnectTimeout, 2519
restart_info
ndbinfo table, 2800
restore, 3878
restore-data option
ndb_restore, 2672
restore-epoch option
ndb_restore, 2673
restore-meta option
ndb_restore, 2673
restore-privilege-tables option
ndb_restore, 2673
restoring backups
in NDB Cluster, 2659
restoring from backup
in NDB Cluster replication, 2842
restoring NDB backups
between NDB release series, 2675
to earlier versions of NDB, 2675
to later versions of NDB, 2676
restrictions
character sets, 1228
events, 2968
InnoDB, 2120
performance_schema database, 3174
pluggable authentication, 813
server-side cursors, 1696
signals, 1719
stored routines, 2968
subqueries, 1648
triggers, 2968
views, 2972
XA transactions, 1669
result-file option
mysqlbinlog, 431
mysqldump, 361
retries option
ndb_desc, 2643
retrieving
data from tables, 227
RETURN, 1693

return (\r), 1124, 1609
REVERSE(), 1376
REVOKE statement, 803, 1736
revoking
  privileges, 1736
rewrite-database option
  ndb_restore, 2673
rhost option
  mysqlaccess, 422
RIGHT JOIN, 982, 1630
RIGHT OUTER JOIN, 1630
RIGHT(), 1376
RLIKE, 1386
ROLLBACK, 1653
  XA transactions, 1667
rollback, 3878
rollback option
  mysqlaccess, 422
rollback segment, 1868, 3878
ROLLBACK TO SAVEPOINT, 1657
Rolling back
  thread state, 1114
rolling restart (NDB Cluster), 2729
ROLLUP, 1476
root password, 189
root user, 766
  password resetting, 3258
ROUND(), 1346
rounding, 1486
rounding errors, 1254
routines
  stored, 2939, 2941
ROUTINES
  INFORMATION_SCHEMA table, 2999
routines option
  mysqldump, 364
ROW, 1640
row, 3878
row constructors, 1640
  optimizations, 999
row format, 3878
row lock, 3879
row size
  maximum, 1037
row subqueries, 1640
row-based replication, 3879
  advantages, 2235
  disadvantages, 2236
row-level locking, 1087, 3879
rowid option
  ndb_select_all, 2682
rows
  counting, 236
  deleting, 3269
  matching problems, 3269
  selecting, 228
  sorting, 230
rows option

mysql_find_rows, 448
ndb_config, 2635
ROW_COUNT(), 1447
ROW_FORMAT
COMPACT, 1037, 1940
COMPRESSED, 1037, 1922, 1942
DYNAMIC, 1037, 1941
REDUNDANT, 1037, 1939
RPAD(), 1376
rpl_recovery_rank
removed features, 20
Rpl_semi_sync_master_clients status variable, 661
rpl_semi_sync_master_enabled system variable, 2259
Rpl_semi_sync_master_net_avg_wait_time status variable, 661
Rpl_semi_sync_master_net_waits status variable, 662
Rpl_semi_sync_master_net_wait_time status variable, 662
Rpl_semi_sync_master_no_times status variable, 662
Rpl_semi_sync_master_no_tx status variable, 662
Rpl_semi_sync_master_status status variable, 662
Rpl_semi_sync_master_timefunc_failures status variable, 662
rpl_semi_sync_master_timeout system variable, 2259
rpl_semi_sync_master_trace_level system variable, 2260
Rpl_semi_sync_master_tx_avg_wait_time status variable, 662
Rpl_semi_sync_master_tx_waits status variable, 662
Rpl_semi_sync_master_tx_wait_time status variable, 662
rpl_semi_sync_master_wait_no_slave system variable, 2260
Rpl_semi_sync_master_wait_pos_backtraverse status variable, 663
Rpl_semi_sync_master_wait_sessions status variable, 663
Rpl_semi_sync_master_yes_tx status variable, 663
rpl_semi_sync_slave_enabled system variable, 2276
Rpl_semi_sync_slave_status status variable, 663
rpl_semi_sync_slave_trace_level system variable, 2277
Rpl_status
removed features, 20
rpl_stop_slave_timeout system variable, 2277
RPM file, 131, 134, 137
rpm option
mysql_install_db, 297
RPM Package Manager, 137
Rsa_public_key status variable, 663
RTRIM(), 1377
Ruby, 3879
Ruby API, 3186, 3879
running
ANSI mode, 40
batch mode, 241
multiple servers, 729
queries, 220
running CMake after prior invocation, 161, 179
rw-lock, 3879
rwlock_instances table
performance_schema, 3112

S

safe statement (replication)
defined, 2238
safe-recover option
myisamchk, 399

safe-updates mode, 329
safe-updates option
  mysql, 317, 329
safe-user-create option
  mysqld, 537
Sakila, 8
same value wins (conflict resolution), 2855
sandbox mode
  for expired-password accounts, 808
SASL, 2089
SAVEPOINT, 1657
savepoint, 3880
Saving state
  thread state, 1114
scalability, 3880
Scalable Coherent Interface (NDB Cluster) (OBSOLETE), 2609
scale
  arithmetic, 1486
  numeric, 1252
scale out, 3880
scale up, 3880
SchedulerExecutionTimer, 2531
SchedulerResponsiveness, 2531
SchedulerSpinTimer, 2531
schema, 3880
  altering, 1496
  creating, 1521
  deleting, 1580
SCHEMA Events (NDB Cluster), 2724
SCHEMA(), 1448
SCHEMATA
  INFORMATION_SCHEMA table, 3001
SCHEMA_PRIVILEGES
  INFORMATION_SCHEMA table, 3002
SCI (NDB Cluster) (OBSOLETE), 2609
script files, 241
scripts, 282, 289
  mysql_install_db, 183
  SQL, 307
search index, 3880
searching
  and case sensitivity, 3265
  full-text, 1393
  MySQL Web pages, 35
  two keys, 246
Searching rows for update
  thread state, 1114
SECOND(), 1359
secondary index, 3881
  InnoDB, 1857
secure connections, 828
  command options, 272
secure-auth option, 272
  mysql, 317
  mysqladmin, 339
  mysqlbinlog, 431
  mysqlcheck, 347
  mysqldump, 355

mysqlimport, 374
mysqlshow, 379
mysqlslap, 388
secure_auth
deprecated features, 19
secure_auth system variable, 609
secure_file_priv system variable, 610
securing an NDB Cluster, 2823
security
against attackers, 775
and malicious SQL statements, 2822
and NDB utilities, 2824
for the InnoDB memcached interface, 2089
new features, 9
plugins, 842
security system, 781
SEC_TO_TIME(), 1359
segment, 3881
SELECT
INTO, 1628
LIMIT, 1622
optimizing, 1053, 1814
Query Cache, 1079
SELECT INTO TABLE, 43
SELECT privilege, 788
select-limit option
mysql, 317
selecting
databases, 224
selectivity, 3881
Select_full_join status variable, 663
Select_full_range_join status variable, 663
Select_range status variable, 663
Select_range_check status variable, 663
Select_scan status variable, 663
SELinux, 930
error log file context, 933
file context, 932
mode, 931
MySQL data directory context, 932
MySQL feature TCP port context, 934
MySQL Server policies, 932
mysqld TCP port context, 934
PID file context, 933
secure_file_priv directory context, 933
status, 931
TCP port context, 933
troubleshooting, 934
Unix domain file context, 933
semi-consistent read, 3881
InnoDB, 2000
semijoins, 1002
semisynchronous replication, 2343
administrative interface, 2344
configuration, 2345
installation, 2345
monitoring, 2347
SendBufferMemory, 2601

Sending binlog event to slave
  thread state, 1118
sending cached result to client
  thread state, 1117
SendSignalId, 2602, 2606
SEQUENCE, 247
sequence emulation, 1447
sequences, 247
SERIAL, 1252, 1254
SERIAL DEFAULT VALUE, 1302
SERIALIZABLE, 3881
  transaction isolation level, 1879
server, 3881
  connecting, 219, 276
  debugging, 736
  disconnecting, 219
  logs, 694
  restart, 188
  shutdown, 188
  signal handling, 458
  starting, 182
  starting and stopping, 193
  starting problems, 185
server administration, 331
server configuration, 462
server connections
  command options, 270
server defaults
  new features, 10
server plugins, 716
server variables, 1800 (see system variables)
server-id option
  mysqlbinlog, 432
server-id-bits option
  mysqlbinlog, 432
  mysqld, 2565
server-log-file option
  ndb_setup.py, 2686
server-name option
  ndb_setup.py, 2687
server-public-key-path option, 273
  mysql, 317
server-side cursors
  restrictions, 1696
server-side prepared statement, 3882
ServerPort, 2484
servers
  multiple, 729
servers table
  system table, 694
server_id system variable, 2249
server_id_bits system variable, 2581
server_operations
  ndbinfo table, 2803
server_transactions
  ndbinfo table, 2805
server_uuid system variable
  mysqld, 2249

service-startup-timeout option
  mysql.server, 289
services
  for plugins, 728
servlet, 3882
SESSION
  SET statement, 1752
session variables
  and replication, 2368
session_account_connect_attrs table
  performance_schema, 3136
session_connect_attrs table
  performance_schema, 3136
SESSION_STATUS
  INFORMATION_SCHEMA table, 2988
SESSION_USER(), 1448
SESSION_VARIABLES
  INFORMATION_SCHEMA table, 2988
SET
  CHARACTER SET, 1191
  NAMES, 1191
  ONE_SHOT, 1755
  size, 1307
SET CHARACTER SET statement, 1756
SET CHARSET statement, 1756
SET data type, 1277, 1285
SET GLOBAL sql_slave_skip_counter, 1678
SET GLOBAL statement, 639
SET NAMES, 1197
SET NAMES statement, 1756
Set option
  thread command, 1110
SET PASSWORD statement, 806, 1737
SET SESSION statement, 639
SET sql_log_bin, 1672
SET statement
  assignment operator, 1334
  CHARACTER SET, 1756
  CHARSET, 1756
  NAMES, 1756
  variable assignment, 1752
SET TRANSACTION, 1663
set-auto-increment[option
  myisamchk, 400
set-charset option
  mysqlbinlog, 432
  mysqldump, 358
set-collation option
  myisamchk, 399
set-gtid-purged option
  mysqldump, 360
setting
  passwords, 806
setting passwords, 1737
setting program variables, 266
setup
  postinstallation, 181
  thread state, 1114

setup.bat
  NDB Cluster (Windows), 2684
setup_actors table
  performance_schema, 3105
setup_consumers table
  performance_schema, 3106
setup_instruments table
  performance_schema, 3107
setup_objects table
  performance_schema, 3108
setup_timers table
  performance_schema, 3109
set_firewall_mode() MySQL Enterprise Firewall UDF, 921
SHA(), 1437
SHA1(), 1437
SHA2(), 1437
sha256_password authentication plugin, 848
sha256_password_private_key_path system variable, 611
sha256_password_public_key_path system variable, 611
shared lock, 1872, 3882
shared memory transporter (see NDB Cluster)
shared tablespace, 3882
shared-memory-base-name option, 272
  mysql, 318
  mysqladmin, 339
  mysqlbinlog, 432
  mysqlcheck, 347
  mysqldump, 366
  mysqlimport, 374
  mysqlshow, 380
  mysqlslap, 388
  mysql_upgrade, 306
SharedGlobalMemory, 2541
shared_memory system variable, 612
shared_memory_base_name system variable, 612
sharp checkpoint, 3882
shell syntax, 4
ShmKey, 2606
ShmSize, 2606
short-form option
  mysqlbinlog, 432
SHOW
  in NDB Cluster management client, 2451
SHOW AUTHORS, 1757
  removed features, 20
SHOW AUTHORS statement, 1756
SHOW BINARY LOGS statement, 1756, 1757
SHOW BINLOG EVENTS statement, 1756, 1758
SHOW CHARACTER SET statement, 1756, 1759
SHOW COLLATION statement, 1756, 1759
SHOW COLUMNS statement, 1756, 1760
SHOW command (NDB Cluster), 2700
SHOW CONTRIBUTORS, 1762
  removed features, 20
SHOW CONTRIBUTORS statement, 1756
SHOW CREATE DATABASE statement, 1756, 1762
SHOW CREATE EVENT statement, 1756
SHOW CREATE FUNCTION statement, 1756, 1763

SHOW CREATE PROCEDURE statement, 1756, 1763
SHOW CREATE SCHEMA statement, 1756, 1762
SHOW CREATE TABLE statement, 1756, 1764
SHOW CREATE TRIGGER statement, 1756, 1765
SHOW CREATE VIEW statement, 1756, 1765
SHOW DATABASES privilege, 788
SHOW DATABASES statement, 1756, 1766
SHOW ENGINE
  and NDB Cluster, 2812
SHOW ENGINE INNODB MUTEX
  removed features, 20
SHOW ENGINE INNODB STATUS
  and innodb_use_sys_malloc, 1902
SHOW ENGINE INNODB STATUS statement, 1766
SHOW ENGINE NDB STATUS, 1766, 2812
SHOW ENGINE NDBCLUSTER STATUS, 1766, 2812
SHOW ENGINE statement, 1756, 1766
SHOW ENGINES
  and NDB Cluster, 2812
SHOW ENGINES statement, 1756, 1770
SHOW ERRORS statement, 1756, 1772
SHOW EVENTS statement, 1756, 1772
SHOW extensions, 3060
SHOW FIELDS statement, 1756, 1760
SHOW FUNCTION CODE statement, 1756, 1775
SHOW FUNCTION STATUS statement, 1756, 1775
SHOW GRANTS statement, 1756, 1775
SHOW INDEX statement, 1756, 1776
SHOW KEYS statement, 1756, 1776
SHOW MASTER LOGS statement, 1756, 1757
SHOW MASTER STATUS statement, 1756, 1777
SHOW OPEN TABLES statement, 1756, 1778
show option
  my_print_defaults, 454
SHOW PLUGINS statement, 1756, 1778
SHOW PRIVILEGES statement, 1756, 1779
SHOW PROCEDURE CODE statement, 1756, 1780
SHOW PROCEDURE STATUS statement, 1756, 1781
SHOW PROCESSLIST statement, 1756, 1782
SHOW PROFILE statement, 1756, 1784
SHOW PROFILES statement, 1756, 1784, 1786
SHOW RELAYLOG EVENTS statement, 1756, 1786
SHOW SCHEDULER STATUS, 2952
SHOW SCHEMAS statement, 1766
SHOW SLAVE HOSTS statement, 1756, 1787
SHOW SLAVE STATUS statement, 1756, 1788
SHOW STATUS
  and NDB Cluster, 2815
SHOW STATUS statement, 1756, 1794
SHOW STORAGE ENGINES statement, 1770
SHOW TABLE STATUS statement, 1756, 1795
SHOW TABLES statement, 1756, 1798
SHOW TRIGGERS statement, 1756, 1799
SHOW VARIABLES
  and NDB Cluster, 2813
SHOW VARIABLES statement, 1756, 1800
SHOW VIEW privilege, 788
SHOW WARNINGS statement, 1756, 1802

SHOW with WHERE, 2976, 3060
show-slave-auth-info option
  mysqld, 2256
show-table-type option
  mysqlshow, 380
show-temp-status option
  ndb_show_tables, 2688
show-warnings option
  mysql, 318
showing
  database information, 375
show_old_temporals system variable, 612
shutdown, 3882
  server, 690
Shutdown
  thread command, 1110
SHUTDOWN command (NDB Cluster), 2702
SHUTDOWN privilege, 789
shutdown-timeout option
  mysqladmin, 339
shutting down
  the server, 188
Shutting down
  thread state, 1121
SIGHUP signal
  log maintenance, 715
  server response, 458, 1806
SIGINT signal
  client response, 460
  mysql client, 318
  server response, 458, 739
sigint-ignore option
  mysql, 318
SIGN(), 1347
SIGNAL, 1708
signal handling, 458
signals
  client response, 460
  restrictions, 1719
  server response, 458
SigNum, 2607
SIGPIPE signal
  client response, 460
SIGTERM signal
  server response, 458
silent column changes, 1572
silent option
  myisamchk, 395
  myisampack, 409
  mysql, 318
  mysqladmin, 339
  mysqlcheck, 347
  mysqld_multi, 290
  mysqlimport, 374
  mysqlslap, 388
  perror, 455
  resolveip, 456
simplified_binlog_gtid_recovery, 2310

SIN(), 1347
single quote (\'), 1124
single user mode (NDB Cluster), 2702, 2731
  and ndb_restore, 2660
single-transaction option
  mysqldump, 366
single-user option
  ndb_waiter, 2692
SINGLEUSER Events (NDB Cluster), 2724
size of tables, 1035
sizes
  display, 1251
--skip option prefix, 266
skip-broken-objects option
  ndb_restore, 2674
skip-column-names option
  mysql, 318
skip-comments option
  mysqldump, 358
skip-database option
  mysqlcheck, 347
skip-grant-tables option
  mysqld, 538
skip-gtids option
  mysqlbinlog, 432
skip-host-cache option
  mysqld, 539
skip-innodb option
  mysqld, 539, 1967
skip-kill-mysqld option
  mysqld_safe, 285
skip-line-numbers option
  mysql, 318
skip-name-resolve option
  mysql_install_db, 297
skip-ndbcluster option
  mysqld, 2566
skip-new option
  mysqld, 539
skip-nodegroup option
  ndb_error_reporter, 2646
skip-opt option
  mysqldump, 365
skip-partition option
  mysqld, 539
skip-show-database option
  mysqld, 539
skip-slave-start option
  mysqld, 2268
skip-ssl option, 273
  mysqld, 541
skip-stack-trace option
  mysqld, 540
skip-symbolic-links option
  mysqld, 542
skip-syslog option
  mysqld_safe, 285
skip-table-check option

ndb_restore, 2674
skip-unknown-objects option
ndb_restore, 2675
skip-use-db option
mysql_find_rows, 448
skip_external_locking system variable, 613
skip_name_resolve system variable, 613
skip_networking system variable, 614
skip_show_database system variable, 614
Slave has read all relay log; waiting for more updates
thread state, 1119
SLAVE START
removed features, 20
SLAVE STOP
removed features, 20
slave-skip-errors option
mysqld, 2269
slave-sql-verify-checksum option
mysqld, 2269
slave_allow_batching, 2839
slave_allow_batching system variable, 2581
slave_checkpoint_group system variable, 2277
slave_checkpoint_period system variable, 2278
slave_compressed_protocol system variable, 2279
slave_exec_mode system variable, 2279
Slave_heartbeat_period status variable, 664
Slave_last_heartbeat status variable, 664
slave_load_tmpdir system variable, 2280
slave_master_info table
system table, 694
slave_max_allowed_packet system variable, 2280
slave_net_timeout system variable, 2281
Slave_open_temp_tables status variable, 664
slave_parallel_workers system variable, 2281
slave_pending_jobs_size_max system variable, 2282
Slave_received_heartbeats status variable, 664
slave_relay_log_info table
system table, 694
Slave_retried_transactions status variable, 664
Slave_rows_last_search_algorithm_used status variable, 664
slave_rows_search_algorithms system variable, 2282
Slave_running status variable, 664
slave_skip_errors system variable, 2283
slave_sql_verify_checksum system variable, 2284
slave_transaction_retries system variable, 2284
slave_type_conversions system variable, 2285
slave_worker_info table
system table, 694
Sleep
thread command, 1110
sleep option
mysqladmin, 339
SLEEP(), 1484
slow queries, 334
slow query log, 712, 3882
slow shutdown, 3882
slow-start-timeout option
mysqld, 540

Slow_launch_threads status variable, 664
slow_launch_time system variable, 614
slow_log table
  system table, 693
Slow_queries status variable, 664
slow_query_log system variable, 614
slow_query_log_file system variable, 615
SMALLINT data type, 1253
snapshot, 3882
SNAPSHOTEND (START BACKUP command), 2744
SNAPSHOTSTART (START BACKUP command), 2744
socket option, 272
  mysql, 318
  mysqladmin, 339
  mysqlbinlog, 432
  mysqlcheck, 347
  mysqld, 540
  mysqldump, 355
  mysqld_safe, 285
  mysqlhotcopy, 446
  mysqlimport, 374
  mysqlshow, 380
  mysqlslap, 388
  mysql_config, 452
  mysql_convert_table_format, 447
  mysql_setpermission, 450
  mysql_upgrade, 306
socket system variable, 615
socket_instances table
  performance_schema, 3113
socket_summary_by_event_name table
  performance_schema, 3148
socket_summary_by_instance table
  performance_schema, 3148
Solaris
  installation, 153
Solaris installation problems, 153
Solaris troubleshooting, 180
Solaris x86_64 issues, 1044
SOME, 1639
sort buffer, 3883
sort-index option
  myisamchk, 400
sort-records option
  myisamchk, 400
sort-recover option
  myisamchk, 399
sorting
  data, 230
  grant tables, 800, 801
  table rows, 230
Sorting for group
  thread state, 1114
Sorting for order
  thread state, 1114
Sorting index
  thread state, 1114
Sorting result

thread state, 1114
sort_buffer_size myisamchk variable, 395
sort_buffer_size system variable, 615
sort_key_blocks myisamchk variable, 395
Sort_merge_passes status variable, 664
Sort_range status variable, 664
Sort_rows status variable, 664
Sort_scan status variable, 664
SOUNDEX(), 1377
SOUNDS LIKE, 1377
source, 3883
source (mysql client command), 242, 327
source command
mysql, 322
source distribution
installing, 156
space ID, 3883
SPACE(), 1377
sparse file, 3883
spassword option
mysqlaccess, 422
spatial data types, 1287
storage requirements, 1307
spatial extensions in MySQL, 1287
spatial functions, 1449
spatial queries
optimization, 1020
speed
increasing with replication, 2221
inserting, 1015
of queries, 964
spin, 3883
sporadic-binlog-dump-fail option
mysqld, 2292
Spring, 3883
sp_reload_firewall_rules() MySQL Enterprise Firewall stored procedure, 919
sp_set_firewall_mode() MySQL Enterprise Firewall stored procedure, 920
SQL, 3883
defined, 5
SQL mode, 668
ALLOW_INVALID_DATES, 670
and partitioning, 2365, 2923
and replication, 2365
ANSI, 669, 674
ANSI_QUOTES, 670
DB2, 674
deprecated features, 19
ERROR_FOR_DIVISION_BY_ZERO, 670
HIGH_NOT_PRECEDENCE, 670
IGNORE_SPACE, 670
MAXDB, 674
MSSQL, 675
MYSQL323, 675
MYSQL40, 675
NO_AUTO_CREATE_USER, 671
NO_AUTO_VALUE_ON_ZERO, 671
NO_BACKSLASH_ESCAPES, 671
NO_DIR_IN_CREATE, 671

NO_ENGINE_SUBSTITUTION, 671
NO_FIELD_OPTIONS, 672
NO_KEY_OPTIONS, 672
NO_TABLE_OPTIONS, 672
NO_UNSIGNED_SUBTRACTION, 672
NO_ZERO_DATE, 673
NO_ZERO_IN_DATE, 673
ONLY_FULL_GROUP_BY, 673, 1479
ORACLE, 675
PAD_CHAR_TO_FULL_LENGTH, 673
PIPES_AS_CONCAT, 674
POSTGRESQL, 675
REAL_AS_FLOAT, 674
strict, 669
STRICT_ALL_TABLES, 674
STRICT_TRANS_TABLES, 669, 674
TRADITIONAL, 669, 675
SQL node (NDB Cluster)
defined, 2382
SQL nodes (NDB Cluster), 2747
SQL scripts, 307
SQL SECURITY
effect on privileges, 2959
SQL statements
replicas, 1672
replication source servers, 1670
SQL statements relating to NDB Cluster, 2812
SQL-92
extensions to, 39
sql-mode option
mysqld, 540
SQLState, 3883
sql_auto_is_null system variable, 616
SQL_BIG_RESULT
SELECT modifier, 1628
sql_big_selects system variable, 616
sql_big_tables
removed features, 20
SQL_BUFFER_RESULT
SELECT modifier, 1628
sql_buffer_result system variable, 617
SQL_CACHE, 1082
SELECT modifier, 1628
SQL_CALC_FOUND_ROWS, 997
SELECT modifier, 1628
sql_log_bin system variable, 2304
sql_log_off system variable, 617
sql_low_priority_updates
removed features, 20
sql_max_join_size
removed features, 20
sql_mode system variable, 617
sql_notes system variable, 618
SQL_NO_CACHE, 1083
SELECT modifier, 1628
sql_quote_show_create system variable, 619
sql_safe_updates system variable, 329, 619
sql_select_limit system variable, 329, 619

sql_slave_skip_counter, 1678
sql_slave_skip_counter system variable, 2285
SQL_SMALL_RESULT
  SELECT modifier, 1628
SQL_THREAD_WAIT_AFTER_GTIDS(), 1471
sql_warnings system variable, 620
SQRT(), 1347
square brackets, 1252
srcdir option
  mysql_install_db, 297
SRID values
  handling by spatial functions, 1452
SRID(), 1458
SSD, 1921, 3883
SSH, 775, 841
SSL, 828, 3884
  command options, 272
  establishing connections, 829
  OpenSSL compared to yaSSL, 840
  X.509 Basics, 828
SSL library
  configuring, 164
ssl option, 273
  mysqld, 541
SSL options
  mysql, 318
  mysqladmin, 339
  mysqlbinlog, 433
  mysqlcheck, 347
  mysqldump, 355
  mysqlimport, 374
  mysqlshow, 380
  mysqlslap, 389
  mysql_upgrade, 306
SSL related options
  GRANT statement, 1732
ssl-ca option, 274
ssl-capath option, 274
ssl-cert option, 274
ssl-cipher option, 274
ssl-crl option, 275
ssl-crlpath option, 275
ssl-key option, 275
ssl-mode option, 275
ssl-verify-server-cert option, 276
Ssl_accepts status variable, 665
Ssl_accept_renegotiates status variable, 665
ssl_ca system variable, 620
Ssl_callback_cache_hits status variable, 665
ssl_capath system variable, 620
ssl_cert system variable, 620
Ssl_cipher status variable, 665
ssl_cipher system variable, 620
Ssl_cipher_list status variable, 665
Ssl_client_connects status variable, 665
Ssl_connect_renegotiates status variable, 665
ssl_crl system variable, 621
ssl_crlpath system variable, 621

Ssl_ctx_verify_depth status variable, 665
Ssl_ctx_verify_mode status variable, 665
Ssl_default_timeout status variable, 665
Ssl_finished_accepts status variable, 665
Ssl_finished_connects status variable, 665
ssl_key system variable, 621
Ssl_server_not_after status variable, 665
Ssl_server_not_before status variable, 665
Ssl_sessions_reused status variable, 666
Ssl_session_cache_hits status variable, 666
Ssl_session_cache_misses status variable, 666
Ssl_session_cache_mode status variable, 666
Ssl_session_cache_overflows status variable, 666
Ssl_session_cache_size status variable, 666
Ssl_session_cache_timeouts status variable, 666
Ssl_used_session_cache_entries status variable, 666
Ssl_verify_depth status variable, 666
Ssl_verify_mode status variable, 666
Ssl_version status variable, 666
staging-tries option
  ndb_move_data, 2654
standalone option
  mysqld, 542
Standard Monitor, 2060, 2064, 2067
Standard SQL
  differences from, 43, 1735
  extensions to, 39, 41
standards compatibility, 39
START
  XA transactions, 1667
START BACKUP
  NOWAIT, 2744
  SNAPSHOTEND, 2744
  SNAPSHOTSTART, 2744
  syntax, 2744
  WAIT COMPLETED, 2744
  WAIT STARTED, 2744
START command (NDB Cluster), 2700
START SLAVE, 1678
START TRANSACTION, 1653
start-datetime option
  mysqlbinlog, 433
start-position option
  mysqlbinlog, 433
StartConnectBackoffMaxTime, 2556
StartFailRetryDelay, 2547
StartFailureTimeout, 2508
starting
  comments, 45
  mysqld, 778
  the server, 182
  the server automatically, 193
Starting many servers, 729
StartNoNodeGroupTimeout, 2509
StartPartialTimeout, 2508
StartPartitionedTimeout, 2508
StartPoint(), 1462
startup, 3884

STARTUP Events (NDB Cluster), 2721
startup options
  default, 259
startup parameters, 462
  mysql, 307
  mysqladmin, 334
  tuning, 1096
StartupStatusReportFrequency, 2524
start_row option
  mysql_find_rows, 448
state-file option
  comp_err, 293
statement interceptor, 3884
statement termination
  Control+C, 307, 318
statement-based replication, 3884
  advantages, 2234
  disadvantages, 2234
  unsafe statements, 2234
statements
  compound, 1687
  CREATE USER, 803
  DROP USER, 803
  GRANT, 803
  replicas, 1672
  replication source servers, 1670
  REVOKE, 803
Statistics
  thread command, 1110
statistics, 3884
  thread state, 1114
STATISTICS
  INFORMATION_SCHEMA table, 3003
STATISTICS Events (NDB Cluster), 2723
stats option
  myisam_ftdump, 391
stats_method myisamchk variable, 395
status
  tables, 1795
status command
  mysql, 322
  results, 334
STATUS command (NDB Cluster), 2701
status option
  MySQLInstallerConsole, 103
  mysqlshow, 380
status variable
  Aborted_clients, 651
  Aborted_connects, 651
  Audit_log_current_size, 907
  Audit_log_events, 907
  Audit_log_events_filtered, 907
  Audit_log_events_lost, 907
  Audit_log_events_written, 907
  Audit_log_event_max_drop_size, 907
  Audit_log_total_size, 907
  Audit_log_write_waits, 907
  Binlog_cache_disk_use, 651

Binlog_cache_use, 651
Binlog_stmt_cache_disk_use, 651
Binlog_stmt_cache_use, 651
Bytes_received, 651
Bytes_sent, 651
Compression, 652
Connections, 653
Connection_control_delay_generated, 877
Connection_errors_accept, 652
Connection_errors_internal, 652
Connection_errors_max_connections, 652
Connection_errors_peer_address, 652
Connection_errors_select, 653
Connection_errors_tcpwrap, 653
Created_tmp_disk_tables, 653
Created_tmp_files, 653
Created_tmp_tables, 653
Delayed_errors, 653
Delayed_insert_threads, 653
Delayed_writes, 653
Firewall_access_denied, 923
Firewall_access_granted, 923
Firewall_access_suspicious, 923
Firewall_cached_entries, 923
Flush_commands, 654
Handler_commit, 654
Handler_delete, 654
Handler_discover, 2584
Handler_external_lock, 654
Handler_mrr_init, 654
Handler_prepare, 654
Handler_read_first, 654
Handler_read_key, 654
Handler_read_last, 654
Handler_read_next, 654
Handler_read_prev, 654
Handler_read_rnd, 655
Handler_read_rnd_next, 655
Handler_rollback, 655
Handler_savepoint, 655
Handler_savepoint_rollback, 655
Handler_update, 655
Handler_write, 655
Innodb_available_undo_logs, 655
Innodb_buffer_pool_bytes_data, 655
Innodb_buffer_pool_bytes_dirty, 656
Innodb_buffer_pool_dump_status, 655
Innodb_buffer_pool_load_status, 655
Innodb_buffer_pool_pages_data, 656
Innodb_buffer_pool_pages_dirty, 656
Innodb_buffer_pool_pages_flushed, 656
Innodb_buffer_pool_pages_free, 656
Innodb_buffer_pool_pages_latched, 656
Innodb_buffer_pool_pages_misc, 656
Innodb_buffer_pool_pages_total, 656
Innodb_buffer_pool_reads, 657
Innodb_buffer_pool_read_ahead, 656
Innodb_buffer_pool_read_ahead_evicted, 656

Innodb_buffer_pool_read_ahead_rnd, 656
Innodb_buffer_pool_read_requests, 656
Innodb_buffer_pool_wait_free, 657
Innodb_buffer_pool_write_requests, 657
Innodb_data_fsyncs, 657
Innodb_data_pending_fsyncs, 657
Innodb_data_pending_reads, 657
Innodb_data_pending_writes, 657
Innodb_data_read, 657
Innodb_data_reads, 657
Innodb_data_writes, 657
Innodb_data_written, 657
Innodb_dblwr_pages_written, 657
Innodb_dblwr_writes, 657
Innodb_have_atomic_builtins, 657
Innodb_log_waits, 658
Innodb_log_writes, 658
Innodb_log_write_requests, 658
Innodb_num_open_files, 658
Innodb_os_log_fsyncs, 658
Innodb_os_log_pending_fsyncs, 658
Innodb_os_log_pending_writes, 658
Innodb_os_log_written, 658
Innodb_pages_created, 658
Innodb_pages_read, 658
Innodb_pages_written, 658
Innodb_page_size, 658
Innodb_rows_deleted, 659
Innodb_rows_inserted, 659
Innodb_rows_read, 659
Innodb_rows_updated, 659
Innodb_row_lock_current_waits, 658
Innodb_row_lock_time, 658
Innodb_row_lock_time_avg, 658
Innodb_row_lock_time_max, 658
Innodb_row_lock_waits, 659
Innodb_truncated_status_writes, 659
Key_blocks_not_flushed, 659
Key_blocks_unused, 659
Key_blocks_used, 659
Key_reads, 659
Key_read_requests, 659
Key_writes, 659
Key_write_requests, 659
Last_query_cost, 659
Last_query_partial_plans, 660
Max_used_connections, 660
Ndb_api_adaptive_send_deferred_count, 2584
Ndb_api_adaptive_send_deferred_count_session, 2584
Ndb_api_adaptive_send_deferred_count_slave, 2584
Ndb_api_adaptive_send_forced_count, 2584
Ndb_api_adaptive_send_forced_count_session, 2584
Ndb_api_adaptive_send_forced_count_slave, 2584
Ndb_api_adaptive_send_unforced_count, 2584
Ndb_api_adaptive_send_unforced_count_session, 2585
Ndb_api_adaptive_send_unforced_count_slave, 2585
Ndb_api_bytes_received_count, 2586
Ndb_api_bytes_received_count_session, 2585

Ndb_api_bytes_received_count_slave, 2585
Ndb_api_bytes_sent_count, 2585
Ndb_api_bytes_sent_count_session, 2585
Ndb_api_bytes_sent_count_slave, 2585
Ndb_api_event_bytes_count, 2586
Ndb_api_event_bytes_count_injector, 2586
Ndb_api_event_data_count, 2586
Ndb_api_event_data_count_injector, 2586
Ndb_api_event_nondata_count, 2586
Ndb_api_event_nondata_count_injector, 2586
Ndb_api_pk_op_count, 2587
Ndb_api_pk_op_count_session, 2586
Ndb_api_pk_op_count_slave, 2587
Ndb_api_pruned_scan_count, 2587
Ndb_api_pruned_scan_count_session, 2587
Ndb_api_pruned_scan_count_slave, 2587
Ndb_api_range_scan_count, 2588
Ndb_api_range_scan_count_session, 2588
Ndb_api_range_scan_count_slave, 2588
Ndb_api_read_row_count, 2588
Ndb_api_read_row_count_session, 2588
Ndb_api_read_row_count_slave, 2588
Ndb_api_scan_batch_count, 2589
Ndb_api_scan_batch_count_session, 2589
Ndb_api_scan_batch_count_slave, 2589
Ndb_api_table_scan_count, 2589
Ndb_api_table_scan_count_session, 2589
Ndb_api_table_scan_count_slave, 2589
Ndb_api_trans_abort_count, 2590
Ndb_api_trans_abort_count_session, 2590
Ndb_api_trans_abort_count_slave, 2590
Ndb_api_trans_close_count, 2591
Ndb_api_trans_close_count_session, 2590
Ndb_api_trans_close_count_slave, 2590
Ndb_api_trans_commit_count, 2591
Ndb_api_trans_commit_count_session, 2591
Ndb_api_trans_commit_count_slave, 2591
Ndb_api_trans_local_read_row_count, 2592
Ndb_api_trans_local_read_row_count_session, 2591
Ndb_api_trans_local_read_row_count_slave, 2591
Ndb_api_trans_start_count, 2592
Ndb_api_trans_start_count_session, 2592
Ndb_api_trans_start_count_slave, 2592
Ndb_api_uk_op_count, 2593
Ndb_api_uk_op_count_session, 2592
Ndb_api_uk_op_count_slave, 2592
Ndb_api_wait_exec_complete_count, 2593
Ndb_api_wait_exec_complete_count_session, 2593
Ndb_api_wait_exec_complete_count_slave, 2593
Ndb_api_wait_meta_request_count, 2594
Ndb_api_wait_meta_request_count_session, 2593
Ndb_api_wait_meta_request_count_slave, 2594
Ndb_api_wait_nanos_count, 2594
Ndb_api_wait_nanos_count_session, 2594
Ndb_api_wait_nanos_count_slave, 2594
Ndb_api_wait_scan_result_count, 2595
Ndb_api_wait_scan_result_count_session, 2594
Ndb_api_wait_scan_result_count_slave, 2595

Ndb_cluster_node_id, 2595
Ndb_config_from_host, 2595
Ndb_config_from_port, 2595
Ndb_conflict_fn_epoch, 2596
Ndb_conflict_fn_epoch2, 2596
Ndb_conflict_fn_epoch2_trans, 2596
Ndb_conflict_fn_epoch_trans, 2596
Ndb_conflict_fn_max, 2595
Ndb_conflict_fn_max_del_win, 2595
Ndb_conflict_fn_old, 2595
Ndb_conflict_last_conflict_epoch, 2596
Ndb_conflict_last_stable_epoch, 2597
Ndb_conflict_reflected_op_discard_count, 2596
Ndb_conflict_reflected_op_prepare_count, 2596
Ndb_conflict_refresh_op_count, 2597
Ndb_conflict_trans_conflict_commit_count, 2597
Ndb_conflict_trans_detect_iter_count, 2597
Ndb_conflict_trans_reject_count, 2597
Ndb_conflict_trans_row_conflict_count, 2597
Ndb_conflict_trans_row_reject_count, 2597
Ndb_epoch_delete_delete_count, 2598
Ndb_execute_count, 2598
Ndb_last_commit_epoch_server, 2598
Ndb_last_commit_epoch_session, 2598
Ndb_number_of_data_nodes, 2598
Ndb_pruned_scan_count, 2598
Ndb_pushed_queries_defined, 2598
Ndb_pushed_queries_dropped, 2598
Ndb_pushed_queries_executed, 2598
Ndb_pushed_reads, 2598
Ndb_scan_count, 2599
Not_flushed_delayed_rows, 660
Opened_files, 660
Opened_tables, 660
Opened_table_definitions, 660
Open_files, 660
Open_streams, 660
Open_tables, 660
Open_table_definitions, 660
Performance_schema_accounts_lost, 3170
Performance_schema_cond_classes_lost, 3170
Performance_schema_cond_instances_lost, 3170
Performance_schema_digest_lost, 3170
Performance_schema_file_classes_lost, 3170
Performance_schema_file_handles_lost, 3170
Performance_schema_file_instances_lost, 3170
Performance_schema_hosts_lost, 3170
Performance_schema_locker_lost, 3170
Performance_schema_mutex_classes_lost, 3170
Performance_schema_mutex_instances_lost, 3170
Performance_schema_rwlock_classes_lost, 3171
Performance_schema_rwlock_instances_lost, 3171
Performance_schema_session_connect_attrs_lost, 3171
Performance_schema_socket_classes_lost, 3171
Performance_schema_socket_instances_lost, 3171
Performance_schema_stage_classes_lost, 3171
Performance_schema_statement_classes_lost, 3171
Performance_schema_table_handles_lost, 3171

Performance_schema_table_instances_lost, 3171
Performance_schema_thread_classes_lost, 3171
Performance_schema_thread_instances_lost, 3171
Performance_schema_users_lost, 3171
Prepared_stmt_count, 660
Qcache_free_blocks, 661
Qcache_free_memory, 661
Qcache_hits, 661
Qcache_inserts, 661
Qcache_lowmem_prunes, 661
Qcache_not_cached, 661
Qcache_queries_in_cache, 661
Qcache_total_blocks, 661
Queries, 661
Questions, 661
Rpl_semi_sync_master_clients, 661
Rpl_semi_sync_master_net_avg_wait_time, 661
Rpl_semi_sync_master_net_waits, 662
Rpl_semi_sync_master_net_wait_time, 662
Rpl_semi_sync_master_no_times, 662
Rpl_semi_sync_master_no_tx, 662
Rpl_semi_sync_master_status, 662
Rpl_semi_sync_master_timefunc_failures, 662
Rpl_semi_sync_master_tx_avg_wait_time, 662
Rpl_semi_sync_master_tx_waits, 662
Rpl_semi_sync_master_tx_wait_time, 662
Rpl_semi_sync_master_wait_pos_backtraverse, 663
Rpl_semi_sync_master_wait_sessions, 663
Rpl_semi_sync_master_yes_tx, 663
Rpl_semi_sync_slave_status, 663
Rsa_public_key, 663
Select_full_join, 663
Select_full_range_join, 663
Select_range, 663
Select_range_check, 663
Select_scan, 663
Slave_heartbeat_period, 664
Slave_last_heartbeat, 664
Slave_open_temp_tables, 664
Slave_received_heartbeats, 664
Slave_retried_transactions, 664
Slave_rows_last_search_algorithm_used, 664
Slave_running, 664
Slow_launch_threads, 664
Slow_queries, 664
Sort_merge_passes, 664
Sort_range, 664
Sort_rows, 664
Sort_scan, 664
Ssl_accepts, 665
Ssl_accept_renegotiates, 665
Ssl_callback_cache_hits, 665
Ssl_cipher, 665
Ssl_cipher_list, 665
Ssl_client_connects, 665
Ssl_connect_renegotiates, 665
Ssl_ctx_verify_depth, 665
Ssl_ctx_verify_mode, 665

Ssl_default_timeout, 665
Ssl_finished_accepts, 665
Ssl_finished_connects, 665
Ssl_server_not_after, 665
Ssl_server_not_before, 665
Ssl_sessions_reused, 666
Ssl_session_cache_hits, 666
Ssl_session_cache_misses, 666
Ssl_session_cache_mode, 666
Ssl_session_cache_overflows, 666
Ssl_session_cache_size, 666
Ssl_session_cache_timeouts, 666
Ssl_used_session_cache_entries, 666
Ssl_verify_depth, 666
Ssl_verify_mode, 666
Ssl_version, 666
Table_locks_immediate, 666
Table_locks_waited, 667
Table_open_cache_hits, 667
Table_open_cache_misses, 667
Table_open_cache_overflows, 667
Tc_log_max_pages_used, 667
Tc_log_page_siz, 667
Tc_log_page_waits, 667
Threads_cached, 667
Threads_connected, 667
Threads_created, 667
Threads_running, 667
Uptime, 668
Uptime_since_flush_status, 668
validate_password_dictionary_file_last_parsed, 883
validate_password_dictionary_file_words_count, 883
status variables, 650, 1794
NDB Cluster, 2584
NDB Cluster replication conflict detection, 2857
STD(), 1475
STDDEV(), 1475
STDDEV_POP(), 1475
STDDEV_SAMP(), 1475
stemming, 3884
STOP command (NDB Cluster), 2700
STOP SLAVE, 1681
stop-datetime option
mysqlbinlog, 433
stop-never option
mysqlbinlog, 433
stop-never-slave-server-id option
mysqlbinlog, 433
stop-position option
mysqlbinlog, 433
StopOnError, 2505
stopping
the server, 193
stopword, 3884
stopword list
user-defined, 1406
stopwords, 1403
storage engine, 3885

ARCHIVE, 2141
InnoDB, 1820
PERFORMANCE_SCHEMA, 3063
storage engines
and application feature requirements, 2400
applications supported, 2399
availability, 2398
choosing, 2123
differences between NDB and InnoDB, 2398
usage scenarios, 2400
storage nodes - see data nodes, ndbd (see data nodes, ndbd)
storage nodes - see data nodes, ndbd, ndbmtd (see data nodes, ndbd, ndbmtd)
storage requirements
data types, 1303
date data types, 1305
InnoDB tables, 1304
NDB Cluster, 1304
numeric data types, 1304
spatial data types, 1307
string data types, 1306
time data types, 1305
storage space
minimizing, 1029
storage_engine system variable, 622
stored functions, 2941
and INSERT DELAYED, 1597
stored object, 3885
stored object privileges, 2959
stored objects, 2939
orphan, 2961
stored procedures, 2941
stored program, 3885
stored programs, 1687, 2939
reparsing, 1086
stored routine, 3885
stored routines, 2939, 2941
and replication, 2359
LAST_INSERT_ID(), 2943
metadata, 2943
restrictions, 2968
stored_program_cache system variable, 622
storing result in query cache
thread state, 1117
storing row into queue
thread state, 1116
STRAIGHT_JOIN, 983, 1054, 1066, 1630, 1815
SELECT modifier, 1002, 1627
STRCMP(), 1385
strict mode, 3885
strict SQL mode, 669
STRICT_ALL_TABLES SQL mode, 674
STRICT_TRANS_TABLES SQL mode, 669, 674
string collating, 1232
string comparison functions, 1383
string comparison operators, 1383
string comparisons
case sensitivity, 1383
string concatenation, 1123, 1370

string data types, 1274
  storage requirements, 1306
string functions, 1368
string literal introducer, 1124, 1186
string literals, 1123
string operators, 1368
string replacement
  replace utility, 455
string types, 1274
StringMemory, 2490
strings
  defined, 1123
  escape sequences, 1123
  nondelimited, 1128
  repertoire, 1178
striping
  defined, 1097
STR_TO_DATE(), 1359
ST_Area(), 1463
ST_AsBinary(), 1457
ST_AsText(), 1457
ST_Buffer(), 1465
ST_Centroid(), 1463
ST_Contains(), 1467
ST_Crosses(), 1467
ST_Difference(), 1465
ST_Dimension(), 1458
ST_Disjoint(), 1467
ST_Distance(), 1467
ST_EndPoint(), 1461
ST_Envelope(), 1458
ST_Equals(), 1467
ST_ExteriorRing(), 1463
ST_GeomCollFromWKB(), 1455
ST_GeometryCollectionFromWKB(), 1455
ST_GeometryFromText(), 1454
ST_GeometryFromWKB(), 1455
ST_GeometryN(), 1464
ST_GeometryType(), 1458
ST_GeomFromText(), 1454
ST_GeomFromWKB(), 1455
ST_InteriorRingN(), 1464
ST_Intersection(), 1465
ST_Intersects(), 1467
ST_IsClosed(), 1461
ST_IsEmpty(), 1459
ST_IsSimple(), 1459
ST_LineFromText(), 1454
ST_LineFromWKB(), 1456
ST_LineStringFromText(), 1454
ST_LineStringFromWKB(), 1456
ST_NumGeometries(), 1464
ST_NumInteriorRings(), 1464
ST_NumPoints(), 1461
ST_Overlaps(), 1467
ST_PointFromText(), 1454
ST_PointFromWKB(), 1456
ST_PointN(), 1462

ST_PolyFromText(), 1454
ST_PolyFromWKB(), 1456
ST_PolygonFromText(), 1454
ST_PolygonFromWKB(), 1456
ST_SRID(), 1459
ST_StartPoint(), 1462
ST_SymDifference(), 1465
ST_Touches(), 1468
ST_Union(), 1466
ST_Within(), 1468
ST_X(), 1459
ST_Y(), 1459
SUBDATE(), 1360
sublist, 3885
SUBPARTITION BY KEY
  known issues, 2929
subpartitioning, 2891
subpartitions, 2891
  known issues, 2929
subqueries, 1637
  correlated, 1642
  errors, 1645
  in FROM clause (see derived tables)
  optimization, 1001, 1005
  restrictions, 1648
  rewriting as joins, 1648
  with ALL, 1640
  with ANY, IN, SOME, 1639
  with EXISTS, 1641
  with NOT EXISTS, 1641
  with row constructors, 1640
subquery (see subqueries)
subquery materialization, 1003
subselects, 1637
SUBSTR(), 1377
SUBSTRING(), 1378
SUBSTRING_INDEX(), 1378
SUBTIME(), 1360
subtraction (-), 1338
suffix option
  mysqlhotcopy, 446
SUM(), 1475
SUM(DISTINCT), 1475
SUNPRO_CXX_LIBRARY option
  CMake, 177
SUPER privilege, 789
super-large-pages option
  mysqld, 542
superuser, 189
superuser option
  mysqlaccess, 423
support
  for operating systems, 60
  for platforms, 60
suppression
  default values, 47
supremum record, 3885
surrogate key, 3885

symbolic links, 1098, 1099
  databases, 1098
  tables, 1098
  Windows, 1099
symbolic-links option
  mysqld, 542
symbols-file option
  resolve_stack_dump, 454
synchronization of source and replica
  in NDB Cluster Replication, 2844
Syncing ndb table schema operation and binlog
  thread state, 1121
sync_binlog system variable, 2305
sync_frm system variable, 622
sync_master_info system variable, 2286
sync_relay_log system variable, 2286
sync_relay_log_info system variable, 2287
syntax
  regular expression, 1386
syntax conventions, 2
synthetic key, 3885
sys schema, 3065
sys-check option
  ndb_index_stat, 2651
sys-create option
  ndb_index_stat, 2650
sys-create-if-not-exist option
  ndb_index_stat, 2650
sys-create-if-not-valid option
  ndb_index_stat, 2650
sys-drop option
  ndb_index_stat, 2650
sys-skip-events option
  ndb_index_stat, 2651
sys-skip-tables option
  ndb_index_stat, 2651
SYSCONFDIR option
  CMake, 171
SYSDATE(), 1361
sysdate-is-now option
  mysqld, 542
syslog option
  mysqld_safe, 285
syslog-tag option
  mysqld_safe, 285
system
  privilege, 781
  security, 766
system command
  mysql, 322
System lock
  thread state, 1114
system optimization, 1096
system option
  ndb_config, 2636
system table
  optimizer, 1058, 1627
system tables

columns_priv table, 693, 791
db table, 189, 693, 791
event table, 693
firewall_users table, 694
firewall_whitelist table, 694
func table, 693, 729
general_log table, 693
help tables, 694
help_category table, 694
help_keyword table, 694
help_relation table, 694
help_topic table, 694
host table, 693
innodb_index_stats table, 694, 1909
innodb_table_stats table, 694, 1909
ndb_binlog_index table, 694, 2834
plugin table, 693
proc table, 693
procs_priv table, 693, 791
proxies_priv table, 189, 693, 791
servers table, 694
slave_master_info table, 694
slave_relay_log_info table, 694
slave_worker_info table, 694
slow_log table, 693
tables_priv table, 693, 791
time zone tables, 694
time_zone table, 694
time_zone_leap_second table, 694
time_zone_name table, 694
time_zone_transition table, 694
time_zone_transition_type table, 694
user table, 189, 693, 791
system tablespace, 3886
system variable
audit_log_buffer_size, 902
audit_log_connection_policy, 902
audit_log_current_session, 903
audit_log_exclude_accounts, 903
audit_log_file, 903
audit_log_flush, 903
audit_log_format, 904
audit_log_include_accounts, 904
audit_log_policy, 905
audit_log_rotate_on_size, 905
audit_log_statement_policy, 906
audit_log_strategy, 906
authentication_windows_log_level, 546
authentication_windows_use_principal_name, 546
autocommit, 547
automatic_sp_privileges, 547
auto_increment_increment, 2256
auto_increment_offset, 2259
avoid_temporal_upgrade, 548
back_log, 548
basedir, 549
big_tables, 549
bind_address, 549

binlogging_impossible_mode, 2296
binlog_cache_size, 2292
binlog_checksum, 2293
binlog_direct_non_transactional_updates, 2293
binlog_error_action, 2294
binlog_format, 2295
binlog_gtid_simple_recovery, 2306
binlog_max_flush_queue_time, 2297
binlog_order_commits, 2297
binlog_rows_query_log_events, 2299
binlog_row_image, 2298
binlog_stmt_cache_size, 2299
block_encryption_mode, 550
bulk_insert_buffer_size, 551, 2130
character_sets_dir, 553
character_set_client, 551
character_set_connection, 551
character_set_database, 552
character_set_filesystem, 552
character_set_results, 552
character_set_server, 552
character_set_system, 553
collation_connection, 553
collation_database, 553
collation_server, 554
completion_type, 554
concurrent_insert, 554
connection_control_failed_connections_threshold, 875
connection_control_max_connection_delay, 876
connection_control_min_connection_delay, 876
connect_timeout, 555
core_file, 556
daemon_memcached_enable_binlog, 1967
daemon_memcached_engine_lib_name, 1967
daemon_memcached_engine_lib_path, 1967
daemon_memcached_option, 1968
daemon_memcached_r_batch_size, 1968
daemon_memcached_w_batch_size, 1969
datadir, 556
datetime_format, 556
date_format, 556
debug, 556
debug_sync, 557
default_storage_engine, 557
default_tmp_storage_engine, 558
default_week_format, 558
delayed_insert_limit, 559
delayed_insert_timeout, 560
delayed_queue_size, 560
delay_key_write, 558, 2130
disconnect_on_expired_password, 560
div_precision_increment, 561
end_markers_in_json, 561
error_count, 562
event_scheduler, 562
expire_logs_days, 2300
explicit_defaults_for_timestamp, 563
external_user, 565

flush, 565
flush_time, 565
foreign_key_checks, 565
ft_boolean_syntax, 566
ft_max_word_len, 567
ft_min_word_len, 567
ft_query_expansion_limit, 568
ft_stopword_file, 568
general_log, 568
general_log_file, 568
group_concat_max_len, 569
gtid_executed, 2307
gtid_purged, 2309
have_compress, 569
have_crypt, 569
have_dynamic_loading, 569
have_geometry, 569
have_openssl, 569
have_profiling, 569
have_query_cache, 569
have_rtree_keys, 570
have_ssl, 570
have_symlink, 570
hostname, 571
identity, 571
ignore_builtin_innodb, 1969
ignore_db_dirs, 571
init_connect, 571
init_file, 571
init_slave, 2270
innodb_adaptive_flushing, 1969
innodb_adaptive_flushing_lwm, 1969
innodb_adaptive_hash_index, 1970
innodb_adaptive_max_sleep_delay, 1970
innodb_additional_mem_pool_size, 1971
innodb_api_bk_commit_interval, 1971
innodb_api_disable_rowlock, 1971
innodb_api_enable_binlog, 1972
innodb_api_enable_mdl, 1972
innodb_api_trx_level, 1972
innodb_autoextend_increment, 1973
innodb_autoinc_lock_mode, 1973
innodb_buffer_pool_dump_at_shutdown, 1973
innodb_buffer_pool_dump_now, 1974
innodb_buffer_pool_filename, 1974
innodb_buffer_pool_instances, 1974
innodb_buffer_pool_load_abort, 1975
innodb_buffer_pool_load_at_startup, 1976
innodb_buffer_pool_load_now, 1976
innodb_buffer_pool_size, 1976
innodb_change_buffering, 1977
innodb_change_buffering_debug, 1978
innodb_change_buffer_max_size, 1977
innodb_checksums, 1980
innodb_checksum_algorithm, 1978
innodb_cmp_per_index_enabled, 1981
innodb_commit_concurrency, 1981
innodb_compression_failure_threshold_pct, 1981

innodb_compression_level, 1982
innodb_compression_pad_pct_max, 1982
innodb_concurrency_tickets, 1982
innodb_data_file_path, 1983
innodb_data_home_dir, 1984
innodb_disable_sort_file_cache, 1984
innodb_doublewrite, 1984
innodb_fast_shutdown, 1985
innodb_file_format, 1985
innodb_file_format_check, 1986
innodb_file_format_max, 1986
innodb_file_per_table, 1987
innodb_fil_make_page_dirty_debug, 1985
innodb_flushing_avg_loops, 1991
innodb_flush_log_at_timeout, 1987
innodb_flush_log_at_trx_commit, 1988
innodb_flush_method, 1989
innodb_flush_neighbors, 1990
innodb_force_load_corrupted, 1991
innodb_force_recovery, 1991
innodb_ft_aux_table, 1992
innodb_ft_cache_size, 1992
innodb_ft_enable_diag_print, 1993
innodb_ft_enable_stopword, 1993
innodb_ft_max_token_size, 1994
innodb_ft_min_token_size, 1994
innodb_ft_num_word_optimize, 1994
innodb_ft_result_cache_limit, 1995
innodb_ft_server_stopword_table, 1995
innodb_ft_sort_pll_degree, 1996
innodb_ft_total_cache_size, 1996
innodb_ft_user_stopword_table, 1997
innodb_io_capacity, 1997
innodb_io_capacity_max, 1997
innodb_limit_optimistic_insert_debug, 1998
innodb_locks_unsafe_for_binlog, 1999
innodb_lock_wait_timeout, 1999
innodb_log_buffer_size, 2002
innodb_log_checkpoint_now, 2002
innodb_log_compressed_pages, 2002
innodb_log_files_in_group, 2003
innodb_log_file_size, 2003
innodb_log_group_home_dir, 2004
innodb_lru_scan_depth, 2004
innodb_max_dirty_pages_pct, 2005
innodb_max_dirty_pages_pct_lwm, 2005
innodb_max_purge_lag, 2006
innodb_max_purge_lag_delay, 2006
innodb_mirrored_log_groups, 2006
innodb_monitor_disable, 2006
innodb_monitor_enable, 2007
innodb_monitor_reset, 2007
innodb_monitor_reset_all, 2007
innodb_numa_interleave, 2007
innodb_old_blocks_pct, 2008
innodb_old_blocks_time, 2008
innodb_online_alter_log_max_size, 2009
innodb_open_files, 2009

innodb_optimize_fulltext_only, 2009
innodb_page_size, 2010
innodb_purge_batch_size, 2011
innodb_purge_threads, 2011
innodb_random_read_ahead, 2012
innodb_read_ahead_threshold, 2012
innodb_read_io_threads, 2013
innodb_read_only, 2013
innodb_replication_delay, 2013
innodb_rollback_on_timeout, 2014
innodb_rollback_segments, 2014
innodb_saved_page_number_debug, 2014
innodb_sort_buffer_size, 2015
innodb_spin_wait_delay, 2016
innodb_stats_auto_recalc, 2016
innodb_stats_include_delete_marked, 1911, 2016
innodb_stats_method, 2017
innodb_stats_on_metadata, 2017
innodb_stats_persistent_sample_pages, 2018
innodb_stats_sample_pages, 2019
innodb_stats_transient_sample_pages, 2019
innodb_status_output, 2019
innodb_status_output_locks, 2020
innodb_strict_mode, 2020
innodb_support_xa, 2021
innodb_sync_array_size, 2021
innodb_sync_spin_loops, 2022
innodb_table_locks, 2022
innodb_thread_concurrency, 2022
innodb_thread_sleep_delay, 2023
innodb_tmpdir, 2024
innodb_trx_purge_view_update_only_debug, 2025
innodb_trx_rseg_n_slots_debug, 2025
innodb_undo_directory, 2025
innodb_undo_tablespaces, 2026
innodb_use_native_aio, 2027
innodb_use_sys_malloc, 2027
innodb_version, 2028
innodb_write_io_threads, 2028
insert_id, 572
interactive_timeout, 572
join_buffer_size, 572
keep_files_on_create, 573
key_buffer_size, 573
key_cache_age_threshold, 574
key_cache_block_size, 575
key_cache_division_limit, 575
large_files_support, 575
large_pages, 575
large_page_size, 576
last_insert_id, 576
lc_messages, 576
lc_messages_dir, 576
lc_time_names, 577
license, 577
local_infile, 577, 779
locked_in_memory, 578
lock_wait_timeout, 577

log_bin, 2300
log_bin_basename, 2300
log_bin_index, 2301
log_bin_trust_function_creators, 2301
log_bin_use_v1_row_events, 2301
log_error, 578
log_output, 578
log_queries_not_using_indexes, 579
log_slave_updates, 2302
log_slow_slave_statements, 2271
log_throttle_queries_not_using_indexes, 579
log_warnings, 580
long_query_time, 580
lower_case_file_system, 581
lower_case_table_names, 581
low_priority_updates, 581
master_info_repository, 2271
master_verify_checksum, 2302
max_allowed_packet, 582
max_binlog_cache_size, 2303
max_binlog_size, 2303
max_binlog_stmt_cache_size, 2304
max_connections, 583
max_connect_errors, 582
max_delayed_threads, 583
max_digest_length, 583
max_error_count, 584
max_heap_table_size, 584
max_insert_delayed_threads, 585
max_join_size, 329, 585
max_length_for_sort_data, 586
max_prepared_stmt_count, 586
max_relay_log_size, 2272
max_seeks_for_key, 586
max_sort_length, 587
max_sp_recursion_depth, 587
max_tmp_tables, 588
max_user_connections, 588
max_write_lock_count, 588
metadata_locks_cache_size, 589
metadata_locks_hash_instances, 589
min_examined_row_limit, 589
myisam_data_pointer_size, 590
myisam_max_sort_file_size, 590, 2131
myisam_mmap_size, 591
myisam_recover_options, 591, 2131
myisam_repair_threads, 592
myisam_sort_buffer_size, 592, 2131
myisam_stats_method, 593
myisam_use_mmap, 593
mysql_firewall_max_query_size, 922
mysql_firewall_mode, 922
mysql_firewall_trace, 922
named_pipe, 593
named_pipe_full_access_group, 594
ndbinfo_database, 2582
ndbinfo_max_bytes, 2582
ndbinfo_max_rows, 2583

ndbinfo_offline, 2583
ndbinfo_show_hidden, 2583
ndbinfo_table_prefix, 2583
ndbinfo_version, 2584, 2584
ndb_autoincrement_prefetch_sz, 2567
ndb_cache_check_time, 2567
ndb_clear_apply_status, 2567
ndb_deferred_constraints, 2568
ndb_distribution, 2568
ndb_eventbuffer_free_percent, 2568
ndb_eventbuffer_max_alloc, 2569
ndb_extra_logging, 2569
ndb_force_send, 2569
ndb_index_stat_enable, 2570
ndb_index_stat_option, 2570
ndb_join_pushdown, 2572
ndb_log_apply_status, 2574
ndb_log_bin, 2574
ndb_log_binlog_index, 2574
ndb_log_empty_epochs, 2575
ndb_log_empty_update, 2575
ndb_log_exclusive_reads, 2575
ndb_log_orig, 2575
ndb_log_transaction_id, 2576
ndb_optimized_node_selection, 2576
ndb_recv_thread_activation_threshold, 2577
ndb_recv_thread_cpu_mask, 2577
ndb_report_thresh_binlog_epoch_slip, 2577
ndb_report_thresh_binlog_mem_usage, 2578
ndb_show_foreign_key_mock_tables, 2578
ndb_slave_conflict_role, 2578
ndb_table_no_logging, 2579
ndb_table_temporary, 2580
ndb_use_copying_alter_table, 2580
ndb_use_exact_count, 2580
ndb_use_transactions, 2580
ndb_version, 2581
ndb_version_string, 2581
net_buffer_length, 594
net_read_timeout, 595
net_retry_count, 595
net_write_timeout, 595
new, 596
old, 596
old_alter_table, 596
old_passwords, 597
open_files_limit, 598
optimizer_prune_level, 598
optimizer_search_depth, 599
optimizer_switch, 599
optimizer_trace, 600
optimizer_trace_features, 600
optimizer_trace_limit, 601
optimizer_trace_max_mem_size, 601
optimizer_trace_offset, 601
performance_schema, 3160
performance_schema_accounts_size, 3160
performance_schema_digests_size, 3160

performance_schema_events_stages_history_long_size, 3161
performance_schema_events_stages_history_size, 3161
performance_schema_events_statements_history_long_size, 3161
performance_schema_events_statements_history_size, 3162
performance_schema_events_waits_history_long_size, 3162
performance_schema_events_waits_history_size, 3162
performance_schema_hosts_size, 3162
performance_schema_max_cond_classes, 3163
performance_schema_max_cond_instances, 3163
performance_schema_max_digest_length, 3163
performance_schema_max_file_classes, 3164
performance_schema_max_file_handles, 3164
performance_schema_max_file_instances, 3164
performance_schema_max_mutex_classes, 3164
performance_schema_max_mutex_instances, 3165
performance_schema_max_rwlock_classes, 3165
performance_schema_max_rwlock_instances, 3165
performance_schema_max_socket_classes, 3165
performance_schema_max_socket_instances, 3166
performance_schema_max_stage_classes, 3166
performance_schema_max_statement_classes, 3166
performance_schema_max_table_handles, 3167
performance_schema_max_table_instances, 3167
performance_schema_max_thread_classes, 3167
performance_schema_max_thread_instances, 3168
performance_schema_session_connect_attrs_size, 3168
performance_schema_setup_actors_size, 3168
performance_schema_setup_objects_size, 3169
performance_schema_users_size, 3169
pid_file, 601
plugin_dir, 602
port, 602
preload_buffer_size, 602
profiling, 603
profiling_history_size, 603
protocol_version, 603
proxy_user, 603
pseudo_slave_mode, 603
pseudo_thread_id, 603
query_alloc_block_size, 604
query_cache_limit, 604
query_cache_min_res_unit, 604
query_cache_size, 605
query_cache_type, 605
query_cache_wlock_invalidate, 606
query_prealloc_size, 606
rand_seed1, 607
rand_seed2, 607
range_alloc_block_size, 607
read_buffer_size, 607
read_only, 608
read_rnd_buffer_size, 609
relay_log, 2272
relay_log_basename, 2273
relay_log_index, 2273
relay_log_info_file, 2273
relay_log_info_repository, 2274
relay_log_purge, 2274

relay_log_recovery, 2274
relay_log_space_limit, 2275
report_host, 2275
report_password, 2275
report_port, 2276
report_user, 2276
rpl_semi_sync_master_enabled, 2259
rpl_semi_sync_master_timeout, 2259
rpl_semi_sync_master_trace_level, 2260
rpl_semi_sync_master_wait_no_slave, 2260
rpl_semi_sync_slave_enabled, 2276
rpl_semi_sync_slave_trace_level, 2277
rpl_stop_slave_timeout, 2277
secure_auth, 609
secure_file_priv, 610
server_id, 2249
server_id_bits, 2581
sha256_password_private_key_path, 611
sha256_password_public_key_path, 611
shared_memory, 612
shared_memory_base_name, 612
show_old_temporals, 612
simplified_binlog_gtid_recovery, 2310
skip_external_locking, 613
skip_name_resolve, 613
skip_networking, 614
skip_show_database, 614
slave_allow_batching, 2581
slave_checkpoint_group, 2277
slave_checkpoint_period, 2278
slave_compressed_protocol, 2279
slave_exec_mode, 2279
slave_load_tmpdir, 2280
slave_max_allowed_packet, 2280
slave_net_timeout, 2281
slave_parallel_workers, 2281
slave_pending_jobs_size_max, 2282
slave_rows_search_algorithms, 2282
slave_skip_errors, 2283
slave_sql_verify_checksum, 2284
slave_transaction_retries, 2284
slave_type_conversions, 2285
slow_launch_time, 614
slow_query_log, 614
slow_query_log_file, 615
socket, 615
sort_buffer_size, 615
sql_auto_is_null, 616
sql_big_selects, 616
sql_buffer_result, 617
sql_log_bin, 2304
sql_log_off, 617
sql_mode, 617
sql_notes, 618
sql_quote_show_create, 619
sql_safe_updates, 329, 619
sql_select_limit, 329, 619
sql_slave_skip_counter, 2285

sql_warnings, 620
ssl_ca, 620
ssl_capath, 620
ssl_cert, 620
ssl_cipher, 620
ssl_crl, 621
ssl_crlpath, 621
ssl_key, 621
storage_engine, 622
stored_program_cache, 622
sync_binlog, 2305
sync_frm, 622
sync_master_info, 2286
sync_relay_log, 2286
sync_relay_log_info, 2287
system_time_zone, 623
table_definition_cache, 623
table_open_cache, 624
table_open_cache_instances, 624
thread_cache_size, 624
thread_concurrency, 625
thread_handling, 625
thread_pool_algorithm, 626
thread_pool_high_priority_connection, 626
thread_pool_max_unused_threads, 627
thread_pool_prio_kickup_timer, 627
thread_pool_size, 627
thread_pool_stall_limit, 628
thread_stack, 628
timed_mutexes, 629
timestamp, 629
time_format, 629
time_zone, 629
tmpdir, 630
tmp_table_size, 630
transaction_alloc_block_size, 631
transaction_allow_batching, 2582
transaction_prealloc_size, 631
tx_isolation, 632
tx_read_only, 633
unique_checks, 634
updatable_views_with_limit, 635
validate_password_dictionary_file, 880
validate_password_length, 881
validate_password_mixed_case_count, 881
validate_password_number_count, 881
validate_password_policy, 882
validate_password_special_char_count, 882
validate_user_plugins, 635
version, 635
version_comment, 636
version_compile_machine, 636
version_compile_os, 636
wait_timeout, 636
warning_count, 636
system variables, 545, 637, 1800
and replication, 2368
enforce_gtid_consistency, 2306

gtid_mode, 2308
gtid_next, 2308
gtid_owned, 2309
mysqld, 462
privileges required, 639
system_time_zone system variable, 623
SYSTEM_USER(), 1448

T

tab (\t), 1124, 1609
tab option
mysqldump, 361
ndb_restore, 2675
table, 3886
changing, 1501, 1509, 3272
deleting, 1583
rebuilding, 208
repair, 208
row size, 1304
table aliases, 1624
table cache, 1032
table description
myisamchk, 400
table is full, 549, 3254
Table is full errors (NDB Cluster), 2488
table lock, 3887
Table Monitor, 2060, 2071
table names
case sensitivity, 41, 1136
table option
mysql, 319
mysqlaccess, 423
ndb_desc, 2644
table pullout
semijoin strategy, 1003
table scan, 1897
table type, 3887
choosing, 2123
table-level locking, 1087
tables
BLACKHOLE, 2143
checking, 397
cloning, 1562
closing, 1032
compressed, 408
compressed format, 2134
const, 1058
constant, 966
copying, 1562
counting rows, 236
creating, 225
CSV, 2140
defragment, 2133
defragmenting, 959, 1744
deleting rows, 3269
displaying, 375
displaying status, 1795
dumping, 348, 444

dynamic, 2133
error checking, 955
EXAMPLE, 2155
FEDERATED, 2150
flush, 334
fragmentation, 1744
HEAP, 2136
importing, 1841
improving performance, 1029
information, 400
information about, 240
InnoDB, 1820
loading data, 226
maintenance, 340
maintenance schedule, 959
maximum size, 1035
MEMORY, 2136
MERGE, 2145
merging, 2145
multiple, 238
MyISAM, 2127
names, 1132
open, 1032
opening, 1032
optimizing, 958
partitioning, 2145
repair, 340
repairing, 956
retrieving data, 227
selecting columns, 229
selecting rows, 228
sorting rows, 230
symbolic links, 1098
system, 1058
TEMPORARY, 1561
too many, 1034
TABLES
INFORMATION_SCHEMA table, 3004
tables option
mysqlcheck, 347
mysqldump, 364
tablespace, 1866, 3887
Tablespace Monitor, 2060, 2068
InnoDB, 1944, 2075
TABLESPACES
INFORMATION_SCHEMA table, 3008
tables_priv table
system table, 693, 791
TABLE_CONSTRAINTS
INFORMATION_SCHEMA table, 3008
table_definition_cache system variable, 623
table_io_waits_summary_by_index_usage table
performance_schema, 3146
table_io_waits_summary_by_table table
performance_schema, 3145
Table_locks_immediate status variable, 666
Table_locks_waited status variable, 667
table_lock_waits_summary_by_table table

performance_schema, 3146
table_open_cache, 1032
table_open_cache system variable, 624
Table_open_cache_hits status variable, 667
table_open_cache_instances system variable, 624
Table_open_cache_misses status variable, 667
Table_open_cache_overflows status variable, 667
TABLE_PRIVILEGES
INFORMATION_SCHEMA table, 3009
TAN(), 1347
tar
problems on Solaris, 153, 153
tc-heuristic-recover option
mysqld, 543
Tcl, 3887
Tcl API, 3187
tcp-ip option
mysqld_multi, 290
TCP/IP, 106, 111, 174, 276, 285, 316, 430, 452, 457, 537, 676, 730, 775, 822, 3113, 3246
TCP_MAXSEG_SIZE, 2602
TCP_RCV_BUF_SIZE, 2602
TCP_SND_BUF_SIZE, 2602
Tc_log_max_pages_used status variable, 667
Tc_log_page_size status variable, 667
Tc_log_page_waits status variable, 667
tc_time_track_stats
ndbinfo table, 2806
tee command
mysql, 323
tee option
mysql, 319
temp-pool option
mysqld, 543
temporal interval syntax, 1170
temporary file
write access, 184
temporary files, 3263
temporary table, 3887
TEMPORARY table privileges, 786, 1562, 1731
temporary tables
and replication, 2365
internal, 1034
problems, 3272
TEMPORARY tables, 1561
renaming, 1586
temporary tablespace, 3888
terminal monitor
defined, 219
test option
myisampack, 409
testing
connection to the server, 798
installation, 182
postinstallation, 181
test_plugin_server authentication plugin, 869
TEXT
size, 1306
text collection, 3888

TEXT columns
  default values, 1281
  indexes, 1527, 1527
  indexing, 1019, 1543
TEXT data type, 1277, 1280
text files
  importing, 327, 368, 1603
The used command is not allowed with this MySQL version
  error message, 780
thread, 3888
thread cache, 676
thread command
  Binlog Dump, 1108
  Change user, 1108
  Close stmt, 1108
  Connect, 1108
  Connect Out, 1108
  Create DB, 1108
  Daemon, 1109
  Debug, 1109
  Delayed insert, 1109
  Drop DB, 1109
  Error, 1109
  Execute, 1109
  Fetch, 1109
  Field List, 1109
  Init DB, 1109
  Kill, 1109
  Long Data, 1109
  Ping, 1109
  Prepare, 1109
  Processlist, 1109
  Query, 1109
  Quit, 1109
  Refresh, 1109
  Register Slave, 1110
  Reset stmt, 1110
  Set option, 1110
  Shutdown, 1110
  Sleep, 1110
  Statistics, 1110
  Time, 1110
thread commands, 1108
thread state
  After create, 1110
  allocating local table, 1116
  altering table, 1110
  Analyzing, 1110
  Changing master, 1120
  Checking master version, 1118
  checking permissions, 1110
  checking privileges on cached query, 1117
  checking query cache for query, 1117
  Checking table, 1110
  cleaning up, 1110
  Clearing, 1121
  closing tables, 1111
  committing alter table to storage engine, 1111

Committing events to binlog, 1120
Connecting to master, 1118
converting HEAP to MyISAM, 1111
copy to tmp table, 1111
Copying to group table, 1111
Copying to tmp table, 1111
Copying to tmp table on disk, 1111
Creating delayed handler, 1116
Creating index, 1111
Creating sort index, 1111
creating table, 1111
Creating tmp table, 1111
deleting from main table, 1111
deleting from reference tables, 1111
discard_or_import_tablespace, 1111
end, 1112
executing, 1112
Execution of init_command, 1112
Finished reading one binlog; switching to next binlog, 1118
freeing items, 1112
FULLTEXT initialization, 1112
got handler lock, 1116
got old table, 1116
init, 1112
Initialized, 1121
insert, 1117
invalidating query cache entries, 1117
Killed, 1112
Killing slave, 1120
logging slow query, 1112
login, 1112
Making temporary file (append) before replaying LOAD DATA INFILE, 1119
Making temporary file (create) before replaying LOAD DATA INFILE, 1119
manage keys, 1112
Master has sent all binlog to slave; waiting for binlog to be updated, 1118
Opening master dump table, 1120
Opening mysql.ndb_apply_status, 1120
Opening table, 1112
Opening tables, 1112
optimizing, 1113
preparing, 1113
preparing for alter table, 1113
Processing events, 1120
Processing events from schema table, 1120
Purging old relay logs, 1113
query end, 1113
Queueing master event to the relay log, 1118
Reading event from the relay log, 1119
Reading from net, 1113
Reading master dump table data, 1120
Rebuilding the index on master dump table, 1120
Reconnecting after a failed binlog dump request, 1118
Reconnecting after a failed master event read, 1118
Registering slave on master, 1118
Removing duplicates, 1113
removing tmp table, 1113
rename, 1113
rename result table, 1113

Reopen tables, 1113
Repair by sorting, 1113
Repair done, 1113
Repair with keycache, 1113
Requesting binlog dump, 1118
reschedule, 1117
Rolling back, 1114
Saving state, 1114
Searching rows for update, 1114
Sending binlog event to slave, 1118
sending cached result to client, 1117
setup, 1114
Shutting down, 1121
Slave has read all relay log; waiting for more updates, 1119
Sorting for group, 1114
Sorting for order, 1114
Sorting index, 1114
Sorting result, 1114
statistics, 1114
storing result in query cache, 1117
storing row into queue, 1116
Syncing ndb table schema operation and binlog, 1121
System lock, 1114
update, 1114
Updating, 1115
updating main table, 1115
updating reference tables, 1115
upgrading lock, 1117
User lock, 1115
User sleep, 1115
Waiting for allowed to take ndbcluster global schema lock, 1121
Waiting for an event from Coordinator, 1119
Waiting for commit lock, 1115
waiting for delay_list, 1116
Waiting for event from ndbcluster, 1121
Waiting for first event from ndbcluster, 1121
Waiting for global read lock, 1115, 1115
waiting for handler insert, 1117
waiting for handler lock, 1117
waiting for handler open, 1117
Waiting for INSERT, 1117
Waiting for master to send event, 1119
Waiting for master update, 1119
Waiting for ndbcluster binlog update to reach current position, 1121
Waiting for ndbcluster global schema lock, 1121
Waiting for ndbcluster to start, 1121
Waiting for next activation, 1121
Waiting for query cache lock, 1117
Waiting for scheduler to stop, 1121
Waiting for schema epoch, 1121
Waiting for schema metadata lock, 1115
Waiting for slave mutex on exit, 1119, 1120
Waiting for Slave Workers to free pending events, 1120
Waiting for stored function metadata lock, 1115
Waiting for stored procedure metadata lock, 1115
Waiting for table flush, 1115
Waiting for table level lock, 1115
Waiting for table metadata lock, 1115

Waiting for tables, 1115
Waiting for the next event in relay log, 1120
Waiting for the slave SQL thread to free enough relay log space, 1119
Waiting for trigger metadata lock, 1115
Waiting on cond, 1116
Waiting on empty queue, 1121
Waiting to finalize termination, 1118
Waiting to reconnect after a failed binlog dump request, 1119
Waiting to reconnect after a failed master event read, 1119
Waiting until MASTER_DELAY seconds after master executed event, 1120
Writing to net, 1116
thread states, 1106
delayed inserts, 1116
event scheduler, 1121
general, 1110
NDB Cluster, 1120
query cache, 1117
replication replica, 1118, 1119, 1120
replication source, 1118
thread table
performance_schema, 3152
threadblocks
ndbinfo table, 2808
ThreadConfig, 2538
ThreadPool (see DiskIOThreadPool)
threads, 334, 1782
display, 1782
monitoring, 1106, 1782, 2996, 3152
threadstat
ndbinfo table, 2808
Threads_cached status variable, 667
Threads_connected status variable, 667
Threads_created status variable, 667
Threads_running status variable, 667
thread_cache_size system variable, 624
thread_concurrency system variable, 625
thread_handling system variable, 625
thread_pool_algorithm system variable, 626
thread_pool_high_priority_connection system variable, 626
thread_pool_max_unused_threads system variable, 627
thread_pool_prio_kickup_timer system variable, 627
thread_pool_size system variable, 627
thread_pool_stall_limit system variable, 628
thread_stack system variable, 628
Time
thread command, 1110
TIME data type, 1262, 1264
time data types
storage requirements, 1305
time literals, 1126
time representation
Event Scheduler, 2951
time zone problems, 3264
time zone tables, 300
system tables, 694
time zones
and replication, 2366
leap seconds, 689

support, 685
upgrading, 687
TIME(), 1361
TimeBetweenEpochs, 2513
TimeBetweenEpochsTimeout, 2513
TimeBetweenGlobalCheckpoints, 2512, 2545
TimeBetweenGlobalCheckpointsTimeout, 2513
TimeBetweenInactiveTransactionAbortCheck, 2514
TimeBetweenLocalCheckpoints, 2512
TimeBetweenWatchDogCheck, 2507
TimeBetweenWatchDogCheckInitial, 2508
TIMEDIFF(), 1361
timed_mutexes
deprecated features, 19
timed_mutexes system variable, 629
timeout, 555, 1438, 1603
timeout option
ndb_waiter, 2692
timeouts (replication), 2366
TIMESTAMP
and NULL values, 3268
and replication, 2349
indexes, 1027
initialization and updating, 1268
TIMESTAMP data type, 1261, 1263
timestamp system variable, 629
TIMESTAMP(), 1362
TIMESTAMPADD(), 1362
TIMESTAMPDIFF(), 1362
timezone option
mysqld_safe, 285
time_format
deprecated features, 19
time_format system variable, 629
TIME_FORMAT(), 1362
TIME_TO_SEC(), 1362
time_zone system variable, 629
time_zone table
system table, 694
time_zone_leap_second table
system table, 694
time_zone_name table
system table, 694
time_zone_transition table
system table, 694
time_zone_transition_type table
system table, 694
TINYBLOB data type, 1276
TINYINT data type, 1253
TINYTEXT data type, 1276
tips
optimization, 1016
TLS, 828
command options, 272
establishing connections, 829
TMPDIR environment variable, 184, 256, 457, 3263
TMPDIR option
CMake, 171

tmpdir option
  myisamchk, 399
  myisampack, 409
  mysqld, 544
  mysqlhotcopy, 447
  mysql_upgrade, 306
tmpdir system variable, 630
tmp_table_size system variable, 630
to-last-log option
  mysqlbinlog, 434
Tomcat, 3888
tools
  command-line, 100, 307
  list of, 55
  mysqld_multi, 289
  mysqld_safe, 282
torn page, 1943, 3888
TotalSendBufferMemory
  API and SQL nodes, 2553
  data nodes, 2546
  management nodes, 2482
Touches(), 1468
TO_BASE64(), 1378
TO_DAYS(), 1363
TO_SECONDS(), 1363
TPS, 3888
TP_THREAD_GROUP_STATE
  INFORMATION_SCHEMA table, 3055
TP_THREAD_GROUP_STATS
  INFORMATION_SCHEMA table, 3056
TP_THREAD_STATE
  INFORMATION_SCHEMA table, 3058
trace DBI method, 739
trace files
  ndbmtd, 2618
trace files (NDB Cluster), 2615
TRADITIONAL SQL mode, 669, 675
trailing spaces
  CHAR, 1276, 1278
  ENUM, 1283
  in comparisons, 1278
  SET, 1285
  VARCHAR, 1276, 1278
transaction, 3888
transaction access mode, 1663
transaction ID, 3888
transaction isolation level, 1663
  NDB Cluster, 2404
  READ COMMITTED, 1877
  READ UNCOMMITTED, 1879
  REPEATABLE READ, 1877
  SERIALIZABLE, 1879
transaction-isolation option
  mysqld, 543
transaction-read-only option
  mysqld, 544
transaction-safe tables, 1820
transactional option

ndb_delete_all, 2639
TransactionBufferMemory, 2494
TransactionDeadlockDetectionTimeout, 2515
TransactionInactiveTimeout, 2514
transactions, 1872
and replication, 2366, 2366
isolation levels, 1876
metadata locking, 1092
support, 1820
transaction_alloc_block_size system variable, 631
transaction_allow_batching session variable (NDB Cluster), 2582
transaction_prealloc_size system variable, 631
Translators
list of, 53
transparent page compression, 3889
transportable tablespace, 3889
Transportable Tablespaces, 1841
transporters
ndbinfo table, 2810
.TRG file, 3886
TRIGGER privilege, 790
triggers, 1574, 1584, 1799, 2939, 2943
and INSERT DELAYED, 1597
and replication, 2359, 2368
LAST_INSERT_ID(), 2943
metadata, 2947
restrictions, 2968
TRIGGERS
INFORMATION_SCHEMA table, 3009
triggers option
mysqldump, 364
TRIM(), 1379
.TRN file, 3886
troubleshooting, 3243, 3889
ALTER TABLE problems, 3272
compiling MySQL server, 179
connection problems, 822
InnoDB deadlocks, 1887, 1888
InnoDB errors, 2118
InnoDB recovery problems, 2113
InnoDB table fragmentation, 1945
replication, 2373
startup problems, 185
with MySQL Enterprise Monitor, 3189
with MySQL Performance Schema, 3172
TRUE, 1126, 1131
testing for, 1330, 1330
truncate, 3889
TRUNCATE TABLE, 1586
and NDB Cluster, 2403
and replication, 2368
performance_schema database, 3102, 3174
TRUNCATE(), 1347
truststore, 3889
tuning, 962
InnoDB compressed tables, 1923
tuple, 3889
tupscan option

ndb_select_all, 2682
tutorial, 219
twiddle option
ndb_redo_log_reader, 2659
two-phase commit, 654, 654, 2021, 3889
TwoPassInitialNodeRestartCopy, 2531
tx_isolation system variable, 632
tx_read_only system variable, 633
type conversions, 1321, 1326
type option
mysql_convert_table_format, 448
ndb_config, 2636
ndb_show_tables, 2688
types
columns, 1251, 1307
data, 1251
date and time, 1259
numeric, 1252
of tables, 2123
portability, 1308
string, 1274
typographical conventions, 2
TZ environment variable, 457, 685, 3264
tz-utc option
mysqldump, 362

U

UCASE(), 1379
UCS-2, 1176
ucs2 character set, 1211
as client character set, 1192
UDF API, 728
UDFs, 728, 1749, 1750
installing, 728
reference, 1320
uninstalling, 728
ulimit, 3256
UMASK environment variable, 457, 3257
UMASK_DIR environment variable, 457, 3258
unary minus (-), 1338
unblocking blocked hosts, 681
unbuffered option
mysql, 319
UNCOMPRESS(), 1437
UNCOMPRESSED_LENGTH(), 1437
undo, 3890
undo log, 1868, 3890
undo log segment, 3890
undo tablespace, 3890
undo tablespaces, 1868
UndoDataBuffer, 2520
UndoIndexBuffer, 2520
unexpected halt
replication, 2274, 2332
UNHEX(), 1379
Unicode, 1176, 3890
Unicode Collation Algorithm, 1216
UNINSTALL PLUGIN statement, 1751

uninstalling plugins, 717, 1751
uninstalling UDFs, 728
UNION, 246, 1635
UNIQUE, 1509
unique constraint, 3890
unique index, 3891
unique key, 3891
  constraint, 46
unique keys
  and partitioning keys, 2930
unique_checks system variable, 634
unique_subquery join type
  optimizer, 1059
Unix signal handling, 458
UNIX_TIMESTAMP(), 1364
UNKNOWN
  testing for, 1330, 1330
Unknown column ... in 'on clause', 1635, 1635
Unknown or incorrect time zone
  error, 686
unloading
  tables, 227
UNLOCK TABLES, 1657
unnamed views, 1643
unpack option
  myisamchk, 399
unqualified option
  ndb_desc, 2644
  ndb_show_tables, 2688
unsafe statement (replication)
  defined, 2238
unsafe statements (replication), 2239
UNSIGNED, 1252, 1257
UNTIL, 1693
updatable views, 2957
updatable_views_with_limit system variable, 635
UPDATE, 44, 1649
update
  thread state, 1114
update option
  MySQLInstallerConsole, 103
  ndb_index_stat, 2649
UPDATE privilege, 790
update-state option
  myisamchk, 398
UpdateXML(), 1421
Updating
  thread state, 1115
updating main table
  thread state, 1115
updating reference tables
  thread state, 1115
upgrade option
  MySQLInstallerConsole, 103
upgrade-system-tables option
  mysql_upgrade, 306
upgrades
  NDB Cluster, 2435, 2729

upgrades and downgrades (NDB Cluster)
  compatibility between versions, 2435
upgrading, 194
  different architecture, 210
  to &current-series;, 195
  with MySQL APT Repository, 206
  with MySQL SLES Repository, 206
  with MySQL Yum Repository, 204
upgrading lock
  thread state, 1117
upgrading MySQL, 300
UPPER(), 1380
uptime, 334
Uptime status variable, 668
Uptime_since_flush_status status variable, 668
URLs for downloading MySQL, 61
usage option
  ndb_config, 2636
usage option (NDB Cluster programs), 2697
USAGE privilege, 790
USE, 1818
use command
  mysql, 323
USE INDEX, 1072
USE KEY, 1072
use-frm option
  mysqlcheck, 347
use-https option
  ndb_setup.py, 2687
use-threads option
  mysqlimport, 374
useHexFormat option
  ndb_select_all, 2681
user accounts
  altering, 1720
  creating, 803, 1721
  renaming, 1735
  resource limits, 588, 820, 1734
USER environment variable, 278, 457
User lock
  thread state, 1115
user management, 781
user names
  and passwords, 782
  in account names, 796
  in default accounts, 189
user option, 272
  mysql, 319
  mysqlaccess, 423
  mysqladmin, 339
  mysqlbinlog, 434
  mysqlcheck, 348
  mysqld, 544
  mysqldump, 355
  mysqld_multi, 291
  mysqld_safe, 286
  mysqlhotcopy, 447
  mysqlimport, 374

mysqlshow, 380
mysqlslap, 389
mysql_convert_table_format, 448
mysql_install_db, 297
mysql_setpermission, 450
mysql_upgrade, 306
user privileges
adding, 803
checking, 805
deleting, 1724
dropping, 1724
revoking, 805
User sleep
thread state, 1115
user table
sorting, 800
system table, 189, 693, 791
user variables
and replication, 2368
USER(), 1448
user-defined function
creating, 1749
deleting, 1750
user-defined functions (see UDFs)
user-defined variables, 1165
users
deleting, 805, 1724
root, 189
users table
performance_schema, 3133
USER_PRIVILEGES
INFORMATION_SCHEMA table, 3011
USING HASH
with NDB tables, 1529
using multiple disks to start data, 1099
using NDB Cluster programs, 2609
USING versus ON
joins, 1634
UTC_DATE(), 1365
UTC_TIME(), 1366
UTC_TIMESTAMP(), 1366
UTF-8, 1176
database object metadata, 1180
utf16 character set, 1211
as client character set, 1192
utf16le character set, 1211
as client character set, 1192
utf16_bin collation, 1219
utf32 character set, 1211
as client character set, 1192
utf8 character set, 1210
alias for utf8mb3, 1210, 1210
utf8mb3 character set, 1210
utf8 alias, 1210, 1210
utf8mb4 character set, 1209
utilities
program-development, 255
utility programs, 254

UUID(), 1485
UUID_SHORT(), 1486

V

valid numbers
  examples, 1126
validate-password option
  mysqld, 880
validate_password plugin, 877
  configuring, 879
  installing, 878
  options, 879
  status variables, 883
  system variables, 880
validate_password_dictionary_file system variable, 880
validate_password_dictionary_file_last_parsed status variable, 883
validate_password_dictionary_file_words_count status variable, 883
validate_password_length system variable, 881
validate_password_mixed_case_count system variable, 881
validate_password_number_count system variable, 881
validate_password_policy system variable, 882
validate_password_special_char_count system variable, 882
VALIDATE_PASSWORD_STRENGTH(), 1437
validate_user_plugins system variable, 635
VALUES(), 1486
VARBINARY data type, 1276, 1279
VARCHAR
  size, 1306
VARCHAR data type, 1274, 1276
VARCHARACTER data type, 1276
variable option
  mysql_config, 452
variable-length type, 3891
variables
  and replication, 2368
  environment, 256
  server, 1800
  status, 650, 1794
  system, 545, 637, 1800
  user defined, 1165
VARIANCE(), 1476
VAR_POP(), 1476
VAR_SAMP(), 1476
verbose option
  myisamchk, 395
  myisampack, 409
  myisam_ftdump, 391
  mysql, 319
  mysqladmin, 340
  mysqlbinlog, 434
  mysqlcheck, 348
  mysqld, 545
  mysqldump, 358
  mysqldumpslow, 444
  mysqld_multi, 291
  mysqlimport, 375
  mysqlshow, 380
  mysqlslap, 389

mysql_config_editor, 417
mysql_convert_table_format, 448
mysql_install_db, 297
mysql_plugin, 299
mysql_upgrade, 306
mysql_waitpid, 450
my_print_defaults, 454
ndbd, 2615
ndbmtd, 2615
ndb_blob_tool, 2629
ndb_index_stat, 2651
ndb_mgmd, 2626
ndb_move_data, 2654
ndb_restore, 2675
perror, 455
verify-binlog-checksum option
mysqlbinlog, 434
version
choosing, 60
latest, 61
VERSION file
CMake, 180
version option
comp_err, 293
myisamchk, 395
myisampack, 409
mysql, 319
mysqlaccess, 423
mysqladmin, 340
mysqlbinlog, 434
mysqlcheck, 348
mysqld, 545
mysqldump, 358
mysqld_multi, 291
mysqlimport, 375
mysqlshow, 380
mysqlslap, 389
mysql_config, 452
mysql_config_editor, 417
mysql_convert_table_format, 448
mysql_plugin, 299
mysql_waitpid, 450
my_print_defaults, 454
ndb_config, 2636
perror, 455
resolveip, 457
resolve_stack_dump, 454
version option (NDB Cluster programs), 2698
version system variable, 635
VERSION(), 1449
version-check option
mysql_upgrade, 306
version_comment system variable, 636
version_compile_machine system variable, 636
version_compile_os system variable, 636
vertical option
mysql, 319
mysqladmin, 340

victim, 3891
Vietnamese, 3219
view, 3891
views, 1576, 2939, 2955
  algorithms, 2956
  and replication, 2370
  limitations, 2973
  metadata, 2959
  privileges, 2973
  problems, 2973
  restrictions, 2972
  updatable, 1576, 2957
VIEWS
  INFORMATION_SCHEMA table, 3012
Visual Studio, 3891

W

wait, 3891
WAIT COMPLETED (START BACKUP command), 2744
wait option
  myisamchk, 395
  myisampack, 410
  mysql, 319
  mysqladmin, 340
WAIT STARTED (START BACKUP command), 2744
wait-nodes option
  ndb_waiter, 2692
Waiting for allowed to take ndbcluster global schema lock
  thread state, 1121
Waiting for an event from Coordinator
  thread state, 1119
Waiting for commit lock
  thread state, 1115
waiting for delay_list
  thread state, 1116
Waiting for event from ndbcluster
  thread state, 1121
Waiting for event metadata lock
  thread state, 1115
Waiting for event read lock
  thread state, 1115
Waiting for first event from ndbcluster
  thread state, 1121
Waiting for global read lock
  thread state, 1115
waiting for handler insert
  thread state, 1117
waiting for handler lock
  thread state, 1117
waiting for handler open
  thread state, 1117
Waiting for INSERT
  thread state, 1117
Waiting for master to send event
  thread state, 1119
Waiting for master update
  thread state, 1119
Waiting for ndbcluster binlog update to reach current position

thread state, 1121
Waiting for ndbcluster global schema lock
thread state, 1121
Waiting for ndbcluster to start
thread state, 1121
Waiting for next activation
thread state, 1121
Waiting for query cache lock
thread state, 1117
Waiting for scheduler to stop
thread state, 1121
Waiting for schema epoch
thread state, 1121
Waiting for schema metadata lock
thread state, 1115
Waiting for slave mutex on exit
thread state, 1119, 1120
Waiting for Slave Workers to free pending events
thread state, 1120
Waiting for stored function metadata lock
thread state, 1115
Waiting for stored procedure metadata lock
thread state, 1115
Waiting for table flush
thread state, 1115
Waiting for table level lock
thread state, 1115
Waiting for table metadata lock
thread state, 1115
Waiting for tables
thread state, 1115
Waiting for the next event in relay log
thread state, 1120
Waiting for the slave SQL thread to free enough relay log space
thread state, 1119
Waiting for trigger metadata lock
thread state, 1115
Waiting on cond
thread state, 1116
Waiting on empty queue
thread state, 1121
Waiting to finalize termination
thread state, 1118
Waiting to reconnect after a failed binlog dump request
thread state, 1119
Waiting to reconnect after a failed master event read
thread state, 1119
Waiting until MASTER_DELAY seconds after master executed event
thread state, 1120
wait_timeout system variable, 636
WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS(), 1471
Wan, 2481, 2555
warm backup, 3892
warm up, 3892
warnings command
mysql, 323
warning_count system variable, 636
websites

MySQL, 35
WEEK(), 1366
WEEKDAY(), 1367
WEEKOFYEAR(), 1367
WEIGHT_STRING(), 1380
Well-Known Binary format
geometry values, 1296
Well-Known Text format
geometry values, 1295
WHERE, 965
with SHOW, 2976, 3060
where option
mysqldump, 364
WHILE, 1694
labels, 1687
widths
display, 1251
Wildcard character (%), 1124
Wildcard character (_), 1125
wildcards
and LIKE, 1023
in account names, 797
in mysql.columns_priv table, 802
in mysql.db table, 801
in mysql.procs_priv table, 802
in mysql.tables_priv table, 802
Windows
MySQL restrictions, 115
path name separators, 262
pluggable authentication, 863
upgrading, 206
windows option
mysql_install_db, 297
WITH ROLLUP, 1476
Within(), 1469
WITH_ASAN option
CMake, 174
WITH_BUNDLED_LIBEVENT option
CMake, 178
WITH_BUNDLED_MEMCACHED option
CMake, 178
WITH_CLASSPATH option
CMake, 178
WITH_DEBUG option
CMake, 175
WITH_DEFAULT_COMPILER_OPTIONS option
CMake, 177
WITH_DEFAULT_FEATURE_SET option
CMake, 175
WITH_EDITLINE option
CMake, 175
WITH_EMBEDDED_SERVER option
CMake, 175
WITH_EMBEDDED_SHARED_LIBRARY option
CMake, 175
WITH_ERROR_INSERT option
CMake, 178
WITH_EXTRA_CHARSETS option

CMake, 175
WITH_GMOCK option
CMake, 175
WITH_INNODB_MEMCACHED option
CMake, 175
WITH_LIBEDIT option
CMake, 176
WITH_LIBEVENT option
CMake, 175
WITH_LIBWRAP option
CMake, 176
WITH_NDBCLUSTER option
CMake, 179
WITH_NDBCLUSTER_STORAGE_ENGINE option
CMake, 178
WITH_NDBMTD option
CMake, 179
WITH_NDB_BINLOG option
CMake, 179
WITH_NDB_DEBUG option
CMake, 179
WITH_NDB_JAVA option
CMake, 179
WITH_NDB_PORT option
CMake, 179
WITH_NDB_TEST option
CMake, 179
WITH_NUMA option
CMake, 176
WITH_SSL option
CMake, 176
WITH_SYMVER16 option
CMake, 176
WITH_UNIT_TESTS option
CMake, 177
WITH_UNIXODBC option
CMake, 177
WITH_VALGRIND option
CMake, 177
WITH_ZLIB option
CMake, 177
WKB format
geometry values, 1296
WKT format
geometry values, 1295
workload, 3892
wrappers
Eiffel, 3187
write access
tmp, 184
write combining, 3892
write-binlog option
mysqlcheck, 348
mysql_upgrade, 306
write_buffer_size myisamchk variable, 395
Writing to net
thread state, 1116

X

X(), 1460
X.509/Certificate, 828
XA, 3892
XA BEGIN, 1667
XA COMMIT, 1667
XA PREPARE, 1667
XA RECOVER, 1667
XA ROLLBACK, 1667
XA START, 1667
XA transactions, 1666
  restrictions, 1669
  transaction identifiers, 1667
xid
  XA transaction identifier, 1667
xml option
  mysql, 319
  mysqldump, 362
  ndb_config, 2636
XOR
  bitwise, 1429
  logical, 1333

Y

Y(), 1460
yaSSL, 164, 828
  compared to OpenSSL, 840
yaSSL versus OpenSSL
  detecting, 841
YEAR data type, 1262, 1265
YEAR(), 1367
YEARWEEK(), 1367
Yen sign (Japanese), 3219
young, 3892
Your password does not satisfy the current policy requirements
  password error, 877

Z

ZEROFILL, 1252, 1257
ZFS, 2161

C Function Index

mysql_affected_rows()

Section 13.2.1, “CALL Statement”
Section 12.16, “Information Functions”
Section 13.2.5, “INSERT Statement”
Section 13.2.8, “REPLACE Statement”

mysql_close()

Section B.3.2.10, “Communication Errors and Aborted Connections”

mysql_errno()

Section 6.4.4.3, “Audit Log File Formats”
Section B.2, “Error Information Interfaces”
Section 13.6.7.5, “SIGNAL Statement”

mysql_error()
Section B.2, “Error Information Interfaces”
Section 13.6.7.5, “SIGNAL Statement”

mysql_escape_string()
Section 6.1.7, “Client Programming Security Guidelines”

mysql_fetch_row()
Section 15.8.1, “FEDERATED Storage Engine Overview”

mysql_free_result()
Section B.3.2.13, “Commands out of sync”

mysql_get_character_set_info()
Section 10.14.2, “Choosing a Collation ID”

mysql_info()
Section 13.1.7, “ALTER TABLE Statement”
Section 13.2.5.3, “INSERT DELAYED Statement”
Section 13.2.5, “INSERT Statement”
Section 13.2.6, “LOAD DATA Statement”
Section 1.7.3.1, “PRIMARY KEY and UNIQUE Index Constraints”
Section 13.2.11, “UPDATE Statement”

mysql_insert_id()
Section 13.1.17, “CREATE TABLE Statement”
Section 12.16, “Information Functions”
Section 13.2.5, “INSERT Statement”
Section 5.1.7, “Server System Variables”
Section 3.6.9, “Using AUTO_INCREMENT”

mysql_library_end()
Section 23.6, “libmysqld, the Embedded MySQL Server Library”

mysql_library_init()
Section 23.6, “libmysqld, the Embedded MySQL Server Library”
Section 23.6.3, “Options with the Embedded Server”

mysql_next_result()
Section 13.2.1, “CALL Statement”

mysql_options()
Section 6.2.1, “Account User Names and Passwords”
Section 6.4.1.5, “Client-Side Cleartext Pluggable Authentication”
Section 10.4, “Connection Character Sets and Collations”
Section 4.2.6, “Connection Compression Control”
Section B.3.2.8, “MySQL server has gone away”
Section 22.12.8, “Performance Schema Connection Attribute Tables”
Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”
Section 6.2.10, “Server Handling of Expired Passwords”
Section 6.4.1.4, “SHA-256 Pluggable Authentication”
Section 5.7.4, “Using Client Programs in a Multiple-Server Environment”

mysql_options4()
Section 22.12.8, “Performance Schema Connection Attribute Tables”

mysql_ping()
Section B.3.2.8, “MySQL server has gone away”

mysql_query()
Section 13.2.1, “CALL Statement”

mysql_real_connect()
Section 13.2.1, “CALL Statement”
Chapter 12, *Functions and Operators*
Section 12.16, “Information Functions”
Section 13.2.5.2, “INSERT ... ON DUPLICATE KEY UPDATE Statement”
Section 13.2.5, “INSERT Statement”
Section 13.5, “Prepared Statements”
Section 6.2.10, “Server Handling of Expired Passwords”
Section 5.1.7, “Server System Variables”
Section 20.2.1, “Stored Routine Syntax”
Section 4.10, “Unix Signal Handling in MySQL”
Section 5.7.4, “Using Client Programs in a Multiple-Server Environment”

mysql_real_escape_string()
Section 6.1.7, “Client Programming Security Guidelines”
Section 11.4.5, “Populating Spatial Columns”
Section 9.1.1, “String Literals”

mysql_real_query()
Section 13.2.1, “CALL Statement”
Section 15.8.1, “FEDERATED Storage Engine Overview”

mysql_shutdown()
Section 6.2.2, “Privileges Provided by MySQL”

mysql_sqlstate()
Section B.2, “Error Information Interfaces”
Section 13.6.7.5, “SIGNAL Statement”

mysql_stmt_attr_set()
Section 13.6.6.5, “Restrictions on Server-Side Cursors”

mysql_stmt_errno()
Section B.2, “Error Information Interfaces”

mysql_stmt_error()
Section B.2, “Error Information Interfaces”

mysql_stmt_execute()
Section 8.10.3.1, “How the Query Cache Operates”

mysql_stmt_next_result()
Section 13.2.1, “CALL Statement”

mysql_stmt_prepare()
Section 8.10.4, “Caching of Prepared Statements and Stored Programs”
Section 8.10.3.1, “How the Query Cache Operates”
Section 13.5, “Prepared Statements”

mysql_stmt_send_long_data()
Section 5.1.7, “Server System Variables”

mysql_stmt_sqlstate()
Section B.2, “Error Information Interfaces”

mysql_store_result()
Section B.3.2.13, “Commands out of sync”
Section 15.8.1, “FEDERATED Storage Engine Overview”
Section 4.5.1, “`mysql` — The MySQL Command-Line Client”

mysql_thread_end()
Section 23.6, “libmysqld, the Embedded MySQL Server Library”

mysql_thread_init()
Section 23.6, “libmysqld, the Embedded MySQL Server Library”

mysql_use_result()
Section B.3.2.13, “Commands out of sync”
Section 4.5.1, “`mysql` — The MySQL Command-Line Client”
Section B.3.2.7, “Out of memory”

mysql_warning_count()
Section B.2, “Error Information Interfaces”
Section 13.7.5.41, “SHOW WARNINGS Statement”

Command Index

A | B | C | D | E | G | H | I | K | L | M | N | O | P | R | S | T | U | V | W | Y | Z

A

[index top]

Access
Section 13.2.2, “DELETE Statement”

addgroup
Section 18.2.1.1, “Installing an NDB Cluster Binary Release on Linux”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”

addr2line
Section 5.8.1.5, “Using a Stack Trace”

adduser
Section 18.2.1.1, “Installing an NDB Cluster Binary Release on Linux”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”

ALL STATUS
Section 18.5.6, “NDB Cluster Single User Mode”

APF
Section 18.5.16.1, “NDB Cluster Security and Networking Issues”

apt-get
Section 16.2.1, “Installing `memcached`”

Section 2.5.7, “Installing MySQL on Linux from the Native Software Repositories”
Section 14.20.4, “Security Considerations for the InnoDB memcached Plugin”
Section 16.2.3.3, “Using `libmemcached` with C and C++”

audit2allow

Section 6.6.6, “Troubleshooting SELinux”

B

[index top]

bash

Section 1.1, “About This Manual”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 2.4.1, “General Notes on Installing MySQL on OS X”
Section 4.2.1, “Invoking MySQL Programs”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 4.2.7, “Setting Environment Variables”

bison

Section 1.8.1, “Contributors to MySQL”
Section 2.9.8, “Dealing with Problems Compiling MySQL”
Section 2.9.2, “Source Installation Prerequisites”

C

[index top]

c++filt

Section 5.8.1.5, “Using a Stack Trace”

cat

Section 4.5.1.1, “mysql Client Options”

cd

Resetting the Root Password: Windows Systems

chkconfig

Section 18.2.1.1, “Installing an NDB Cluster Binary Release on Linux”
Section 2.5.7, “Installing MySQL on Linux from the Native Software Repositories”
Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”

chroot

Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”

CMake

Section 10.13, “Adding a Character Set”
Section 18.2.1.4, “Building NDB Cluster from Source on Linux”
Section B.3.2.16, “Can't initialize character set”
Section 18.2.2.2, “Compiling and Installing NDB Cluster from Source on Windows”
Section 10.5, “Configuring Application Character Set and Collation”
Section 2.9.6, “Configuring SSL Library Support”
Section 2.9.8, “Dealing with Problems Compiling MySQL”
Section 4.9, “Environment Variables”
Section B.3.3.6, “How to Protect or Change the MySQL Unix Socket File”
Section 14.14, “InnoDB Startup Options and System Variables”

Section 2.9.5, “Installing MySQL Using a Development Source Tree”
Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”
Chapter 18, *MySQL NDB Cluster 7.3 and NDB Cluster 7.4*
Section 18.5.9, “MySQL Server Usage for NDB Cluster”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.6.7, “`mysqlaccess` — Client for Checking Access Privileges”
Section 22.2, “Performance Schema Build Configuration”
Section 5.7.3, “Running Multiple MySQL Instances on Unix”
Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”
Section 10.3.2, “Server Character Set and Collation”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 2.9.2, “Source Installation Prerequisites”
Section 15.5, “The ARCHIVE Storage Engine”
Section 15.6, “The BLACKHOLE Storage Engine”
Section 15.9, “The EXAMPLE Storage Engine”
Section 15.8, “The FEDERATED Storage Engine”
Section 1.2.2, “The Main Features of MySQL”
Section 5.8.4, “Tracing mysqld Using DTrace”
Section 4.2.2.2, “Using Option Files”
Section 18.1.4.1, “What is New in NDB Cluster 7.3”
Section B.3.3.3, “What to Do If MySQL Keeps Crashing”

cmake

Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”
Section 14.20.4, “Security Considerations for the InnoDB memcached Plugin”

cmd

Resetting the Root Password: Windows Systems

cmd.exe

Section 1.1, “About This Manual”
Section 4.2.1, “Invoking MySQL Programs”

command.com

Section 1.1, “About This Manual”
Section 4.2.1, “Invoking MySQL Programs”

comp_err

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”
Section 4.1, “Overview of MySQL Programs”

configure

Section 1.1, “About This Manual”
Section 1.6, “How to Report Bugs or Problems”
Section 16.2.1, “Installing `memcached`”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 16.2.3.3, “Using `libmemcached` with C and C++”
Section 16.2.3.6, “Using MySQL and `memcached` with PHP”

copy

Section 17.1.1.6, “Creating a Data Snapshot Using Raw Data Files”

coreadm

Section 2.7, “Installing MySQL on Solaris”

Section 5.1.6, “Server Command Options”

cp

Section 17.3.1.2, “Backing Up Raw Data from a Replica”
Section 7.1, “Backup and Recovery Types”
Section 17.1.1.6, “Creating a Data Snapshot Using Raw Data Files”
Section 17.1.1.9, “Introducing Additional Replicas to an Existing Replication Environment”

crash-me

Section 8.13.2, “The MySQL Benchmark Suite”

cron

Section B.3.2.2, “Can't connect to [local] MySQL server”
Section 13.7.2.2, “CHECK TABLE Statement”
Section 15.2.1, “MyISAM Startup Options”
Section 5.4.7, “Server Log Maintenance”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”
Section 3.5, “Using mysql in Batch Mode”

csh

Section 1.1, “About This Manual”
Section 4.2.1, “Invoking MySQL Programs”
Section 4.2.7, “Setting Environment Variables”

D

[index top]

daemon_memcached

Section 14.20.5.2, “Adapting a memcached Application for the InnoDB memcached Plugin”
Section 14.20.2, “InnoDB memcached Architecture”

date

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

df

Section B.3.1, “How to Determine What Is Causing a Problem”

Directory Utility

Section 2.4.1, “General Notes on Installing MySQL on OS X”

dnf

Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository”
Section 2.11.5, “Upgrading MySQL with the MySQL Yum Repository”

docker exec

Section 2.5.8.1, “Basic Steps for MySQL Server Deployment with Docker”

docker inspect

Section 2.5.8.2, “More Topics on Deploying MySQL Server with Docker”

docker logs mysqld-container

Section 2.5.8.2, “More Topics on Deploying MySQL Server with Docker”

docker ps

Section 2.5.8.1, “Basic Steps for MySQL Server Deployment with Docker”

docker pull
Section 2.5.8.1, “Basic Steps for MySQL Server Deployment with Docker”

docker rm
Section 2.5.8.1, “Basic Steps for MySQL Server Deployment with Docker”

docker run
Section 2.5.8.1, “Basic Steps for MySQL Server Deployment with Docker”
Section 2.5.8.2, “More Topics on Deploying MySQL Server with Docker”

docker stop
Section 2.5.8.1, “Basic Steps for MySQL Server Deployment with Docker”

dump
Section 17.1.1.6, “Creating a Data Snapshot Using Raw Data Files”

E

[index top]

emerge
Section 16.2.1, “Installing `memcached`”
Section 2.5.7, “Installing MySQL on Linux from the Native Software Repositories”

G

[index top]

gcc
Section 23.6.1, “Compiling Programs with libmysqld”
Section 2.13.3, “Problems Using the Perl DBI/DBD Interface”
Section 1.8.4, “Tools that were used to create MySQL”

gdb
Section 5.8.1.1, “Compiling MySQL for Debugging”
Section 5.8.1.4, “Debugging mysqld under gdb”
Section 1.8.4, “Tools that were used to create MySQL”
Section B.3.3.3, “What to Do If MySQL Keeps Crashing”

getenforce
Section 6.6.2, “Changing the SELinux Mode”

git branch
Section 2.9.5, “Installing MySQL Using a Development Source Tree”

git checkout
Section 2.9.5, “Installing MySQL Using a Development Source Tree”

gmake
Section 2.8, “Installing MySQL on FreeBSD”
Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”
Section 2.9.2, “Source Installation Prerequisites”

GnuPG
Section 2.1.4.2, “Signature Checking Using GnuPG”

gnutar
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”
Section 2.9.2, “Source Installation Prerequisites”

gogoc
Section 5.1.12.5, “Obtaining an IPv6 Address from a Broker”

gpg
Section 2.1.4.2, “Signature Checking Using GnuPG”

grep
Section 4.6.9, “`mysqldumpslow` — Summarize Slow Query Log Files”
Section 3.3.4.7, “Pattern Matching”

groupadd
Section 18.2.1.1, “Installing an NDB Cluster Binary Release on Linux”
Section 2.7, “Installing MySQL on Solaris”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”

gtar
Section 2.7, “Installing MySQL on Solaris”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”
Section 2.9.2, “Source Installation Prerequisites”

gunzip
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”
Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”

gzip
Section 18.3.3.6, “Defining NDB Cluster Data Nodes”
Section 1.6, “How to Report Bugs or Problems”
Section 2.4, “Installing MySQL on OS X”

H

[index top]

help contents
Section 4.5.1.4, “mysql Client Server-Side Help”

hostname
Section B.3.2.2, “Can't connect to [local] MySQL server”

I

[index top]

icc
Section 2.1.6, “Compiler-Specific Build Characteristics”

ifconfig
Section 5.1.12.1, “Verifying System Support for IPv6”

innochecksum
Section 13.7.2.2, “CHECK TABLE Statement”
Section 4.6.1, “`innochecksum` — Offline InnoDB File Checksum Utility”

MySQL Glossary
Section 4.1, “Overview of MySQL Programs”
Section 1.3, “What Is New in MySQL 5.6”

install.rb

Section 16.2.3.7, “Using MySQL and memcached with Ruby”

iptables

Section 18.5.16.1, “NDB Cluster Security and Networking Issues”

K

[index top]

kill

Section B.3.2.2, “Can't connect to [local] MySQL server”
Section 18.3.3.6, “Defining NDB Cluster Data Nodes”
Section 18.4.1, “ndbd — The NDB Cluster Data Node Daemon”
Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 13.3.7.3, “Restrictions on XA Transactions”
Section 4.10, “Unix Signal Handling in MySQL”

ksh

Section 4.2.1, “Invoking MySQL Programs”
Section 4.2.7, “Setting Environment Variables”

kswapd

Section 18.3.3.6, “Defining NDB Cluster Data Nodes”

L

[index top]

less

Section 4.5.1.2, “mysql Client Commands”
Section 4.5.1.1, “mysql Client Options”

libmemcached

libmemcached Command-Line Utilities

ln

Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”
Section 8.12.3.2, “Using Symbolic Links for MyISAM Tables on Unix”

logger

Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”

ls

Section 6.6, “SELinux”

lsof +L1

Section B.3.3.5, “Where MySQL Stores Temporary Files”

M

[index top]

m4
Section 2.9.2, “Source Installation Prerequisites”

make
Section 2.9.8, “Dealing with Problems Compiling MySQL”
Section 16.2.1, “Installing `memcached`”
Section 2.8, “Installing MySQL on FreeBSD”
Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”
Section 2.13.3, “Problems Using the Perl DBI/DBD Interface”
Section 2.9.2, “Source Installation Prerequisites”

make && make install
Section 18.2.1.4, “Building NDB Cluster from Source on Linux”

make install
Section 18.2.1.4, “Building NDB Cluster from Source on Linux”
Section 16.2.1, “Installing `memcached`”
Section 2.9.7, “MySQL Source-Configuration Options”

make package
Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”
Section 2.9.7, “MySQL Source-Configuration Options”

make test
Section 2.9.5, “Installing MySQL Using a Development Source Tree”
Section 2.13.1, “Installing Perl on Unix”

make VERBOSE=1
Section 2.9.8, “Dealing with Problems Compiling MySQL”

md5
Section 2.1.4.1, “Verifying the MD5 Checksum”

md5.exe
Section 2.1.4.1, “Verifying the MD5 Checksum”

md5sum
Section 2.1.4.1, “Verifying the MD5 Checksum”

memcache
Section 16.2.2.5, “`memcached` Hashing/Distribution Types”
Section 16.2.3.5, “Using MySQL and `memcached` with Python”

memcached
Section 14.20.5.2, “Adapting a memcached Application for the InnoDB memcached Plugin”
Section 14.20.5.1, “Adapting an Existing MySQL Schema for the InnoDB memcached Plugin”
Section 14.20.5.5, “Adapting DML Statements to memcached Operations”
Section 16.2.3.1, “Basic `memcached` Operations”
Section 14.20.1, “Benefits of the InnoDB memcached Plugin”
Section 14.20.5.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”
Section 16.2.2.4, “Data Expiry”
Section 16.2.3, “Developing a `memcached` Application”
Section 16.2.4, “Getting `memcached` Statistics”
Section 14.20.2, “InnoDB memcached Architecture”
Section 14.20, “InnoDB memcached Plugin”
Section 14.20.7, “InnoDB memcached Plugin Internals”

Section 14.14, “InnoDB Startup Options and System Variables”
Section 16.2.1, “Installing memcached”
libmemcached Command-Line Utilities
libmemcached Set Functions
Section 16.2.2.1, “memcached Command-Line Options”
Section 16.2.2.2, “memcached Deployment”
Section 16.2.4.5, “memcached Detail Statistics”
Section 16.2.5, “memcached FAQ”
Section 16.2.4.1, “memcached General Statistics”
Section 16.2.2.5, “memcached Hashing/Distribution Types”
Section 16.2.4.3, “memcached Item Statistics”
Section 16.2.2.9, “memcached Logs”
Section 16.2.4.4, “memcached Size Statistics”
Section 16.2.4.2, “memcached Slabs Statistics”
Section 16.2.2.8, “memcached Thread Support”
Section 16.2.2.7, “Memory Allocation within memcached”
MySQL Glossary
Section 2.9.7, “MySQL Source-Configuration Options”
Section 14.20.5.6, “Performing DML and DDL Statements on the Underlying InnoDB Table”
Section 14.20.4, “Security Considerations for the InnoDB memcached Plugin”
Section 14.20.3, “Setting Up the InnoDB memcached Plugin”
Section 14.20.6, “The InnoDB memcached Plugin and Replication”
Section 14.20.8, “Troubleshooting the InnoDB memcached Plugin”
Section 14.20.5.3, “Tuning InnoDB memcached Plugin Performance”
Section 16.2.3.3, “Using libmemcached with C and C++”
Section 16.2.2, “Using memcached”
Section 16.2.2.6, “Using memcached and DTrace”
Section 16.2.3.2, “Using memcached as a MySQL Caching Layer”
Section 16.2.4.6, “Using memcached-tool”
Section 16.2.3.8, “Using MySQL and memcached with Java”
Section 16.2.3.4, “Using MySQL and memcached with Perl”
Section 16.2.3.6, “Using MySQL and memcached with PHP”
Section 16.2.3.5, “Using MySQL and memcached with Python”
Section 16.2.3.7, “Using MySQL and memcached with Ruby”
Section 16.2, “Using MySQL with memcached”
Section 16.2.2.3, “Using Namespaces”
Section 16.2.3.9, “Using the memcached TCP Text Protocol”
Section 1.3, “What Is New in MySQL 5.6”
Section 14.20.5, “Writing Applications for the InnoDB memcached Plugin”

memcached-1.2.5 directory:

Section 16.2.1, “Installing memcached”

memcached-tool

Section 16.2.4, “Getting memcached Statistics”
Section 16.2.4.6, “Using memcached-tool”

memcapable

Section 14.20.2, “InnoDB memcached Architecture”

memcat

Section 14.20.2, “InnoDB memcached Architecture”
libmemcached Command-Line Utilities

memcp

Section 14.20.2, “InnoDB memcached Architecture”
libmemcached Command-Line Utilities

memflush

Section 14.20.2, “InnoDB memcached Architecture”
`libmemcached` Command-Line Utilities

memrm

Section 14.20.2, “InnoDB memcached Architecture”
`libmemcached` Command-Line Utilities

memslap

`libmemcached` Command-Line Utilities
Section 14.20.5.3, “Tuning InnoDB memcached Plugin Performance”

mgmd

Section 18.2.2.1, “Installing NDB Cluster on Windows from a Binary Release”
Section 18.2, “NDB Cluster Installation”

mkdir

Section 13.1.10, “CREATE DATABASE Statement”

mklink

Section 8.12.3.3, “Using Symbolic Links for Databases on Windows”

more

Section 4.5.1.2, “mysql Client Commands”
Section 4.5.1.1, “mysql Client Options”

msql2mysql

Section 4.7.1, “`msql2mysql` — Convert mSQL Programs for Use with MySQL”
Section 4.1, “Overview of MySQL Programs”
Section 4.8.2, “`replace` — A String-Replacement Utility”
Section 1.3, “What Is New in MySQL 5.6”

mv

Section 5.4.2.5, “Error Log File Flushing and Renaming”
Section 5.4.7, “Server Log Maintenance”
Section 5.4.3, “The General Query Log”

my_print_defaults

Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”
Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”
Section 4.1, “Overview of MySQL Programs”
Section 4.7, “Program Development Utilities”

myisam_ftdump

Section 12.10, “Full-Text Search Functions”
Section 4.6.2, “`myisam_ftdump` — Display Full-Text Index information”
Section 4.1, “Overview of MySQL Programs”

myisamchk

Section 13.7.2.1, “ANALYZE TABLE Statement”
Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 13.7.2.2, “CHECK TABLE Statement”
Section 15.2.3.3, “Compressed Table Characteristics”
Section 15.2.4.1, “Corrupted MyISAM Tables”
Section 7.2, “Database Backup Methods”

Section 5.8.1, “Debugging a MySQL Server”
Section 13.2.2, “DELETE Statement”
Section 15.2.3.2, “Dynamic Table Characteristics”
Section 8.8.2, “EXPLAIN Output Format”
Section 8.11.5, “External Locking”
Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”
Section 7.6.2, “How to Check MyISAM Tables for Errors”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 1.6, “How to Report Bugs or Problems”
Section 8.3.7, “InnoDB and MyISAM Index Statistics Collection”
Section 8.4.6, “Limits on Table Size”
Section 13.7.6.5, “LOAD INDEX INTO CACHE Statement”
Section 19.3.4, “Maintenance of Partitions”
Section 5.8.1.7, “Making a Test Case If You Experience Table Corruption”
Section 15.2.1, “MyISAM Startup Options”
Section 7.6, “MyISAM Table Maintenance and Crash Recovery”
Section 7.6.4, “MyISAM Table Optimization”
Section 15.2.3, “MyISAM Table Storage Formats”
Section 4.6.3.2, “myisamchk Check Options”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.3.6, “myisamchk Memory Usage”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”
Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”
Section 4.6.3.5, “Obtaining Table Information with myisamchk”
Section 8.6.1, “Optimizing MyISAM Queries”
Section 8.6.3, “Optimizing REPAIR TABLE Statements”
Section 4.6.3.4, “Other myisamchk Options”
Section 4.1, “Overview of MySQL Programs”
Section 15.2.4.2, “Problems from Tables Not Being Closed Properly”
Section 13.7.2.5, “REPAIR TABLE Statement”
Section 19.6, “Restrictions and Limitations on Partitioning”
Section 5.1.7, “Server System Variables”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”
Section 13.7.5.23, “SHOW INDEX Statement”
Section 13.7.5.37, “SHOW TABLE STATUS Statement”
Section 15.2.3.1, “Static (Fixed-Length) Table Characteristics”
Section 8.12.1, “System Factors”
Section 21.22, “The INFORMATION_SCHEMA STATISTICS Table”
Section 21.23, “The INFORMATION_SCHEMA TABLES Table”
Section 1.2.2, “The Main Features of MySQL”
Section 15.2, “The MyISAM Storage Engine”
Section 7.6.1, “Using myisamchk for Crash Recovery”
Section 5.8.1.6, “Using Server Logs to Find Causes of Errors in mysqld”
Section 8.12.3.2, “Using Symbolic Links for MyISAM Tables on Unix”
Section B.3.3.3, “What to Do If MySQL Keeps Crashing”

myisamchk *.MYI

Section 7.6.3, “How to Repair MyISAM Tables”

myisamchk tbl_name

Section 7.6.2, “How to Check MyISAM Tables for Errors”

myisamlog

Section 4.6.4, “`myisamlog` — Display MyISAM Log File Contents”

Section 4.1, “Overview of MySQL Programs”

myisampack

Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 15.2.3.3, “Compressed Table Characteristics”
Section 13.1.17, “CREATE TABLE Statement”
Section 8.11.5, “External Locking”
Section 8.4.6, “Limits on Table Size”
Section 15.7.1, “MERGE Table Advantages and Disadvantages”
Section 15.2.3, “MyISAM Table Storage Formats”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.6.3.5, “Obtaining Table Information with myisamchk”
Section 8.4.1, “Optimizing Data Size”
Section 4.1, “Overview of MySQL Programs”
Section 19.6, “Restrictions and Limitations on Partitioning”
Section 13.1.17.6, “Silent Column Specification Changes”
Section 15.7, “The MERGE Storage Engine”
Section 15.2, “The MyISAM Storage Engine”

mysql

Section 1.7.2.4, “'--' as the Start of a Comment”
Section 1.1, “About This Manual”
Section 6.2.7, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 18.5.7.2, “Adding NDB Cluster Data Nodes Online: Basic procedure”
Section 18.5.7.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 18.5.7.1, “Adding NDB Cluster Data Nodes Online: General Issues”
Section 13.1.7.1, “ALTER TABLE Partition Operations”
Section 7.1, “Backup and Recovery Types”
Section 2.5.8.1, “Basic Steps for MySQL Server Deployment with Docker”
Section 13.6.1, “BEGIN ... END Compound Statement”
Section 6.4.1.5, “Client-Side Cleartext Pluggable Authentication”
Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”
Section 18.5.1, “Commands in the NDB Cluster Management Client”
Section 9.6, “Comments”
Section 18.3, “Configuration of NDB Cluster”
Section 10.5, “Configuring Application Character Set and Collation”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 14.8.11.2, “Configuring Non-Persistent Optimizer Statistics Parameters”
Section 3.1, “Connecting to and Disconnecting from the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 5.1.12.4, “Connecting Using IPv6 Nonlocal Host Addresses”
Section 5.1.12.3, “Connecting Using the IPv6 Local Host Address”
Section 10.4, “Connection Character Sets and Collations”
Section 4.2.5, “Connection Transport Protocols”
Section 1.8.1, “Contributors to MySQL”
Section 14.6.1.5, “Converting Tables from MyISAM to InnoDB”
Section 2.11.11, “Copying MySQL Databases to Another Machine”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 3.3.1, “Creating and Selecting a Database”
Section 2.3.4.6, “Customizing the PATH for MySQL Tools”
Section 5.8.2, “Debugging a MySQL Client”
Section 18.3.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”
Section 20.1, “Defining Stored Programs”
Section 18.5.12, “Distributed Privileges Using Shared Grant Tables”
Section 2.12.5, “Downgrade Troubleshooting”
Section 2.12, “Downgrading MySQL”

Section 14.17.2, “Enabling InnoDB Monitors”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 3.2, “Entering Queries”
Section 4.9, “Environment Variables”
Section B.2, “Error Information Interfaces”
Section B.1, “Error Message Sources and Elements”
Section 20.4.2, “Event Scheduler Configuration”
Section 7.3, “Example Backup and Recovery Strategy”
Section 3.6, “Examples of Common Queries”
Section 19.3.3, “Exchanging Partitions and Subpartitions with Tables”
Section 4.5.1.5, “Executing SQL Statements from a Text File”
Chapter 12, *Functions and Operators*
Section 2.4.1, “General Notes on Installing MySQL on OS X”
Section 13.6.7.3, “GET DIAGNOSTICS Statement”
Section 13.7.1.4, “GRANT Statement”
Section 13.8.3, “HELP Statement”
Section B.3.1, “How to Determine What Is Causing a Problem”
Section 14.7.5.3, “How to Minimize and Handle Deadlocks”
Section 1.6, “How to Report Bugs or Problems”
Section 6.1.5, “How to Run MySQL as a Normal User”
Section B.3.2.14, “Ignoring user”
Section 12.16, “Information Functions”
Section 18.2.2.3, “Initial Startup of NDB Cluster on Windows”
Section 14.18.2, “InnoDB Recovery”
Section 2.5.5, “Installing MySQL on Linux Using RPM Packages from Oracle”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”
Section 18.2.1.2, “Installing NDB Cluster from RPM”
Section 18.2.2.1, “Installing NDB Cluster on Windows from a Binary Release”
Section 4.2.1, “Invoking MySQL Programs”
Section 18.1.7.8, “Issues Exclusive to NDB Cluster”
Section 8.2.1.16, “LIMIT Query Optimization”
Section 13.2.6, “LOAD DATA Statement”
Section 13.2.7, “LOAD XML Statement”
Section 7.4.5.1, “Making a Copy of a Database”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 18.5, “Management of NDB Cluster”
Section 8.13.1, “Measuring the Speed of Expressions and Functions”
Section 2.5.8.2, “More Topics on Deploying MySQL Server with Docker”
Section A.11, “MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
Section 4.5.1.2, “mysql Client Commands”
Section 4.5.1.3, “mysql Client Logging”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.4, “mysql Client Server-Side Help”
Section 4.5.1.6, “mysql Client Tips”
MySQL Glossary
Section 2.3.3.1, “MySQL Installer Initial Setup”
Chapter 18, *MySQL NDB Cluster 7.3 and NDB Cluster 7.4*
Section 5.1.13, “MySQL Server Time Zone Support”
Section 18.5.9, “MySQL Server Usage for NDB Cluster”
Section 4.5.1, “`mysql` — The MySQL Command-Line Client”
Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”
Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”
Section 4.4.6, “`mysql_tzinfo_to_sql` — Load the Time Zone Tables”
Section 4.6.7, “`mysqlaccess` — Client for Checking Access Privileges”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 18.5.13, “NDB API Statistics Counters and Variables”
Section 18.6.9, “NDB Cluster Backups With NDB Cluster Replication”
Section 18.5.10.1, “NDB Cluster Disk Data Objects”
Section 18.2.5, “NDB Cluster Example with Tables and Data”
Section 18.6, “NDB Cluster Replication”
Section 18.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”
Section 18.4.5, “`ndb_mgm` — The NDB Cluster Management Client”
Section 18.5.14, “ndbinfo: The NDB Cluster Information Database”
Section 18.4.2, “`ndbinfo_select_all` — Select From ndbinfo Tables”
Section 4.2.2.6, “Option Defaults, Options Expecting Values, and the = Sign”
Section B.3.2.7, “Out of memory”
Section 4.1, “Overview of MySQL Programs”
Section B.3.2.9, “Packet Too Large”
Section 6.4.1.6, “PAM Pluggable Authentication”
Section 6.1.2.4, “Password Hashing in MySQL”
Section 6.2.11, “Pluggable Authentication”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”
Section 13.5, “Prepared Statements”
Section 18.6.5, “Preparing the NDB Cluster for Replication”
Section 4.2.2.4, “Program Option Modifiers”
Section 19.2.3.1, “RANGE COLUMNS partitioning”
Section 2.11.10, “Rebuilding or Repairing Tables or Indexes”
Section 7.4.4, “Reloading Delimited-Text Format Backups”
Section 7.4.2, “Reloading SQL-Format Backups”
Resetting the Root Password: Generic Instructions
Restoring an NDB backup to a later version of NDB Cluster
Restoring to More Nodes Than the Original
Section 13.7.1.6, “REVOKE Statement”
Section 2.10.4, “Securing the Initial MySQL Accounts”
Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”
Section 6.2.10, “Server Handling of Expired Passwords”
Section 5.1.7, “Server System Variables”
Section 5.1.14, “Server-Side Help Support”
Section 6.4.1.4, “SHA-256 Pluggable Authentication”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”
Section 13.7.5.41, “SHOW WARNINGS Statement”
Section 13.6.7.5, “SIGNAL Statement”
Section 6.4.1.8, “Socket Peer-Credential Pluggable Authentication”
Section 4.2.2, “Specifying Program Options”
Section 2.3.4.7, “Starting MySQL as a Windows Service”
Section 18.6.6, “Starting NDB Cluster Replication (Single Replication Channel)”
Section 9.1.1, “String Literals”
Section 2.10.3, “Testing the Server”
Section 11.3.4, “The BLOB and TEXT Types”
Section 18.5.14.17, “The ndbinfo memory_per_fragment Table”
Section 18.5.14.27, “The ndbinfo transporters Table”
Section 20.3.1, “Trigger Syntax and Examples”
Section 14.21.3, “Troubleshooting InnoDB Data Dictionary Operations”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Chapter 3, *Tutorial*
Section 4.10, “Unix Signal Handling in MySQL”
Section 2.11, “Upgrading MySQL”
Section 7.3.2, “Using Backups for Recovery”
Section 17.1.3.3, “Using GTIDs for Failover and Scaleout”
Section 6.4.5.3, “Using MySQL Enterprise Firewall”
Section 3.5, “Using mysql in Batch Mode”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

Section 7.4, “Using mysqldump for Backups”
Section 4.2.2.2, “Using Option Files”
Section 4.2.2.1, “Using Options on the Command Line”
Section 4.2.2.5, “Using Options to Set Program Variables”
Section 5.8.1.6, “Using Server Logs to Find Causes of Errors in mysqld”
Section 18.5.10.2, “Using Symbolic Links with Disk Data Objects”
Section 1.3, “What Is New in MySQL 5.6”
Section 2.3.6, “Windows Postinstallation Procedures”
Section 12.12, “XML Functions”

mysql ...

Section 5.8.1.1, “Compiling MySQL for Debugging”

mysql-server

Section 2.8, “Installing MySQL on FreeBSD”

mysql-test-run.pl

Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.2.2.2, “Using Option Files”

mysql.exe

Section 18.2.2.3, “Initial Startup of NDB Cluster on Windows”
Section 18.2.2.1, “Installing NDB Cluster on Windows from a Binary Release”
Section 4.5.1.6, “mysql Client Tips”

mysql.server

Section 2.5, “Installing MySQL on Linux”
Section 18.2.1.2, “Installing NDB Cluster from RPM”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”
Section 4.6.9, “`mysqldumpslow` — Summarize Slow Query Log Files”
Section 4.1, “Overview of MySQL Programs”
Section 5.1.6, “Server Command Options”
Section 2.10.5, “Starting and Stopping MySQL Automatically”
Section B.3.3.7, “Time Zone Problems”

mysql.server stop

Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”

mysql_config

Section 23.6.1, “Compiling Programs with libmysqld”
Section 2.9.8, “Dealing with Problems Compiling MySQL”
Section 4.7.2, “`mysql_config` — Display Options for Compiling Clients”
Section 4.1, “Overview of MySQL Programs”

mysql_config_editor

Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 4.9, “Environment Variables”
Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”
Section 4.6.3.1, “myisamchk General Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 4.1, “Overview of MySQL Programs”
Section 4.2.2.2, “Using Option Files”
Section 1.3, “What Is New in MySQL 5.6”

mysql_convert_table_format

Section 4.6.11, “mysql_convert_table_format — Convert Tables to Use a Given Storage Engine”
Section 4.1, “Overview of MySQL Programs”
Section 1.3, “What Is New in MySQL 5.6”

mysql_find_rows

Section 4.6.12, “mysql_find_rows — Extract SQL Statements from Files”
Section 4.1, “Overview of MySQL Programs”
Section 1.3, “What Is New in MySQL 5.6”

mysql_fix_extensions

Section 4.6.13, “mysql_fix_extensions — Normalize Table File Name Extensions”
Section 4.1, “Overview of MySQL Programs”
Section 1.3, “What Is New in MySQL 5.6”

mysql_install_db

Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 2.10.1, “Initializing the Data Directory”
Section 2.5.5, “Installing MySQL on Linux Using RPM Packages from Oracle”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”
Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.4.3, “mysql_install_db — Initialize MySQL Data Directory”
Section 18.6.4, “NDB Cluster Replication Schema and Tables”
Section 4.1, “Overview of MySQL Programs”
Section 2.10.1.1, “Problems Running mysql_install_db”
Section 5.1.6, “Server Command Options”
Section 5.1.10, “Server SQL Modes”
Section 5.1.7, “Server System Variables”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 5.1.2.2, “Using a Sample Default Server Configuration File”
Section 1.3, “What Is New in MySQL 5.6”

mysql_plugin

Section 4.4.4, “mysql_plugin — Configure MySQL Server Plugins”
Section 4.1, “Overview of MySQL Programs”

mysql_secure_installation

Section 2.5.7, “Installing MySQL on Linux from the Native Software Repositories”
Section 2.5.5, “Installing MySQL on Linux Using RPM Packages from Oracle”
Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository”
Section 2.7.1, “Installing MySQL on Solaris Using a Solaris PKG”
Section 4.4.3, “mysql_install_db — Initialize MySQL Data Directory”
Section 4.4.5, “mysql_secure_installation — Improve MySQL Installation Security”
Section 4.1, “Overview of MySQL Programs”
Section 2.10.4, “Securing the Initial MySQL Accounts”

mysql_setpermission

Section 1.8.1, “Contributors to MySQL”

Section 4.6.14, “mysql_setpermission — Interactively Set Permissions in Grant Tables”
Section 4.1, “Overview of MySQL Programs”
Section 1.3, “What Is New in MySQL 5.6”

mysql_setpermissions

Section 4.6.14, “mysql_setpermission — Interactively Set Permissions in Grant Tables”

mysql_stmt_execute()

Section 5.1.9, “Server Status Variables”

mysql_stmt_prepare()

Section 5.1.9, “Server Status Variables”

mysql_tzinfo_to_sql

Section 5.1.13, “MySQL Server Time Zone Support”
Section 4.4.6, “mysql_tzinfo_to_sql — Load the Time Zone Tables”
Section 4.1, “Overview of MySQL Programs”

mysql_upgrade

Section 11.2.5, “2-Digit YEAR(2) Limitations and Migrating to 4-Digit YEAR”
Section 6.2.7, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 13.1.1, “ALTER DATABASE Statement”
Section 2.12.4, “Downgrading Binary and Package-based Installations on Unix/Linux”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 6.2.3, “Grant Tables”
Section B.3.2.14, “Ignoring user”
Section 6.4.1.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.1, “Overview of MySQL Programs”
Section 6.1.2.4, “Password Hashing in MySQL”
Section 2.11.10, “Rebuilding or Repairing Tables or Indexes”
Section 2.5.2, “Replacing a Third-Party Distribution of MySQL Using the MySQL Yum Repository”
Section 17.1.3.4, “Restrictions on Replication with GTIDs”
Section 5.1.6, “Server Command Options”
Section 17.4.3, “Upgrading a Replication Setup”
Section 18.2.7, “Upgrading and Downgrading NDB Cluster”
Section 2.11.4, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 2.11.8, “Upgrading MySQL on Windows”
Section 2.11.5, “Upgrading MySQL with the MySQL Yum Repository”
Section 1.3, “What Is New in MySQL 5.6”

mysql_waitpid

Section 4.6.15, “mysql_waitpid — Kill Process and Wait for Its Termination”
Section 4.1, “Overview of MySQL Programs”
Section 1.3, “What Is New in MySQL 5.6”

mysql_waitpid()

Section 4.6.15, “mysql_waitpid — Kill Process and Wait for Its Termination”

mysql_zap

Section 4.6.16, “mysql_zap — Kill Processes That Match a Pattern”
Section 4.1, “Overview of MySQL Programs”
Section 1.3, “What Is New in MySQL 5.6”

mysqlaccess

Section 1.8.1, “Contributors to MySQL”
Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”

Section 4.1, “Overview of MySQL Programs”
Section 1.3, “What Is New in MySQL 5.6”

mysqladmin

Section 6.2.9, “Assigning Account Passwords”
Section 17.3.1.1, “Backing Up a Replica Using mysqldump”
Section B.3.2.2, “Can't connect to [local] MySQL server”
Section 6.4.1.5, “Client-Side Cleartext Pluggable Authentication”
Section 5.1.1, “Configuring the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 10.4, “Connection Character Sets and Collations”
Section 1.8.1, “Contributors to MySQL”
Section 13.1.10, “CREATE DATABASE Statement”
Section 2.3.4.6, “Customizing the PATH for MySQL Tools”
Section 5.8.1, “Debugging a MySQL Server”
Section 17.1.3.5, “Disabling GTID Transactions”
Section 13.1.21, “DROP DATABASE Statement”
Section 13.7.6.3, “FLUSH Statement”
Section 2.4.1, “General Notes on Installing MySQL on OS X”
Section B.3.1, “How to Determine What Is Causing a Problem”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 1.6, “How to Report Bugs or Problems”
Section 2.5.5, “Installing MySQL on Linux Using RPM Packages from Oracle”
Section A.11, “MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 2.3.3.1, “MySQL Installer Initial Setup”
Section 5.4, “MySQL Server Logs”
Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.1, “Overview of MySQL Programs”
Section 22.12.8, “Performance Schema Connection Attribute Tables”
Section 6.2.11, “Pluggable Authentication”
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.7.3, “Running Multiple MySQL Instances on Unix”
Section 2.10.4, “Securing the Initial MySQL Accounts”
Section 6.2.10, “Server Handling of Expired Passwords”
Section 17.1.3.2, “Setting Up Replication Using GTIDs”
Section 2.3.4.7, “Starting MySQL as a Windows Service”
Section 2.3.4.5, “Starting MySQL from the Windows Command Line”
Section 2.10.3, “Testing the Server”
Section 1.2.2, “The Main Features of MySQL”
Section 5.1.15, “The Server Shutdown Process”
Section 2.11.8, “Upgrading MySQL on Windows”
Section 4.2.2.2, “Using Option Files”
Section 4.2.2.1, “Using Options on the Command Line”
Section B.3.3.3, “What to Do If MySQL Keeps Crashing”

mysqladmin debug

Section 5.8.1, “Debugging a MySQL Server”
Section 20.4.5, “Event Scheduler Status”
Section 13.7.1.4, “GRANT Statement”
Section 6.2.2, “Privileges Provided by MySQL”

mysqladmin extended-status

Section 13.2.5.3, “INSERT DELAYED Statement”
Section 13.7.5.36, “SHOW STATUS Statement”

mysqladmin flush-hosts

Section 5.1.11.2, “DNS Lookups and the Host Cache”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”

mysqladmin flush-logs

Section 7.3.3, “Backup Strategy Summary”
Section 5.4.2.5, “Error Log File Flushing and Renaming”
Section 7.3.1, “Establishing a Backup Policy”
Section 5.4.7, “Server Log Maintenance”
Section 5.4.4, “The Binary Log”
Section 17.2.2.1, “The Relay Log”

mysqladmin flush-privileges

Section 2.11.11, “Copying MySQL Databases to Another Machine”
Section 6.2.3, “Grant Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 5.1.6, “Server Command Options”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 6.2.8, “When Privilege Changes Take Effect”

mysqladmin flush-tables

Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 8.11.5, “External Locking”
Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section 8.12.4.1, “How MySQL Uses Memory”
Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 7.6.1, “Using myisamchk for Crash Recovery”

mysqladmin flush-xxx

Section 6.2.7, “Adding Accounts, Assigning Privileges, and Dropping Accounts”

mysqladmin kill

Section B.3.3.4, “How MySQL Handles a Full Disk”
Section 13.7.6.4, “KILL Statement”
Section 12.15, “Locking Functions”
Section B.3.2.8, “MySQL server has gone away”
Section 6.2.2, “Privileges Provided by MySQL”

mysqladmin password

Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 6.2.10, “Server Handling of Expired Passwords”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”

mysqladmin processlist

Section 8.14.1, “Accessing the Process List”
Section 6.2.7, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 13.7.6.4, “KILL Statement”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 6.2.2, “Privileges Provided by MySQL”

mysqladmin processlist status

Section 5.8.1, “Debugging a MySQL Server”

mysqladmin refresh

Section 6.2.7, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 8.4.3.1, “How MySQL Opens and Closes Tables”

Section 5.4.7, “Server Log Maintenance”

mysqladmin reload

Section 6.2.7, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 6.2.3, “Grant Tables”
Section 1.6, “How to Report Bugs or Problems”
Section 4.6.7, “`mysqlaccess` — Client for Checking Access Privileges”
Section 5.1.6, “Server Command Options”
Section 6.2.13, “Setting Account Resource Limits”
Section 6.2.8, “When Privilege Changes Take Effect”

mysqladmin reload version

Section 1.6, “How to Report Bugs or Problems”

mysqladmin shutdown

Section 6.2.6, “Access Control, Stage 2: Request Verification”
Section 18.5.7.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 5.8.1.2, “Creating Trace Files”
Section 13.7.1.4, “GRANT Statement”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 6.1.5, “How to Run MySQL as a Normal User”
Section 2.4.2, “Installing MySQL on OS X Using Native Packages”
Section 5.8.1.7, “Making a Test Case If You Experience Table Corruption”
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.4.1.29, “Replication and Temporary Tables”
Section 18.2.6, “Safe Shutdown and Restart of NDB Cluster”
Section 2.3.4.7, “Starting MySQL as a Windows Service”
Section 5.1.15, “The Server Shutdown Process”
Section B.3.3.3, “What to Do If MySQL Keeps Crashing”

mysqladmin status

Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”

mysqladmin variables

Section B.3.2.8, “MySQL server has gone away”
Section 13.7.5.40, “SHOW VARIABLES Statement”

mysqladmin variables extended-status processlist

Section 1.6, “How to Report Bugs or Problems”

mysqladmin ver

Section 5.8.1.1, “Compiling MySQL for Debugging”

mysqladmin version

Section B.3.2.2, “Can't connect to [local] MySQL server”
Section 1.6, “How to Report Bugs or Problems”
Section B.3.2.8, “MySQL server has gone away”
Section 2.10.3, “Testing the Server”
Section B.3.3.3, “What to Do If MySQL Keeps Crashing”

mysqlanalyze

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

mysqlbackup

Section 7.1, “Backup and Recovery Types”
Section 17.1.1.6, “Creating a Data Snapshot Using Raw Data Files”
Section 14.18.1, “InnoDB Backup”
MySQL Glossary
Section 4.5.4, “`mysqldump` — A Database Backup Program”

mysqlbinlog

Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 13.7.6.1, “BINLOG Statement”
Command Probes
Section 17.1.3.1, “GTID Concepts”
Section 17.4.5, “How to Report Replication Bugs or Problems”
Section 14.18.2, “InnoDB Recovery”
Section B.3.7, “Known Issues in MySQL”
Section 12.20, “Miscellaneous Functions”
Section 4.5.1.1, “mysql Client Options”
MySQL Glossary
MySQL Server Options for NDB Cluster
Section 4.6.8.1, “mysqlbinlog Hex Dump Format”
Section 4.6.8.2, “mysqlbinlog Row Event Display”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.1, “Overview of MySQL Programs”
Section 22.12.8, “Performance Schema Connection Attribute Tables”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”
Section 7.5.2, “Point-in-Time Recovery Using Event Positions”
Section 7.5.1, “Point-in-Time Recovery Using Event Times”
Section 18.6.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.4.1.18, “Replication and LOAD DATA”
Section 17.4.1.35, “Replication and Variables”
Section 17.1.3.4, “Restrictions on Replication with GTIDs”
Section 13.7.5.3, “SHOW BINLOG EVENTS Statement”
Section 13.7.5.33, “SHOW RELAYLOG EVENTS Statement”
Section 4.6.8.4, “Specifying the mysqlbinlog Server ID”
Section 13.4.2.5, “START SLAVE Statement”
Section 5.4.4, “The Binary Log”
Section 5.4.3, “The General Query Log”
Section 17.2.2.1, “The Relay Log”
Section 17.1.2.2, “Usage of Row-Based Logging and Replication”
Section 7.3.2, “Using Backups for Recovery”
Section 17.1.3.3, “Using GTIDs for Failover and Scaleout”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”
Section 1.3, “What Is New in MySQL 5.6”

mysqlbinlog binary-log-file | mysql

Section 5.8.1.7, “Making a Test Case If You Experience Table Corruption”

mysqlbinlog|mysql

Section B.3.7, “Known Issues in MySQL”

mysqlbug

Section 4.4.2, “`mysqlbug` — Generate Bug Report”
Section 1.3, “What Is New in MySQL 5.6”

mysqlcheck

Section 13.1.1, “ALTER DATABASE Statement”

Section 6.4.1.5, “Client-Side Cleartext Pluggable Authentication”
Section 10.4, “Connection Character Sets and Collations”
Section 19.3.4, “Maintenance of Partitions”
Section 9.2.4, “Mapping of Identifiers to File Names”
Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”
Section 7.6, “MyISAM Table Maintenance and Crash Recovery”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.1, “Overview of MySQL Programs”
Section 2.11.10, “Rebuilding or Repairing Tables or Indexes”
Section 19.6, “Restrictions and Limitations on Partitioning”
Section 5.1.7, “Server System Variables”
Section 1.2.2, “The Main Features of MySQL”
Section 15.2, “The MyISAM Storage Engine”

mysqld

Section 1.1, “About This Manual”
Section 8.2.1.19, “Avoiding Full Table Scans”
Section 2.5.8.1, “Basic Steps for MySQL Server Deployment with Docker”
Section 17.1.4.4, “Binary Log Options and Variables”
Section 5.4.4.1, “Binary Logging Formats”
Section 18.2.1.4, “Building NDB Cluster from Source on Linux”
Section B.3.2.2, “Can't connect to [local] MySQL server”
Section B.3.2.12, “Can't create/write to file”
Section B.3.2.16, “Can't initialize character set”
Section B.3.2.4, “Client does not support authentication protocol”
Section 18.5.1, “Commands in the NDB Cluster Management Client”
Section 9.6, “Comments”
Section B.3.2.10, “Communication Errors and Aborted Connections”
Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”
Section 5.8.1.1, “Compiling MySQL for Debugging”
Section 18.3, “Configuration of NDB Cluster”
Section 2.9.6, “Configuring SSL Library Support”
Section 5.1.1, “Configuring the Server”
Section 5.1.11.1, “Connection Interfaces”
Section 14.20.5.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”
Section 15.2.4.1, “Corrupted MyISAM Tables”
Section 13.1.17, “CREATE TABLE Statement”
Section 5.8.1.2, “Creating Trace Files”
Section 14.7.5, “Deadlocks in InnoDB”
Section 5.8.1, “Debugging a MySQL Server”
Section 5.8.1.4, “Debugging mysqld under gdb”
Section 18.3.3.6, “Defining NDB Cluster Data Nodes”
Section 18.3.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”
Section 17.1.3.5, “Disabling GTID Transactions”
Section 5.1.11.2, “DNS Lookups and the Host Cache”
Section 14.6.5, “Doublewrite Buffer”
Section 14.10.1, “Enabling File Formats”
Section 14.17.2, “Enabling InnoDB Monitors”
Section 4.9, “Environment Variables”
Section 5.4.2.5, “Error Log File Flushing and Renaming”
Section 5.4.2.2, “Error Logging on Unix and Unix-Like Systems”
Section 5.4.2.1, “Error Logging on Windows”
Section 5.4.2.3, “Error Logging to the System Log”
Section 8.11.5, “External Locking”
Section B.3.2.17, “File Not Found and Similar Errors”
Section 14.6.3.2, “File-Per-Table Tablespaces”
Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”

Section 14.21.2, “Forcing InnoDB Recovery”
Section 18.6.2, “General Requirements for NDB Cluster Replication”
Section 8.14.3, “General Thread States”
Section 16.1.3, “Handling MySQL Recovery with ZFS”
Section 8.12.4.1, “How MySQL Uses Memory”
Section B.3.1, “How to Determine What Is Causing a Problem”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 1.6, “How to Report Bugs or Problems”
Section 6.1.5, “How to Run MySQL as a Normal User”
Section 9.2.3, “Identifier Case Sensitivity”
Section B.3.2.14, “Ignoring user”
Section 18.6.8, “Implementing Failover with NDB Cluster Replication”
Section 12.16, “Information Functions”
Section 18.2.3, “Initial Configuration of NDB Cluster”
Section 18.2.4, “Initial Startup of NDB Cluster”
Section 14.18.1, “InnoDB Backup”
Section 14.12.1, “InnoDB Disk I/O”
Section 14.20.2, “InnoDB memcached Architecture”
Section 14.18.2, “InnoDB Recovery”
Section 14.8.1, “InnoDB Startup Configuration”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.21, “InnoDB Troubleshooting”
Section 13.2.5.2, “INSERT ... ON DUPLICATE KEY UPDATE Statement”
Section 13.2.5.3, “INSERT DELAYED Statement”
Section 13.2.5, “INSERT Statement”
Section 18.2.1, “Installation of NDB Cluster on Linux”
Section 18.2.1.1, “Installing an NDB Cluster Binary Release on Linux”
Section 2.5.5, “Installing MySQL on Linux Using RPM Packages from Oracle”
Section 2.4.2, “Installing MySQL on OS X Using Native Packages”
Section 2.7, “Installing MySQL on Solaris”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”
Section 18.2.1.2, “Installing NDB Cluster from RPM”
Section 18.2.2.1, “Installing NDB Cluster on Windows from a Binary Release”
Section 18.2.2.4, “Installing NDB Cluster Processes as Windows Services”
Section 18.1.7.8, “Issues Exclusive to NDB Cluster”
Section 13.7.6.4, “KILL Statement”
Section 18.6.3, “Known Issues in NDB Cluster Replication”
Section 13.2.6, “LOAD DATA Statement”
Section 5.8.1.7, “Making a Test Case If You Experience Table Corruption”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 18.5, “Management of NDB Cluster”
Section 12.20, “Miscellaneous Functions”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 14.6.1.4, “Moving or Copying InnoDB Tables”
Section 15.2.1, “MyISAM Startup Options”
Section 4.6.3.2, “myisamchk Check Options”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”
Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section A.1, “MySQL 5.6 FAQ: General”
Section A.11, “MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
Section A.3, “MySQL 5.6 FAQ: Server SQL Mode”
Section 4.5.1.6, “mysql Client Tips”
MySQL Glossary
Section 2.3.1, “MySQL Installation Layout on Microsoft Windows”
Chapter 18, *MySQL NDB Cluster 7.3 and NDB Cluster 7.4*
Chapter 5, *MySQL Server Administration*

Section B.3.2.8, “MySQL server has gone away”
Section 5.4, “MySQL Server Logs”
MySQL Server Options for NDB Cluster
Section 5.1.13, “MySQL Server Time Zone Support”
Section 18.5.9, “MySQL Server Usage for NDB Cluster”
Section 18.1.6, “MySQL Server Using InnoDB Compared with NDB Cluster”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 1.7, “MySQL Standards Compliance”
Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”
Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”
Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.3.1, “`mysqld` — The MySQL Server”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”
Section 18.5.13, “NDB API Statistics Counters and Variables”
Section 18.5.16.3, “NDB Cluster and MySQL Security Procedures”
Section 18.2.8.1, “NDB Cluster Auto-Installer Requirements”
Section 18.3.3.1, “NDB Cluster Configuration: Basic Example”
Section 18.1.1, “NDB Cluster Core Concepts”
Section 18.2, “NDB Cluster Installation”
Section 18.3.2.5, “NDB Cluster mysqld Option and Variable Reference”
Section 18.1.2, “NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions”
Section 18.1, “NDB Cluster Overview”
Section 18.4, “NDB Cluster Programs”
Section 18.6, “NDB Cluster Replication”
Section 18.6.11, “NDB Cluster Replication Conflict Resolution”
Section 18.6.4, “NDB Cluster Replication Schema and Tables”
Section 18.6.10, “NDB Cluster Replication: Bidirectional and Circular Replication”
NDB Cluster Status Variables
NDB Cluster System Variables
Section 18.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”
Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 18.4.26, “`ndb_show_tables` — Display List of NDB Tables”
Section 18.5.14, “ndbinfo: The NDB Cluster Information Database”
Section 13.7.2.4, “OPTIMIZE TABLE Statement”
Section B.3.5, “Optimizer-Related Issues”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 23.6.3, “Options with the Embedded Server”
Section 18.1.5.1, “Options, Variables, and Parameters Added, Deprecated or Removed in NDB 7.3”
Section 18.1.5.2, “Options, Variables, and Parameters Added, Deprecated or Removed in NDB 7.4”
Section 4.1, “Overview of MySQL Programs”
Section 18.3.2, “Overview of NDB Cluster Configuration Parameters, Options, and Variables”
Section B.3.2.9, “Packet Too Large”
Section 22.3, “Performance Schema Startup Configuration”
Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 18.6.5, “Preparing the NDB Cluster for Replication”
Section 15.2.4.2, “Problems from Tables Not Being Closed Properly”
Section 2.10.1.1, “Problems Running mysql_install_db”
Section B.3.3.1, “Problems with File Permissions”
Section 4.2.2.4, “Program Option Modifiers”
Section 8.10.3.3, “Query Cache Configuration”
Section 18.3.3.2, “Recommended Starting Configuration for NDB Cluster”
Section 14.6.6, “Redo Log”

Section 13.7.2.5, “REPAIR TABLE Statement”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.1.4.1, “Replication and Binary Logging Option and Variable Reference”
Section 17.1.4, “Replication and Binary Logging Options and Variables”
Section 17.4.1.26, “Replication and Source or Replica Shutdowns”
Section 17.2.2.2, “Replication Metadata Repositories”
Section 17.1.4.2, “Replication Source Options and Variables”
Section 13.4.2.3, “RESET SLAVE Statement”
Resetting the Root Password: Unix and Unix-Like Systems
Resetting the Root Password: Windows Systems
Section 4.7.4, “`resolve_stack_dump` — Resolve Numeric Stack Trace Dump to Symbols”
Restoring an NDB backup to a later version of NDB Cluster
Restoring an NDB backup to a previous version of NDB Cluster
Section 17.1.3.4, “Restrictions on Replication with GTIDs”
Section B.3.4.5, “Rollback Failure for Nontransactional Tables”
Section 5.7, “Running Multiple MySQL Instances on One Machine”
Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”
Section 6.1.4, “Security-Related mysqld Options and Variables”
Section 13.2.9.1, “SELECT ... INTO Statement”
Section 2.3.4.3, “Selecting a MySQL Server Type”
Section 6.6, “SELinux”
Section 4.3, “Server and Server-Startup Programs”
Section 10.3.2, “Server Character Set and Collation”
Section 5.1.6, “Server Command Options”
Section 5.4.7, “Server Log Maintenance”
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”
Section 10.12, “Setting the Error Message Language”
Section 6.6.5.1, “Setting the TCP Port Context for mysqld”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”
Section 17.1.3.2, “Setting Up Replication Using GTIDs”
Section 14.20.3, “Setting Up the InnoDB memcached Plugin”
Section 13.7.5.16, “SHOW ENGINE Statement”
Section 4.2.2, “Specifying Program Options”
Section 2.10.5, “Starting and Stopping MySQL Automatically”
Section 5.7.2.2, “Starting Multiple MySQL Instances as Windows Services”
Section 5.7.2.1, “Starting Multiple MySQL Instances at the Windows Command Line”
Section 2.3.4.7, “Starting MySQL as a Windows Service”
Section 2.3.4.5, “Starting MySQL from the Windows Command Line”
Section 18.6.6, “Starting NDB Cluster Replication (Single Replication Channel)”
Section 1.8.5, “Supporters of MySQL”
Section 17.3.7, “Switching Sources During Failover”
Section 8.11.2, “Table Locking Issues”
Section B.3.2.18, “Table-Corruption Issues”
Section 2.3.4.8, “Testing The MySQL Installation”
Section 2.10.3, “Testing the Server”
Section 5.4.4, “The Binary Log”
Section 15.6, “The BLACKHOLE Storage Engine”
Section 5.8.3, “The DBUG Package”
Section 5.4.6, “The DDL Log”
Section 5.4.2, “The Error Log”
Section 5.4.3, “The General Query Log”
Section 14.20.6, “The InnoDB memcached Plugin and Replication”
Section 15.2, “The MyISAM Storage Engine”
Section 8.10.3, “The MySQL Query Cache”
Section 5.1, “The MySQL Server”
Section 18.5.14.19, “The ndbinfo operations_per_fragment Table”
Section 5.4.5, “The Slow Query Log”

Section B.3.3.7, “Time Zone Problems”
Section B.3.2.6, “Too many connections”
Section 5.8.4, “Tracing mysqld Using DTrace”
Section 2.3.5, “Troubleshooting a Microsoft Windows MySQL Server Installation”
Section 14.21.1, “Troubleshooting InnoDB I/O Problems”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”
Section 4.10, “Unix Signal Handling in MySQL”
Section 2.11.9, “Upgrade Troubleshooting”
Section 2.11.8, “Upgrading MySQL on Windows”
Section 5.8.1.5, “Using a Stack Trace”
Section 7.6.1, “Using myisamchk for Crash Recovery”
Section 4.2.2.2, “Using Option Files”
Section 5.8.1.6, “Using Server Logs to Find Causes of Errors in mysqld”
Section 8.12.3.3, “Using Symbolic Links for Databases on Windows”
Section 8.12.3.2, “Using Symbolic Links for MyISAM Tables on Unix”
Section 18.2.8.2, “Using the NDB Cluster Auto-Installer”
Section 18.6.7, “Using Two Replication Channels for NDB Cluster Replication”
Section 5.8.1.3, “Using WER with PDB to create a Windows crashdump”
Section 18.1.4.2, “What is New in NDB Cluster 7.4”
Section B.3.3.3, “What to Do If MySQL Keeps Crashing”
Section 6.2.8, “When Privilege Changes Take Effect”
Section B.3.3.5, “Where MySQL Stores Temporary Files”
Section 2.1.2, “Which MySQL Version and Distribution to Install”

mysqld mysqld.trace

Section 5.8.1.2, “Creating Trace Files”

mysqld-debug

Section 5.8.1.2, “Creating Trace Files”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”
Section 4.3.1, “`mysqld` — The MySQL Server”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 2.3.4.3, “Selecting a MySQL Server Type”

mysqld.exe

Section 18.2.2.3, “Initial Startup of NDB Cluster on Windows”
Section 18.2.2.1, “Installing NDB Cluster on Windows from a Binary Release”
Section 18.2.2.4, “Installing NDB Cluster Processes as Windows Services”

mysqld_multi

Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.1, “Overview of MySQL Programs”
Section 5.7.3, “Running Multiple MySQL Instances on Unix”

mysqld_safe

Section 18.5.7.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 5.8.1.1, “Compiling MySQL for Debugging”
Section 5.1.1, “Configuring the Server”
Section 17.1.3.5, “Disabling GTID Transactions”
Section 8.12.4.2, “Enabling Large Page Support”
Section 5.4.2.2, “Error Logging on Unix and Unix-Like Systems”
Section 5.4.2.3, “Error Logging to the System Log”
Section B.3.2.17, “File Not Found and Similar Errors”
Section B.3.3.6, “How to Protect or Change the MySQL Unix Socket File”
Section 14.21, “InnoDB Troubleshooting”
Section 18.2.1.2, “Installing NDB Cluster from RPM”

Section 6.1.3, “Making MySQL Secure Against Attackers”
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
Section 5.1.13, “MySQL Server Time Zone Support”
Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”
Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 18.5.16.3, “NDB Cluster and MySQL Security Procedures”
Section 4.2.2.6, “Option Defaults, Options Expecting Values, and the = Sign”
Section 4.1, “Overview of MySQL Programs”
Section B.3.2.9, “Packet Too Large”
Section B.3.3.1, “Problems with File Permissions”
Section 5.7, “Running Multiple MySQL Instances on One Machine”
Section 5.7.3, “Running Multiple MySQL Instances on Unix”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 17.1.3.2, “Setting Up Replication Using GTIDs”
Section 2.10.5, “Starting and Stopping MySQL Automatically”
Section 2.10.2, “Starting the Server”
Section 2.10.3, “Testing the Server”
Section 5.4.2, “The Error Log”
Section B.3.3.7, “Time Zone Problems”
Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”
Section 5.1.2.2, “Using a Sample Default Server Configuration File”
Section 4.2.2.2, “Using Option Files”

mysqldump

Section 11.2.5, “2-Digit YEAR(2) Limitations and Migrating to 4-Digit YEAR”
Section 17.3.1.1, “Backing Up a Replica Using mysqldump”
Section 17.3.1.3, “Backing Up a Source or Replica by Making It Read Only”
Chapter 7, *Backup and Recovery*
Section 7.1, “Backup and Recovery Types”
Section 7.3.3, “Backup Strategy Summary”
Section 14.5.1, “Buffer Pool”
Section 8.5.5, “Bulk Data Loading for InnoDB Tables”
Section 2.11.3, “Changes in MySQL 5.6”
Section 6.4.1.5, “Client-Side Cleartext Pluggable Authentication”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 4.2.5, “Connection Transport Protocols”
Section 1.8.1, “Contributors to MySQL”
Section 10.9.8, “Converting Between 3-Byte and 4-Byte Unicode Character Sets”
Section 7.4.5.2, “Copy a Database from one Server to Another”
Section 2.11.11, “Copying MySQL Databases to Another Machine”
Section 13.1.17, “CREATE TABLE Statement”
Section 17.1.1.5, “Creating a Data Snapshot Using mysqldump”
Section 17.1.1.6, “Creating a Data Snapshot Using Raw Data Files”
Section 14.6.1.1, “Creating InnoDB Tables”
Section 2.3.4.6, “Customizing the PATH for MySQL Tools”
Section 7.2, “Database Backup Methods”
Section 14.12.4, “Defragmenting a Table”
Section 18.5.12, “Distributed Privileges Using Shared Grant Tables”
Section 2.12.3, “Downgrade Notes”
Section 2.12.5, “Downgrade Troubleshooting”
Section 2.12.4, “Downgrading Binary and Package-based Installations on Unix/Linux”
Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 7.4.1, “Dumping Data in SQL Format with mysqldump”
Section 7.4.5.3, “Dumping Stored Programs”
Section 7.4.5.4, “Dumping Table Definitions and Content Separately”

Section 7.3.1, “Establishing a Backup Policy”
Section 7.3, “Example Backup and Recovery Strategy”
Section 13.1.17.5, “FOREIGN KEY Constraints”
Section 1.6, “How to Report Bugs or Problems”
Section 17.1.1, “How to Set Up Replication”
Section 9.2.3, “Identifier Case Sensitivity”
Section 4.6.1, “`innochecksum` — Offline InnoDB File Checksum Utility”
Section 14.18.1, “InnoDB Backup”
Section 2.6, “Installing MySQL Using Unbreakable Linux Network (ULN)”
Section 13.2.6, “LOAD DATA Statement”
Section 13.2.7, “LOAD XML Statement”
Section 7.4.5.1, “Making a Copy of a Database”
Section 9.2.4, “Mapping of Identifiers to File Names”
Section 14.6.1.4, “Moving or Copying InnoDB Tables”
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
Section 4.5.1.1, “mysql Client Options”
Section 5.4, “MySQL Server Logs”
Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”
Section 7.4.5, “mysqldump Tips”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 18.2.5, “NDB Cluster Example with Tables and Data”
Section 18.1, “NDB Cluster Overview”
Section 18.5.14, “ndbinfo: The NDB Cluster Information Database”
Section 17.1.1.4, “Obtaining the Replication Source Binary Log Coordinates”
Section 18.5.8, “Online Backup of NDB Cluster”
Section 4.1, “Overview of MySQL Programs”
Section 22.12.8, “Performance Schema Connection Attribute Tables”
Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 18.6.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”
Section 18.6.5, “Preparing the NDB Cluster for Replication”
Section 6.2.2, “Privileges Provided by MySQL”
Section B.3.4.8, “Problems with Floating-Point Values”
Section 14.8.10, “Purge Configuration”
Section 2.11.10, “Rebuilding or Repairing Tables or Indexes”
Section 7.4.4, “Reloading Delimited-Text Format Backups”
Section 7.4.2, “Reloading SQL-Format Backups”
Section 17.3.5, “Replicating Different Databases to Different Replicas”
Restoring an NDB backup to a later version of NDB Cluster
Restoring an NDB backup to a previous version of NDB Cluster
Restoring to More Nodes Than the Original
Section 22.19, “Restrictions on Performance Schema”
Section 17.1.3.4, “Restrictions on Replication with GTIDs”
Section 20.9, “Restrictions on Views”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.4.7, “Server Log Maintenance”
Section 5.1.10, “Server SQL Modes”
Section 5.1.7, “Server System Variables”
Section 17.1.1.8, “Setting Up Replication with Existing Data”
Section B.3.4.7, “Solving Problems with No Matching Rows”
Section 4.2.2, “Specifying Program Options”
Section 2.3.4.7, “Starting MySQL as a Windows Service”
Section 11.3.4, “The BLOB and TEXT Types”
Section 14.20.6, “The InnoDB memcached Plugin and Replication”
Section 1.2.2, “The Main Features of MySQL”
Section 14.6.3.1, “The System Tablespace”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 13.7.3.4, “UNINSTALL PLUGIN Statement”
Section 17.4.3, “Upgrading a Replication Setup”

Section 2.11.4, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 17.1.3.3, “Using GTIDs for Failover and Scaleout”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”
Section 7.4, “Using mysqldump for Backups”
Section 4.2.2.2, “Using Option Files”
Section 17.3.1, “Using Replication for Backups”
Section 17.3.3, “Using Replication with Different Source and Replica Storage Engines”
Section 1.3, “What Is New in MySQL 5.6”
Section 12.12, “XML Functions”

mysqldump mysql

Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”

mysqldumpslow

Section 4.6.9, “`mysqldumpslow` — Summarize Slow Query Log Files”
Section 4.1, “Overview of MySQL Programs”
Section 5.4.5, “The Slow Query Log”

mysqlhotcopy

Chapter 7, *Backup and Recovery*
Section 7.1, “Backup and Recovery Types”
Section 1.8.1, “Contributors to MySQL”
Section 7.2, “Database Backup Methods”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”
Section 4.1, “Overview of MySQL Programs”
Section 1.3, “What Is New in MySQL 5.6”

mysqlimport

Section 7.1, “Backup and Recovery Types”
Section 10.4, “Connection Character Sets and Collations”
Section 2.11.11, “Copying MySQL Databases to Another Machine”
Section 7.2, “Database Backup Methods”
Section 2.12.5, “Downgrade Troubleshooting”
Section 13.2.6, “LOAD DATA Statement”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.1, “Overview of MySQL Programs”
Section 7.4.4, “Reloading Delimited-Text Format Backups”
Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”

MySQLInstallerConsole.exe

Section 2.3.3.5, “MySQLInstallerConsole Reference”

mysqloptimize

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

mysqlrepair

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

mysqlshow

Section 6.4.1.5, “Client-Side Cleartext Pluggable Authentication”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 10.4, “Connection Character Sets and Collations”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.1, “Overview of MySQL Programs”
Section 13.7.5.15, “SHOW DATABASES Statement”

Section 13.7.5.23, “SHOW INDEX Statement”
Section 13.7.5.37, “SHOW TABLE STATUS Statement”
Section 2.3.4.8, “Testing The MySQL Installation”
Section 2.10.3, “Testing the Server”
Section 2.3.6, “Windows Postinstallation Procedures”

mysqlshow db_name

Section 13.7.5.38, “SHOW TABLES Statement”

mysqlshow db_name tbl_name

Section 13.7.5.6, “SHOW COLUMNS Statement”

mysqlslap

Section 6.4.1.5, “Client-Side Cleartext Pluggable Authentication”
Section 14.16.1, “Monitoring InnoDB Mutex Waits Using Performance Schema”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”
Section 4.1, “Overview of MySQL Programs”
Section 8.13.3, “Using Your Own Benchmarks”

mysqltest

Section 6.2.10, “Server Handling of Expired Passwords”
Section 6.4.1.4, “SHA-256 Pluggable Authentication”

N

[index top]

nbdmtd

Section 18.3.3.6, “Defining NDB Cluster Data Nodes”

ndb_blob_tool

Section 18.4.6, “`ndb_blob_tool` — Check and Repair BLOB and TEXT columns of NDB Cluster Tables”

ndb_config

Section 18.4, “NDB Cluster Programs”
Section 18.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”

ndb_cpcd

Section 18.4.8, “`ndb_cpcd` — Automate Testing for NDB Development”

ndb_delete_all

Section 18.4.9, “`ndb_delete_all` — Delete All Rows from an NDB Table”

ndb_desc

Section 18.5.7.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 18.3.3.6, “Defining NDB Cluster Data Nodes”
Section 19.2.5, “KEY Partitioning”
MySQL Server Options for NDB Cluster
Section 18.5.16.3, “NDB Cluster and MySQL Security Procedures”
Section 18.5.10.1, “NDB Cluster Disk Data Objects”
Section 18.4.10, “`ndb_desc` — Describe NDB Tables”
Section 21.31.1, “The INFORMATION_SCHEMA FILES Table”
Section 21.14, “The INFORMATION_SCHEMA PARTITIONS Table”
Section 18.5.14.4, “The ndbinfo cluster_operations Table”

Section 18.5.14.22, “The ndbinfo server_operations Table”

ndb_drop_index

Section 18.4.11, “ndb_drop_index — Drop Index from an NDB Table”

ndb_drop_table

Section 18.4.11, “ndb_drop_index — Drop Index from an NDB Table”
Section 18.4.12, “ndb_drop_table — Drop an NDB Table”

ndb_error_reporter

Section 18.4.13, “ndb_error_reporter — NDB Error-Reporting Utility”

ndb_index_stat

Section 18.4.14, “ndb_index_stat — NDB Index Statistics Utility”

ndb_mgm

Section 18.5.7.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 18.5.7.1, “Adding NDB Cluster Data Nodes Online: General Issues”
Section 18.2.1.4, “Building NDB Cluster from Source on Linux”
Section 18.5.1, “Commands in the NDB Cluster Management Client”
Section 18.3.3.6, “Defining NDB Cluster Data Nodes”
Section 18.5.12, “Distributed Privileges Using Shared Grant Tables”
Section 18.2.4, “Initial Startup of NDB Cluster”
Section 18.2.2.3, “Initial Startup of NDB Cluster on Windows”
Section 18.2.1, “Installation of NDB Cluster on Linux”
Section 18.2.1.1, “Installing an NDB Cluster Binary Release on Linux”
Section 18.2.1.2, “Installing NDB Cluster from RPM”
Section 18.2.1.3, “Installing NDB Cluster Using .deb Files”
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
Chapter 18, *MySQL NDB Cluster 7.3 and NDB Cluster 7.4*
MySQL Server Options for NDB Cluster
Section 18.1.1, “NDB Cluster Core Concepts”
Section 18.5.3.1, “NDB Cluster Logging Management Commands”
Section 18.4, “NDB Cluster Programs”
Section 18.5.16.1, “NDB Cluster Security and Networking Issues”
Section 18.5.6, “NDB Cluster Single User Mode”
Section 18.4.5, “ndb_mgm — The NDB Cluster Management Client”
Section 18.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”
Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Section 18.5.8, “Online Backup of NDB Cluster”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 18.6.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”
Restoring an NDB backup to a later version of NDB Cluster
Restoring to More Nodes Than the Original
Section 18.2.6, “Safe Shutdown and Restart of NDB Cluster”
Section 18.5.14.1, “The ndbinfo arbitrator_validity_detail Table”
Section 18.5.14.15, “The ndbinfo membership Table”
Section 18.5.14.16, “The ndbinfo memoryusage Table”
Section 18.5.14.18, “The ndbinfo nodes Table”
Section 18.5.14.27, “The ndbinfo transporters Table”
Section 18.5.8.2, “Using The NDB Cluster Management Client to Create a Backup”

ndb_mgm.exe

Section 18.2.2.3, “Initial Startup of NDB Cluster on Windows”

Section 18.2.2.1, “Installing NDB Cluster on Windows from a Binary Release”

ndb_mgmd

Section 18.2.1.4, “Building NDB Cluster from Source on Linux”
Section 18.5.1, “Commands in the NDB Cluster Management Client”
Section 18.3.3.5, “Defining an NDB Cluster Management Server”
Section 18.3.3.6, “Defining NDB Cluster Data Nodes”
Section 18.2.4, “Initial Startup of NDB Cluster”
Section 18.2.2.3, “Initial Startup of NDB Cluster on Windows”
Section 18.2.1, “Installation of NDB Cluster on Linux”
Section 18.2.1.1, “Installing an NDB Cluster Binary Release on Linux”
Section 18.2.1.2, “Installing NDB Cluster from RPM”
Section 18.2.2.1, “Installing NDB Cluster on Windows from a Binary Release”
Section 18.2.1.3, “Installing NDB Cluster Using .deb Files”
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
MySQL Server Options for NDB Cluster
Section 2.9.7, “MySQL Source-Configuration Options”
Section 18.3.3.1, “NDB Cluster Configuration: Basic Example”
Section 18.3.3.3, “NDB Cluster Connection Strings”
Section 18.1.1, “NDB Cluster Core Concepts”
Section 18.5.3.1, “NDB Cluster Logging Management Commands”
Section 18.1.2, “NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions”
Section 18.4, “NDB Cluster Programs”
Section 18.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”
Section 18.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”
Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 18.3.1, “Quick Test Setup of NDB Cluster”
Section 18.2.6, “Safe Shutdown and Restart of NDB Cluster”
Section 18.5.4, “Summary of NDB Cluster Start Phases”
Section 18.2.8.2, “Using the NDB Cluster Auto-Installer”

ndb_mgmd.exe

Section 18.2.2.3, “Initial Startup of NDB Cluster on Windows”
Section 18.2.2.1, “Installing NDB Cluster on Windows from a Binary Release”
Section 18.2.2.4, “Installing NDB Cluster Processes as Windows Services”

ndb_move_data

Section 18.4.15, “`ndb_move_data` — NDB Data Copy Utility”

ndb_print_backup_file

Section 18.4.16, “`ndb_print_backup_file` — Print NDB Backup File Contents”
Section 18.4.18, “`ndb_print_frag_file` — Print NDB Fragment List File Contents”
Section 18.4.19, “`ndb_print_schema_file` — Print NDB Schema File Contents”
Section 18.4.20, “`ndb_print_sys_file` — Print NDB System File Contents”
Section 18.4.21, “`ndb_redo_log_reader` — Check and Print Content of Cluster Redo Log”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

ndb_print_file

Section 18.4.17, “`ndb_print_file` — Print NDB Disk Data File Contents”

ndb_print_frag_file

Section 18.4.18, “`ndb_print_frag_file` — Print NDB Fragment List File Contents”

ndb_print_schema_file

Section 18.4.16, “`ndb_print_backup_file` — Print NDB Backup File Contents”

Section 18.4.17, “ndb_print_file — Print NDB Disk Data File Contents”
Section 18.4.18, “ndb_print_frag_file — Print NDB Fragment List File Contents”
Section 18.4.19, “ndb_print_schema_file — Print NDB Schema File Contents”
Section 18.4.20, “ndb_print_sys_file — Print NDB System File Contents”
Section 18.4.21, “ndb_redo_log_reader — Check and Print Content of Cluster Redo Log”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

ndb_print_sys_file

Section 18.4.16, “ndb_print_backup_file — Print NDB Backup File Contents”
Section 18.4.17, “ndb_print_file — Print NDB Disk Data File Contents”
Section 18.4.18, “ndb_print_frag_file — Print NDB Fragment List File Contents”
Section 18.4.19, “ndb_print_schema_file — Print NDB Schema File Contents”
Section 18.4.20, “ndb_print_sys_file — Print NDB System File Contents”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

ndb_redo_log_reader

Section 18.4.21, “ndb_redo_log_reader — Check and Print Content of Cluster Redo Log”

ndb_restore

Section 7.1, “Backup and Recovery Types”
Section 18.3.3.6, “Defining NDB Cluster Data Nodes”
Section 18.5.12, “Distributed Privileges Using Shared Grant Tables”
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
Section 18.6.9, “NDB Cluster Backups With NDB Cluster Replication”
Section 18.1.1, “NDB Cluster Core Concepts”
Section 18.1, “NDB Cluster Overview”
Section 18.4, “NDB Cluster Programs”
Section 18.6.4, “NDB Cluster Replication Schema and Tables”
Section 18.6.10, “NDB Cluster Replication: Bidirectional and Circular Replication”
Section 18.5.6, “NDB Cluster Single User Mode”
Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Section 18.5.8, “Online Backup of NDB Cluster”
Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 18.6.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”
Restoring an NDB backup to a later version of NDB Cluster
Restoring an NDB backup to a previous version of NDB Cluster
Restoring to Fewer Nodes Than the Original
Restoring to More Nodes Than the Original
Section 18.1.4.1, “What is New in NDB Cluster 7.3”
Section 18.1.4.2, “What is New in NDB Cluster 7.4”

ndb_select_all

Section 18.5.16.3, “NDB Cluster and MySQL Security Procedures”
Section 18.4.23, “ndb_select_all — Print Rows from an NDB Table”
Section 18.4.26, “ndb_show_tables — Display List of NDB Tables”

ndb_select_count

Section 18.4.24, “ndb_select_count — Print Row Counts for NDB Tables”

ndb_setup.py

Section 18.2.8.1, “NDB Cluster Auto-Installer Requirements”
Section 18.4, “NDB Cluster Programs”
Section 18.4.25, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”

Section 18.2.8, “The NDB Cluster Auto-Installer (No longer supported)”
Section 18.2.8.2, “Using the NDB Cluster Auto-Installer”
Section 18.1.4.1, “What is New in NDB Cluster 7.3”
Section 18.1.4.2, “What is New in NDB Cluster 7.4”

ndb_show_tables

MySQL Server Options for NDB Cluster
Section 18.5.16.3, “NDB Cluster and MySQL Security Procedures”
Section 18.4, “NDB Cluster Programs”
Section 18.4.26, “`ndb_show_tables` — Display List of NDB Tables”
Section 18.5.14.4, “The ndbinfo cluster_operations Table”
Section 18.5.14.19, “The ndbinfo operations_per_fragment Table”
Section 18.5.14.22, “The ndbinfo server_operations Table”
Section 18.1.4.2, “What is New in NDB Cluster 7.4”

ndb_size.pl

Section 11.6, “Data Type Storage Requirements”
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
MySQL Server Options for NDB Cluster
Section 18.4.27, “`ndb_size.pl` — NDBCLUSTER Size Requirement Estimator”

ndb_waiter

Section 18.4.28, “`ndb_waiter` — Wait for NDB Cluster to Reach a Given Status”
Section 18.1.4.2, “What is New in NDB Cluster 7.4”

ndbd

Section 18.5.7.2, “Adding NDB Cluster Data Nodes Online: Basic procedure”
Section 18.5.7.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 18.2.1.4, “Building NDB Cluster from Source on Linux”
Section 18.5.1, “Commands in the NDB Cluster Management Client”
Section 18.3.3.6, “Defining NDB Cluster Data Nodes”
Section 18.2.4, “Initial Startup of NDB Cluster”
Section 18.2.1, “Installation of NDB Cluster on Linux”
Section 18.2.1.1, “Installing an NDB Cluster Binary Release on Linux”
Section 18.2.1.2, “Installing NDB Cluster from RPM”
Section 18.2.2.1, “Installing NDB Cluster on Windows from a Binary Release”
Section 18.2.1.3, “Installing NDB Cluster Using .deb Files”
Section 18.5, “Management of NDB Cluster”
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
MySQL Server Options for NDB Cluster
Section 18.6.9, “NDB Cluster Backups With NDB Cluster Replication”
Section 18.3.3.1, “NDB Cluster Configuration: Basic Example”
Section 18.1.1, “NDB Cluster Core Concepts”
Section 18.3.2.1, “NDB Cluster Data Node Configuration Parameters”
Section 18.2, “NDB Cluster Installation”
Section 18.1.2, “NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions”
Section 18.4, “NDB Cluster Programs”
Section 18.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”
Section 18.4.28, “`ndb_waiter` — Wait for NDB Cluster to Reach a Given Status”
Section 18.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”
Section 18.4.3, “`ndbmtd` — The NDB Cluster Data Node Daemon (Multi-Threaded)”
Section 18.3.2, “Overview of NDB Cluster Configuration Parameters, Options, and Variables”
Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 18.6.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”
Section 18.3.1, “Quick Test Setup of NDB Cluster”
Section 18.3.3.2, “Recommended Starting Configuration for NDB Cluster”
Section 18.2.6, “Safe Shutdown and Restart of NDB Cluster”

Section 18.5.4, “Summary of NDB Cluster Start Phases”
Section 18.5.14.18, “The ndbinfo nodes Table”
Section 18.5.3.3, “Using CLUSTERLOG STATISTICS in the NDB Cluster Management Client”
Section 18.2.8.2, “Using the NDB Cluster Auto-Installer”

ndbd.exe

Section 18.2.2.3, “Initial Startup of NDB Cluster on Windows”
Section 18.2.2.1, “Installing NDB Cluster on Windows from a Binary Release”
Section 18.2.2.4, “Installing NDB Cluster Processes as Windows Services”

ndbinfo_select_all

Section 18.4.2, “`ndbinfo_select_all` — Select From ndbinfo Tables”

ndbmtd

Section 18.5.7.2, “Adding NDB Cluster Data Nodes Online: Basic procedure”
Section 18.5.7.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 18.2.1.4, “Building NDB Cluster from Source on Linux”
Section 18.3.3.6, “Defining NDB Cluster Data Nodes”
Section 18.2.1, “Installation of NDB Cluster on Linux”
Section 18.2.1.1, “Installing an NDB Cluster Binary Release on Linux”
Section 18.2.1.2, “Installing NDB Cluster from RPM”
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 18.1.1, “NDB Cluster Core Concepts”
Section 18.3.2.1, “NDB Cluster Data Node Configuration Parameters”
Section 18.1.2, “NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions”
Section 18.4, “NDB Cluster Programs”
Section 18.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”
Section 18.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”
Section 18.4.3, “`ndbmtd` — The NDB Cluster Data Node Daemon (Multi-Threaded)”
Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 18.6.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”
Section 18.3.3.2, “Recommended Starting Configuration for NDB Cluster”
Restoring to Fewer Nodes Than the Original
Section 18.2.6, “Safe Shutdown and Restart of NDB Cluster”
Section 18.5.14.18, “The ndbinfo nodes Table”
Section 18.5.14.20, “The ndbinfo resources Table”
Section 18.2.8.2, “Using the NDB Cluster Auto-Installer”

ndbmtd.exe

Section 18.2.2.3, “Initial Startup of NDB Cluster on Windows”
Section 18.2.2.1, “Installing NDB Cluster on Windows from a Binary Release”
Section 18.2.2.4, “Installing NDB Cluster Processes as Windows Services”

NET

Section 18.2.2.4, “Installing NDB Cluster Processes as Windows Services”
Section 2.3.4.7, “Starting MySQL as a Windows Service”

NET START

Section 18.2.2.4, “Installing NDB Cluster Processes as Windows Services”
Section 5.7.2.2, “Starting Multiple MySQL Instances as Windows Services”

NET START mysqld_service_name

Section 2.3.4.7, “Starting MySQL as a Windows Service”
Section 2.3.5, “Troubleshooting a Microsoft Windows MySQL Server Installation”
Section 2.11.8, “Upgrading MySQL on Windows”

NET STOP
Section 18.2.2.4, “Installing NDB Cluster Processes as Windows Services”
Section 5.7.2.2, “Starting Multiple MySQL Instances as Windows Services”

NET STOP mysqld_service_name
Section 2.3.4.7, “Starting MySQL as a Windows Service”
Section 2.11.8, “Upgrading MySQL on Windows”

NET STOP service_name
Section 18.2.6, “Safe Shutdown and Restart of NDB Cluster”

nm
Section 4.7.4, “`resolve_stack_dump` — Resolve Numeric Stack Trace Dump to Symbols”
Section 5.8.1.5, “Using a Stack Trace”

numactl
Section 18.3.3.6, “Defining NDB Cluster Data Nodes”

O

[index top]

openssl
Section 6.3.3.2, “Creating RSA Keys Using openssl”
Section 6.3.3, “Creating SSL and RSA Certificates and Keys”
Section 6.3.3.1, “Creating SSL Certificates and Keys Using openssl”

openssl md5 package_name
Section 2.1.4.1, “Verifying the MD5 Checksum”

P

[index top]

perror
Section B.3.2.12, “Can't create/write to file”
Section B.2, “Error Information Interfaces”
Section B.3.2.17, “File Not Found and Similar Errors”
Section 7.6.3, “How to Repair MyISAM Tables”
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
Section 4.1, “Overview of MySQL Programs”
Section 4.8.1, “`perror` — Display MySQL Error Message Information”

pfexec
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”

PGP
Section 2.1.4.2, “Signature Checking Using GnuPG”

ping6
Section 5.1.12.5, “Obtaining an IPv6 Address from a Broker”

pkgadd
Section 2.7.1, “Installing MySQL on Solaris Using a Solaris PKG”

pkgrm
Section 2.7.1, “Installing MySQL on Solaris Using a Solaris PKG”

ppm
Section 2.13, “Perl Installation Notes”

ps
Section 6.2.9, “Assigning Account Passwords”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 8.12.4.1, “How MySQL Uses Memory”
Section B.3.1, “How to Determine What Is Causing a Problem”
Section 4.6.16, “`mysql_zap` — Kill Processes That Match a Pattern”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 6.6, “SELinux”
Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”

ps xa | grep mysqld
Section B.3.2.2, “Can't connect to [local] MySQL server”

R

[index top]

rename
Section 5.4.2.5, “Error Log File Flushing and Renaming”
Section 5.4.7, “Server Log Maintenance”
Section 5.4.3, “The General Query Log”

replace
Section 4.7.1, “`msql2mysql` — Convert mSQL Programs for Use with MySQL”
Section 4.1, “Overview of MySQL Programs”
Section 4.8.2, “`replace` — A String-Replacement Utility”
Section 17.3.4, “Using Replication for Scale-Out”

resolve_stack_dump
Section 4.1, “Overview of MySQL Programs”
Section 4.7.4, “`resolve_stack_dump` — Resolve Numeric Stack Trace Dump to Symbols”
Section 5.8.1.5, “Using a Stack Trace”

resolveip
Section 4.1, “Overview of MySQL Programs”
Section 4.8.3, “`resolveip` — Resolve Host name to IP Address or Vice Versa”

rm
Section 13.4.1.1, “PURGE BINARY LOGS Statement”

rpm
Section 2.5.5, “Installing MySQL on Linux Using RPM Packages from Oracle”
Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”
Section 2.1.4.4, “Signature Checking Using RPM”

rpmbuild
Section 2.5.5, “Installing MySQL on Linux Using RPM Packages from Oracle”

Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”
Section 2.9.2, “Source Installation Prerequisites”

rsync

Section 7.1, “Backup and Recovery Types”
Section 17.1.1.6, “Creating a Data Snapshot Using Raw Data Files”
Section 17.1.1.9, “Introducing Additional Replicas to an Existing Replication Environment”

S

[index top]

SC

Section 2.3.4.7, “Starting MySQL as a Windows Service”

SC DELETE

Section 18.2.2.4, “Installing NDB Cluster Processes as Windows Services”

SC DELETE mysqld_service_name

Section 5.7.2.2, “Starting Multiple MySQL Instances as Windows Services”
Section 2.3.4.7, “Starting MySQL as a Windows Service”

SC DELETE service_name

Section 18.2.2.4, “Installing NDB Cluster Processes as Windows Services”

SC START

Section 18.2.2.4, “Installing NDB Cluster Processes as Windows Services”
Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 5.7.2.2, “Starting Multiple MySQL Instances as Windows Services”

SC START mysqld_service_name

Section 2.3.5, “Troubleshooting a Microsoft Windows MySQL Server Installation”
Section 2.11.8, “Upgrading MySQL on Windows”

sc start mysqld_service_name

Section 2.3.4.7, “Starting MySQL as a Windows Service”

SC STOP

Section 18.2.2.4, “Installing NDB Cluster Processes as Windows Services”
Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 5.7.2.2, “Starting Multiple MySQL Instances as Windows Services”

SC STOP mysqld_service_name

Section 2.3.4.7, “Starting MySQL as a Windows Service”

sc stop mysqld_service_name

Section 2.3.4.7, “Starting MySQL as a Windows Service”

SC STOP service_name

Section 18.2.6, “Safe Shutdown and Restart of NDB Cluster”

scp

Section 7.1, “Backup and Recovery Types”
Section 17.1.1.6, “Creating a Data Snapshot Using Raw Data Files”

sed
Section 3.3.4.7, “Pattern Matching”

SELECT
Section 18.2.5, “NDB Cluster Example with Tables and Data”

semanage
Section 6.6.6, “Troubleshooting SELinux”

semodule
Section 6.6.3, “MySQL Server SELinux Policies”

service
Section 2.5.7, “Installing MySQL on Linux from the Native Software Repositories”

Service Control Manager
Section 2.3, “Installing MySQL on Microsoft Windows”
Section 2.3.4.7, “Starting MySQL as a Windows Service”

Services
Section 18.2.2.4, “Installing NDB Cluster Processes as Windows Services”
Section 2.3.4.7, “Starting MySQL as a Windows Service”

sestatus
Section 6.6.2, “Changing the SELinux Mode”
Section 6.6.1, “Check if SELinux is Enabled”

setenforce
Section 6.6.2, “Changing the SELinux Mode”
Section 6.6.6, “Troubleshooting SELinux”

setenv
Section 4.2.7, “Setting Environment Variables”

setrlimit
Section 16.2.2.1, “`memcached` Command-Line Options”
Section 16.2.2, “Using `memcached`”

setup.bat
Section 18.2.8.2, “Using the NDB Cluster Auto-Installer”

sh
Section 1.1, “About This Manual”
Section B.3.2.17, “File Not Found and Similar Errors”
Section 4.2.1, “Invoking MySQL Programs”
Section 4.2.7, “Setting Environment Variables”

SHOW
Section 18.3.1, “Quick Test Setup of NDB Cluster”

SHOW ERRORS
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”

SHOW WARNINGS
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”

sleep
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

ssh
Section 18.5.16.1, “NDB Cluster Security and Networking Issues”
Section 16.1.1, “Using ZFS for File System Replication”

Start>Run>cmd.exe
Section 6.3.3.1, “Creating SSL Certificates and Keys Using openssl”

strings
Section 6.1.1, “Security Guidelines”

su root
Section 18.2.1.1, “Installing an NDB Cluster Binary Release on Linux”

sudo
Section 18.2.1.1, “Installing an NDB Cluster Binary Release on Linux”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”
Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”

System Preferences...
Section 2.4.4, “Installing and Using the MySQL Preference Pane”

T

[index top]

tar
Section 17.3.1.2, “Backing Up Raw Data from a Replica”
Section 7.1, “Backup and Recovery Types”
Section 17.1.1.6, “Creating a Data Snapshot Using Raw Data Files”
Section 3.3, “Creating and Using a Database”
Section 1.6, “How to Report Bugs or Problems”
Section 2.5.5, “Installing MySQL on Linux Using RPM Packages from Oracle”
Section 2.4, “Installing MySQL on OS X”
Section 2.7, “Installing MySQL on Solaris”
Section 2.7.1, “Installing MySQL on Solaris Using a Solaris PKG”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”
Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”
Section 2.13.1, “Installing Perl on Unix”
Section 17.1.1.9, “Introducing Additional Replicas to an Existing Replication Environment”
Section 2.9.1, “Source Installation Methods”
Section 2.9.2, “Source Installation Prerequisites”
Section 2.1.2, “Which MySQL Version and Distribution to Install”

tcpdump
Section 6.1.1, “Security Guidelines”

tcsh
Section 1.1, “About This Manual”
Section B.3.2.17, “File Not Found and Similar Errors”
Section 2.4.1, “General Notes on Installing MySQL on OS X”
Section 4.2.1, “Invoking MySQL Programs”
Section 4.2.7, “Setting Environment Variables”

tee
Section 4.5.1.2, “mysql Client Commands”

Telnet
Section 16.2.4, “Getting `memcached` Statistics”

telnet
Section 16.2.4, “Getting `memcached` Statistics”
Section 14.20.2, “InnoDB memcached Architecture”
Section 6.1.1, “Security Guidelines”
Section 14.20.3, “Setting Up the InnoDB memcached Plugin”

Terminal
Section 2.4, “Installing MySQL on OS X”

Text in this style
Section 1.1, “About This Manual”

top
Section B.3.1, “How to Determine What Is Causing a Problem”

U

[index top]

ulimit
Section 18.3.3.6, “Defining NDB Cluster Data Nodes”
Section 8.12.4.2, “Enabling Large Page Support”
Section B.3.2.17, “File Not Found and Similar Errors”
Section 16.2.2.1, “`memcached` Command-Line Options”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section B.3.2.9, “Packet Too Large”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”

unix_chkpwd
Section 6.4.1.6, “PAM Pluggable Authentication”

update-rc.d
Section 18.2.1.1, “Installing an NDB Cluster Binary Release on Linux”

useradd
Section 18.2.1.1, “Installing an NDB Cluster Binary Release on Linux”
Section 2.7, “Installing MySQL on Solaris”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”

V

[index top]

vi
Section 18.2.3, “Initial Configuration of NDB Cluster”
Section 4.5.1.2, “mysql Client Commands”

Section 3.3.4.7, “Pattern Matching”

vmstat

Section 16.2.2.1, “`memcached` Command-Line Options”

W

[index top]

WinDbg

Section 5.8.1.3, “Using WER with PDB to create a Windows crashdump”

windbg.exe

Section 5.8.1.3, “Using WER with PDB to create a Windows crashdump”

winMd5Sum

Section 2.1.4.1, “Verifying the MD5 Checksum”

WinZip

Section 17.3.1.2, “Backing Up Raw Data from a Replica”
Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”
Section 2.9.2, “Source Installation Prerequisites”

WordPad

Section 13.2.6, “LOAD DATA Statement”

Y

[index top]

yacc

Section 2.9.8, “Dealing with Problems Compiling MySQL”
Section 9.3, “Keywords and Reserved Words”

yum

Section 16.2.1, “Installing `memcached`”
Section 2.5.7, “Installing MySQL on Linux from the Native Software Repositories”
Section 2.5.5, “Installing MySQL on Linux Using RPM Packages from Oracle”
Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository”
Section 2.11.5, “Upgrading MySQL with the MySQL Yum Repository”
Section 16.2.3.3, “Using `libmemcached` with C and C++”

yum install

Section 2.5.5, “Installing MySQL on Linux Using RPM Packages from Oracle”

yum update

Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository”
Section 2.5.2, “Replacing a Third-Party Distribution of MySQL Using the MySQL Yum Repository”

yum update mysql-server

Section 2.5.2, “Replacing a Third-Party Distribution of MySQL Using the MySQL Yum Repository”

yum-config-manager

Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository”

Section 2.5.2, “Replacing a Third-Party Distribution of MySQL Using the MySQL Yum Repository”

Z

[index top]

zfs recv
Section 16.1.1, “Using ZFS for File System Replication”

zip
Section 17.1.1.6, “Creating a Data Snapshot Using Raw Data Files”
Section 1.6, “How to Report Bugs or Problems”

zsh
Section 4.2.7, “Setting Environment Variables”

zypper
Section 2.5.5, “Installing MySQL on Linux Using RPM Packages from Oracle”

Function Index

Symbols | A | B | C | D | E | F | G | H | I | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y

Symbols

[index top]

%
Section 1.7.1, “MySQL Extensions to Standard SQL”

A

[index top]

ABS()
Section 13.7.3.1, “CREATE FUNCTION Statement for User-Defined Functions”
Section 12.6.2, “Mathematical Functions”
Section 19.6.3, “Partitioning Limitations Relating to Functions”

ACOS()
Section 12.6.2, “Mathematical Functions”

add()
Section 16.2.3.1, “Basic `memcached` Operations”

ADDDATE()
Section 12.7, “Date and Time Functions”

addslashes()
Section 6.1.7, “Client Programming Security Guidelines”

ADDTIME()
Section 12.7, “Date and Time Functions”

AES_DECRYPT()

Section 12.14, “Encryption and Compression Functions”
Section 6.5.4, “MySQL Enterprise Encryption User-Defined Function Descriptions”
Section 5.1.7, “Server System Variables”
Section 6.3.4, “SSL Library-Dependent Capabilities”

AES_ENCRYPT()

Section 12.14, “Encryption and Compression Functions”
Section 6.5.4, “MySQL Enterprise Encryption User-Defined Function Descriptions”
Section 5.1.7, “Server System Variables”
Section 6.3.4, “SSL Library-Dependent Capabilities”

Area()

Section 12.17.7.4, “Polygon and MultiPolygon Property Functions”

AsBinary()

Section 11.4.6, “Fetching Spatial Data”
Section 12.17.6, “Geometry Format Conversion Functions”

ASCII()

Section 13.8.3, “HELP Statement”
Section 12.8, “String Functions and Operators”

ASIN()

Section 12.6.2, “Mathematical Functions”

AsText()

Section 11.4.6, “Fetching Spatial Data”
Section 12.17.6, “Geometry Format Conversion Functions”

AsWKB()

Section 12.17.6, “Geometry Format Conversion Functions”

AsWKT()

Section 12.17.6, “Geometry Format Conversion Functions”

ATAN()

Section 12.6.2, “Mathematical Functions”

ATAN2()

Section 12.6.2, “Mathematical Functions”

AVG()

Section 12.19.1, “Aggregate Function Descriptions”
Section 11.2.1, “Date and Time Data Type Syntax”
Section 8.2.1.14, “GROUP BY Optimization”
Section 11.3.5, “The ENUM Type”
Section 1.2.2, “The Main Features of MySQL”
Section 11.3.6, “The SET Type”

B

[index top]

BENCHMARK()
Section 13.2.10.8, “Derived Tables”
Section 8.10.3.1, “How the Query Cache Operates”
Section 12.16, “Information Functions”
Section 8.13.1, “Measuring the Speed of Expressions and Functions”
Section 13.2.10.10, “Optimizing Subqueries”

BIN()
Section 9.1.5, “Bit-Value Literals”
Section 12.8, “String Functions and Operators”

BIT_AND()
Section 12.19.1, “Aggregate Function Descriptions”
Section 12.13, “Bit Functions and Operators”
Section 1.7.1, “MySQL Extensions to Standard SQL”

BIT_COUNT()
Section 12.13, “Bit Functions and Operators”
Section 1.7.1, “MySQL Extensions to Standard SQL”

BIT_LENGTH()
Section 12.8, “String Functions and Operators”

BIT_OR()
Section 12.19.1, “Aggregate Function Descriptions”
Section 12.13, “Bit Functions and Operators”
Section 1.7.1, “MySQL Extensions to Standard SQL”

BIT_XOR()
Section 12.19.1, “Aggregate Function Descriptions”
Section 12.13, “Bit Functions and Operators”
Section 1.7.1, “MySQL Extensions to Standard SQL”

Buffer()
Section 12.17.8, “Spatial Operator Functions”

C

[index top]

CAST()
Section 9.1.5, “Bit-Value Literals”
Section 12.11, “Cast Functions and Operators”
Section 12.8.3, “Character Set and Collation of Function Results”
Section 12.4.2, “Comparison Functions and Operators”
Section 11.2.8, “Conversion Between Date and Time Types”
Section 12.7, “Date and Time Functions”
Section 9.5, “Expressions”
Section 9.1.4, “Hexadecimal Literals”
Section 1.7.2, “MySQL Differences from Standard SQL”
Section 12.3, “Type Conversion in Expression Evaluation”
Section 9.4, “User-Defined Variables”

CEIL()
Section 12.6.2, “Mathematical Functions”

CEILING()
Section 19.2.4.1, “LINEAR HASH Partitioning”
Section 12.6.2, “Mathematical Functions”
Section 19.6.3, “Partitioning Limitations Relating to Functions”

Centroid()
Section 12.17.7.4, “Polygon and MultiPolygon Property Functions”

CHAR()
Section 12.14, “Encryption and Compression Functions”
Section 4.5.1.1, “mysql Client Options”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 12.8, “String Functions and Operators”

CHAR_LENGTH()
Section 12.8, “String Functions and Operators”
Section 10.10.1, “Unicode Character Sets”

CHARACTER_LENGTH()
Section 12.8, “String Functions and Operators”

CHARSET()
Section 12.8.3, “Character Set and Collation of Function Results”
Section 12.16, “Information Functions”

COALESCE()
Section 12.4.2, “Comparison Functions and Operators”
Section 13.2.9.2, “JOIN Clause”

COERCIBILITY()
Section 10.8.4, “Collation Coercibility in Expressions”
Section 12.16, “Information Functions”

COLLATION()
Section B.3.4.1, “Case Sensitivity in String Searches”
Section 12.8.3, “Character Set and Collation of Function Results”
Section 12.16, “Information Functions”

COMPRESS()
Section 12.14, “Encryption and Compression Functions”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 5.1.7, “Server System Variables”

CONCAT()
Section 12.19.1, “Aggregate Function Descriptions”
Section 12.11, “Cast Functions and Operators”
Section 12.8.3, “Character Set and Collation of Function Results”
Section 10.2.1, “Character Set Repertoire”
Section 10.8.4, “Collation Coercibility in Expressions”
Section 13.7.3.1, “CREATE FUNCTION Statement for User-Defined Functions”
Section 12.4.3, “Logical Operators”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 5.1.10, “Server SQL Modes”
Section 13.7.5.14, “SHOW CREATE VIEW Statement”
Section 12.8, “String Functions and Operators”

Section 21.29, “The INFORMATION_SCHEMA VIEWS Table”
Section 12.3, “Type Conversion in Expression Evaluation”
Section 12.12, “XML Functions”

CONCAT_WS()

Section 12.19.1, “Aggregate Function Descriptions”
Section 12.8, “String Functions and Operators”

CONNECTION_ID()

Section 6.4.4.3, “Audit Log File Formats”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.10.3.1, “How the Query Cache Operates”
Section 12.16, “Information Functions”
Section 13.7.6.4, “KILL Statement”
Section 5.1.7, “Server System Variables”
Section 13.7.5.30, “SHOW PROCESSLIST Statement”
Section 21.16, “The INFORMATION_SCHEMA PROCESSLIST Table”
Section 22.12.10.3, “The threads Table”

Contains()

Section 12.17.9.2, “Spatial Relation Functions That Use Minimum Bounding Rectangles”
Section 1.3, “What Is New in MySQL 5.6”

CONV()

Section 12.8.3, “Character Set and Collation of Function Results”
Section 12.6.2, “Mathematical Functions”
Section 12.8, “String Functions and Operators”

CONVERT()

Section 12.11, “Cast Functions and Operators”
Section 10.3.8, “Character Set Introducers”
Section 10.3.6, “Character String Literal Character Set and Collation”
Section 12.4.2, “Comparison Functions and Operators”
Section A.11, “MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 4.5.1.1, “mysql Client Options”

CONVERT_TZ()

Section 12.7, “Date and Time Functions”
Section 8.10.3.1, “How the Query Cache Operates”
Section 8.3.10, “Indexed Lookups from TIMESTAMP Columns”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”

COS()

Section 12.6.2, “Mathematical Functions”

COT()

Section 12.6.2, “Mathematical Functions”

COUNT()

Section 12.19.1, “Aggregate Function Descriptions”
Section 3.3.4.8, “Counting Rows”
Section 13.7.3.1, “CREATE FUNCTION Statement for User-Defined Functions”
Section 8.2.1.14, “GROUP BY Optimization”
Section 14.23, “InnoDB Restrictions and Limitations”
Section 8.4.4, “Internal Temporary Table Use in MySQL”

Section 18.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 1.7.1, “MySQL Extensions to Standard SQL”
NDB Cluster Status Variables
Section B.3.4.3, “Problems with NULL Values”
Section 5.1.10, “Server SQL Modes”
Section 1.2.2, “The Main Features of MySQL”
Section 20.5.3, “Updatable and Insertable Views”
Section 20.5.2, “View Processing Algorithms”
Section 8.2.1.1, “WHERE Clause Optimization”

CRC32()

Section 12.6.2, “Mathematical Functions”

Crosses()

Section 12.17.9.1, “Spatial Relation Functions That Use Object Shapes”

crypt()

Section 12.14, “Encryption and Compression Functions”
Section 5.1.7, “Server System Variables”

CURDATE()

Section 12.7, “Date and Time Functions”
Section 3.3.4.5, “Date Calculations”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.10.3.1, “How the Query Cache Operates”

CURRENT_DATE

Section 11.5, “Data Type Default Values”
Section 12.7, “Date and Time Functions”

CURRENT_DATE()

Section 11.2.8, “Conversion Between Date and Time Types”
Section 12.7, “Date and Time Functions”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.10.3.1, “How the Query Cache Operates”

CURRENT_TIME

Section 12.7, “Date and Time Functions”

CURRENT_TIME()

Section 12.7, “Date and Time Functions”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.10.3.1, “How the Query Cache Operates”

CURRENT_TIMESTAMP

Section 11.2.6, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”
Section 13.1.11, “CREATE EVENT Statement”
Section 11.5, “Data Type Default Values”
Section 12.7, “Date and Time Functions”
Section 5.1.7, “Server System Variables”

CURRENT_TIMESTAMP()

Section 11.2.6, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”
Section 12.7, “Date and Time Functions”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”

Section 8.10.3.1, “How the Query Cache Operates”

CURRENT_USER

Section 13.7.1.1, “ALTER USER Statement”
Section 13.1.11, “CREATE EVENT Statement”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 13.1.19, “CREATE TRIGGER Statement”
Section 13.1.20, “CREATE VIEW Statement”
Section 6.2.3, “Grant Tables”
Section 12.16, “Information Functions”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.4.1.15, “Replication and System Functions”
Section 17.4.1.8, “Replication of CURRENT_USER()”
Section 6.2.4, “Specifying Account Names”
Section 20.6, “Stored Object Access Control”

CURRENT_USER()

Section 6.2.5, “Access Control, Stage 1: Connection Verification”
Section 13.7.1.1, “ALTER USER Statement”
Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 13.1.11, “CREATE EVENT Statement”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 13.1.19, “CREATE TRIGGER Statement”
Section 13.1.20, “CREATE VIEW Statement”
Section 8.10.3.1, “How the Query Cache Operates”
Section 12.16, “Information Functions”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 6.2.12, “Proxy Users”
Section 17.4.1.15, “Replication and System Functions”
Section 17.4.1.8, “Replication of CURRENT_USER()”
Section 13.7.1.7, “SET PASSWORD Statement”
Section 6.2.4, “Specifying Account Names”
Section 6.2.15, “SQL-Based Account Activity Auditing”
Section 20.6, “Stored Object Access Control”
Section 10.2.2, “UTF-8 for Metadata”

CURTIME()

Section 12.7, “Date and Time Functions”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 11.2.7, “Fractional Seconds in Time Values”
Section 8.10.3.1, “How the Query Cache Operates”
Section 5.1.13, “MySQL Server Time Zone Support”
Section 17.4.1.14, “Replication and Fractional Seconds Support”

D

[index top]

DATABASE()

Section 17.1.4.4, “Binary Log Options and Variables”
Section 3.3.1, “Creating and Selecting a Database”
Section 13.1.21, “DROP DATABASE Statement”
Section 3.4, “Getting Information About Databases and Tables”
Section 8.10.3.1, “How the Query Cache Operates”
Section 12.16, “Information Functions”
Section B.3.7, “Known Issues in MySQL”
Section 10.2.2, “UTF-8 for Metadata”

DATE()

Section 12.7, “Date and Time Functions”

DATE_ADD()

Section 12.6.1, “Arithmetic Operators”
Section 11.2, “Date and Time Data Types”
Section 12.7, “Date and Time Functions”
Section 3.3.4.5, “Date Calculations”
Section 9.5, “Expressions”

DATE_FORMAT()

Section 12.11, “Cast Functions and Operators”
Section 12.7, “Date and Time Functions”
Section 10.16, “MySQL Server Locale Support”
Section 5.1.7, “Server System Variables”

DATE_SUB()

Section 11.2, “Date and Time Data Types”
Section 12.7, “Date and Time Functions”
Section 9.5, “Expressions”

DATEDIFF()

Section 12.7, “Date and Time Functions”
Section 19.6.3, “Partitioning Limitations Relating to Functions”

DAY()

Section 12.7, “Date and Time Functions”
Section 19.6.3, “Partitioning Limitations Relating to Functions”

DAYNAME()

Section 12.7, “Date and Time Functions”
Section 10.16, “MySQL Server Locale Support”
Section 5.1.7, “Server System Variables”

DAYOFMONTH()

Section 12.7, “Date and Time Functions”
Section 3.3.4.5, “Date Calculations”
Section 19.6.3, “Partitioning Limitations Relating to Functions”

DAYOFWEEK()

Section 12.7, “Date and Time Functions”
Section 19.6.3, “Partitioning Limitations Relating to Functions”

DAYOFYEAR()

Section 12.7, “Date and Time Functions”
Section 19.6.3, “Partitioning Limitations Relating to Functions”
Section 19.2, “Partitioning Types”

DECODE()

Section 12.14, “Encryption and Compression Functions”
Section 1.7.1, “MySQL Extensions to Standard SQL”

decr()

Section 16.2.3.1, “Basic `memcached` Operations”

DEFAULT()
Section 11.5, “Data Type Default Values”
Section 13.2.5, “INSERT Statement”
Section 12.20, “Miscellaneous Functions”
Section 13.2.8, “REPLACE Statement”

DEGREES()
Section 12.6.2, “Mathematical Functions”

delete()
Section 16.2.3.1, “Basic `memcached` Operations”

DES_DECRYPT()
Section 12.14, “Encryption and Compression Functions”
Section 5.1.6, “Server Command Options”

DES_ENCRYPT()
Section 12.14, “Encryption and Compression Functions”
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.1.6, “Server Command Options”

Dimension()
Section 12.17.7.1, “General Geometry Property Functions”

Disjoint()
Section 12.17.9.2, “Spatial Relation Functions That Use Minimum Bounding Rectangles”

E

[index top]

ELT()
Section 12.8.3, “Character Set and Collation of Function Results”
Section B.3.7, “Known Issues in MySQL”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 12.8, “String Functions and Operators”

ENCODE()
Section 12.14, “Encryption and Compression Functions”
Section 1.7.1, “MySQL Extensions to Standard SQL”

ENCRYPT()
Section 12.14, “Encryption and Compression Functions”
Section 8.10.3.1, “How the Query Cache Operates”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 10.11, “Restrictions on Character Sets”
Section 5.1.7, “Server System Variables”

EndPoint()
Section 12.17.7.3, “LineString and MultiLineString Property Functions”

Envelope()
Section 12.17.7.1, “General Geometry Property Functions”

Equals()
Section 12.17.9.2, “Spatial Relation Functions That Use Minimum Bounding Rectangles”

EXP()
Section 12.6.2, “Mathematical Functions”

EXPORT_SET()
Section 12.8, “String Functions and Operators”

ExteriorRing()
Section 12.17.7.4, “Polygon and MultiPolygon Property Functions”

EXTRACT()
Section 12.11, “Cast Functions and Operators”
Section 12.7, “Date and Time Functions”
Section 9.5, “Expressions”
Section 19.6.3, “Partitioning Limitations Relating to Functions”

ExtractValue()
Section 12.12, “XML Functions”

F

[index top]

FIELD()
Section 12.8, “String Functions and Operators”

FIND_IN_SET()
Section 12.8, “String Functions and Operators”
Section 11.3.6, “The SET Type”

FLOOR()
Section 12.6.2, “Mathematical Functions”
Section 19.6.3, “Partitioning Limitations Relating to Functions”

flush_all
Section 16.2.3.1, “Basic `memcached` Operations”

FORMAT()
Section 12.8.3, “Character Set and Collation of Function Results”
Section 12.6.2, “Mathematical Functions”
Section 12.20, “Miscellaneous Functions”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 10.16, “MySQL Server Locale Support”
Section 12.8, “String Functions and Operators”

FOUND_ROWS()
Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.10.3.1, “How the Query Cache Operates”
Section 12.16, “Information Functions”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.4.1.15, “Replication and System Functions”

FROM_BASE64()
Section 12.8, “String Functions and Operators”

FROM_DAYS()
Section 12.7, “Date and Time Functions”
Section 1.7.1, “MySQL Extensions to Standard SQL”

FROM_UNIXTIME()
Section 6.4.4.4, “Audit Log Logging Control”
Section 12.7, “Date and Time Functions”
Section 8.3.10, “Indexed Lookups from TIMESTAMP Columns”
Section 17.4.1.31, “Replication and Time Zones”

G

[index top]

GeomCollFromText()
Section 12.17.3, “Functions That Create Geometry Values from WKT Values”

GeomCollFromWKB()
Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

GeometryCollection()
Section 12.17.5, “MySQL-Specific Functions That Create Geometry Values”

GeometryCollectionFromText()
Section 12.17.3, “Functions That Create Geometry Values from WKT Values”

GeometryCollectionFromWKB()
Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

GeometryFromText()
Section 12.17.3, “Functions That Create Geometry Values from WKT Values”

GeometryFromWKB()
Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

GeometryN()
Section 12.17.7.5, “GeometryCollection Property Functions”

GeometryType()
Section 12.17.7.1, “General Geometry Property Functions”

GeomFromText()
Section 12.17.3, “Functions That Create Geometry Values from WKT Values”
Section 11.4.5, “Populating Spatial Columns”

GeomFromWKB()
Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

get()
Section 16.2.3.1, “Basic `memcached` Operations”

GET_FORMAT()
Section 12.7, “Date and Time Functions”
Section 10.16, “MySQL Server Locale Support”

GET_LOCK()
Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 13.1.11, “CREATE EVENT Statement”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 20.4.1, “Event Scheduler Overview”
Section 8.14.3, “General Thread States”
Section 8.10.3.1, “How the Query Cache Operates”
Section 8.11.1, “Internal Locking Methods”
Section 13.7.6.4, “KILL Statement”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 12.15, “Locking Functions”
Section 17.4.1.15, “Replication and System Functions”

gethostbyaddr()
Section 5.1.11.2, “DNS Lookups and the Host Cache”

gethostbyname()
Section 5.1.11.2, “DNS Lookups and the Host Cache”

getrusage()
Section 18.5.14.26, “The ndbinfo threadstat Table”

gettimeofday()
Section 18.5.14.26, “The ndbinfo threadstat Table”

GLength()
Section 12.17.7.3, “LineString and MultiLineString Property Functions”
Section 11.4, “Spatial Data Types”
Section 12.8, “String Functions and Operators”

GREATEST()
Section 12.8.3, “Character Set and Collation of Function Results”
Section 12.4.2, “Comparison Functions and Operators”

GROUP_CONCAT()
Section 12.19.1, “Aggregate Function Descriptions”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section B.3.7, “Known Issues in MySQL”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 5.1.7, “Server System Variables”
Section 1.2.2, “The Main Features of MySQL”

GTID_SUBSET()
Section 12.18, “Functions Used with Global Transaction IDs”
Section 17.1.3.1, “GTID Concepts”

GTID_SUBTRACT()
Section 12.18, “Functions Used with Global Transaction IDs”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 17.1.3.1, “GTID Concepts”

H

[index top]

HEX()
Section 9.1.5, “Bit-Value Literals”
Section 12.8.3, “Character Set and Collation of Function Results”
Section 10.3.6, “Character String Literal Character Set and Collation”
Section 9.1.4, “Hexadecimal Literals”
Section 12.6.2, “Mathematical Functions”
Section 12.20, “Miscellaneous Functions”
Section A.11, “MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 12.8, “String Functions and Operators”

HOUR()
Section 12.7, “Date and Time Functions”
Section 19.6.3, “Partitioning Limitations Relating to Functions”

I

[index top]

IF()
Section 12.8.3, “Character Set and Collation of Function Results”
Section 12.5, “Flow Control Functions”
Section 13.6.5.2, “IF Statement”
Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”
Section B.3.7, “Known Issues in MySQL”
Section 1.7.1, “MySQL Extensions to Standard SQL”

IFNULL()
Section 12.5, “Flow Control Functions”
Section B.3.4.3, “Problems with NULL Values”

incr()
Section 16.2.3.1, “Basic `memcached` Operations”

INET6_ATON()
Section 5.1.12, “IPv6 Support”
Section 12.20, “Miscellaneous Functions”

INET6_NTOA()
Section 5.1.12, “IPv6 Support”
Section 12.20, “Miscellaneous Functions”

INET_ATON()
Section 5.1.12, “IPv6 Support”
Section 12.20, “Miscellaneous Functions”

INET_NTOA()
Section 5.1.12, “IPv6 Support”
Section 12.20, “Miscellaneous Functions”

INSERT()
Section 12.8, “String Functions and Operators”

INSTR()
Section 12.8.3, “Character Set and Collation of Function Results”
Section 12.8, “String Functions and Operators”

InteriorRingN()
Section 12.17.7.4, “Polygon and MultiPolygon Property Functions”

Intersects()
Section 12.17.9.2, “Spatial Relation Functions That Use Minimum Bounding Rectangles”

INTERVAL()
Section 12.4.2, “Comparison Functions and Operators”

IS_FREE_LOCK()
Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.10.3.1, “How the Query Cache Operates”
Section 12.15, “Locking Functions”
Section 17.4.1.15, “Replication and System Functions”

IS_IPV4()
Section 12.20, “Miscellaneous Functions”

IS_IPV4_COMPAT()
Section 12.20, “Miscellaneous Functions”

IS_IPV4_MAPPED()
Section 12.20, “Miscellaneous Functions”

IS_IPV6()
Section 12.20, “Miscellaneous Functions”

IS_USED_LOCK()
Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.10.3.1, “How the Query Cache Operates”
Section 12.15, “Locking Functions”
Section 17.4.1.15, “Replication and System Functions”

IsClosed()
Section 12.17.7.3, “LineString and MultiLineString Property Functions”

IsEmpty()
Section 12.17.7.1, “General Geometry Property Functions”

ISNULL()
Section 12.4.2, “Comparison Functions and Operators”

IsSimple()
Section 12.17.7.1, “General Geometry Property Functions”

L

[index top]

LAST_DAY()

Section 12.7, “Date and Time Functions”

LAST_INSERT_ID()

Section 12.4.2, “Comparison Functions and Operators”
Section 13.1.17, “CREATE TABLE Statement”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.10.3.1, “How the Query Cache Operates”
Section 12.16, “Information Functions”
Section 13.2.5.2, “INSERT ... ON DUPLICATE KEY UPDATE Statement”
Section 13.2.5.3, “INSERT DELAYED Statement”
Section 13.2.5, “INSERT Statement”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 17.4.1.1, “Replication and AUTO_INCREMENT”
Section 17.4.1.15, “Replication and System Functions”
Section 5.1.7, “Server System Variables”
Section 20.2.4, “Stored Procedures, Functions, Triggers, and LAST_INSERT_ID()”
Section 17.4.4, “Troubleshooting Replication”
Section 20.5.3, “Updatable and Insertable Views”
Section 3.6.9, “Using AUTO_INCREMENT”

LCASE()

Section 12.8.3, “Character Set and Collation of Function Results”
Section 12.8, “String Functions and Operators”

LEAST()

Section 12.8.3, “Character Set and Collation of Function Results”
Section 12.4.2, “Comparison Functions and Operators”

LEFT()

Section 12.8, “String Functions and Operators”

LENGTH()

Section 11.6, “Data Type Storage Requirements”
Section 12.8, “String Functions and Operators”
Section 11.4.3, “Supported Spatial Data Formats”

Length()

Section 12.17.7.3, “LineString and MultiLineString Property Functions”
Section 11.4, “Spatial Data Types”

LineFromText()

Section 12.17.3, “Functions That Create Geometry Values from WKT Values”

LineFromWKB()

Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

LineString()

Section 12.17.5, “MySQL-Specific Functions That Create Geometry Values”

LineStringFromText()

Section 12.17.3, “Functions That Create Geometry Values from WKT Values”

LineStringFromWKB()
Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

LN()
Section 12.6.2, “Mathematical Functions”

LOAD_FILE()
Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.10.3.1, “How the Query Cache Operates”
Section 13.2.7, “LOAD XML Statement”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 6.5.2, “MySQL Enterprise Encryption Usage and Examples”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.4.1.15, “Replication and System Functions”
Section 5.1.7, “Server System Variables”
Section 12.8, “String Functions and Operators”

LOCALTIME
Section 11.2.6, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”
Section 12.7, “Date and Time Functions”

LOCALTIME()
Section 11.2.6, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”
Section 12.7, “Date and Time Functions”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”

LOCALTIMESTAMP
Section 11.2.6, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”
Section 12.7, “Date and Time Functions”

LOCALTIMESTAMP()
Section 11.2.6, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”
Section 12.7, “Date and Time Functions”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”

LOCATE()
Section 12.8, “String Functions and Operators”

LOG()
Section 19.2.4.1, “LINEAR HASH Partitioning”
Section 12.6.2, “Mathematical Functions”

LOG10()
Section 12.6.2, “Mathematical Functions”

LOG2()
Section 12.6.2, “Mathematical Functions”

LOWER()
Section 12.11, “Cast Functions and Operators”
Section 12.8.3, “Character Set and Collation of Function Results”
Section 12.8, “String Functions and Operators”
Section 10.10.1, “Unicode Character Sets”
Section 10.8.7, “Using Collation in INFORMATION_SCHEMA Searches”

LPAD()
Section 12.8, “String Functions and Operators”

LTRIM()
Section 12.8.3, “Character Set and Collation of Function Results”
Section 12.8, “String Functions and Operators”

M

[index top]

MAKE_SET()
Section 12.8, “String Functions and Operators”

MAKEDATE()
Section 12.7, “Date and Time Functions”

MAKETIME()
Section 12.7, “Date and Time Functions”

MASTER_POS_WAIT()
Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.10.3.1, “How the Query Cache Operates”
Section 12.20, “Miscellaneous Functions”
Section A.14, “MySQL 5.6 FAQ: Replication”

MATCH
Section 9.5, “Expressions”

MATCH ()
Section 12.10, “Full-Text Search Functions”

MATCH()
Section 12.10.2, “Boolean Full-Text Searches”
Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”
Section 12.10.5, “Full-Text Restrictions”
Section 12.10, “Full-Text Search Functions”
Section 14.6.2.3, “InnoDB FULLTEXT Indexes”
MySQL Glossary
Section 12.10.1, “Natural Language Full-Text Searches”
Section 1.3, “What Is New in MySQL 5.6”

MAX()
Section 11.2.9, “2-Digit Years in Dates”
Section 12.19.1, “Aggregate Function Descriptions”
Section 8.2.1.14, “GROUP BY Optimization”
Section 8.3.1, “How MySQL Uses Indexes”
Section B.3.7, “Known Issues in MySQL”
Section 11.1.1, “Numeric Data Type Syntax”
Section 13.2.10.10, “Optimizing Subqueries”
Section 5.1.10, “Server SQL Modes”
Section 21.32.1, “The INFORMATION_SCHEMA TP_THREAD_GROUP_STATE Table”
Section 1.2.2, “The Main Features of MySQL”
Section 20.5.3, “Updatable and Insertable Views”

Section 8.3.9, “Use of Index Extensions”
Section 3.6.9, “Using AUTO_INCREMENT”
Section 20.5.2, “View Processing Algorithms”

MBRContains()

Section 12.17.9.2, “Spatial Relation Functions That Use Minimum Bounding Rectangles”
Section 11.4.9, “Using Spatial Indexes”

MBRDisjoint()

Section 12.17.9.2, “Spatial Relation Functions That Use Minimum Bounding Rectangles”

MBREqual()

Section 12.17.9.2, “Spatial Relation Functions That Use Minimum Bounding Rectangles”

MBRIntersects()

Section 12.17.9.2, “Spatial Relation Functions That Use Minimum Bounding Rectangles”

MBROverlaps()

Section 12.17.9.2, “Spatial Relation Functions That Use Minimum Bounding Rectangles”

MBRTouches()

Section 12.17.9.2, “Spatial Relation Functions That Use Minimum Bounding Rectangles”

MBRWithin()

Section 12.17.9.2, “Spatial Relation Functions That Use Minimum Bounding Rectangles”
Section 11.4.9, “Using Spatial Indexes”

MD5()

Section 12.14, “Encryption and Compression Functions”
Section 6.1.2.5, “Implications of Password Hashing Changes in MySQL 4.1 for Application Programs”
Section 19.2.5, “KEY Partitioning”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 9.2, “Schema Object Names”

MICROSECOND()

Section 12.7, “Date and Time Functions”
Section 19.6.3, “Partitioning Limitations Relating to Functions”

MID()

Section 12.8.3, “Character Set and Collation of Function Results”
Section 12.8, “String Functions and Operators”

MIN()

Section 11.2.9, “2-Digit Years in Dates”
Section 12.19.1, “Aggregate Function Descriptions”
Section 8.2.1.14, “GROUP BY Optimization”
Section 8.3.1, “How MySQL Uses Indexes”
Section B.3.7, “Known Issues in MySQL”
Section 11.1.1, “Numeric Data Type Syntax”
Section 13.2.10.10, “Optimizing Subqueries”
Section B.3.4.3, “Problems with NULL Values”
Section 1.2.2, “The Main Features of MySQL”
Section 20.5.3, “Updatable and Insertable Views”
Section 8.3.9, “Use of Index Extensions”
Section 20.5.2, “View Processing Algorithms”

Section 8.2.1.1, “WHERE Clause Optimization”

MINUTE()

Section 12.7, “Date and Time Functions”
Section 19.6.3, “Partitioning Limitations Relating to Functions”

MLineFromText()

Section 12.17.3, “Functions That Create Geometry Values from WKT Values”

MLineFromWKB()

Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

MOD()

Section 12.6.1, “Arithmetic Operators”
Section 3.3.4.5, “Date Calculations”
Section 12.6.2, “Mathematical Functions”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 19.6.3, “Partitioning Limitations Relating to Functions”
Section 5.1.10, “Server SQL Modes”

MONTH()

Section 12.7, “Date and Time Functions”
Section 3.3.4.5, “Date Calculations”
Section 19.6.3, “Partitioning Limitations Relating to Functions”
Section 19.2, “Partitioning Types”

MONTHNAME()

Section 12.7, “Date and Time Functions”
Section 10.16, “MySQL Server Locale Support”
Section 5.1.7, “Server System Variables”

MPointFromText()

Section 12.17.3, “Functions That Create Geometry Values from WKT Values”

MPointFromWKB()

Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

MPolyFromText()

Section 12.17.3, “Functions That Create Geometry Values from WKT Values”

MPolyFromWKB()

Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

MultiLineString()

Section 12.17.5, “MySQL-Specific Functions That Create Geometry Values”

MultiLineStringFromText()

Section 12.17.3, “Functions That Create Geometry Values from WKT Values”

MultiLineStringFromWKB()

Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

MultiPoint()

Section 12.17.5, “MySQL-Specific Functions That Create Geometry Values”

MultiPointFromText()
Section 12.17.3, “Functions That Create Geometry Values from WKT Values”

MultiPointFromWKB()
Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

MultiPolygon()
Section 12.17.5, “MySQL-Specific Functions That Create Geometry Values”

MultiPolygonFromText()
Section 12.17.3, “Functions That Create Geometry Values from WKT Values”

MultiPolygonFromWKB()
Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

my_open()
Section 5.1.9, “Server Status Variables”

N

[index top]

NAME_CONST()
Section 12.20, “Miscellaneous Functions”
Section 20.7, “Stored Program Binary Logging”

NOW()
Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 11.2.6, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 11.5, “Data Type Default Values”
Section 12.7, “Date and Time Functions”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 11.2.7, “Fractional Seconds in Time Values”
Section 8.10.3.1, “How the Query Cache Operates”
Section A.1, “MySQL 5.6 FAQ: General”
Section 5.1.13, “MySQL Server Time Zone Support”
Section 17.4.1.15, “Replication and System Functions”
Section 17.4.1.31, “Replication and Time Zones”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 11.2.4, “The YEAR Type”

NULLIF()
Section 12.5, “Flow Control Functions”

NumGeometries()
Section 12.17.7.5, “GeometryCollection Property Functions”

NumInteriorRings()
Section 12.17.7.4, “Polygon and MultiPolygon Property Functions”

NumPoints()
Section 12.17.7.3, “LineString and MultiLineString Property Functions”

O

[index top]

OCT()
Section 12.8, “String Functions and Operators”

OCTET_LENGTH()
Section 12.8, “String Functions and Operators”

OLD_PASSWORD()
Section 2.11.3, “Changes in MySQL 5.6”
Section B.3.2.4, “Client does not support authentication protocol”
Section 13.7.1.2, “CREATE USER Statement”
Section 12.14, “Encryption and Compression Functions”
Section 6.1.2.5, “Implications of Password Hashing Changes in MySQL 4.1 for Application Programs”
Section 6.1.2.4, “Password Hashing in MySQL”
Section 5.1.7, “Server System Variables”
Section 13.7.1.7, “SET PASSWORD Statement”
Section 6.4.3, “The Password Validation Plugin”
Section 1.3, “What Is New in MySQL 5.6”

ORD()
Section 12.8, “String Functions and Operators”

Overlaps()
Section 12.17.9.2, “Spatial Relation Functions That Use Minimum Bounding Rectangles”

P

[index top]

PASSWORD()
Section 6.2.5, “Access Control, Stage 1: Connection Verification”
Section 6.2.9, “Assigning Account Passwords”
Section 2.11.3, “Changes in MySQL 5.6”
Section 13.7.1.2, “CREATE USER Statement”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 12.14, “Encryption and Compression Functions”
Section 8.10.3.1, “How the Query Cache Operates”
Section B.3.2.14, “Ignoring user”
Section 6.1.2.5, “Implications of Password Hashing Changes in MySQL 4.1 for Application Programs”
Section 19.2.5, “KEY Partitioning”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 6.1.2.4, “Password Hashing in MySQL”
Section 6.1.2.3, “Passwords and Logging”
Section 5.1.7, “Server System Variables”
Section 13.7.1.7, “SET PASSWORD Statement”
Section 6.4.1.4, “SHA-256 Pluggable Authentication”
Section 6.4.3, “The Password Validation Plugin”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 1.3, “What Is New in MySQL 5.6”

PERIOD_ADD()
Section 12.7, “Date and Time Functions”
Section 1.7.1, “MySQL Extensions to Standard SQL”

PERIOD_DIFF()
Section 12.7, “Date and Time Functions”
Section 1.7.1, “MySQL Extensions to Standard SQL”

PI()
Section 9.2.5, “Function Name Parsing and Resolution”
Section 12.6.2, “Mathematical Functions”

Point()
Section 12.17.5, “MySQL-Specific Functions That Create Geometry Values”
Section 11.4.3, “Supported Spatial Data Formats”

PointFromText()
Section 12.17.3, “Functions That Create Geometry Values from WKT Values”

PointFromWKB()
Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

PointN()
Section 12.17.7.3, “LineString and MultiLineString Property Functions”

PolyFromText()
Section 12.17.3, “Functions That Create Geometry Values from WKT Values”

PolyFromWKB()
Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

Polygon()
Section 12.17.5, “MySQL-Specific Functions That Create Geometry Values”

PolygonFromText()
Section 12.17.3, “Functions That Create Geometry Values from WKT Values”

PolygonFromWKB()
Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

POSITION()
Section 12.8, “String Functions and Operators”

POW()
Section 8.2.1.17, “Function Call Optimization”
Section 19.2.4, “HASH Partitioning”
Section 12.6.2, “Mathematical Functions”

POWER()
Section 19.2.4.1, “LINEAR HASH Partitioning”
Section 12.6.2, “Mathematical Functions”

pthread_mutex()
Section 1.8.1, “Contributors to MySQL”

Q

[index top]

QUARTER()
Section 12.7, “Date and Time Functions”
Section 19.6.3, “Partitioning Limitations Relating to Functions”

QUOTE()
Section 12.8, “String Functions and Operators”
Section 9.1.1, “String Literals”

R

[index top]

RADIANS()
Section 12.6.2, “Mathematical Functions”

RAND()
Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.2.1.17, “Function Call Optimization”
Section 8.10.3.1, “How the Query Cache Operates”
Section 12.6.2, “Mathematical Functions”
Section 17.4.1.15, “Replication and System Functions”
Section 5.1.7, “Server System Variables”

RANDOM_BYTES()
Section 12.14, “Encryption and Compression Functions”
Section 8.10.3.1, “How the Query Cache Operates”

RELEASE_ALL_LOCKS()
Section 8.10.3.1, “How the Query Cache Operates”

RELEASE_LOCK()
Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 13.2.3, “DO Statement”
Section 8.10.3.1, “How the Query Cache Operates”
Section 8.11.1, “Internal Locking Methods”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 12.15, “Locking Functions”
Section 17.4.1.15, “Replication and System Functions”

REPEAT()
Section 12.8.3, “Character Set and Collation of Function Results”
Section 12.8, “String Functions and Operators”

REPLACE()
Section 12.8.3, “Character Set and Collation of Function Results”
Section 12.8, “String Functions and Operators”

replace()
Section 16.2.3.1, “Basic `memcached` Operations”

REVERSE()
Section 12.8.3, “Character Set and Collation of Function Results”

Section 12.8, “String Functions and Operators”

RIGHT()

Section 12.8.3, “Character Set and Collation of Function Results”
Section 12.8, “String Functions and Operators”

ROUND()

Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”
Section 12.6.2, “Mathematical Functions”
Section 12.21, “Precision Math”
Section 12.21.5, “Precision Math Examples”
Section 12.21.4, “Rounding Behavior”

ROW_COUNT()

Section 13.2.1, “CALL Statement”
Section 13.2.2, “DELETE Statement”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 12.16, “Information Functions”
Section 13.2.5, “INSERT Statement”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.4.1.15, “Replication and System Functions”
Section 13.6.7.7, “The MySQL Diagnostics Area”

RPAD()

Section 12.8.3, “Character Set and Collation of Function Results”
Section 12.8, “String Functions and Operators”

RTRIM()

Section 12.8.3, “Character Set and Collation of Function Results”
Section 12.8, “String Functions and Operators”

S

[index top]

SCHEMA()

Section 12.16, “Information Functions”

SEC_TO_TIME()

Section 12.7, “Date and Time Functions”

SECOND()

Section 12.7, “Date and Time Functions”
Section 19.6.3, “Partitioning Limitations Relating to Functions”

SESSION_USER()

Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 12.16, “Information Functions”
Section 10.2.2, “UTF-8 for Metadata”

set()

Section 16.2.3.1, “Basic `memcached` Operations”

setrlimit()

Section 5.1.7, “Server System Variables”

SHA()
Section 12.14, “Encryption and Compression Functions”

SHA1()
Section 12.14, “Encryption and Compression Functions”
Section 6.1.2.5, “Implications of Password Hashing Changes in MySQL 4.1 for Application Programs”

SHA2()
Section 12.14, “Encryption and Compression Functions”
Section 6.1.2.5, “Implications of Password Hashing Changes in MySQL 4.1 for Application Programs”
Section 6.1.1, “Security Guidelines”

SIGN()
Section 12.6.2, “Mathematical Functions”

SIN()
Section 12.6.2, “Mathematical Functions”

SLEEP()
Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.14.3, “General Thread States”
Section 8.10.3.1, “How the Query Cache Operates”
Section 12.20, “Miscellaneous Functions”
Section 21.32.2, “The INFORMATION_SCHEMA TP_THREAD_GROUP_STATS Table”

SOUNDEX()
Section 12.8.3, “Character Set and Collation of Function Results”
Section 12.8, “String Functions and Operators”

SPACE()
Section 12.8.3, “Character Set and Collation of Function Results”
Section 12.8, “String Functions and Operators”

SQL_THREAD_WAIT_AFTER_GTIDS()
Section 12.18, “Functions Used with Global Transaction IDs”

SQRT()
Section 12.6.2, “Mathematical Functions”

SRID()
Section 12.17.7.1, “General Geometry Property Functions”

ST_Area()
Section 12.17.7, “Geometry Property Functions”
Section 12.17.7.4, “Polygon and MultiPolygon Property Functions”

ST_AsBinary()
Section 12.17.6, “Geometry Format Conversion Functions”

ST_AsText()
Section 12.17.6, “Geometry Format Conversion Functions”

ST_AsWKB()
Section 12.17.6, “Geometry Format Conversion Functions”

ST_AsWKT()

Section 12.17.6, “Geometry Format Conversion Functions”

ST_Buffer()

Section 12.17.8, “Spatial Operator Functions”

ST_Centroid()

Section 12.17.7.4, “Polygon and MultiPolygon Property Functions”

ST_Contains()

Section 12.17.9.1, “Spatial Relation Functions That Use Object Shapes”
Section 1.3, “What Is New in MySQL 5.6”

ST_Crosses()

Section 12.17.9.1, “Spatial Relation Functions That Use Object Shapes”

ST_Difference()

Section 12.17.8, “Spatial Operator Functions”

ST_Dimension()

Section 12.17.7.1, “General Geometry Property Functions”

ST_Disjoint()

Section 12.17.9.1, “Spatial Relation Functions That Use Object Shapes”

ST_Distance()

Section 12.17.9.1, “Spatial Relation Functions That Use Object Shapes”

ST_EndPoint()

Section 12.17.7.3, “LineString and MultiLineString Property Functions”
Section 12.17.8, “Spatial Operator Functions”

ST_Envelope()

Section 12.17.7.1, “General Geometry Property Functions”
Section 12.17.8, “Spatial Operator Functions”

ST_Equals()

Section 12.17.9.1, “Spatial Relation Functions That Use Object Shapes”

ST_ExteriorRing()

Section 12.17.7.4, “Polygon and MultiPolygon Property Functions”
Section 12.17.8, “Spatial Operator Functions”

ST_GeomCollFromText()

Section 12.17.3, “Functions That Create Geometry Values from WKT Values”

ST_GeomCollFromWKB()

Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

ST_GeometryCollectionFromText()

Section 12.17.3, “Functions That Create Geometry Values from WKT Values”

ST_GeometryCollectionFromWKB()

Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

ST_GeometryFromText()

Section 12.17.3, “Functions That Create Geometry Values from WKT Values”

ST_GeometryFromWKB()

Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

ST_GeometryN()

Section 12.17.7.5, “GeometryCollection Property Functions”
Section 12.17.8, “Spatial Operator Functions”

ST_GeometryType()

Section 12.17.7.1, “General Geometry Property Functions”

ST_GeomFromText()

Section 12.17.3, “Functions That Create Geometry Values from WKT Values”
Section 11.4.3, “Supported Spatial Data Formats”

ST_GeomFromWKB()

Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

ST_InteriorRingN()

Section 12.17.7.4, “Polygon and MultiPolygon Property Functions”
Section 12.17.8, “Spatial Operator Functions”

ST_Intersection()

Section 12.17.8, “Spatial Operator Functions”

ST_Intersects()

Section 12.17.9.1, “Spatial Relation Functions That Use Object Shapes”

ST_IsClosed()

Section 12.17.7.3, “LineString and MultiLineString Property Functions”

ST_IsEmpty()

Section 12.17.7.1, “General Geometry Property Functions”

ST_IsSimple()

Section 12.17.7.1, “General Geometry Property Functions”

ST_LineFromText()

Section 12.17.3, “Functions That Create Geometry Values from WKT Values”

ST_LineFromWKB()

Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

ST_LineStringFromText()

Section 12.17.3, “Functions That Create Geometry Values from WKT Values”

ST_LineStringFromWKB()

Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

ST_NumGeometries()

Section 12.17.7.5, “GeometryCollection Property Functions”

ST_NumInteriorRings()
Section 12.17.7.4, “Polygon and MultiPolygon Property Functions”

ST_NumPoints()
Section 12.17.7.3, “LineString and MultiLineString Property Functions”

ST_Overlaps()
Section 12.17.9.1, “Spatial Relation Functions That Use Object Shapes”

ST_PointFromText()
Section 12.17.3, “Functions That Create Geometry Values from WKT Values”

ST_PointFromWKB()
Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

ST_PointN()
Section 12.17.7.3, “LineString and MultiLineString Property Functions”
Section 12.17.8, “Spatial Operator Functions”

ST_PolyFromText()
Section 12.17.3, “Functions That Create Geometry Values from WKT Values”

ST_PolyFromWKB()
Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

ST_PolygonFromText()
Section 12.17.3, “Functions That Create Geometry Values from WKT Values”

ST_PolygonFromWKB()
Section 12.17.4, “Functions That Create Geometry Values from WKB Values”

ST_SRID()
Section 12.17.7.1, “General Geometry Property Functions”

ST_StartPoint()
Section 12.17.7.3, “LineString and MultiLineString Property Functions”

ST_SymDifference()
Section 12.17.8, “Spatial Operator Functions”

ST_Touches()
Section 12.17.9.1, “Spatial Relation Functions That Use Object Shapes”

ST_Union()
Section 12.17.8, “Spatial Operator Functions”

ST_Within()
Section 12.17.9.1, “Spatial Relation Functions That Use Object Shapes”

ST_X()
Section 12.17.7.2, “Point Property Functions”
Section 11.4.3, “Supported Spatial Data Formats”

ST_Y()
Section 12.17.7.2, “Point Property Functions”

StartPoint()
Section 12.17.7.3, “LineString and MultiLineString Property Functions”
Section 12.17.8, “Spatial Operator Functions”

STD()
Section 12.19.1, “Aggregate Function Descriptions”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 1.2.2, “The Main Features of MySQL”

STDDEV()
Section 12.19.1, “Aggregate Function Descriptions”

STDDEV_POP()
Section 12.19.1, “Aggregate Function Descriptions”

STDDEV_SAMP()
Section 12.19.1, “Aggregate Function Descriptions”

STR_TO_DATE()
Section 11.2, “Date and Time Data Types”
Section 12.7, “Date and Time Functions”
Section 10.16, “MySQL Server Locale Support”

STRCMP()
Section B.3.4.2, “Problems Using DATE Columns”
Section 12.8.1, “String Comparison Functions and Operators”

SUBDATE()
Section 12.7, “Date and Time Functions”

SUBSTR()
Section 12.8, “String Functions and Operators”

SUBSTRING()
Section 12.8.3, “Character Set and Collation of Function Results”
Section 12.8, “String Functions and Operators”

SUBSTRING_INDEX()
Section 6.2.15, “SQL-Based Account Activity Auditing”
Section 12.8, “String Functions and Operators”

SUBTIME()
Section 12.7, “Date and Time Functions”

SUM()
Section 12.19.1, “Aggregate Function Descriptions”
Section 13.7.3.1, “CREATE FUNCTION Statement for User-Defined Functions”
Section 11.2.1, “Date and Time Data Type Syntax”
Section 8.2.1.14, “GROUP BY Optimization”
Section B.3.4.3, “Problems with NULL Values”
Section 11.3.5, “The ENUM Type”

Section 1.2.2, “The Main Features of MySQL”
Section 11.3.6, “The SET Type”
Section 20.5.3, “Updatable and Insertable Views”
Section 20.5.2, “View Processing Algorithms”

SYSDATE()

Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 12.7, “Date and Time Functions”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 11.2.7, “Fractional Seconds in Time Values”
Section 8.10.3.1, “How the Query Cache Operates”
Section 17.4.1.14, “Replication and Fractional Seconds Support”
Section 17.4.1.15, “Replication and System Functions”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”

SYSTEM_USER()

Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 12.16, “Information Functions”
Section 10.2.2, “UTF-8 for Metadata”

T

[index top]

TAN()

Section 12.6.2, “Mathematical Functions”

thr_setconcurrency()

Section 5.1.7, “Server System Variables”

TIME()

Section 12.7, “Date and Time Functions”

TIME_FORMAT()

Section 12.11, “Cast Functions and Operators”
Section 12.7, “Date and Time Functions”

TIME_TO_SEC()

Section 12.7, “Date and Time Functions”
Section 19.6.3, “Partitioning Limitations Relating to Functions”

TIMEDIFF()

Section 12.7, “Date and Time Functions”

TIMESTAMP()

Section 12.7, “Date and Time Functions”

TIMESTAMPADD()

Section 12.7, “Date and Time Functions”

TIMESTAMPDIFF()

Section 12.7, “Date and Time Functions”
Section 3.3.4.5, “Date Calculations”

TO_BASE64()
Section 12.8, “String Functions and Operators”

TO_DAYS()
Section 12.7, “Date and Time Functions”
Section 19.2.4, “HASH Partitioning”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 19.4, “Partition Pruning”
Section 19.6.3, “Partitioning Limitations Relating to Functions”
Section 19.2, “Partitioning Types”

TO_SECONDS()
Section 12.7, “Date and Time Functions”
Section 19.4, “Partition Pruning”
Section 19.6.3, “Partitioning Limitations Relating to Functions”
Section 19.2, “Partitioning Types”

Touches()
Section 12.17.9.1, “Spatial Relation Functions That Use Object Shapes”

TRIM()
Section 12.8.3, “Character Set and Collation of Function Results”
Section 10.7, “Column Character Set Conversion”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 12.8, “String Functions and Operators”

TRUNCATE()
Section 12.6.2, “Mathematical Functions”

U

[index top]

UCASE()
Section 12.8.3, “Character Set and Collation of Function Results”
Section 12.8, “String Functions and Operators”

UNCOMPRESS()
Section 12.14, “Encryption and Compression Functions”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 5.1.7, “Server System Variables”

UNCOMPRESSED_LENGTH()
Section 12.14, “Encryption and Compression Functions”

UNHEX()
Section 12.14, “Encryption and Compression Functions”
Section 4.5.1.1, “mysql Client Options”
Section 12.8, “String Functions and Operators”

UNIX_TIMESTAMP()
Section 12.7, “Date and Time Functions”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.10.3.1, “How the Query Cache Operates”

Section 8.3.10, “Indexed Lookups from TIMESTAMP Columns”
Section 19.6.3, “Partitioning Limitations Relating to Functions”
Section 19.2.1, “RANGE Partitioning”
Section 5.1.7, “Server System Variables”
Section B.3.3.7, “Time Zone Problems”

UpdateXML()

Section 12.12, “XML Functions”

UPPER()

Section 12.11, “Cast Functions and Operators”
Section 12.8.3, “Character Set and Collation of Function Results”
Section 10.2.1, “Character Set Repertoire”
Section 12.8, “String Functions and Operators”
Section 10.10.1, “Unicode Character Sets”
Section 10.8.7, “Using Collation in INFORMATION_SCHEMA Searches”

USER()

Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 10.8.4, “Collation Coercibility in Expressions”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.10.3.1, “How the Query Cache Operates”
Section 12.16, “Information Functions”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 6.2.12, “Proxy Users”
Section 17.4.1.15, “Replication and System Functions”
Section 6.2.15, “SQL-Based Account Activity Auditing”
Section 10.2.2, “UTF-8 for Metadata”

UTC_DATE

Section 12.7, “Date and Time Functions”

UTC_DATE()

Section 12.7, “Date and Time Functions”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”

UTC_TIME

Section 12.7, “Date and Time Functions”

UTC_TIME()

Section 12.7, “Date and Time Functions”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”

UTC_TIMESTAMP

Section 12.7, “Date and Time Functions”

UTC_TIMESTAMP()

Section 12.7, “Date and Time Functions”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 11.2.7, “Fractional Seconds in Time Values”
Section 5.1.13, “MySQL Server Time Zone Support”
Section 17.4.1.14, “Replication and Fractional Seconds Support”

UUID()

Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”

Section 17.1.4.4, “Binary Log Options and Variables”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.2.1.17, “Function Call Optimization”
Section 8.10.3.1, “How the Query Cache Operates”
Section 12.20, “Miscellaneous Functions”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.4.1.15, “Replication and System Functions”
Section 5.4.4.2, “Setting The Binary Log Format”
Section 20.7, “Stored Program Binary Logging”

UUID_SHORT()

Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 8.10.3.1, “How the Query Cache Operates”
Section 12.20, “Miscellaneous Functions”

V

[index top]

VALIDATE_PASSWORD_STRENGTH()

Section 12.14, “Encryption and Compression Functions”
Section 6.4.3.2, “Password Validation Plugin Options and Variables”
Section 6.4.3, “The Password Validation Plugin”
Section 1.3, “What Is New in MySQL 5.6”

VALUES()

Section 13.2.5.2, “INSERT ... ON DUPLICATE KEY UPDATE Statement”
Section 12.20, “Miscellaneous Functions”

VAR_POP()

Section 12.19.1, “Aggregate Function Descriptions”

VAR_SAMP()

Section 12.19.1, “Aggregate Function Descriptions”

VARIANCE()

Section 12.19.1, “Aggregate Function Descriptions”

VERSION()

Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 6.4.4.3, “Audit Log File Formats”
Section B.3.4.1, “Case Sensitivity in String Searches”
Section 10.8.4, “Collation Coercibility in Expressions”
Section 12.16, “Information Functions”
Section 17.4.1.15, “Replication and System Functions”
Section 10.2.2, “UTF-8 for Metadata”

W

[index top]

WAIT_UNTIL_SQL_THREAD_AFTER_GTIDS()

Section 12.18, “Functions Used with Global Transaction IDs”

WEEK()
Section 12.7, “Date and Time Functions”
Section 5.1.7, “Server System Variables”

WEEKDAY()
Section 12.7, “Date and Time Functions”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 19.6.3, “Partitioning Limitations Relating to Functions”
Section 19.2, “Partitioning Types”

WEEKOFYEAR()
Section 12.7, “Date and Time Functions”

WEIGHT_STRING()
Section 10.14, “Adding a Collation to a Character Set”
Section B.3.4.1, “Case Sensitivity in String Searches”
Section 12.8, “String Functions and Operators”
Section 10.10.1, “Unicode Character Sets”

Within()
Section 12.17.9.2, “Spatial Relation Functions That Use Minimum Bounding Rectangles”

X

[index top]

X()
Section 12.17.7.2, “Point Property Functions”

Y

[index top]

Y()
Section 12.17.7.2, “Point Property Functions”

YEAR()
Section 12.7, “Date and Time Functions”
Section 3.3.4.5, “Date Calculations”
Section 19.2.4, “HASH Partitioning”
Section 19.2.7, “How MySQL Partitioning Handles NULL”
Section 19.3.1, “Management of RANGE and LIST Partitions”
Section 19.4, “Partition Pruning”
Section 19.6.3, “Partitioning Limitations Relating to Functions”
Section 19.2, “Partitioning Types”
Section 19.2.1, “RANGE Partitioning”

YEARWEEK()
Section 12.7, “Date and Time Functions”
Section 19.6.3, “Partitioning Limitations Relating to Functions”

INFORMATION_SCHEMA Index

C | E | F | G | I | K | M | N | O | P | R | S | T | U | V

C

[index top]

CHARACTER_SETS

Section 10.3.8, “Character Set Introducers”
Section 10.2, “Character Sets and Collations in MySQL”
Section 10.3.6, “Character String Literal Character Set and Collation”
Section 10.3.5, “Column Character Set and Collation”
Section 10.3.3, “Database Character Set and Collation”
Section 13.7.5.4, “SHOW CHARACTER SET Statement”
Section 10.10, “Supported Character Sets and Collations”
Section 10.3.4, “Table Character Set and Collation”
Section 21.2, “The INFORMATION_SCHEMA CHARACTER_SETS Table”

COLLATION_CHARACTER_SET_APPLICABILITY

Section 21.4, “The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table”

COLLATIONS

Section 10.15, “Character Set Configuration”
Section 10.2, “Character Sets and Collations in MySQL”
Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”
Section 13.7.5.5, “SHOW COLLATION Statement”
Section 21.3, “The INFORMATION_SCHEMA COLLATIONS Table”

COLUMN_PRIVILEGES

Section 21.6, “The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table”

COLUMNS

Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”
Section 13.7.5.6, “SHOW COLUMNS Statement”
Section 21.5, “The INFORMATION_SCHEMA COLUMNS Table”
Section 21.30.1, “The INFORMATION_SCHEMA INNODB_BUFFER_PAGE Table”
Section 21.30.2, “The INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU Table”
Section 21.30.3, “The INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS Table”
Section 21.30.4, “The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables”
Section 21.30.6, “The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables”
Section 21.30.5, “The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables”
Section 21.30.7, “The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table”
Section 21.30.8, “The INFORMATION_SCHEMA INNODB_FT_CONFIG Table”
Section 21.30.9, “The INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD Table”
Section 21.30.10, “The INFORMATION_SCHEMA INNODB_FT_DELETED Table”
Section 21.30.11, “The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table”
Section 21.30.12, “The INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE Table”
Section 21.30.14, “The INFORMATION_SCHEMA INNODB_LOCK_WAITS Table”
Section 21.30.13, “The INFORMATION_SCHEMA INNODB_LOCKS Table”
Section 21.30.15, “The INFORMATION_SCHEMA INNODB_METRICS Table”
Section 21.30.16, “The INFORMATION_SCHEMA INNODB_SYS_COLUMNS Table”
Section 21.30.17, “The INFORMATION_SCHEMA INNODB_SYS_DATAFILES Table”
Section 21.30.18, “The INFORMATION_SCHEMA INNODB_SYS_FIELDS Table”
Section 21.30.19, “The INFORMATION_SCHEMA INNODB_SYS_FOREIGN Table”
Section 21.30.20, “The INFORMATION_SCHEMA INNODB_SYS_FOREIGN_COLS Table”
Section 21.30.21, “The INFORMATION_SCHEMA INNODB_SYS_INDEXES Table”

Section 21.30.22, “The INFORMATION_SCHEMA INNODB_SYS_TABLES Table”
Section 21.30.23, “The INFORMATION_SCHEMA INNODB_SYS_TABLESPACES Table”
Section 21.30.24, “The INFORMATION_SCHEMA INNODB_SYS_TABLESTATS View”
Section 21.30.25, “The INFORMATION_SCHEMA INNODB_TRX Table”

CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS

Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 6.4.2.2, “Connection-Control System and Status Variables”
Section 6.4.2, “The Connection-Control Plugins”
Section 21.33.1, “The INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS Table”

E

[index top]

ENGINES

Section 18.5.15, “Quick Reference: NDB Cluster SQL Statements”
Section 5.1.7, “Server System Variables”
Section 13.7.5.17, “SHOW ENGINES Statement”
Section 21.7, “The INFORMATION_SCHEMA ENGINES Table”

EVENTS

Section 20.4.4, “Event Metadata”
Section 20.4.2, “Event Scheduler Configuration”
Section 17.4.1.16, “Replication of Invoked Features”
Section 13.7.5.19, “SHOW EVENTS Statement”
Section 21.8, “The INFORMATION_SCHEMA EVENTS Table”

F

[index top]

FILES

Section 21.31, “INFORMATION_SCHEMA NDB Cluster Tables”
Section 18.5.10.1, “NDB Cluster Disk Data Objects”
Section 18.5.14, “ndbinfo: The NDB Cluster Information Database”
Section 6.2.2, “Privileges Provided by MySQL”
Section 21.31.1, “The INFORMATION_SCHEMA FILES Table”
Section 21.24, “The INFORMATION_SCHEMA TABLESPACES Table”

G

[index top]

GLOBAL_STATUS

Section 18.5, “Management of NDB Cluster”
Section 18.5.13, “NDB API Statistics Counters and Variables”
Section 18.6, “NDB Cluster Replication”
Section 13.7.5.36, “SHOW STATUS Statement”
Section 21.9, “The INFORMATION_SCHEMA GLOBAL_STATUS and SESSION_STATUS Tables”

GLOBAL_VARIABLES

Section 13.7.5.40, “SHOW VARIABLES Statement”
Section 21.10, “The INFORMATION_SCHEMA GLOBAL_VARIABLES and SESSION_VARIABLES Tables”

I

[index top]

INFORMATION_SCHEMA
Section 2.12.3, “Downgrade Notes”
Section 14.15, “InnoDB INFORMATION_SCHEMA Tables”
MySQL Glossary
Section 5.2, “The MySQL Data Directory”

INFORMATION_SCHEMA.CHARACTER_SETS
Section A.11, “MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”

INFORMATION_SCHEMA.COLLATIONS
Section 10.14.2, “Choosing a Collation ID”

INFORMATION_SCHEMA.COLUMNS
Section A.11, “MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 18.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”
Section 22.1, “Performance Schema Quick Start”
Section 5.1.7, “Server System Variables”

INFORMATION_SCHEMA.ENGINES
Section 14.1.3, “Checking InnoDB Availability”
Section 22.1, “Performance Schema Quick Start”
Section 18.5.15, “Quick Reference: NDB Cluster SQL Statements”

INFORMATION_SCHEMA.EVENTS
Section 20.4.4, “Event Metadata”
Section 17.4.1.16, “Replication of Invoked Features”
Section 20.8, “Restrictions on Stored Programs”
Section 20.4.6, “The Event Scheduler and MySQL Privileges”

INFORMATION_SCHEMA.FILES
Section 13.1.4, “ALTER LOGFILE GROUP Statement”
Section 13.1.8, “ALTER TABLESPACE Statement”
Section 13.1.14, “CREATE LOGFILE GROUP Statement”
Section 13.1.18, “CREATE TABLESPACE Statement”
Section 18.5.10.3, “NDB Cluster Disk Data Storage Requirements”
Section 18.4.10, “`ndb_desc` — Describe NDB Tables”

INFORMATION_SCHEMA.GLOBAL_STATUS
Section 18.5.13, “NDB API Statistics Counters and Variables”

INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
Section 14.5.2, “Change Buffer”

INFORMATION_SCHEMA.INNODB_CMP
MySQL Glossary
Section 14.9.3, “Tuning Compression for InnoDB Tables”
Section 14.15.1.3, “Using the Compression Information Schema Tables”

INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX
Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.9.3, “Tuning Compression for InnoDB Tables”

INFORMATION_SCHEMA.INNODB_CMPMEM

Section 14.15.1.3, “Using the Compression Information Schema Tables”

INFORMATION_SCHEMA.INNODB_FT_CONFIG

Section 14.6.2.3, “InnoDB FULLTEXT Indexes”

INFORMATION_SCHEMA.INNODB_FT_DEFAULT_STOPWORD

Section 12.10.4, “Full-Text Stopwords”

INFORMATION_SCHEMA.INNODB_FT_INDEX_CACHE

Section 14.6.2.3, “InnoDB FULLTEXT Indexes”

INFORMATION_SCHEMA.INNODB_FT_INDEX_TABLE

Section 12.10.4, “Full-Text Stopwords”

INFORMATION_SCHEMA.INNODB_LOCK_WAITS

Section 14.15.2.1, “Using InnoDB Transaction and Locking Information”

INFORMATION_SCHEMA.INNODB_LOCKS

Section 14.15.2.1, “Using InnoDB Transaction and Locking Information”

INFORMATION_SCHEMA.INNODB_METRICS

Section 14.5.2, “Change Buffer”
Section 14.15.6, “InnoDB INFORMATION_SCHEMA Metrics Table”
Section 14.14, “InnoDB Startup Options and System Variables”

INFORMATION_SCHEMA.INNODB_SYS_INDEXES

Section 14.6.2.3, “InnoDB FULLTEXT Indexes”

INFORMATION_SCHEMA.INNODB_SYS_TABLES

Section 19.3.3, “Exchanging Partitions and Subpartitions with Tables”
Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”
Section 14.11, “InnoDB Row Formats”
Section 14.21.3, “Troubleshooting InnoDB Data Dictionary Operations”

INFORMATION_SCHEMA.INNODB_SYS_TABLESPACES

Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”

INFORMATION_SCHEMA.INNODB_SYS_TABLESTATS

Section 13.7.2.1, “ANALYZE TABLE Statement”
Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”

INFORMATION_SCHEMA.INNODB_TRX

Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.15.2.1, “Using InnoDB Transaction and Locking Information”

INFORMATION_SCHEMA.KEY_COLUMN_USAGE

Section 2.11.3, “Changes in MySQL 5.6”
Section 1.7.3.2, “FOREIGN KEY Constraints”
Section 13.1.17.5, “FOREIGN KEY Constraints”

INFORMATION_SCHEMA.MYSQL_FIREWALL_USERS

Section 6.4.5.4, “MySQL Enterprise Firewall Reference”

INFORMATION_SCHEMA.MYSQL_FIREWALL_WHITELIST

Section 6.4.5.4, “MySQL Enterprise Firewall Reference”

INFORMATION_SCHEMA.OPTIMIZER_TRACE

Section 1.3, “What Is New in MySQL 5.6”

INFORMATION_SCHEMA.PARTITIONS

Section 19.3.3, “Exchanging Partitions and Subpartitions with Tables”
Section 19.2.7, “How MySQL Partitioning Handles NULL”
Section 19.2.5, “KEY Partitioning”
Section 19.3.5, “Obtaining Information About Partitions”
Section 19.2.3.1, “RANGE COLUMNS partitioning”
Section 5.1.6, “Server Command Options”

INFORMATION_SCHEMA.PLUGINS

Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 5.5.1, “Installing and Uninstalling Plugins”
Section 6.4.4.1, “Installing MySQL Enterprise Audit”
Section 5.5.2, “Obtaining Server Plugin Information”
Section 6.4.1.6, “PAM Pluggable Authentication”
Chapter 19, *Partitioning*
Section 6.4.3.1, “Password Validation Plugin Installation”
Section 17.3.9.2, “Semisynchronous Replication Installation and Configuration”
Section 6.4.1.8, “Socket Peer-Credential Pluggable Authentication”
Section 6.4.1.9, “Test Pluggable Authentication”
Section 5.5.3.2, “Thread Pool Installation”
Section 6.4.1.7, “Windows Pluggable Authentication”

INFORMATION_SCHEMA.PROCESSLIST

Section 12.16, “Information Functions”
Section 22.6, “Performance Schema Instrument Naming Conventions”
Section 22.12.5, “Performance Schema Stage Event Tables”
Section 6.2.2, “Privileges Provided by MySQL”
Section 14.15.2.1, “Using InnoDB Transaction and Locking Information”

INFORMATION_SCHEMA.ROUTINES

Section A.4, “MySQL 5.6 FAQ: Stored Procedures and Functions”

INFORMATION_SCHEMA.SESSION_STATUS

Section 18.5.13, “NDB API Statistics Counters and Variables”

INFORMATION_SCHEMA.STATISTICS

Section 14.8.11.2, “Configuring Non-Persistent Optimizer Statistics Parameters”
Section 8.9.3, “Index Hints”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 18.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

INFORMATION_SCHEMA.TABLE_CONSTRAINTS

Section 2.11.3, “Changes in MySQL 5.6”
Section 14.13.1, “Online DDL Operations”

INFORMATION_SCHEMA.TABLES

Section 18.5.7.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 13.1.7, “ALTER TABLE Statement”
Section 14.8.11.2, “Configuring Non-Persistent Optimizer Statistics Parameters”

Section 14.14, “InnoDB Startup Options and System Variables”
Section 5.1.6, “Server Command Options”

INFORMATION_SCHEMA.TRIGGERS

Section A.5, “MySQL 5.6 FAQ: Triggers”

INFORMATION_SCHEMA.VIEWS

Section 20.5.3, “Updatable and Insertable Views”

INNODB_BUFFER_PAGE

Section 14.5.2, “Change Buffer”
Section 14.15.5, “InnoDB INFORMATION_SCHEMA Buffer Pool Tables”
Section 21.30.1, “The INFORMATION_SCHEMA INNODB_BUFFER_PAGE Table”
Section 21.30.2, “The INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU Table”

INNODB_BUFFER_PAGE_LRU

Section 14.15.5, “InnoDB INFORMATION_SCHEMA Buffer Pool Tables”
Section 14.8.3.5, “Saving and Restoring the Buffer Pool State”
Section 21.30.2, “The INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU Table”

INNODB_BUFFER_POOL_STATS

Section 14.5.1, “Buffer Pool”
Section 14.15.5, “InnoDB INFORMATION_SCHEMA Buffer Pool Tables”
Section 21.30.3, “The INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS Table”

INNODB_CMP

Section 14.15.1, “InnoDB INFORMATION_SCHEMA Tables about Compression”
Section 14.15.1.1, “INNODB_CMP and INNODB_CMP_RESET”
Section 14.15.1.2, “INNODB_CMPMEM and INNODB_CMPMEM_RESET”
Section 14.9.4, “Monitoring InnoDB Table Compression at Runtime”
Section 21.30.4, “The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables”
Section 14.15.1.3, “Using the Compression Information Schema Tables”

INNODB_CMP_PER_INDEX

Section 14.9.4, “Monitoring InnoDB Table Compression at Runtime”
Section 21.30.6, “The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables”
Section 14.15.1.3, “Using the Compression Information Schema Tables”

INNODB_CMP_PER_INDEX_RESET

Section 21.30.6, “The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables”

INNODB_CMP_RESET

Section 14.15.1, “InnoDB INFORMATION_SCHEMA Tables about Compression”
Section 14.15.1.1, “INNODB_CMP and INNODB_CMP_RESET”
Section 14.15.1.2, “INNODB_CMPMEM and INNODB_CMPMEM_RESET”
Section 21.30.4, “The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables”

INNODB_CMPMEM

Section 14.15.1, “InnoDB INFORMATION_SCHEMA Tables about Compression”
Section 14.15.1.2, “INNODB_CMPMEM and INNODB_CMPMEM_RESET”
Section 21.30.5, “The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables”
Section 14.15.1.3, “Using the Compression Information Schema Tables”

INNODB_CMPMEM_RESET

Section 14.15.1, “InnoDB INFORMATION_SCHEMA Tables about Compression”
Section 14.15.1.2, “INNODB_CMPMEM and INNODB_CMPMEM_RESET”
Section 21.30.5, “The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables”

INNODB_FT_BEING_DELETED

Section 14.6.2.3, “InnoDB FULLTEXT Indexes”
Section 14.15.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 21.30.7, “The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table”

INNODB_FT_CONFIG

Section 14.6.2.3, “InnoDB FULLTEXT Indexes”
Section 14.15.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 21.30.8, “The INFORMATION_SCHEMA INNODB_FT_CONFIG Table”

INNODB_FT_DEFAULT_STOPWORD

Section 12.10.4, “Full-Text Stopwords”
Section 14.6.2.3, “InnoDB FULLTEXT Indexes”
Section 14.15.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”
Section 21.30.9, “The INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD Table”

INNODB_FT_DELETED

Section 14.6.2.3, “InnoDB FULLTEXT Indexes”
Section 14.15.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 21.30.7, “The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table”
Section 21.30.10, “The INFORMATION_SCHEMA INNODB_FT_DELETED Table”

INNODB_FT_INDEX_CACHE

Section 14.6.2.3, “InnoDB FULLTEXT Indexes”
Section 14.15.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 21.30.11, “The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table”

INNODB_FT_INDEX_TABLE

Section 14.6.2.3, “InnoDB FULLTEXT Indexes”
Section 14.15.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 21.30.7, “The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table”
Section 21.30.10, “The INFORMATION_SCHEMA INNODB_FT_DELETED Table”
Section 21.30.12, “The INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE Table”

INNODB_LOCK_WAITS

Section 14.15.2, “InnoDB INFORMATION_SCHEMA Transaction and Locking Information”
Section 14.15.2.2, “InnoDB Lock and Lock-Wait Information”
Section 14.15.2.3, “Persistence and Consistency of InnoDB Transaction and Locking Information”
Section 21.30.14, “The INFORMATION_SCHEMA INNODB_LOCK_WAITS Table”

INNODB_LOCKS

Section 14.15.2, “InnoDB INFORMATION_SCHEMA Transaction and Locking Information”
Section 14.15.2.2, “InnoDB Lock and Lock-Wait Information”
Section 14.15.2.3, “Persistence and Consistency of InnoDB Transaction and Locking Information”

Section 21.30.14, “The INFORMATION_SCHEMA INNODB_LOCK_WAITS Table”
Section 21.30.13, “The INFORMATION_SCHEMA INNODB_LOCKS Table”
Section 21.30.25, “The INFORMATION_SCHEMA INNODB_TRX Table”

INNODB_METRICS

Section 14.5.2, “Change Buffer”
Section 14.8.12, “Configuring the Merge Threshold for Index Pages”
Section 14.15.6, “InnoDB INFORMATION_SCHEMA Metrics Table”
MySQL Glossary
Section 21.30.15, “The INFORMATION_SCHEMA INNODB_METRICS Table”

INNODB_SYS_COLUMNS

Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”
Section 21.30.16, “The INFORMATION_SCHEMA INNODB_SYS_COLUMNS Table”

INNODB_SYS_DATAFILES

Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”
Section 21.30.17, “The INFORMATION_SCHEMA INNODB_SYS_DATAFILES Table”
Section 21.24, “The INFORMATION_SCHEMA TABLESPACES Table”

INNODB_SYS_FIELDS

Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”
Section 21.30.18, “The INFORMATION_SCHEMA INNODB_SYS_FIELDS Table”

INNODB_SYS_FOREIGN

Section 1.7.3.2, “FOREIGN KEY Constraints”
Section 13.1.17.5, “FOREIGN KEY Constraints”
Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”
Section 21.30.19, “The INFORMATION_SCHEMA INNODB_SYS_FOREIGN Table”

INNODB_SYS_FOREIGN_COLS

Section 1.7.3.2, “FOREIGN KEY Constraints”
Section 13.1.17.5, “FOREIGN KEY Constraints”
Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”
Section 21.30.20, “The INFORMATION_SCHEMA INNODB_SYS_FOREIGN_COLS Table”

INNODB_SYS_INDEXES

Section 14.8.12, “Configuring the Merge Threshold for Index Pages”
Section 14.6.2.3, “InnoDB FULLTEXT Indexes”
Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”
Section 21.30.21, “The INFORMATION_SCHEMA INNODB_SYS_INDEXES Table”

INNODB_SYS_TABLES

Section 14.6.2.3, “InnoDB FULLTEXT Indexes”
Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”
Section 21.30.22, “The INFORMATION_SCHEMA INNODB_SYS_TABLES Table”

INNODB_SYS_TABLESPACES

Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”
Section 21.30.23, “The INFORMATION_SCHEMA INNODB_SYS_TABLESPACES Table”
Section 21.24, “The INFORMATION_SCHEMA TABLESPACES Table”

INNODB_SYS_TABLESTATS

Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”
Section 21.30.24, “The INFORMATION_SCHEMA INNODB_SYS_TABLESTATS View”

INNODB_TRX
Section 14.15.2, “InnoDB INFORMATION_SCHEMA Transaction and Locking Information”
Section 14.15.2.2, “InnoDB Lock and Lock-Wait Information”
Section 14.15.2.3, “Persistence and Consistency of InnoDB Transaction and Locking Information”
Section 21.30.13, “The INFORMATION_SCHEMA INNODB_LOCKS Table”
Section 21.30.25, “The INFORMATION_SCHEMA INNODB_TRX Table”
Section 14.15.2.1, “Using InnoDB Transaction and Locking Information”

K

[index top]

KEY_COLUMN_USAGE
Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”
Section 21.11, “The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table”

M

[index top]

MYSQL_FIREWALL_USERS
Section 21.34.1, “The INFORMATION_SCHEMA MYSQL_FIREWALL_USERS Table”

MYSQL_FIREWALL_WHITELIST
Section 21.34.2, “The INFORMATION_SCHEMA MYSQL_FIREWALL_WHITELIST Table”

N

[index top]

NDB_TRANSID_MYSQL_CONNECTION_MAP
Section 18.5.14.22, “The ndbinfo server_operations Table”
Section 18.5.14.23, “The ndbinfo server_transactions Table”

ndb_transid_mysql_connection_map
Section 21.31, “INFORMATION_SCHEMA NDB Cluster Tables”
MySQL Server Options for NDB Cluster
Section 18.5.14, “ndbinfo: The NDB Cluster Information Database”
Section 21.31.2, “The INFORMATION_SCHEMA ndb_transid_mysql_connection_map Table”

O

[index top]

OPTIMIZER_TRACE
Section 21.12, “The INFORMATION_SCHEMA OPTIMIZER_TRACE Table”

P

[index top]

PARAMETERS
Section A.4, “MySQL 5.6 FAQ: Stored Procedures and Functions”
Section 13.7.5.29, “SHOW PROCEDURE STATUS Statement”
Section 21.13, “The INFORMATION_SCHEMA PARAMETERS Table”
Section 21.19, “The INFORMATION_SCHEMA ROUTINES Table”

PARTITIONS

Section 19.3.3, “Exchanging Partitions and Subpartitions with Tables”
Section 19.2.7, “How MySQL Partitioning Handles NULL”
Section 19.3.5, “Obtaining Information About Partitions”
Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”
Chapter 19, *Partitioning*
Section 13.7.5.37, “SHOW TABLE STATUS Statement”
Section 21.14, “The INFORMATION_SCHEMA PARTITIONS Table”
Section 21.23, “The INFORMATION_SCHEMA TABLES Table”

PLUGINS

Section 13.7.3.3, “INSTALL PLUGIN Statement”
Section 5.5.2, “Obtaining Server Plugin Information”
Section 21.15, “The INFORMATION_SCHEMA PLUGINS Table”
Section 1.3, “What Is New in MySQL 5.6”

PROCESSLIST

Section 8.14.1, “Accessing the Process List”
Section 13.7.6.4, “KILL Statement”
Section 14.15.2.3, “Persistence and Consistency of InnoDB Transaction and Locking Information”
Section 13.7.5.30, “SHOW PROCESSLIST Statement”
Section 21.30.25, “The INFORMATION_SCHEMA INNODB_TRX Table”
Section 21.16, “The INFORMATION_SCHEMA PROCESSLIST Table”
Section 22.12.10.3, “The threads Table”
Section 14.15.2.1, “Using InnoDB Transaction and Locking Information”

PROFILING

Section 13.7.5.31, “SHOW PROFILE Statement”
Section 21.17, “The INFORMATION_SCHEMA PROFILING Table”

R

[index top]

REFERENTIAL_CONSTRAINTS

Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”
Section 21.18, “The INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS Table”

ROUTINES

Section 21.1, “Introduction”
Section A.4, “MySQL 5.6 FAQ: Stored Procedures and Functions”
Section 13.7.5.29, “SHOW PROCEDURE STATUS Statement”
Section 20.6, “Stored Object Access Control”
Section 20.2.3, “Stored Routine Metadata”
Section 21.19, “The INFORMATION_SCHEMA ROUTINES Table”

S

[index top]

SCHEMA_PRIVILEGES

Section 21.21, “The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table”

SCHEMATA

Section 6.2.3, “Grant Tables”

Section 6.2.2, “Privileges Provided by MySQL”
Section 5.1.7, “Server System Variables”
Section 13.7.5.15, “SHOW DATABASES Statement”
Section 21.20, “The INFORMATION_SCHEMA SCHEMATA Table”

SESSION_STATUS

Section 18.5, “Management of NDB Cluster”
Section 18.5.13, “NDB API Statistics Counters and Variables”
Section 18.6, “NDB Cluster Replication”
Section 13.7.5.36, “SHOW STATUS Statement”
Section 21.9, “The INFORMATION_SCHEMA GLOBAL_STATUS and SESSION_STATUS Tables”

SESSION_VARIABLES

Section 13.7.5.40, “SHOW VARIABLES Statement”
Section 21.10, “The INFORMATION_SCHEMA GLOBAL_VARIABLES and SESSION_VARIABLES Tables”

STATISTICS

Section 13.7.2.1, “ANALYZE TABLE Statement”
Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”
Section 13.7.5.23, “SHOW INDEX Statement”
Section 21.22, “The INFORMATION_SCHEMA STATISTICS Table”

T

[index top]

TABLE_CONSTRAINTS

Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”
Section 21.18, “The INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS Table”
Section 21.25, “The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table”

TABLE_PRIVILEGES

Section 21.26, “The INFORMATION_SCHEMA TABLE_PRIVILEGES Table”

TABLES

Section 21.1, “Introduction”
Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”
Section 13.7.5.37, “SHOW TABLE STATUS Statement”
Section 13.7.5.38, “SHOW TABLES Statement”
Section 21.23, “The INFORMATION_SCHEMA TABLES Table”

TP_THREAD_GROUP_STATE

Section 21.32, “INFORMATION_SCHEMA Thread Pool Tables”
Section 21.32.1, “The INFORMATION_SCHEMA TP_THREAD_GROUP_STATE Table”
Section 5.5.3.1, “Thread Pool Elements”

TP_THREAD_GROUP_STATS

Section 21.32, “INFORMATION_SCHEMA Thread Pool Tables”
Section 21.32.2, “The INFORMATION_SCHEMA TP_THREAD_GROUP_STATS Table”
Section 21.32.3, “The INFORMATION_SCHEMA TP_THREAD_STATE Table”
Section 5.5.3.1, “Thread Pool Elements”

TP_THREAD_STATE

Section 21.32, “INFORMATION_SCHEMA Thread Pool Tables”

Section 21.32.1, “The INFORMATION_SCHEMA TP_THREAD_GROUP_STATE Table”
Section 21.32.3, “The INFORMATION_SCHEMA TP_THREAD_STATE Table”
Section 5.5.3.1, “Thread Pool Elements”
Section 5.5.3.2, “Thread Pool Installation”

TRIGGERS

Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”
Section 13.7.5.13, “SHOW CREATE TRIGGER Statement”
Section 13.7.5.39, “SHOW TRIGGERS Statement”
Section 21.27, “The INFORMATION_SCHEMA TRIGGERS Table”
Section 20.3.2, “Trigger Metadata”

U

[index top]

USER_PRIVILEGES

Section 21.28, “The INFORMATION_SCHEMA USER_PRIVILEGES Table”

V

[index top]

VIEWS

Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”
Section 13.7.5.14, “SHOW CREATE VIEW Statement”
Section 21.29, “The INFORMATION_SCHEMA VIEWS Table”
Section 20.5.5, “View Metadata”

Join Types Index

A | C | E | F | I | R | S | U

A

[index top]

ALL

Section 8.2.1.19, “Avoiding Full Table Scans”
Section 8.2.1.11, “Block Nested-Loop and Batched Key Access Joins”
Section 8.8.2, “EXPLAIN Output Format”
Section 8.2.1.6, “Nested-Loop Join Algorithms”

C

[index top]

const

Section 8.8.2, “EXPLAIN Output Format”
Section 8.8.3, “Extended EXPLAIN Output Format”
NDB Cluster System Variables
Section 8.2.1.13, “ORDER BY Optimization”
Section 8.2.1.2, “Range Optimization”
Section 13.2.9, “SELECT Statement”

E

[index top]

eq_ref

Section 8.2.1.11, “Block Nested-Loop and Batched Key Access Joins”
Section 8.8.2, “EXPLAIN Output Format”
Section 8.2.1.5, “Index Condition Pushdown Optimization”
Section 15.7.1, “MERGE Table Advantages and Disadvantages”
NDB Cluster System Variables
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”

F

[index top]

fulltext

Section 8.8.2, “EXPLAIN Output Format”

I

[index top]

index

Section 8.2.1.11, “Block Nested-Loop and Batched Key Access Joins”
Section 8.8.2, “EXPLAIN Output Format”
Section 8.2.1.6, “Nested-Loop Join Algorithms”

index_merge

Section 8.8.2, “EXPLAIN Output Format”
Section 8.2.1.3, “Index Merge Optimization”

index_subquery

Section 8.8.2, “EXPLAIN Output Format”
Section 13.2.10.10, “Optimizing Subqueries”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”

R

[index top]

range

Section 8.2.1.11, “Block Nested-Loop and Batched Key Access Joins”
Section 8.8.2, “EXPLAIN Output Format”
Section 8.2.1.14, “GROUP BY Optimization”
Section 8.2.1.5, “Index Condition Pushdown Optimization”
Section 8.2.1.3, “Index Merge Optimization”
Section 8.2.1.6, “Nested-Loop Join Algorithms”
Section 8.2.1.2, “Range Optimization”

ref

Section 8.2.1.11, “Block Nested-Loop and Batched Key Access Joins”
Section 8.8.2, “EXPLAIN Output Format”
Section 8.8.3, “Extended EXPLAIN Output Format”

Section 8.2.1.5, “Index Condition Pushdown Optimization”
Section 8.3.7, “InnoDB and MyISAM Index Statistics Collection”
Section 15.7.1, “MERGE Table Advantages and Disadvantages”
NDB Cluster System Variables
Section 8.2.2.4, “Optimizing Derived Tables”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”

ref_or_null

Section 8.8.2, “EXPLAIN Output Format”
Section 8.2.1.5, “Index Condition Pushdown Optimization”
Section 8.2.1.12, “IS NULL Optimization”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”

S

[index top]

system

Section 8.8.2, “EXPLAIN Output Format”
Section 8.8.3, “Extended EXPLAIN Output Format”
Section 8.2.1.2, “Range Optimization”
Section 13.2.9, “SELECT Statement”

U

[index top]

unique_subquery

Section 8.8.2, “EXPLAIN Output Format”
Section 13.2.10.10, “Optimizing Subqueries”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”

Operator Index

Symbols | A | B | C | D | E | I | L | N | O | R | X

Symbols

[index top]

-

Section 12.6.1, “Arithmetic Operators”
Section 12.11, “Cast Functions and Operators”
Section 9.5, “Expressions”
Section 11.1.1, “Numeric Data Type Syntax”
Section 19.6, “Restrictions and Limitations on Partitioning”

!

Section 9.5, “Expressions”
Section 12.4.3, “Logical Operators”
Section 12.4.1, “Operator Precedence”

!=

Section 12.4.2, “Comparison Functions and Operators”
Section 12.4.1, “Operator Precedence”

Section 8.2.1.2, “Range Optimization”

%

Section 12.6.1, “Arithmetic Operators”

&

Section 12.13, “Bit Functions and Operators”
Section 13.1.17, “CREATE TABLE Statement”
Section 19.6, “Restrictions and Limitations on Partitioning”

&&

Section 12.4.3, “Logical Operators”
Section 1.7.1, “MySQL Extensions to Standard SQL”

>

Section 12.4.2, “Comparison Functions and Operators”
Section 8.3.8, “Comparison of B-Tree and Hash Indexes”
Section 8.8.2, “EXPLAIN Output Format”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 12.4.1, “Operator Precedence”
Section 8.2.1.2, “Range Optimization”

>>

Section 12.13, “Bit Functions and Operators”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 19.6, “Restrictions and Limitations on Partitioning”

>=

Section 12.4.2, “Comparison Functions and Operators”
Section 8.3.8, “Comparison of B-Tree and Hash Indexes”
Section 8.8.2, “EXPLAIN Output Format”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 12.4.1, “Operator Precedence”
Section 8.2.1.2, “Range Optimization”

<

Section 12.4.2, “Comparison Functions and Operators”
Section 8.3.8, “Comparison of B-Tree and Hash Indexes”
Section 8.8.2, “EXPLAIN Output Format”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 12.4.1, “Operator Precedence”
Section 8.2.1.2, “Range Optimization”
Section 3.3.4.6, “Working with NULL Values”

<>

Section 12.4.2, “Comparison Functions and Operators”
Section 8.8.2, “EXPLAIN Output Format”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 12.4.1, “Operator Precedence”
Section 8.2.1.2, “Range Optimization”
Section 3.3.4.6, “Working with NULL Values”

<<

Section 12.13, “Bit Functions and Operators”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 19.6, “Restrictions and Limitations on Partitioning”

<=
Section 12.4.2, “Comparison Functions and Operators”
Section 8.3.8, “Comparison of B-Tree and Hash Indexes”
Section 8.8.2, “EXPLAIN Output Format”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 12.4.1, “Operator Precedence”
Section 8.2.1.2, “Range Optimization”

<=>
Section 12.4.2, “Comparison Functions and Operators”
Section 8.8.2, “EXPLAIN Output Format”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 12.4.1, “Operator Precedence”
Section 8.2.1.2, “Range Optimization”
Section 12.3, “Type Conversion in Expression Evaluation”

*****
Section 12.6.1, “Arithmetic Operators”
Section 11.1.1, “Numeric Data Type Syntax”
Section 19.6, “Restrictions and Limitations on Partitioning”

+
Section 12.6.1, “Arithmetic Operators”
Section 12.11, “Cast Functions and Operators”
Section 9.5, “Expressions”
Section 11.1.1, “Numeric Data Type Syntax”
Section 19.6, “Restrictions and Limitations on Partitioning”

/
Section 12.6.1, “Arithmetic Operators”
Section 19.6, “Restrictions and Limitations on Partitioning”
Section 5.1.7, “Server System Variables”

:=
Section 12.4.4, “Assignment Operators”
Section 12.4.1, “Operator Precedence”
Section 13.7.4.1, “SET Syntax for Variable Assignment”
Section 9.4, “User-Defined Variables”

=
Section 12.4.4, “Assignment Operators”
Section 12.4.2, “Comparison Functions and Operators”
Section 8.3.8, “Comparison of B-Tree and Hash Indexes”
Section 8.8.2, “EXPLAIN Output Format”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 12.4.1, “Operator Precedence”
Section 8.2.1.2, “Range Optimization”
Section 13.2.10.12, “Restrictions on Subqueries”
Section 13.7.4.1, “SET Syntax for Variable Assignment”
Section 12.8.1, “String Comparison Functions and Operators”
Section 9.4, “User-Defined Variables”
Section 3.3.4.6, “Working with NULL Values”

^
Section 12.13, “Bit Functions and Operators”

Section 9.5, “Expressions”
Section 12.4.1, “Operator Precedence”
Section 19.6, “Restrictions and Limitations on Partitioning”

|

Section 12.13, “Bit Functions and Operators”
Section 19.6, “Restrictions and Limitations on Partitioning”

||

Section 12.8.3, “Character Set and Collation of Function Results”
Section 10.8.2, “COLLATE Clause Precedence”
Section 9.5, “Expressions”
Section 12.4.3, “Logical Operators”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 12.4.1, “Operator Precedence”
Section 5.1.10, “Server SQL Modes”

~

Section 12.13, “Bit Functions and Operators”
Section 19.6, “Restrictions and Limitations on Partitioning”

A

[index top]

AND

Section 8.3.8, “Comparison of B-Tree and Hash Indexes”
Section 13.1.17, “CREATE TABLE Statement”
Section 8.2.1.3, “Index Merge Optimization”
Section 12.4.3, “Logical Operators”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 8.2.2.1, “Optimizing Subqueries with Semijoin Transformations”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”
Section 8.2.1.2, “Range Optimization”
Section 13.2.10.12, “Restrictions on Subqueries”
Section 8.2.1.18, “Row Constructor Expression Optimization”
Section 3.6.7, “Searching on Two Keys”
Section 3.3.4.2, “Selecting Particular Rows”
Section 12.8.1, “String Comparison Functions and Operators”
Section 20.5.2, “View Processing Algorithms”

B

[index top]

BETWEEN

Section 12.4.2, “Comparison Functions and Operators”
Section 8.3.8, “Comparison of B-Tree and Hash Indexes”
Section 8.8.2, “EXPLAIN Output Format”
Section 8.2.1.2, “Range Optimization”
Section 12.3, “Type Conversion in Expression Evaluation”

BINARY

Section 12.11, “Cast Functions and Operators”
Section 8.4.2.2, “Optimizing for Character and String Types”
Section 3.3.4.7, “Pattern Matching”

Section 3.3.4.4, “Sorting Rows”

C

[index top]

CASE

Section 13.6.5.1, “CASE Statement”
Section 9.5, “Expressions”
Section 12.5, “Flow Control Functions”
Section 1.7.1, “MySQL Extensions to Standard SQL”

CASE value WHEN compare_value THEN result END

Section 12.5, “Flow Control Functions”

CASE WHEN condition THEN result END

Section 12.5, “Flow Control Functions”

CASE WHEN expr1 = expr2 THEN NULL ELSE expr1 END

Section 12.5, “Flow Control Functions”

D

[index top]

DIV

Section 12.6.1, “Arithmetic Operators”
Section 19.6, “Restrictions and Limitations on Partitioning”

E

[index top]

expr BETWEEN min AND max

Section 12.4.2, “Comparison Functions and Operators”

expr IN ()

Section 12.4.2, “Comparison Functions and Operators”

expr LIKE pat

Section 12.8.1, “String Comparison Functions and Operators”

expr NOT BETWEEN min AND max

Section 12.4.2, “Comparison Functions and Operators”

expr NOT IN ()

Section 12.4.2, “Comparison Functions and Operators”

expr NOT LIKE pat

Section 12.8.1, “String Comparison Functions and Operators”

expr NOT REGEXP pat

Section 12.8.2, “Regular Expressions”

expr NOT RLIKE pat
Section 12.8.2, “Regular Expressions”

expr REGEXP pat
Section 12.8.2, “Regular Expressions”

expr RLIKE pat
Section 12.8.2, “Regular Expressions”

expr1 SOUNDS LIKE expr2
Section 12.8, “String Functions and Operators”

I

[index top]

IN()
Section 8.8.2, “EXPLAIN Output Format”
Section 12.4.1, “Operator Precedence”
Section 8.2.1.2, “Range Optimization”
Section 8.2.1.18, “Row Constructor Expression Optimization”
Section 12.3, “Type Conversion in Expression Evaluation”

IS
Section 12.4.1, “Operator Precedence”

IS boolean_value
Section 12.4.2, “Comparison Functions and Operators”

IS NOT boolean_value
Section 12.4.2, “Comparison Functions and Operators”

IS NOT NULL
Section 12.4.2, “Comparison Functions and Operators”
Section B.3.4.3, “Problems with NULL Values”
Section 8.2.1.2, “Range Optimization”
Section 3.3.4.6, “Working with NULL Values”

IS NULL
Section 12.4.2, “Comparison Functions and Operators”
Section 8.8.2, “EXPLAIN Output Format”
Section 8.2.1.12, “IS NULL Optimization”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”
Section B.3.4.3, “Problems with NULL Values”
Section 8.2.1.2, “Range Optimization”
Section 5.1.7, “Server System Variables”
Section 3.3.4.6, “Working with NULL Values”

L

[index top]

LIKE
Section 6.2.6, “Access Control, Stage 2: Request Verification”

Section 12.11, “Cast Functions and Operators”
Section 10.2, “Character Sets and Collations in MySQL”
Section 8.3.8, “Comparison of B-Tree and Hash Indexes”
Section 8.8.2, “EXPLAIN Output Format”
Section 21.35, “Extensions to SHOW Statements”
Section 13.8.3, “HELP Statement”
Section A.11, “MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 4.5.1.4, “mysql Client Server-Side Help”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 18.6.11, “NDB Cluster Replication Conflict Resolution”
Section 12.4.1, “Operator Precedence”
Section 3.3.4.7, “Pattern Matching”
Section 22.4.4, “Pre-Filtering by Instrument”
Section 18.5.15, “Quick Reference: NDB Cluster SQL Statements”
Section 8.2.1.2, “Range Optimization”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 13.7.5.4, “SHOW CHARACTER SET Statement”
Section 13.7.5.5, “SHOW COLLATION Statement”
Section 13.7.5.6, “SHOW COLUMNS Statement”
Section 13.7.5.15, “SHOW DATABASES Statement”
Section 13.7.5.19, “SHOW EVENTS Statement”
Section 13.7.5.25, “SHOW OPEN TABLES Statement”
Section 13.7.5.29, “SHOW PROCEDURE STATUS Statement”
Section 13.7.5.36, “SHOW STATUS Statement”
Section 13.7.5.37, “SHOW TABLE STATUS Statement”
Section 13.7.5.38, “SHOW TABLES Statement”
Section 13.7.5.39, “SHOW TRIGGERS Statement”
Section 13.7.5.40, “SHOW VARIABLES Statement”
Section 6.2.4, “Specifying Account Names”
Section 12.8.1, “String Comparison Functions and Operators”
Section 9.1.1, “String Literals”
Section 5.1.8.3, “Structured System Variables”
Section 11.3.2, “The CHAR and VARCHAR Types”
Section 11.3.6, “The SET Type”
Section 5.1.8, “Using System Variables”

LIKE '_A%'

Section A.11, “MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”

LIKE 'pattern'

Section 8.2.1.2, “Range Optimization”
Section 13.7.5, “SHOW Statements”

LIKE ... ESCAPE

Section B.3.7, “Known Issues in MySQL”

N

[index top]

N % M

Section 12.6.1, “Arithmetic Operators”
Section 12.6.2, “Mathematical Functions”

N MOD M

Section 12.6.1, “Arithmetic Operators”
Section 12.6.2, “Mathematical Functions”

NOT
Section 12.4.3, “Logical Operators”
Section 5.1.10, “Server SQL Modes”

NOT LIKE
Section 3.3.4.7, “Pattern Matching”
Section 12.8.1, “String Comparison Functions and Operators”

NOT REGEXP
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 3.3.4.7, “Pattern Matching”
Section 12.8.1, “String Comparison Functions and Operators”

NOT RLIKE
Section 3.3.4.7, “Pattern Matching”
Section 12.8.1, “String Comparison Functions and Operators”

O

[index top]

OR
Section 9.5, “Expressions”
Section 13.7.1.4, “GRANT Statement”
Section 8.2.1.3, “Index Merge Optimization”
Section 12.4.3, “Logical Operators”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 12.4.1, “Operator Precedence”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”
Section 8.2.1.2, “Range Optimization”
Section 8.2.1.18, “Row Constructor Expression Optimization”
Section 3.6.7, “Searching on Two Keys”
Section 3.3.4.2, “Selecting Particular Rows”
Section 5.1.10, “Server SQL Modes”
Section 12.8.1, “String Comparison Functions and Operators”

R

[index top]

REGEXP
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 12.4.1, “Operator Precedence”
Section 3.3.4.7, “Pattern Matching”
Section 12.8.2, “Regular Expressions”
Section 10.11, “Restrictions on Character Sets”

RLIKE
Section 3.3.4.7, “Pattern Matching”
Section 12.8.2, “Regular Expressions”
Section 10.11, “Restrictions on Character Sets”

X

[index top]

XOR

Section 12.19.1, “Aggregate Function Descriptions”
Section 12.4.3, “Logical Operators”

Option Index

Symbols | A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W | X | Y | Z

Symbols

[index top]

--

Section 1.7.2.4, “'--' as the Start of a Comment”
Section 4.8.2, “`replace` — A String-Replacement Utility”

-#

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”
Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”
Section 4.8.2, “`replace` — A String-Replacement Utility”
Section 5.1.6, “Server Command Options”
Section 5.8.3, “The DBUG Package”

-do-

Section 17.2.3, “How Servers Evaluate Replication Filtering Rules”

-ignore-

Section 17.2.3, “How Servers Evaluate Replication Filtering Rules”

-1

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

-?

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”
Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”
Section 4.6.2, “`myisam_ftdump` — Display Full-Text Index information”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.4, “`myisamlog` — Display MyISAM Log File Contents”
Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”
Section 4.6.15, “`mysql_waitpid` — Kill Process and Wait for Its Termination”
Section 4.6.16, “`mysql_zap` — Kill Processes That Match a Pattern”

Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.10, “mysqlhotcopy — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 18.4.17, “ndb_print_file — Print NDB Disk Data File Contents”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 4.8.1, “perror — Display MySQL Error Message Information”
Section 4.8.2, “replace — A String-Replacement Utility”
Section 4.8.3, “resolveip — Resolve Host name to IP Address or Vice Versa”
Section 5.1.6, “Server Command Options”
Section 1.2.2, “The Main Features of MySQL”
Section 4.2.2.1, “Using Options on the Command Line”

A

[index top]

-A

Section 4.5.1.1, “mysql Client Options”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 18.4.15, “ndb_move_data — NDB Data Copy Utility”
Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Section 4.6.3.4, “Other myisamchk Options”

-a

Section 18.5.1, “Commands in the NDB Cluster Management Client”
Section 16.2.2.1, “memcached Command-Line Options”
Section 7.6.4, “MyISAM Table Optimization”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 18.4.25, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”
Section 4.6.3.4, “Other myisamchk Options”

--abort-on-error

Section 18.4.15, “ndb_move_data — NDB Data Copy Utility”

--abort-slave-event-count

Section 17.1.4.3, “Replica Server Options and Variables”

--add-drop-database

Section 7.4.1, “Dumping Data in SQL Format with mysqldump”
Section 4.5.4, “mysqldump — A Database Backup Program”

--add-drop-table

Section 4.5.4, “mysqldump — A Database Backup Program”

--add-drop-trigger
Section 4.5.4, “mysqldump — A Database Backup Program”

--add-locks
Section 4.5.4, “mysqldump — A Database Backup Program”

--addtodest
Section 4.6.10, “mysqlhotcopy — A Database Backup Program”

--all
Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”

--all-databases
Section 7.4.1, “Dumping Data in SQL Format with mysqldump”
Section 9.2.4, “Mapping of Identifiers to File Names”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 18.5.14, “ndbinfo: The NDB Cluster Information Database”
Section 2.11.10, “Rebuilding or Repairing Tables or Indexes”
Section 7.4.2, “Reloading SQL-Format Backups”
Section 2.11.4, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

--all-in-1
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--all-tablespaces
Section 4.5.4, “mysqldump — A Database Backup Program”

--allow-keywords
Section 4.5.4, “mysqldump — A Database Backup Program”

--allow-suspicious-udfs
Section 5.1.6, “Server Command Options”

--allowold
Section 4.6.10, “mysqlhotcopy — A Database Backup Program”

--analyze
Section 7.6.4, “MyISAM Table Optimization”
Section 4.6.3.1, “myisamchk General Options”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.6.3.4, “Other myisamchk Options”

--ansi
Section 1.7, “MySQL Standards Compliance”
Section 5.1.6, “Server Command Options”

antonio
Section 6.4.1.6, “PAM Pluggable Authentication”

--append
Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”

--apply-slave-statements
Section 4.5.4, “mysqldump — A Database Backup Program”

--audit-log
Section 6.4.4.6, “Audit Log Reference”
Section 6.4.4.1, “Installing MySQL Enterprise Audit”

--auto-generate-sql
Section 4.5.7, “mysqlslap — A Load Emulation Client”

--auto-generate-sql-add-autoincrement
Section 4.5.7, “mysqlslap — A Load Emulation Client”

--auto-generate-sql-execute-number
Section 4.5.7, “mysqlslap — A Load Emulation Client”

--auto-generate-sql-guid-primary
Section 4.5.7, “mysqlslap — A Load Emulation Client”

--auto-generate-sql-load-type
Section 4.5.7, “mysqlslap — A Load Emulation Client”

--auto-generate-sql-secondary-indexes
Section 4.5.7, “mysqlslap — A Load Emulation Client”

--auto-generate-sql-unique-query-number
Section 4.5.7, “mysqlslap — A Load Emulation Client”

--auto-generate-sql-unique-write-number
Section 4.5.7, “mysqlslap — A Load Emulation Client”

--auto-generate-sql-write-number
Section 4.5.7, “mysqlslap — A Load Emulation Client”

--auto-rehash
Section 14.8.11.2, “Configuring Non-Persistent Optimizer Statistics Parameters”
Section 4.5.1.2, “mysql Client Commands”
Section 4.5.1.1, “mysql Client Options”

auto-rehash
Section 14.8.11.2, “Configuring Non-Persistent Optimizer Statistics Parameters”

--auto-repair
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--auto-vertical-output
Section 4.5.1.1, “mysql Client Options”

--autocommit
Section 5.1.7, “Server System Variables”

B

[index top]

-B

Section 16.2.2.1, “memcached Command-Line Options”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”

-b

Section 16.2.2.1, “memcached Command-Line Options”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.4, “mysql_plugin — Configure MySQL Server Plugins”
Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 18.4.10, “ndb_desc — Describe NDB Tables”
Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Section 4.6.3.4, “Other myisamchk Options”
Section 5.1.6, “Server Command Options”

--back_log

Section 2.7, “Installing MySQL on Solaris”

--backup

Section 4.6.3.3, “myisamchk Repair Options”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”

--backup-path

Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Restoring to Fewer Nodes Than the Original

backup-path

Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”

--backupid

Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Restoring to Fewer Nodes Than the Original

--base64-output

Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.1.3.1, “GTID Concepts”
Section 4.6.8.2, “mysqlbinlog Row Event Display”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 17.1.2.2, “Usage of Row-Based Logging and Replication”

--basedir

Section 2.10.1, “Initializing the Data Directory”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.4.3, “mysql_install_db — Initialize MySQL Data Directory”
Section 4.4.4, “mysql_plugin — Configure MySQL Server Plugins”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”
Section 18.1.7.1, “Noncompliance with SQL Syntax in NDB Cluster”
Section 5.7, “Running Multiple MySQL Instances on One Machine”
Section 5.1.6, “Server Command Options”

Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”

basedir

Section 2.3.4.2, “Creating an Option File”
Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”
Section 2.3.5, “Troubleshooting a Microsoft Windows MySQL Server Installation”

--batch

Section 4.5.1.3, “mysql Client Logging”
Section 4.5.1.1, “mysql Client Options”

--binary-as-hex

Section 4.5.1.1, “mysql Client Options”

--binary-mode

Section 4.5.1.2, “mysql Client Commands”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.6, “mysql Client Tips”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”

--bind-address

Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 18.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”
Section 18.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”

--binlog-checksum

Section 17.1.4.4, “Binary Log Options and Variables”

--binlog-do-db

Section 17.1.4.4, “Binary Log Options and Variables”
Section 17.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 17.2.3, “How Servers Evaluate Replication Filtering Rules”
Section 18.6.3, “Known Issues in NDB Cluster Replication”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 5.4.4, “The Binary Log”

--binlog-format

Section 5.4.4.1, “Binary Logging Formats”
Section 18.6.2, “General Requirements for NDB Cluster Replication”
Section A.4, “MySQL 5.6 FAQ: Stored Procedures and Functions”
Section 5.4.4.2, “Setting The Binary Log Format”
Section 18.6.6, “Starting NDB Cluster Replication (Single Replication Channel)”
Section 18.1.7.6, “Unsupported or Missing Features in NDB Cluster”

--binlog-ignore-db

Section 17.1.4.4, “Binary Log Options and Variables”
Section 17.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 17.2.3, “How Servers Evaluate Replication Filtering Rules”

Section 18.6.3, “Known Issues in NDB Cluster Replication”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 5.4.4, “The Binary Log”

--binlog-row-event-max-size

Section 17.1.4.4, “Binary Log Options and Variables”
Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 5.4.4.2, “Setting The Binary Log Format”

--blob-info

Section 18.4.10, “`ndb_desc` — Describe NDB Tables”

--block-search

Section 4.6.3.4, “Other myisamchk Options”

--bootstrap

Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”

--brief

Section 4.6.7, “`mysqlaccess` — Client for Checking Access Privileges”

--browser-start-page

Section 18.4.25, “`ndb_setup.py` — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”

--builddir

Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”

C

[index top]

-C

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”
Section 16.2.2.1, “`memcached` Command-Line Options”
Section 4.6.3.2, “myisamchk Check Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”

-c

Section 4.6.1, “`innochecksum` — Offline InnoDB File Checksum Utility”
Section 18.2.2.1, “Installing NDB Cluster on Windows from a Binary Release”
Section 16.2.2.1, “`memcached` Command-Line Options”
Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”
Section 4.6.2, “`myisam_ftdump` — Display Full-Text Index information”
Section 4.6.3.2, “myisamchk Check Options”
Section 4.6.4, “`myisamlog` — Display MyISAM Log File Contents”

Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”
Section 18.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”
Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 18.4.25, “`ndb_setup.py` — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Restoring to Fewer Nodes Than the Original

--ca-certs-file

Section 18.2.8.1, “NDB Cluster Auto-Installer Requirements”
Section 18.4.25, “`ndb_setup.py` — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”

--cert-file

Section 18.2.8.1, “NDB Cluster Auto-Installer Requirements”
Section 18.4.25, “`ndb_setup.py` — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”

--cflags

Section 2.9.8, “Dealing with Problems Compiling MySQL”
Section 4.7.2, “`mysql_config` — Display Options for Compiling Clients”

--character-set-client-handshake

Section A.11, “MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 5.1.6, “Server Command Options”
Section 10.10.7.1, “The cp932 Character Set”

--character-set-server

Section 10.15, “Character Set Configuration”
Section 10.5, “Configuring Application Character Set and Collation”
Section A.11, “MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 10.3.2, “Server Character Set and Collation”

--character-sets-dir

Section B.3.2.16, “Can't initialize character set”
Section 10.15, “Character Set Configuration”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 18.4.15, “`ndb_move_data` — NDB Data Copy Utility”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

--character_set_server
Section 2.9.7, “MySQL Source-Configuration Options”

--charset
Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”

--check
Section 4.6.3.2, “myisamchk Check Options”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

--check-only-changed
Section 4.6.3.2, “myisamchk Check Options”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

--check-orphans
Section 18.4.6, “`ndb_blob_tool` — Check and Repair BLOB and TEXT columns of NDB Cluster Tables”

--check-upgrade
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

--checkpoint
Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”

--chroot
Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”
Section 5.1.6, “Server Command Options”

CMAKE_BUILD_TYPE
Section 2.9.7, “MySQL Source-Configuration Options”

CMAKE_C_FLAGS
Section 5.8.1.1, “Compiling MySQL for Debugging”
Section 2.9.8, “Dealing with Problems Compiling MySQL”
Section 2.9.7, “MySQL Source-Configuration Options”

CMAKE_C_FLAGS_build_type
Section 2.9.7, “MySQL Source-Configuration Options”

CMAKE_C_FLAGS_RELWITHDEBINFO
Section 2.9.7, “MySQL Source-Configuration Options”

CMAKE_CXX_FLAGS
Section 5.8.1.1, “Compiling MySQL for Debugging”
Section 2.9.8, “Dealing with Problems Compiling MySQL”
Section 2.9.7, “MySQL Source-Configuration Options”

CMAKE_CXX_FLAGS_build_type
Section 2.9.7, “MySQL Source-Configuration Options”

CMAKE_CXX_FLAGS_RELWITHDEBINFO
Section 2.9.7, “MySQL Source-Configuration Options”

CMAKE_INSTALL_PREFIX

Section 2.9.5, “Installing MySQL Using a Development Source Tree”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 5.7.3, “Running Multiple MySQL Instances on Unix”
Section 5.1.7, “Server System Variables”

CMAKE_PREFIX_PATH

Section 2.9.7, “MySQL Source-Configuration Options”

--collation-server

Section 10.15, “Character Set Configuration”
Section 10.5, “Configuring Application Character Set and Collation”
Section 10.3.2, “Server Character Set and Collation”

--collation_server

Section 2.9.7, “MySQL Source-Configuration Options”

--column-names

Section 4.5.1.1, “mysql Client Options”
Section 4.2.2.4, “Program Option Modifiers”

--column-type-info

Section 8.2.1.16, “LIMIT Query Optimization”
Section 4.5.1.1, “mysql Client Options”

--columns

Section 4.5.5, “`mysqlimport` — A Data Import Program”

--comments

Section 4.5.1.1, “mysql Client Options”
Section 4.5.4, “`mysqldump` — A Database Backup Program”

--commit

Section 4.6.7, “`mysqlaccess` — Client for Checking Access Privileges”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”

--comp

Section 4.2.2, “Specifying Program Options”

--compact

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--compatible

Section 4.5.4, “`mysqldump` — A Database Backup Program”

COMPILATION_COMMENT

Section 5.1.7, “Server System Variables”

--complete-insert

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--compr

Section 4.2.2, “Specifying Program Options”

--compress

Section 4.2.6, “Connection Compression Control”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”
Section 4.2.2, “Specifying Program Options”

--concurrency

Section 4.5.7, “`mysqlslap` — A Load Emulation Client”

--config-cache

Section 18.3.3, “NDB Cluster Configuration Files”
Section 18.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”

--config-dir

Section 18.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”

--config-file

Section 18.2.4, “Initial Startup of NDB Cluster”
Section 18.2.2.3, “Initial Startup of NDB Cluster on Windows”
Section 18.2.2.4, “Installing NDB Cluster Processes as Windows Services”
Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
Section 18.3.3.1, “NDB Cluster Configuration: Basic Example”
Section 18.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”
Section 18.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”

--config-from-node

Section 18.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”

--config_from_node

Section 18.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”

--configdir

Section 18.3.3, “NDB Cluster Configuration Files”
Section 18.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”

--configinfo

Section 18.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”

--connect

Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”

--connect-delay

Section 18.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”

--connect-expired-password

Section 4.5.1.1, “mysql Client Options”
Section 6.2.10, “Server Handling of Expired Passwords”

--connect-retries

Section 18.4.5, “ndb_mgm — The NDB Cluster Management Client”
Section 18.4.1, “ndbd — The NDB Cluster Data Node Daemon”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

--connect-retry-delay

Section 18.4.5, “ndb_mgm — The NDB Cluster Management Client”
Section 18.4.1, “ndbd — The NDB Cluster Data Node Daemon”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

--connect-string

Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

--connect-timeout

Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”

--connection-control

Section 6.4.2.1, “Connection-Control Plugin Installation”

--connection-control-failed-login-attempts

Section 6.4.2.1, “Connection-Control Plugin Installation”

--connection-server-id

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”

--connection-timeout

Section 18.4.13, “ndb_error_reporter — NDB Error-Reporting Utility”

--connections

Section 18.4.7, “ndb_config — Extract NDB Cluster Configuration Information”

--console

Section 14.17.2, “Enabling InnoDB Monitors”
Section 5.4.2.1, “Error Logging on Windows”
Section 18.2.2.3, “Initial Startup of NDB Cluster on Windows”
Section 14.21, “InnoDB Troubleshooting”
Resetting the Root Password: Windows Systems
Section 5.1.6, “Server Command Options”
Section 2.3.4.5, “Starting MySQL from the Windows Command Line”
Section 2.3.4.4, “Starting the Server for the First Time”

--copy

Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”

--core-file

Section 5.8.1.4, “Debugging mysqld under gdb”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”

core-file
Section 5.8.1.3, “Using WER with PDB to create a Windows crashdump”

--core-file-size
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 5.1.6, “Server Command Options”

--correct-checksum
Section 4.6.3.3, “myisamchk Repair Options”

--count
Section 4.6.2, “`myisam_ftdump` — Display Full-Text Index information”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”

--create
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”

--create-old-temporals
NDB Cluster System Variables

--create-options
Section 4.5.4, “`mysqldump` — A Database Backup Program”

--create-schema
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”

--cross-bootstrap
Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”

--csv
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”

--cxxflags
Section 2.9.8, “Dealing with Problems Compiling MySQL”
Section 4.7.2, “`mysql_config` — Display Options for Compiling Clients”

D

[index top]

-D
Section 10.13, “Adding a Character Set”
Section 18.2.1.4, “Building NDB Cluster from Source on Linux”
Section B.3.2.16, “Can't initialize character set”
Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”
Section 18.2.2.2, “Compiling and Installing NDB Cluster from Source on Windows”
Section 5.8.1.1, “Compiling MySQL for Debugging”
Section 23.6.1, “Compiling Programs with libmysqld”
Section 2.9.6, “Configuring SSL Library Support”
Section 5.8.2, “Debugging a MySQL Client”
Section 2.9.4, “Installing MySQL Using a Standard Source Distribution”
Section 16.2.2.1, “`memcached` Command-Line Options”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.5.1.1, “mysql Client Options”

Section 18.5.9, “MySQL Server Usage for NDB Cluster”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 18.4.23, “ndb_select_all — Print Rows from an NDB Table”
Chapter 19, *Partitioning*
Section 22.2, “Performance Schema Build Configuration”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 14.20.3, “Setting Up the InnoDB memcached Plugin”
Section 15.5, “The ARCHIVE Storage Engine”
Section 15.6, “The BLACKHOLE Storage Engine”
Section 15.9, “The EXAMPLE Storage Engine”
Section 15.8, “The FEDERATED Storage Engine”
Section 5.8.4, “Tracing mysqld Using DTrace”
Section B.3.3.3, “What to Do If MySQL Keeps Crashing”
Section 2.1.2, “Which MySQL Version and Distribution to Install”

-d

Section 2.5.8.1, “Basic Steps for MySQL Server Deployment with Docker”
Section 4.6.1, “innochecksum — Offline InnoDB File Checksum Utility”
Section 16.2.2.1, “memcached Command-Line Options”
Section 4.6.2, “myisam_ftdump — Display Full-Text Index information”
Section 4.6.3.1, “myisamchk General Options”
Section 4.4.4, “mysql_plugin — Configure MySQL Server Plugins”
Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”
Section 18.4.6, “ndb_blob_tool — Check and Repair BLOB and TEXT columns of NDB Cluster Tables”
Section 18.4.9, “ndb_delete_all — Delete All Rows from an NDB Table”
Section 18.4.10, “ndb_desc — Describe NDB Tables”
Section 18.4.11, “ndb_drop_index — Drop Index from an NDB Table”
Section 18.4.12, “ndb_drop_table — Drop an NDB Table”
Section 18.4.14, “ndb_index_stat — NDB Index Statistics Utility”
Section 18.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”
Section 18.4.15, “ndb_move_data — NDB Data Copy Utility”
Section 18.4.21, “ndb_redo_log_reader — Check and Print Content of Cluster Redo Log”
Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Section 18.4.23, “ndb_select_all — Print Rows from an NDB Table”
Section 18.4.24, “ndb_select_count — Print Row Counts for NDB Tables”
Section 18.4.25, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”
Section 18.4.26, “ndb_show_tables — Display List of NDB Tables”
Section 18.4.1, “ndbd — The NDB Cluster Data Node Daemon”
Section 18.4.2, “ndbinfo_select_all — Select From ndbinfo Tables”
Section 4.6.3.4, “Other myisamchk Options”
Section 5.1.7, “Server System Variables”

--daemon

Section 18.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”
Section 18.4.1, “ndbd — The NDB Cluster Data Node Daemon”

--data-file-length

Section 4.6.3.3, “myisamchk Repair Options”

--database

Section 4.5.1.1, “mysql Client Options”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 18.4.6, “ndb_blob_tool — Check and Repair BLOB and TEXT columns of NDB Cluster Tables”
Section 18.4.9, “ndb_delete_all — Delete All Rows from an NDB Table”
Section 18.4.10, “ndb_desc — Describe NDB Tables”
Section 18.4.11, “ndb_drop_index — Drop Index from an NDB Table”
Section 18.4.12, “ndb_drop_table — Drop an NDB Table”
Section 18.4.14, “ndb_index_stat — NDB Index Statistics Utility”
Section 18.4.15, “ndb_move_data — NDB Data Copy Utility”
Section 18.4.23, “ndb_select_all — Print Rows from an NDB Table”
Section 18.4.24, “ndb_select_count — Print Row Counts for NDB Tables”
Section 18.4.26, “ndb_show_tables — Display List of NDB Tables”
Section 18.4.27, “ndb_size.pl — NDBCLUSTER Size Requirement Estimator”
Section 18.4.2, “ndbinfo_select_all — Select From ndbinfo Tables”

--databases

Section 7.4.5.2, “Copy a Database from one Server to Another”
Section 7.4.1, “Dumping Data in SQL Format with mysqldump”
Section 7.4.5.1, “Making a Copy of a Database”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 18.5.14, “ndbinfo: The NDB Cluster Information Database”
Section 2.11.10, “Rebuilding or Repairing Tables or Indexes”
Section 7.4.2, “Reloading SQL-Format Backups”

--datadir

Section 2.3.4.2, “Creating an Option File”
Section 2.10.1, “Initializing the Data Directory”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.4.3, “mysql_install_db — Initialize MySQL Data Directory”
Section 4.4.4, “mysql_plugin — Configure MySQL Server Plugins”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”
Section 18.5.16.3, “NDB Cluster and MySQL Security Procedures”
Section 5.7, “Running Multiple MySQL Instances on One Machine”
Section 5.7.3, “Running Multiple MySQL Instances on Unix”
Section 5.1.6, “Server Command Options”
Section 5.7.1, “Setting Up Multiple Data Directories”
Section 5.2, “The MySQL Data Directory”
Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”
Section 4.2.2.2, “Using Option Files”

datadir

Section 2.3.4.2, “Creating an Option File”
Section 2.4.1, “General Notes on Installing MySQL on OS X”
Section 4.3.3, “mysql.server — MySQL Server Startup Script”
Section 2.3.5, “Troubleshooting a Microsoft Windows MySQL Server Installation”
Section 2.3.7, “Windows Platform Restrictions”

--db

Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”

--debug

Section 4.4.1, “comp_err — Compile MySQL Error Message File”

Section 5.8.1.1, “Compiling MySQL for Debugging”
Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.6.7, “`mysqlaccess` — Client for Checking Access Privileges”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.6.9, “`mysqldumpslow` — Summarize Slow Query Log Files”
Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 2.3.4.5, “Starting MySQL from the Windows Command Line”
Section 5.8.3, “The DBUG Package”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”

--debug-check

Section 4.5.1.1, “mysql Client Options”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”

--debug-info

Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”

--debug-level

Section 18.4.25, “`ndb_setup.py` — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”

--debug-sync-timeout

Section 2.9.7, “MySQL Source-Configuration Options”
Section 5.1.6, “Server Command Options”

Section 5.1.7, “Server System Variables”

--default-auth

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”
Section 6.4.1.1, “Native Pluggable Authentication”
Section 6.4.1.2, “Old Native Pluggable Authentication”
Section 6.2.11, “Pluggable Authentication”

--default-authentication-plugin

Section 6.4.1, “Authentication Plugins”
Section 13.7.1.2, “CREATE USER Statement”
Section 6.2.11, “Pluggable Authentication”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”

default-authentication-plugin

Section 6.4.1.4, “SHA-256 Pluggable Authentication”

--default-character-set

Section 6.2.1, “Account User Names and Passwords”
Section 10.15, “Character Set Configuration”
Section 10.5, “Configuring Application Character Set and Collation”
Section 10.4, “Connection Character Sets and Collations”
Section 4.5.1.5, “Executing SQL Statements from a Text File”
Section 13.2.6, “LOAD DATA Statement”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.6, “mysql Client Tips”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 5.1.7, “Server System Variables”

--default-storage-engine

Section 14.14, “InnoDB Startup Options and System Variables”
Section 5.1.6, “Server Command Options”
Section 14.1.5, “Turning Off InnoDB”

--default-time-zone

Section 5.1.13, “MySQL Server Time Zone Support”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”

--default-tmp-storage-engine

Section 14.14, “InnoDB Startup Options and System Variables”

Section 5.1.6, “Server Command Options”
Section 14.1.5, “Turning Off InnoDB”

--default.key_buffer_size

Section 5.1.8.3, “Structured System Variables”

DEFAULT_CHARSET

Section 10.5, “Configuring Application Character Set and Collation”
Section 10.3.2, “Server Character Set and Collation”

DEFAULT_COLLATION

Section 10.5, “Configuring Application Character Set and Collation”
Section 10.3.2, “Server Character Set and Collation”

--defaults-extra-file

Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”
Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”
Section 4.6.3.1, “myisamchk General Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 5.1.6, “Server Command Options”
Section 4.2.2.2, “Using Option Files”

--defaults-file

Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 14.8.1, “InnoDB Startup Configuration”
Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”
Section 4.6.3.1, “myisamchk General Options”
Section 4.5.1.1, “mysql Client Options”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 23.6.3, “Options with the Embedded Server”

Resetting the Root Password: Unix and Unix-Like Systems
Resetting the Root Password: Windows Systems
Section 5.7, “Running Multiple MySQL Instances on One Machine”
Section 5.7.3, “Running Multiple MySQL Instances on Unix”
Section 5.1.6, “Server Command Options”
Section 5.7.2.2, “Starting Multiple MySQL Instances as Windows Services”
Section 5.7.2.1, “Starting Multiple MySQL Instances at the Windows Command Line”
Section 2.3.4.7, “Starting MySQL as a Windows Service”

--defaults-group-suffix

Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”
Section 4.9, “Environment Variables”
Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”
Section 4.6.3.1, “myisamchk General Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 5.1.6, “Server Command Options”

--delay

Section 18.4.2, “`ndbinfo_select_all` — Select From ndbinfo Tables”

--delay-key-write

Section 8.11.5, “External Locking”

--delay_key_write

Section 5.1.8, “Using System Variables”

delay_key_write

Section 15.2.1, “MyISAM Startup Options”

--delayed-insert

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--delete

Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 18.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

--delete-master-logs

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--delete-orphans

Section 18.4.6, “`ndb_blob_tool` — Check and Repair BLOB and TEXT columns of NDB Cluster Tables”

--delimiter

Section 4.5.1.1, “mysql Client Options”

Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.4.23, “ndb_select_all — Print Rows from an NDB Table”

--demangle

Section 5.8.1.5, “Using a Stack Trace”

--des-key-file

Section 12.14, “Encryption and Compression Functions”
Section 13.7.6.3, “FLUSH Statement”
Section 5.1.6, “Server Command Options”

--descending

Section 18.4.23, “ndb_select_all — Print Rows from an NDB Table”

--description

Section 4.6.3.4, “Other myisamchk Options”

--detach

Section 4.5.7, “mysqlslap — A Load Emulation Client”

--diff-default

Section 18.4.7, “ndb_config — Extract NDB Cluster Configuration Information”

--disable

Section 4.2.2.4, “Program Option Modifiers”

--disable-auto-rehash

Section 14.8.11.2, “Configuring Non-Persistent Optimizer Statistics Parameters”
Section 4.5.1.1, “mysql Client Options”

--disable-indexes

Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Restoring to More Nodes Than the Original

--disable-innodb

Section 14.1.5, “Turning Off InnoDB”
Section 1.3, “What Is New in MySQL 5.6”

--disable-keys

Section 4.5.4, “mysqldump — A Database Backup Program”

--disable-log-bin

Section 17.1.4.4, “Binary Log Options and Variables”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”

--disable-named-commands

Section 4.5.1.1, “mysql Client Options”

--disable-plugin_name

Section 5.5.1, “Installing and Uninstalling Plugins”

--disable-ssl

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”

Section 5.1.6, “Server Command Options”

--disconnect-slave-event-count

Section 17.1.4.3, “Replica Server Options and Variables”

--disk

Section 18.4.23, “`ndb_select_all` — Print Rows from an NDB Table”

--diskscan

Section 18.4.9, “`ndb_delete_all` — Delete All Rows from an NDB Table”

--dont-ignore-systab-0

Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”

--drop-source

Section 18.4.15, “`ndb_move_data` — NDB Data Copy Utility”

--dry-scp

Section 18.4.13, “`ndb_error_reporter` — NDB Error-Reporting Utility”

--dryrun

Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”

--dump

Section 4.6.2, “`myisam_ftdump` — Display Full-Text Index information”
Section 18.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

--dump-date

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--dump-file

Section 18.4.6, “`ndb_blob_tool` — Check and Repair BLOB and TEXT columns of NDB Cluster Tables”

--dump-slave

Section 4.5.4, “`mysqldump` — A Database Backup Program”

E

[index top]

-E

Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”

-e

Section 7.6.2, “How to Check MyISAM Tables for Errors”
Section 4.6.1, “`innochecksum` — Offline InnoDB File Checksum Utility”
Section 13.2.7, “LOAD XML Statement”
Section 2.5.8.2, “More Topics on Deploying MySQL Server with Docker”
Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”
Section 4.6.3.2, “myisamchk Check Options”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.3.3, “myisamchk Repair Options”

Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.6.9, “NDB Cluster Backups With NDB Cluster Replication”
Section 18.4.5, “ndb_mgm — The NDB Cluster Management Client”
Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Section 4.6.3.5, “Obtaining Table Information with myisamchk”
Section 18.2.6, “Safe Shutdown and Restart of NDB Cluster”
Section 4.2.2.1, “Using Options on the Command Line”
Section 18.5.8.2, “Using The NDB Cluster Management Client to Create a Backup”

--embedded

Section 4.7.2, “mysql_config — Display Options for Compiling Clients”

--embedded-libs

Section 4.7.2, “mysql_config — Display Options for Compiling Clients”

--enable-64bit

Section 16.2.1, “Installing memcached”

--enable-cleartext-plugin

Section 6.4.1.5, “Client-Side Cleartext Pluggable Authentication”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 6.4.1.6, “PAM Pluggable Authentication”

--enable-dtrace

Section 16.2.1, “Installing memcached”
Section 16.2.2.6, “Using memcached and DTrace”

--enable-memcache

Section 16.2.3.6, “Using MySQL and memcached with PHP”

--enable-plugin_name

Section 5.5.1, “Installing and Uninstalling Plugins”

--enable-threads

Section 16.2.1, “Installing memcached”

ENABLE_DEBUG_SYNC

Section 2.9.7, “MySQL Source-Configuration Options”

enabled

Section 2.5.1, “Installing MySQL on Linux Using the MySQL Yum Repository”

ENABLED_LOCAL_INFILE

Section 2.9.7, “MySQL Source-Configuration Options”
Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”

--enforce-gtid-consistency
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 17.1.3.4, “Restrictions on Replication with GTIDs”

--engine
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”

--env
Section 2.5.8.2, “More Topics on Deploying MySQL Server with Docker”

--error-insert
Section 18.4.15, “`ndb_move_data` — NDB Data Copy Utility”

--event-scheduler
Section 20.4.2, “Event Scheduler Configuration”

event-scheduler
Section 20.4.2, “Event Scheduler Configuration”

--events
Section 7.4.5.3, “Dumping Stored Programs”
Section 7.4.5.4, “Dumping Table Definitions and Content Separately”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 2.11.4, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

--example
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”

--exclude-*
Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”

--exclude-databases
Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”

--exclude-gtids
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”

--exclude-intermediate-sql-tables
Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”

--exclude-missing-columns
Section 18.4.15, “`ndb_move_data` — NDB Data Copy Utility”
Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”

--exclude-missing-tables
Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”

--exclude-tables
Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”

--excludedbs
Section 18.4.27, “`ndb_size.pl` — NDBCLUSTER Size Requirement Estimator”

--excludetables

Section 18.4.27, “ndb_size.pl — NDBCLUSTER Size Requirement Estimator”

--execute

Section 4.5.1.3, “mysql Client Logging”
Section 4.5.1.1, “mysql Client Options”
Section 18.4.5, “ndb_mgm — The NDB Cluster Management Client”
Section 4.2.2.1, “Using Options on the Command Line”
Section 18.5.8.2, “Using The NDB Cluster Management Client to Create a Backup”

--exit-info

Section 5.1.6, “Server Command Options”

--extend-check

Section 4.6.3.2, “myisamchk Check Options”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.3.3, “myisamchk Repair Options”

--extended

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--extended-insert

Section 4.5.4, “mysqldump — A Database Backup Program”

--external-locking

Section 8.11.5, “External Locking”
Section 15.2.1, “MyISAM Startup Options”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 8.12.1, “System Factors”

--extra-file

Section 4.7.3, “my_print_defaults — Display Options from Option Files”

--extra-node-info

Section 18.4.10, “ndb_desc — Describe NDB Tables”

--extra-partition-info

Section 18.4.10, “ndb_desc — Describe NDB Tables”
Section 18.5.14.4, “The ndbinfo cluster_operations Table”
Section 18.5.14.22, “The ndbinfo server_operations Table”

F

[index top]

-F

Section 4.4.1, “comp_err — Compile MySQL Error Message File”
Section 4.6.3.2, “myisamchk Check Options”
Section 4.6.4, “myisamlog — Display MyISAM Log File Contents”
Section 4.5.1.2, “mysql Client Commands”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”

Section 4.5.7, “mysqlslap — A Load Emulation Client”

-f

Section 18.5.1, “Commands in the NDB Cluster Management Client”
Section 18.2.4, “Initial Startup of NDB Cluster”
Section 18.2.2.3, “Initial Startup of NDB Cluster on Windows”
Section 16.2.2.1, “memcached Command-Line Options”
Section 4.6.3.2, “myisamchk Check Options”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.6.4, “myisamlog — Display MyISAM Log File Contents”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.16, “mysql_zap — Kill Processes That Match a Pattern”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 18.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 18.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”
Section 18.4.21, “ndb_redo_log_reader — Check and Print Content of Cluster Redo Log”
Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Section 5.8.1.5, “Using a Stack Trace”

--fast

Section 4.6.3.2, “myisamchk Check Options”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--federated

Section 15.8, “The FEDERATED Storage Engine”

--fields

Section 18.4.7, “ndb_config — Extract NDB Cluster Configuration Information”

--fields-enclosed-by

Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”

--fields-escaped-by

Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”

--fields-optionally-enclosed-by

Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”

--fields-terminated-by

Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”

Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”

--fields-xxx

Section 4.5.4, “mysqldump — A Database Backup Program”

--fix-db-names

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--fix-table-names

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--flush

Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section B.3.3.3, “What to Do If MySQL Keeps Crashing”

--flush-logs

Section 7.3.1, “Establishing a Backup Policy”
Section 5.4, “MySQL Server Logs”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 6.2.2, “Privileges Provided by MySQL”

--flush-privileges

Section 4.5.4, “mysqldump — A Database Backup Program”

--flushlog

Section 4.6.10, “mysqlhotcopy — A Database Backup Program”

--force

Section 4.6.3.2, “myisamchk Check Options”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.11, “mysql_convert_table_format — Convert Tables to Use a Given Storage Engine”
Section 4.4.3, “mysql_install_db — Initialize MySQL Data Directory”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 3.5, “Using mysql in Batch Mode”

--force-if-open

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”

--force-read

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”

--foreground

Section 18.4.1, “ndbd — The NDB Cluster Data Node Daemon”

--format

Section 18.4.27, “ndb_size.pl — NDBCLUSTER Size Requirement Estimator”

--fs

Section 18.4.13, “ndb_error_reporter — NDB Error-Reporting Utility”

G

[index top]

-G
Section 4.5.1.1, “mysql Client Options”
Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”

-g
Section 5.8.1.1, “Compiling MySQL for Debugging”
Section 4.7.3, “my_print_defaults — Display Options from Option Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”

--gci
Section 18.4.23, “ndb_select_all — Print Rows from an NDB Table”

--gci64
Section 18.4.23, “ndb_select_all — Print Rows from an NDB Table”

--gdb
Section 5.8.1.4, “Debugging mysqld under gdb”
Section 5.1.6, “Server Command Options”
Section 4.10, “Unix Signal Handling in MySQL”

--general-log
Section 4.2.2.1, “Using Options on the Command Line”
Section 5.1.8, “Using System Variables”

--general_log
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.4.3, “The General Query Log”
Section 4.2.2.1, “Using Options on the Command Line”
Section 5.1.8, “Using System Variables”

--general_log_file
Section 5.7, “Running Multiple MySQL Instances on One Machine”
Section 5.1.7, “Server System Variables”
Section 5.4.3, “The General Query Log”

H

[index top]

-H
Section 4.4.1, “comp_err — Compile MySQL Error Message File”
Section 4.6.3.1, “myisamchk General Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”

-h
Section 1.1, “About This Manual”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”

Section 4.2.1, “Invoking MySQL Programs”
Section 16.2.2.1, “memcached Command-Line Options”
Section 4.6.2, “myisam_ftdump — Display Full-Text Index information”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.6.10, “mysqlhotcopy — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.4.17, “ndb_print_file — Print NDB Disk Data File Contents”
Section 18.4.23, “ndb_select_all — Print Rows from an NDB Table”
Section 18.4.25, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”
Section 4.7.4, “resolve_stack_dump — Resolve Numeric Stack Trace Dump to Symbols”
Section 5.1.6, “Server Command Options”
Section 4.2.2.1, “Using Options on the Command Line”

--header

Section 18.4.23, “ndb_select_all — Print Rows from an NDB Table”

--header-file

Section 4.4.1, “comp_err — Compile MySQL Error Message File”

--HELP

Section 4.6.3.1, “myisamchk General Options”

--help

Section 4.4.1, “comp_err — Compile MySQL Error Message File”
Section 18.2.2.4, “Installing NDB Cluster Processes as Windows Services”
Section 4.7.3, “my_print_defaults — Display Options from Option Files”
Section 4.6.2, “myisam_ftdump — Display Full-Text Index information”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”
Section 4.6.11, “mysql_convert_table_format — Convert Tables to Use a Given Storage Engine”
Section 4.6.12, “mysql_find_rows — Extract SQL Statements from Files”
Section 4.4.3, “mysql_install_db — Initialize MySQL Data Directory”
Section 4.4.4, “mysql_plugin — Configure MySQL Server Plugins”
Section 4.6.14, “mysql_setpermission — Interactively Set Permissions in Grant Tables”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.6.15, “mysql_waitpid — Kill Process and Wait for Its Termination”
Section 4.6.16, “mysql_zap — Kill Processes That Match a Pattern”
Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”
Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”
Section 4.5.4, “mysqldump — A Database Backup Program”

Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.6.10, “mysqlhotcopy — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.4.6, “ndb_blob_tool — Check and Repair BLOB and TEXT columns of NDB Cluster Tables”
Section 18.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 18.4.13, “ndb_error_reporter — NDB Error-Reporting Utility”
Section 18.4.17, “ndb_print_file — Print NDB Disk Data File Contents”
Section 18.4.21, “ndb_redo_log_reader — Check and Print Content of Cluster Redo Log”
Section 18.4.25, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 4.1, “Overview of MySQL Programs”
Section 4.8.1, “perror — Display MySQL Error Message Information”
Section 4.7.4, “resolve_stack_dump — Resolve Numeric Stack Trace Dump to Symbols”
Section 4.8.3, “resolveip — Resolve Host name to IP Address or Vice Versa”
Section 5.1.6, “Server Command Options”
Section 2.10.3, “Testing the Server”
Section 1.2.2, “The Main Features of MySQL”
Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”
Chapter 3, *Tutorial*
Section 4.2.2.2, “Using Option Files”
Section 4.2.2.1, “Using Options on the Command Line”

--hex

Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”

--hex-blob

Section 4.5.4, “mysqldump — A Database Backup Program”

--hexdump

Section 4.6.8.1, “mysqlbinlog Hex Dump Format”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”

--histignore

Section 4.5.1.3, “mysql Client Logging”
Section 4.5.1.1, “mysql Client Options”

--host

Section 1.1, “About This Manual”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 4.2.1, “Invoking MySQL Programs”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”
Section 4.6.11, “mysql_convert_table_format — Convert Tables to Use a Given Storage Engine”
Section 4.6.14, “mysql_setpermission — Interactively Set Permissions in Grant Tables”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.10, “mysqlhotcopy — A Database Backup Program”

Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 4.2.2.6, “Option Defaults, Options Expecting Values, and the = Sign”
Section 5.1.7, “Server System Variables”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 5.7.4, “Using Client Programs in a Multiple-Server Environment”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”
Section 4.2.2.2, “Using Option Files”
Section 4.2.2.1, “Using Options on the Command Line”

host

Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”
Section 4.2.2.2, “Using Option Files”

--hostname

Section 18.4.27, “ndb_size.pl — NDBCLUSTER Size Requirement Estimator”

--howto

Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”

--html

Section 4.5.1.1, “mysql Client Options”

I

[index top]

-I

Section 16.2.2.1, “memcached Command-Line Options”
Section 16.2.5, “memcached FAQ”
Section 4.6.4, “myisamlog — Display MyISAM Log File Contents”
Section 4.6.15, “mysql_waitpid — Kill Process and Wait for Its Termination”
Section 4.6.16, “mysql_zap — Kill Processes That Match a Pattern”
Section 4.8.1, “perror — Display MySQL Error Message Information”
Section 4.8.2, “replace — A String-Replacement Utility”
Section 4.8.3, “resolveip — Resolve Host name to IP Address or Vice Versa”

-i

Section 18.5.1, “Commands in the NDB Cluster Management Client”
Section 7.6.2, “How to Check MyISAM Tables for Errors”
Section 16.2.2.1, “memcached Command-Line Options”
Section 4.6.3.2, “myisamchk Check Options”
Section 4.6.4, “myisamlog — Display MyISAM Log File Contents”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.4, “mysql_plugin — Configure MySQL Server Plugins”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”

--i-am-a-dummy

Section 4.5.1.1, “mysql Client Options”

Section 4.5.1.6, “mysql Client Tips”

--id

Section 18.4.7, “ndb_config — Extract NDB Cluster Configuration Information”

--ignore

Section 4.5.5, “mysqlimport — A Data Import Program”

--ignore-db-dir

Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”

--ignore-lines

Section 4.5.5, “mysqlimport — A Data Import Program”

--ignore-spaces

Section 4.5.1.1, “mysql Client Options”

--ignore-table

Section 4.5.4, “mysqldump — A Database Backup Program”

--in-file

Section 4.4.1, “comp_err — Compile MySQL Error Message File”

--include

Section 4.7.2, “mysql_config — Display Options for Compiling Clients”

--include-*

Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”

--include-databases

Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”

--include-gtids

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”

--include-master-host-port

Section 4.5.4, “mysqldump — A Database Backup Program”

--include-tables

Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”

--info

Section 4.8.1, “perror — Display MySQL Error Message Information”
Section 4.8.3, “resolveip — Resolve Host name to IP Address or Vice Versa”

--Information

Section 4.6.12, “mysql_find_rows — Extract SQL Statements from Files”

--information

Section 4.6.3.2, “myisamchk Check Options”

--init-command

Section 4.5.1.1, “mysql Client Options”

--init-rpl-role
Section 1.3, “What Is New in MySQL 5.6”

--init_connect
Section 10.5, “Configuring Application Character Set and Collation”

--initial
Section 18.5.7.2, “Adding NDB Cluster Data Nodes Online: Basic procedure”
Section 18.5.7.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 18.5.1, “Commands in the NDB Cluster Management Client”
Section 18.3.3.5, “Defining an NDB Cluster Management Server”
Section 18.3.3.4, “Defining Computers in an NDB Cluster”
Section 18.3.3.6, “Defining NDB Cluster Data Nodes”
Section 18.3.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”
Section 18.5.12, “Distributed Privileges Using Shared Grant Tables”
Section 18.2.2.3, “Initial Startup of NDB Cluster on Windows”
Section 18.6.3, “Known Issues in NDB Cluster Replication”
Section 18.1.7.10, “Limitations Relating to Multiple NDB Cluster Nodes”
Section 18.3.3, “NDB Cluster Configuration Files”
Section 18.5.10.3, “NDB Cluster Disk Data Storage Requirements”
Section 18.3.3.11, “NDB Cluster Shared-Memory Connections”
Section 18.3.3.9, “NDB Cluster TCP/IP Connections”
Section 18.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”
Section 18.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”
Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 18.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”
Section 18.4.3, “`ndbmtd` — The NDB Cluster Data Node Daemon (Multi-Threaded)”
Section 18.3.2, “Overview of NDB Cluster Configuration Parameters, Options, and Variables”
Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 18.6.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”
Restoring to Fewer Nodes Than the Original
Section 18.5.4, “Summary of NDB Cluster Start Phases”

--initial-start
Section 18.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”

--innodb
Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.1.5, “Turning Off InnoDB”
Section 1.3, “What Is New in MySQL 5.6”

--innodb-adaptive-hash-index
Section 14.14, “InnoDB Startup Options and System Variables”

--innodb-file-per-table
Section 5.1.6, “Server Command Options”

innodb-file-per-table
Section 5.1.6, “Server Command Options”

--innodb-rollback-on-timeout
Section 14.21.4, “InnoDB Error Handling”
Section 14.14, “InnoDB Startup Options and System Variables”

--innodb-status-file
Section 14.17.2, “Enabling InnoDB Monitors”

Section 14.14, “InnoDB Startup Options and System Variables”

--innodb-xxx

Section 5.1.6, “Server Command Options”

--innodb_file_per_table

Section 8.12.3.3, “Using Symbolic Links for Databases on Windows”

innodb_file_per_table

Section 17.1.1.6, “Creating a Data Snapshot Using Raw Data Files”

--innodb_support_xa

Section 5.4.4, “The Binary Log”

--insert-ignore

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--install

Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”
Section 18.2.2.4, “Installing NDB Cluster Processes as Windows Services”
Section 18.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”
Section 18.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”
Section 5.1.6, “Server Command Options”
Section 5.7.2.2, “Starting Multiple MySQL Instances as Windows Services”
Section 2.3.4.7, “Starting MySQL as a Windows Service”

--install-manual

Section 5.1.6, “Server Command Options”
Section 5.7.2.2, “Starting Multiple MySQL Instances as Windows Services”
Section 2.3.4.7, “Starting MySQL as a Windows Service”

INSTALL_LAYOUT

Section 2.9.7, “MySQL Source-Configuration Options”
Section 5.1.7, “Server System Variables”

INSTALL_SECURE_FILE_PRIV_EMBEDDEDDIR

Section 2.9.7, “MySQL Source-Configuration Options”
Section 5.1.7, “Server System Variables”

INSTALL_SECURE_FILE_PRIVDIR

Section 5.1.7, “Server System Variables”

--interactive

Section 18.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”

--iterations

Section 4.5.7, “`mysqlslap` — A Load Emulation Client”

J

[index top]

-j

Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”

--join

Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”

K

[index top]

-K

Section 4.5.4, “mysqldump — A Database Backup Program”

-k

Section 16.2.2.1, “memcached Command-Line Options”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 18.4.25, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”

--keep-my-cnf

Section 4.4.3, “mysql_install_db — Initialize MySQL Data Directory”

--keep_files_on_create

Section 13.1.17, “CREATE TABLE Statement”

--keepold

Section 4.6.10, “mysqlhotcopy — A Database Backup Program”

--key-file

Section 18.4.25, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”

--key_buffer_size

Section 5.1.6, “Server Command Options”

--keys

Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”

--keys-used

Section 4.6.3.3, “myisamchk Repair Options”

L

[index top]

-L

Section 16.2.2.1, “memcached Command-Line Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Section 2.13.3, “Problems Using the Perl DBI/DBD Interface”

-l

Section 16.2.2.1, “memcached Command-Line Options”

Section 4.7.3, “my_print_defaults — Display Options from Option Files”
Section 4.6.2, “myisam_ftdump — Display Full-Text Index information”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 18.4.15, “ndb_move_data — NDB Data Copy Utility”
Section 18.4.21, “ndb_redo_log_reader — Check and Print Content of Cluster Redo Log”
Section 18.4.23, “ndb_select_all — Print Rows from an NDB Table”
Section 18.4.24, “ndb_select_count — Print Row Counts for NDB Tables”
Section 18.4.26, “ndb_show_tables — Display List of NDB Tables”
Section 18.4.2, “ndbinfo_select_all — Select From ndbinfo Tables”

--language

Section 5.1.6, “Server Command Options”
Section 1.3, “What Is New in MySQL 5.6”

--large-pages

Section 8.12.4.2, “Enabling Large Page Support”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”

--lc-messages

Section 5.1.6, “Server Command Options”
Section 1.3, “What Is New in MySQL 5.6”

--lc-messages-dir

Section 5.1.6, “Server Command Options”
Section 1.3, “What Is New in MySQL 5.6”

--ldata

Section 4.4.3, “mysql_install_db — Initialize MySQL Data Directory”

--ledir

Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”

--length

Section 4.6.2, “myisam_ftdump — Display Full-Text Index information”

--libmysqld-libs

Section 4.7.2, “mysql_config — Display Options for Compiling Clients”

--libs

Section 4.7.2, “mysql_config — Display Options for Compiling Clients”

--libs_r

Section 4.7.2, “mysql_config — Display Options for Compiling Clients”

--line-numbers

Section 4.5.1.1, “mysql Client Options”

--lines-terminated-by

Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”

--loadqueries

Section 18.4.27, “ndb_size.pl — NDBCLUSTER Size Requirement Estimator”

--local

Section 4.5.5, “mysqlimport — A Data Import Program”
Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”

--local-infile

Section 13.2.7, “LOAD XML Statement”
Section 4.5.1.1, “mysql Client Options”
Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”

--local-load

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”

--local-service

Section 5.1.6, “Server Command Options”
Section 2.3.4.7, “Starting MySQL as a Windows Service”

--lock

Section 18.4.23, “ndb_select_all — Print Rows from an NDB Table”
Section 18.4.24, “ndb_select_count — Print Row Counts for NDB Tables”

--lock-all-tables

Section 4.5.4, “mysqldump — A Database Backup Program”

--lock-tables

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”

--log

Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”
Section 1.3, “What Is New in MySQL 5.6”

--log-bin

Section 7.3.3, “Backup Strategy Summary”
Section 17.1.4.4, “Binary Log Options and Variables”
Section 7.2, “Database Backup Methods”
Section 7.3.1, “Establishing a Backup Policy”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 17.4.5, “How to Report Replication Bugs or Problems”
Section B.3.7, “Known Issues in MySQL”
Section 18.1.7.1, “Noncompliance with SQL Syntax in NDB Cluster”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”
Section 13.4.1.1, “PURGE BINARY LOGS Statement”
Section 5.7, “Running Multiple MySQL Instances on One Machine”
Section 17.3.7, “Switching Sources During Failover”
Section 5.4.4, “The Binary Log”
Section 17.4.4, “Troubleshooting Replication”
Section 17.4.3, “Upgrading a Replication Setup”
Section 7.3.2, “Using Backups for Recovery”

Section 1.3, “What Is New in MySQL 5.6”

--log-bin-index

Section 17.1.4.4, “Binary Log Options and Variables”
Section 18.1.7.1, “Noncompliance with SQL Syntax in NDB Cluster”
Section 5.4.4, “The Binary Log”

--log-bin-use-v1-row-events

Section 2.12.3, “Downgrade Notes”

--log-error

Section 5.4.2.2, “Error Logging on Unix and Unix-Like Systems”
Section 5.4.2.1, “Error Logging on Windows”
Section 5.4.2.3, “Error Logging to the System Log”
Section 2.5.8.2, “More Topics on Deploying MySQL Server with Docker”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.2.2.6, “Option Defaults, Options Expecting Values, and the = Sign”
Section 5.7, “Running Multiple MySQL Instances on One Machine”
Section 5.1.6, “Server Command Options”
Section 5.4.7, “Server Log Maintenance”
Section 2.3.4.5, “Starting MySQL from the Windows Command Line”
Section 2.3.4.4, “Starting the Server for the First Time”

--log-isam

Section 4.6.4, “`myisamlog` — Display MyISAM Log File Contents”
Section 5.1.6, “Server Command Options”

--log-name

Section 18.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”

--log-raw

Section 6.1.2.3, “Passwords and Logging”
Section 5.1.6, “Server Command Options”
Section 5.4.3, “The General Query Log”

--log-short-format

Section 5.1.6, “Server Command Options”
Section 5.4.5, “The Slow Query Log”

--log-slave-updates

Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 17.4.5, “How to Report Replication Bugs or Problems”
Section 5.4.4, “The Binary Log”

--log-slow-admin-statements

Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”

--log-slow-slave-statements

Section 17.1.4.3, “Replica Server Options and Variables”

--log-tc

Section 5.1.6, “Server Command Options”

--log-tc-size

Section 5.1.6, “Server Command Options”
Section 5.1.9, “Server Status Variables”

--log-warnings

Section B.3.2.10, “Communication Errors and Aborted Connections”
Section 5.4.2.4, “Error Log Filtering”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section B.3.2.8, “MySQL server has gone away”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 5.1.6, “Server Command Options”

--log_output

Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”

--login-path

Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”
Section 4.7.3, “`my_print_defaults` — Display Options from Option Files”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 4.2.2.2, “Using Option Files”

--loops

Section 18.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”
Section 18.4.26, “`ndb_show_tables` — Display List of NDB Tables”
Section 18.4.2, “`ndbinfo_select_all` — Select From ndbinfo Tables”

--loose

Section 4.2.2.4, “Program Option Modifiers”

--loose-opt_name

Section 4.2.2.2, “Using Option Files”

--lossy-conversions

Section 18.4.15, “`ndb_move_data` — NDB Data Copy Utility”
Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”

--low-priority

Section 4.5.5, “`mysqlimport` — A Data Import Program”

--low-priority-updates

Section 8.11.3, “Concurrent Inserts”
Section 13.2.5, “INSERT Statement”
Section 8.11.2, “Table Locking Issues”

--lower-case-table-names
Section 9.2.3, “Identifier Case Sensitivity”

M

[index top]

-M
Section 16.2.2.1, “`memcached` Command-Line Options”

-m
Section 16.2.2.1, “`memcached` Command-Line Options”
Section 4.6.3.2, “myisamchk Check Options”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 18.6.9, “NDB Cluster Backups With NDB Cluster Replication”
Section 18.4.21, “`ndb_redo_log_reader` — Check and Print Content of Cluster Redo Log”
Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”

--malloc-lib
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

--master-connect-retry
Section 17.1.4.3, “Replica Server Options and Variables”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”

--master-data
Section 17.1.1.5, “Creating a Data Snapshot Using mysqldump”
Section 7.3.1, “Establishing a Backup Policy”
Section 5.4, “MySQL Server Logs”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.1.3.3, “Using GTIDs for Failover and Scaleout”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

--master-host
Section 17.1.4.3, “Replica Server Options and Variables”

--master-info-file
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.2.2.2, “Replication Metadata Repositories”

--master-password
Section 17.1.4.3, “Replica Server Options and Variables”

--master-port
Section 17.1.4.3, “Replica Server Options and Variables”

--master-retry-count
Section 13.4.2.1, “CHANGE MASTER TO Statement”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”

--master-ssl
Section 17.1.4.3, “Replica Server Options and Variables”

--master-ssl-ca
Section 17.1.4.3, “Replica Server Options and Variables”

--master-ssl-capath
Section 17.1.4.3, “Replica Server Options and Variables”

--master-ssl-cert
Section 17.1.4.3, “Replica Server Options and Variables”

--master-ssl-cipher
Section 17.1.4.3, “Replica Server Options and Variables”

--master-ssl-key
Section 17.1.4.3, “Replica Server Options and Variables”

--master-user
Section 17.1.4.3, “Replica Server Options and Variables”

--max
Section 4.2.2.5, “Using Options to Set Program Variables”

--max-allowed-packet
Section 4.5.1.1, “mysql Client Options”
Section 4.5.4, “`mysqldump` — A Database Backup Program”

--max-binlog-dump-events
Section 17.1.4.4, “Binary Log Options and Variables”

--max-binlog-size
Section 17.1.4.3, “Replica Server Options and Variables”

--max-join-size
Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.6, “mysql Client Tips”

--max-record-length
Section 4.6.3.3, “myisamchk Repair Options”
Section 13.7.2.5, “REPAIR TABLE Statement”

--max-relay-log-size
Section 17.1.4.3, “Replica Server Options and Variables”

--max-seeks-for-key
Section 8.2.1.19, “Avoiding Full Table Scans”
Section B.3.5, “Optimizer-Related Issues”

--max_a
Section 4.2.2.5, “Using Options to Set Program Variables”

--maximum
Section 4.2.2.4, “Program Option Modifiers”

--maximum-back_log
Section 4.2.2.4, “Program Option Modifiers”

--maximum-innodb-log-file-size
Section 5.1.8, “Using System Variables”

--maximum-max_heap_table_size
Section 4.2.2.4, “Program Option Modifiers”

--maximum-query_cache_size
Section 8.10.3.3, “Query Cache Configuration”

--maximum-var_name
Section 5.1.6, “Server Command Options”
Section 5.1.8, “Using System Variables”

--medium-check
Section 4.6.3.2, “myisamchk Check Options”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

--memlock
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 14.6.3.1, “The System Tablespace”

--method
Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”

--mount
Section 2.5.8.2, “More Topics on Deploying MySQL Server with Docker”

--my-plugin
Section 5.5.1, “Installing and Uninstalling Plugins”

--my-print-defaults
Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”

--my_plugin
Section 5.5.1, “Installing and Uninstalling Plugins”

--mycnf
Section 18.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”
Section 18.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”

--myisam-block-size
Section 8.10.2.5, “Key Cache Block Size”
Section 5.1.6, “Server Command Options”

--myisam_sort_buffer_size
Section 4.6.3.6, “myisamchk Memory Usage”

MYSQL_ALLOW_EMPTY_PASSWORD
Section 2.5.8.2, “More Topics on Deploying MySQL Server with Docker”

MYSQL_DATABASE
Section 2.5.8.2, “More Topics on Deploying MySQL Server with Docker”

MYSQL_LOG_CONSOLE
Section 2.5.8.2, “More Topics on Deploying MySQL Server with Docker”

MYSQL_MAINTAINER_MODE
Section 2.9.8, “Dealing with Problems Compiling MySQL”

MYSQL_ONETIME_PASSWORD
Section 2.5.8.1, “Basic Steps for MySQL Server Deployment with Docker”
Section 2.5.8.2, “More Topics on Deploying MySQL Server with Docker”

MYSQL_PASSWORD
Section 2.5.8.2, “More Topics on Deploying MySQL Server with Docker”

MYSQL_RANDOM_ROOT_PASSWORD
Section 2.5.8.2, “More Topics on Deploying MySQL Server with Docker”

MYSQL_ROOT_HOST
Section 2.5.8.2, “More Topics on Deploying MySQL Server with Docker”

MYSQL_ROOT_PASSWORD
Section 2.5.8.2, “More Topics on Deploying MySQL Server with Docker”

MYSQL_TCP_PORT
Section 2.9.5, “Installing MySQL Using a Development Source Tree”
Section 2.9.7, “MySQL Source-Configuration Options”

MYSQL_UNIX_ADDR
Section B.3.3.6, “How to Protect or Change the MySQL Unix Socket File”
Section 2.9.5, “Installing MySQL Using a Development Source Tree”
Section 2.9.7, “MySQL Source-Configuration Options”

MYSQL_USER
Section 2.5.8.2, “More Topics on Deploying MySQL Server with Docker”

--mysqladmin
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”

--mysqld
Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

--mysqld-version
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

N

[index top]

-N
Section 4.4.1, “`comp_err` — Compile MySQL Error Message File”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 18.4.25, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”

-n

Section 18.5.1, “Commands in the NDB Cluster Management Client”
Section 16.2.2.1, “memcached Command-Line Options”
Section 4.7.3, “my_print_defaults — Display Options from Option Files”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.6.10, “mysqlhotcopy — A Database Backup Program”
Section 18.4.10, “ndb_desc — Describe NDB Tables”
Section 18.4.21, “ndb_redo_log_reader — Check and Print Content of Cluster Redo Log”
Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Section 18.4.25, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”
Section 18.4.28, “ndb_waiter — Wait for NDB Cluster to Reach a Given Status”
Section 18.4.1, “ndbd — The NDB Cluster Data Node Daemon”
Section 4.7.4, “resolve_stack_dump — Resolve Numeric Stack Trace Dump to Symbols”

--name

Section 2.5.8.1, “Basic Steps for MySQL Server Deployment with Docker”

--name-file

Section 4.4.1, “comp_err — Compile MySQL Error Message File”

--named-commands

Section 4.5.1.1, “mysql Client Options”

--ndb

Section 4.8.1, “perror — Display MySQL Error Message Information”

--ndb-batch-size

MySQL Server Options for NDB Cluster
Section 18.6.5, “Preparing the NDB Cluster for Replication”

--ndb-blob-read-batch-bytes

MySQL Server Options for NDB Cluster

--ndb-blob-write-batch-bytes

MySQL Server Options for NDB Cluster
Section 18.6.5, “Preparing the NDB Cluster for Replication”

--ndb-cluster

Section A.10, “MySQL 5.6 FAQ: NDB Cluster”

--ndb-cluster-connection-pool

MySQL Server Options for NDB Cluster

--ndb-connectstring

Section 18.2.2.1, “Installing NDB Cluster on Windows from a Binary Release”
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
MySQL Server Options for NDB Cluster
Section 18.5.9, “MySQL Server Usage for NDB Cluster”

Section 18.5.16.2, “NDB Cluster and MySQL Privileges”
Section 18.1.1, “NDB Cluster Core Concepts”
Section 18.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”
Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 18.6.5, “Preparing the NDB Cluster for Replication”
Section 18.3.3.2, “Recommended Starting Configuration for NDB Cluster”
Restoring to Fewer Nodes Than the Original
Restoring to More Nodes Than the Original

--ndb-deferred-constraints

MySQL Server Options for NDB Cluster

--ndb-distribution

MySQL Server Options for NDB Cluster

--ndb-force-send

Section 18.3.3.2, “Recommended Starting Configuration for NDB Cluster”

--ndb-index-stat-enable

Section 18.3.3.2, “Recommended Starting Configuration for NDB Cluster”

--ndb-log-apply-status

MySQL Server Options for NDB Cluster
NDB Cluster System Variables

--ndb-log-empty-epochs

MySQL Server Options for NDB Cluster
Section 18.6.4, “NDB Cluster Replication Schema and Tables”

--ndb-log-empty-update

MySQL Server Options for NDB Cluster

--ndb-log-exclusive-reads

MySQL Server Options for NDB Cluster

--ndb-log-fail-terminate

MySQL Server Options for NDB Cluster
Section 18.1.4.2, “What is New in NDB Cluster 7.4”

--ndb-log-orig

MySQL Server Options for NDB Cluster
Section 18.6.4, “NDB Cluster Replication Schema and Tables”
NDB Cluster System Variables

--ndb-log-transaction-id

Section 17.1.4.4, “Binary Log Options and Variables”
MySQL Server Options for NDB Cluster
Section 18.6.11, “NDB Cluster Replication Conflict Resolution”
NDB Cluster System Variables

--ndb-log-update-as-write

Section 18.6.3, “Known Issues in NDB Cluster Replication”

Section 18.6.11, “NDB Cluster Replication Conflict Resolution”

--ndb-log-update-minimal

MySQL Server Options for NDB Cluster

--ndb-log-updated-only

Section 18.6.11, “NDB Cluster Replication Conflict Resolution”

--ndb-mgmd-host

MySQL Server Options for NDB Cluster
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

--ndb-nodegroup-map

Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”

--ndb-nodeid

MySQL Server Options for NDB Cluster
Section 18.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”
Section 18.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

--ndb-optimization-delay

MySQL Server Options for NDB Cluster
Section 13.7.2.4, “OPTIMIZE TABLE Statement”

--ndb-optimized-node-selection

Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

--ndb-transid-mysql-connection-map

MySQL Server Options for NDB Cluster
Section 21.31.2, “The INFORMATION_SCHEMA ndb_transid_mysql_connection_map Table”

--ndb-use-exact-count

Section 18.3.3.2, “Recommended Starting Configuration for NDB Cluster”

--ndb-wait-connected

MySQL Server Options for NDB Cluster

--ndb-wait-setup

MySQL Server Options for NDB Cluster

--ndbcluster

Section 18.3, “Configuration of NDB Cluster”
Section 18.2.2.1, “Installing NDB Cluster on Windows from a Binary Release”
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
MySQL Server Options for NDB Cluster
Section 18.5.9, “MySQL Server Usage for NDB Cluster”
Section 18.5.16.2, “NDB Cluster and MySQL Privileges”
Section 18.1.1, “NDB Cluster Core Concepts”
Section 18.5.14, “ndbinfo: The NDB Cluster Information Database”
Section 18.3.3.2, “Recommended Starting Configuration for NDB Cluster”

Section 13.7.5.17, “SHOW ENGINES Statement”
Section 21.7, “The INFORMATION_SCHEMA ENGINES Table”

--net-buffer-length

Section 4.5.1.1, “mysql Client Options”
Section 4.5.4, “mysqldump — A Database Backup Program”

net_retry_count

Section 17.2.1.1, “Monitoring Replication Main Threads”

net_write_timeout

Section 17.2.1.1, “Monitoring Replication Main Threads”

--network

Section 2.5.8.2, “More Topics on Deploying MySQL Server with Docker”

--nice

Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”

--no-auto-rehash

Section 4.5.1.1, “mysql Client Options”

--no-autocommit

Section 4.5.4, “mysqldump — A Database Backup Program”

--no-beep

Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”

--no-binlog

Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”

--no-browser

Section 18.4.25, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”

--no-contact

Section 18.4.28, “ndb_waiter — Wait for NDB Cluster to Reach a Given Status”

--no-create-db

Section 4.5.4, “mysqldump — A Database Backup Program”

--no-create-info

Section 7.4.5.4, “Dumping Table Definitions and Content Separately”
Section 4.5.4, “mysqldump — A Database Backup Program”

--no-data

Section 7.4.5.4, “Dumping Table Definitions and Content Separately”
Section 4.5.4, “mysqldump — A Database Backup Program”

--no-defaults

Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”
Section 4.7.3, “my_print_defaults — Display Options from Option Files”

Section 4.6.3.1, “myisamchk General Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”
Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 5.1.6, “Server Command Options”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 4.2.2.2, “Using Option Files”

--no-drop

Section 4.5.7, “`mysqlslap` — A Load Emulation Client”

--no-log

Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”

--no-nodeid-checks

Section 18.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”

--no-restore-disk-objects

Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”

--no-set-names

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--no-symlinks

Section 4.6.3.3, “myisamchk Repair Options”

--no-tablespaces

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--no-upgrade

Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”

--nodaemon

Section 18.4.4, “`ndb_mgmd` — The NDB Cluster Management Server Daemon”
Section 18.4.1, “`ndbd` — The NDB Cluster Data Node Daemon”

--nodata

Section 18.4.23, “`ndb_select_all` — Print Rows from an NDB Table”

--nodeid

Section 18.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”

Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Restoring to Fewer Nodes Than the Original
Restoring to More Nodes Than the Original

--nodes

Section 18.4.7, “ndb_config — Extract NDB Cluster Configuration Information”

--noindices

Section 4.6.10, “mysqlhotcopy — A Database Backup Program”

--nostart

Section 18.5.1, “Commands in the NDB Cluster Management Client”
Section 18.4.1, “ndbd — The NDB Cluster Data Node Daemon”

--not-started

Section 18.4.28, “ndb_waiter — Wait for NDB Cluster to Reach a Given Status”

--nowait-nodes

Section 18.5.7.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 18.3.3.6, “Defining NDB Cluster Data Nodes”
Section 18.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”
Section 18.4.28, “ndb_waiter — Wait for NDB Cluster to Reach a Given Status”
Section 18.4.1, “ndbd — The NDB Cluster Data Node Daemon”

--number-char-cols

Section 4.5.7, “mysqlslap — A Load Emulation Client”

--number-int-cols

Section 4.5.7, “mysqlslap — A Load Emulation Client”

--number-of-queries

Section 4.5.7, “mysqlslap — A Load Emulation Client”

--numeric-dump-file

Section 4.7.4, “resolve_stack_dump — Resolve Numeric Stack Trace Dump to Symbols”

O

[index top]

-O

Section 4.4.1, “comp_err — Compile MySQL Error Message File”
Section 2.9.7, “MySQL Source-Configuration Options”

-o

Section 23.6.1, “Compiling Programs with libmysqld”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.6.4, “myisamlog — Display MyISAM Log File Contents”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 18.4.23, “ndb_select_all — Print Rows from an NDB Table”
Section 18.4.25, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”

Section 8.12.2, “Optimizing Disk I/O”

--offset

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”

--old-style-user-limits

Section 5.1.6, “Server Command Options”
Section 6.2.13, “Setting Account Resource Limits”

--old_server

Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”
Section 4.6.10, “mysqlhotcopy — A Database Backup Program”

ON

Section 3.3.4.9, “Using More Than one Table”

--one-database

Section 4.5.1.1, “mysql Client Options”

--one-thread

Section 1.3, “What Is New in MySQL 5.6”

--only-print

Section 4.5.7, “mysqlslap — A Load Emulation Client”

--open-files-limit

Section B.3.2.17, “File Not Found and Similar Errors”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”
Section 5.1.7, “Server System Variables”

--opt

Section 8.5.5, “Bulk Data Loading for InnoDB Tables”
Section 4.5.4, “mysqldump — A Database Backup Program”

--opt_name

Section 4.2.2.2, “Using Option Files”

--optimize

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--optimizer-switch

Section 18.3.3.2, “Recommended Starting Configuration for NDB Cluster”

--order

Section 18.4.23, “ndb_select_all — Print Rows from an NDB Table”

--order-by-primary

Section 4.5.4, “mysqldump — A Database Backup Program”

--out-dir

Section 4.4.1, “comp_err — Compile MySQL Error Message File”

--out-file

Section 4.4.1, “comp_err — Compile MySQL Error Message File”

P

[index top]

-P

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 4.2.1, “Invoking MySQL Programs”
Section 16.2.2.1, “memcached Command-Line Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”
Section 4.4.4, “mysql_plugin — Configure MySQL Server Plugins”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.10, “mysqlhotcopy — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”
Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Section 5.1.6, “Server Command Options”

-p

Section 6.2.1, “Account User Names and Passwords”
Section 18.5.7.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 2.12, “Downgrading MySQL”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 4.6.1, “innochecksum — Offline InnoDB File Checksum Utility”
Section 4.2.1, “Invoking MySQL Programs”
Section 19.2.5, “KEY Partitioning”
Section 16.2.2.1, “memcached Command-Line Options”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.6.4, “myisamlog — Display MyISAM Log File Contents”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”
Section 4.4.4, “mysql_plugin — Configure MySQL Server Plugins”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.10, “mysqlhotcopy — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.4.10, “ndb_desc — Describe NDB Tables”
Section 18.4.21, “ndb_redo_log_reader — Check and Print Content of Cluster Redo Log”

Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Section 18.4.23, “ndb_select_all — Print Rows from an NDB Table”
Section 18.4.24, “ndb_select_count — Print Row Counts for NDB Tables”
Section 18.4.25, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”
Section 18.4.26, “ndb_show_tables — Display List of NDB Tables”
Section 18.4.2, “ndbinfo_select_all — Select From ndbinfo Tables”
Section B.3.2.5, “Password Fails When Entered Interactively”
Section 2.10.4, “Securing the Initial MySQL Accounts”
Section 2.3.4.7, “Starting MySQL as a Windows Service”
Section 2.3.4.5, “Starting MySQL from the Windows Command Line”
Section 2.3.4.8, “Testing The MySQL Installation”
Section 2.10.3, “Testing the Server”
Section 18.5.14.4, “The ndbinfo cluster_operations Table”
Section 18.5.14.22, “The ndbinfo server_operations Table”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 2.11, “Upgrading MySQL”
Section 2.11.8, “Upgrading MySQL on Windows”
Section 4.2.2.1, “Using Options on the Command Line”
Section 2.3.6, “Windows Postinstallation Procedures”

--pager

Section 4.5.1.2, “mysql Client Commands”
Section 4.5.1.1, “mysql Client Options”

--parallel-recover

Section 4.6.3.3, “myisamchk Repair Options”

--parallelism

Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Section 18.4.23, “ndb_select_all — Print Rows from an NDB Table”
Section 18.4.24, “ndb_select_count — Print Row Counts for NDB Tables”
Section 18.4.2, “ndbinfo_select_all — Select From ndbinfo Tables”

parallelism

Section 18.4.23, “ndb_select_all — Print Rows from an NDB Table”

--parsable

Section 18.4.26, “ndb_show_tables — Display List of NDB Tables”

--partition

Section 5.1.6, “Server Command Options”

--password

Section 6.2.1, “Account User Names and Passwords”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 7.3, “Example Backup and Recovery Strategy”
Section 4.2.1, “Invoking MySQL Programs”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”
Section 4.6.11, “mysql_convert_table_format — Convert Tables to Use a Given Storage Engine”
Section 4.6.14, “mysql_setpermission — Interactively Set Permissions in Grant Tables”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”

Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.10, “mysqlhotcopy — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.4.27, “ndb_size.pl — NDBCLUSTER Size Requirement Estimator”
Section B.3.2.5, “Password Fails When Entered Interactively”
Section 6.4.1.9, “Test Pluggable Authentication”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”
Section 4.2.2.1, “Using Options on the Command Line”

password

Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”
Section 4.2.2.2, “Using Option Files”

--performance-schema-consumer-consumer_name

Section 22.14, “Performance Schema Command Options”

--performance-schema-consumer-events-stages-current

Section 22.14, “Performance Schema Command Options”

--performance-schema-consumer-events-stages-history

Section 22.14, “Performance Schema Command Options”

--performance-schema-consumer-events-stages-history-long

Section 22.14, “Performance Schema Command Options”

--performance-schema-consumer-events-statements-current

Section 22.14, “Performance Schema Command Options”

--performance-schema-consumer-events-statements-history

Section 22.14, “Performance Schema Command Options”

--performance-schema-consumer-events-statements-history-long

Section 22.14, “Performance Schema Command Options”

--performance-schema-consumer-events-waits-current

Section 22.14, “Performance Schema Command Options”

--performance-schema-consumer-events-waits-history

Section 22.14, “Performance Schema Command Options”

--performance-schema-consumer-events-waits-history-long

Section 22.14, “Performance Schema Command Options”

--performance-schema-consumer-global-instrumentation

Section 22.14, “Performance Schema Command Options”

--performance-schema-consumer-statements-digest

Section 22.14, “Performance Schema Command Options”

--performance-schema-consumer-thread-instrumentation

Section 22.14, “Performance Schema Command Options”

--performance-schema-instrument

Section 22.14, “Performance Schema Command Options”
Section 22.3, “Performance Schema Startup Configuration”

--performance-schema-xxx

Section 5.1.6, “Server Command Options”

--performance_schema_max_mutex_classes

Section 22.7, “Performance Schema Status Monitoring”

--performance_schema_max_mutex_instances

Section 22.7, “Performance Schema Status Monitoring”

--pid-file

Section 17.1.4.4, “Binary Log Options and Variables”
Section 5.4.2.1, “Error Logging on Windows”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 5.7, “Running Multiple MySQL Instances on One Machine”
Section 5.1.6, “Server Command Options”
Section 5.4.4, “The Binary Log”

pid-file

Section 4.3.3, “`mysql.server` — MySQL Server Startup Script”

--pipe

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”
Section 2.3.4.8, “Testing The MySQL Installation”

--plan

Section 4.6.7, “`mysqlaccess` — Client for Checking Access Privileges”

--plugin

Section 5.1.6, “Server Command Options”

--plugin-dir

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 6.2.11, “Pluggable Authentication”

--plugin-ini

Section 4.4.4, “mysql_plugin — Configure MySQL Server Plugins”

--plugin-innodb-file-per-table

Section 5.1.6, “Server Command Options”

--plugin-load

Section 6.4.4.6, “Audit Log Reference”
Section 13.7.3.3, “INSTALL PLUGIN Statement”
Section 5.5.1, “Installing and Uninstalling Plugins”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.4.4, “mysql_plugin — Configure MySQL Server Plugins”
Section 5.1.6, “Server Command Options”

--plugin-load-add

Section 6.4.4.6, “Audit Log Reference”
Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 5.5.1, “Installing and Uninstalling Plugins”
Section 6.4.4.1, “Installing MySQL Enterprise Audit”
Section 6.4.1.6, “PAM Pluggable Authentication”
Section 6.4.3.1, “Password Validation Plugin Installation”
Section 6.4.3.2, “Password Validation Plugin Options and Variables”
Section 5.1.6, “Server Command Options”
Section 6.4.1.8, “Socket Peer-Credential Pluggable Authentication”
Section 6.4.1.9, “Test Pluggable Authentication”
Section 5.5.3.2, “Thread Pool Installation”
Section 6.4.1.7, “Windows Pluggable Authentication”

--plugin-sql-mode

Section 5.1.6, “Server Command Options”

--plugin-xxx

Section 5.1.6, “Server Command Options”

--plugin_dir

Section 2.9.7, “MySQL Source-Configuration Options”

--plugin_name

Section 5.5.1, “Installing and Uninstalling Plugins”
Section 4.4.4, “mysql_plugin — Configure MySQL Server Plugins”

--plugindir

Section 4.7.2, “mysql_config — Display Options for Compiling Clients”

--port

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 4.2.1, “Invoking MySQL Programs”

Section 4.5.1.1, “mysql Client Options”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.7.2, “mysql_config — Display Options for Compiling Clients”
Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”
Section 4.6.11, “mysql_convert_table_format — Convert Tables to Use a Given Storage Engine”
Section 4.6.14, “mysql_setpermission — Interactively Set Permissions in Grant Tables”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.10, “mysqlhotcopy — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.4.25, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”
Section 5.7, “Running Multiple MySQL Instances on One Machine”
Section 5.7.3, “Running Multiple MySQL Instances on Unix”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”
Section 5.7.4, “Using Client Programs in a Multiple-Server Environment”

port

Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”
Section 4.2.2.2, “Using Option Files”

--port-open-timeout

Section 5.1.6, “Server Command Options”

--post-query

Section 4.5.7, “mysqlslap — A Load Emulation Client”

--post-system

Section 4.5.7, “mysqlslap — A Load Emulation Client”

--pre-query

Section 4.5.7, “mysqlslap — A Load Emulation Client”

--pre-system

Section 4.5.7, “mysqlslap — A Load Emulation Client”

PREFIX

Section 18.2.1.4, “Building NDB Cluster from Source on Linux”

--prefix

Section 16.2.1, “Installing memcached”

--preserve-trailing-spaces

Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”

--preview

Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”

--print

Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”

--print-data

Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”

--print-defaults

Section 4.2.2.3, “Command-Line Options that Affect Option-File Handling”
Section 4.6.3.1, “myisamchk General Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.4, “mysql_plugin — Configure MySQL Server Plugins”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 5.1.6, “Server Command Options”
Section 2.11.9, “Upgrade Troubleshooting”

--print-full-config

Section 18.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”

--print-log

Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”

--print-meta

Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”

--print_*

Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”

--progress-frequency

Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”

--promote-attributes

Section 18.4.15, “ndb_move_data — NDB Data Copy Utility”
Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”

--prompt

Section 4.5.1.2, “mysql Client Commands”
Section 4.5.1.1, “mysql Client Options”

--protocol

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 4.2.5, “Connection Transport Protocols”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 5.7.3, “Running Multiple MySQL Instances on Unix”
Section 2.3.4.4, “Starting the Server for the First Time”
Section 2.3.4.8, “Testing The MySQL Installation”
Section 1.2.2, “The Main Features of MySQL”
Section 5.7.4, “Using Client Programs in a Multiple-Server Environment”

Q

[index top]

-Q

Section 4.5.4, “mysqldump — A Database Backup Program”

-q

Section 4.6.3.3, “myisamchk Repair Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.10, “mysqlhotcopy — A Database Backup Program”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 18.4.17, “ndb_print_file — Print NDB Disk Data File Contents”

--query

Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 18.4.14, “ndb_index_stat — NDB Index Statistics Utility”

--query-all

Section 18.4.7, “ndb_config — Extract NDB Cluster Configuration Information”

--query-cache-size

Section 8.11.5, “External Locking”

--quick

Section 4.6.3.6, “myisamchk Memory Usage”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.1, “mysql — The MySQL Command-Line Client”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section B.3.2.7, “Out of memory”
Section 7.6.1, “Using myisamchk for Crash Recovery”
Section 4.2.2.2, “Using Option Files”

--quiet

Section 4.6.10, “mysqlhotcopy — A Database Backup Program”

--quote-names

Section 4.5.4, “mysqldump — A Database Backup Program”

R

[index top]

-R

Section 16.2.2.1, “memcached Command-Line Options”
Section 16.2.4.1, “memcached General Statistics”
Section 7.6.4, “MyISAM Table Optimization”
Section 4.6.4, “myisamlog — Display MyISAM Log File Contents”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.3.4, “Other myisamchk Options”
Section 6.2.2, “Privileges Provided by MySQL”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

-r

Section 7.6.3, “How to Repair MyISAM Tables”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”
Section 16.2.2.1, “memcached Command-Line Options”
Section 4.6.3.2, “myisamchk Check Options”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.6.4, “myisamlog — Display MyISAM Log File Contents”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 18.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 18.4.10, “ndb_desc — Describe NDB Tables”
Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Section 5.1.6, “Server Command Options”

--random-passwords

Section 2.10.1, “Initializing the Data Directory”
Section 2.5.5, “Installing MySQL on Linux Using RPM Packages from Oracle”
Section 4.4.3, “mysql_install_db — Initialize MySQL Data Directory”
Section 1.3, “What Is New in MySQL 5.6”

--raw

Section 4.5.1.1, “mysql Client Options”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 17.1.3.3, “Using GTIDs for Failover and Scaleout”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”
Section 1.3, “What Is New in MySQL 5.6”

--read-from-remote-master

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.1.3.3, “Using GTIDs for Failover and Scaleout”

--read-from-remote-server

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 6.2.2, “Privileges Provided by MySQL”
Section 4.6.8.4, “Specifying the mysqlbinlog Server ID”

Section 17.1.3.3, “Using GTIDs for Failover and Scaleout”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”
Section 1.3, “What Is New in MySQL 5.6”

--read-only

Section 4.6.3.2, “myisamchk Check Options”

--real_table_name

Section 18.4.27, “`ndb_size.pl` — NDBCLUSTER Size Requirement Estimator”

--rebuild-indexes

Section 18.3.3.6, “Defining NDB Cluster Data Nodes”
Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”

--reconnect

Section 4.5.1.1, “mysql Client Options”

--record_log_pos

Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”

--recover

Section 4.6.3.2, “myisamchk Check Options”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.3.6, “myisamchk Memory Usage”
Section 4.6.3.3, “myisamchk Repair Options”

--regexp

Section 4.6.12, “`mysql_find_rows` — Extract SQL Statements from Files”
Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”

--relative

Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”

--relay-log-purge

Section 17.1.4.3, “Replica Server Options and Variables”

--relay-log-recovery

Section 17.2.2, “Relay Log and Replication Metadata Repositories”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.2.2.2, “Replication Metadata Repositories”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”
Section 5.1.15, “The Server Shutdown Process”

--relay-log-space-limit

Section 17.1.4.3, “Replica Server Options and Variables”

--relnotes

Section 4.6.7, “`mysqlaccess` — Client for Checking Access Privileges”

--reload

Section 18.5.7.2, “Adding NDB Cluster Data Nodes Online: Basic procedure”
Section 18.5.7.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 18.2.2.3, “Initial Startup of NDB Cluster on Windows”

Section 18.1.7.10, “Limitations Relating to Multiple NDB Cluster Nodes”
Section 18.3.3, “NDB Cluster Configuration Files”
Section 18.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”
Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”

--remove

Section 18.2.2.4, “Installing NDB Cluster Processes as Windows Services”
Section 18.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”
Section 18.4.1, “ndbd — The NDB Cluster Data Node Daemon”
Section 5.1.6, “Server Command Options”
Section 5.7.2.2, “Starting Multiple MySQL Instances as Windows Services”
Section 2.3.4.7, “Starting MySQL as a Windows Service”

--remove{

Section 18.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”

--repair

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--replace

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”

--replicate-*

Section 17.2.3, “How Servers Evaluate Replication Filtering Rules”
Section 17.1.4.3, “Replica Server Options and Variables”

--replicate-*-db

Section 17.1.4.3, “Replica Server Options and Variables”
Section 20.8, “Restrictions on Stored Programs”

--replicate-do-*

Section 18.6.3, “Known Issues in NDB Cluster Replication”

--replicate-do-db

Section 17.1.4.4, “Binary Log Options and Variables”
Section 17.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 17.2.3, “How Servers Evaluate Replication Filtering Rules”
Section 18.6.3, “Known Issues in NDB Cluster Replication”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.3.5, “Replicating Different Databases to Different Replicas”
Section 17.4.1.25, “Replication and Reserved Words”
Section 17.4.1.29, “Replication and Temporary Tables”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”
Section 5.4.4, “The Binary Log”
Section 17.1.2.2, “Usage of Row-Based Logging and Replication”

--replicate-do-table

Section 17.2.3.2, “Evaluation of Table-Level Replication Options”
Section 17.2.3.3, “Interactions Between Replication Filtering Options”
Section 18.6.3, “Known Issues in NDB Cluster Replication”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.4.1.25, “Replication and Reserved Words”
Section 17.4.1.15, “Replication and System Functions”

Section 17.4.1.29, “Replication and Temporary Tables”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”
Section 15.6, “The BLACKHOLE Storage Engine”
Section 17.1.2.2, “Usage of Row-Based Logging and Replication”

--replicate-ignore-*

Section 18.6.3, “Known Issues in NDB Cluster Replication”

--replicate-ignore-db

Section 17.1.4.4, “Binary Log Options and Variables”
Section 17.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 17.2.3, “How Servers Evaluate Replication Filtering Rules”
Section 17.2.3.3, “Interactions Between Replication Filtering Options”
Section 18.6.3, “Known Issues in NDB Cluster Replication”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.4.1.25, “Replication and Reserved Words”
Section 17.4.1.15, “Replication and System Functions”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”
Section 5.4.4, “The Binary Log”
Section 17.1.2.2, “Usage of Row-Based Logging and Replication”

--replicate-ignore-table

Section 17.2.3.2, “Evaluation of Table-Level Replication Options”
Section 18.6.3, “Known Issues in NDB Cluster Replication”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.4.1.25, “Replication and Reserved Words”
Section 17.4.1.29, “Replication and Temporary Tables”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”
Section 15.6, “The BLACKHOLE Storage Engine”
Section 17.1.2.2, “Usage of Row-Based Logging and Replication”

--replicate-rewrite-db

Section 17.2.3, “How Servers Evaluate Replication Filtering Rules”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.1.2.2, “Usage of Row-Based Logging and Replication”

--replicate-same-server-id

Section 13.4.2.1, “CHANGE MASTER TO Statement”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.1.4, “Replication and Binary Logging Options and Variables”

--replicate-wild-do-table

Section 17.2.3.2, “Evaluation of Table-Level Replication Options”
Section 17.2.3, “How Servers Evaluate Replication Filtering Rules”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.3.5, “Replicating Different Databases to Different Replicas”
Section 17.4.1.29, “Replication and Temporary Tables”
Section 20.8, “Restrictions on Stored Programs”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”

--replicate-wild-ignore-table

Section 17.2.3.2, “Evaluation of Table-Level Replication Options”
Section 18.6.3, “Known Issues in NDB Cluster Replication”
Section A.14, “MySQL 5.6 FAQ: Replication”
Section 17.1.4.3, “Replica Server Options and Variables”

Section 17.4.1.29, “Replication and Temporary Tables”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”

replication-ignore-table

Section 17.4.1.36, “Replication and Views”

--report-host

Section 17.1.5.1, “Checking Replication Status”
Section 13.7.5.34, “SHOW SLAVE HOSTS Statement”

--report-password

Section 17.1.4.2, “Replication Source Options and Variables”
Section 13.7.5.34, “SHOW SLAVE HOSTS Statement”

--report-port

Section 13.7.5.34, “SHOW SLAVE HOSTS Statement”

--report-user

Section 17.1.4.2, “Replication Source Options and Variables”
Section 13.7.5.34, “SHOW SLAVE HOSTS Statement”

--resetmaster

Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”

--resetslave

Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”

--restore-data

Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”
Restoring an NDB backup to a previous version of NDB Cluster
Restoring to Fewer Nodes Than the Original

--restore-epoch

Section 18.6.9, “NDB Cluster Backups With NDB Cluster Replication”
Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 18.6.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”

--restore-meta

Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”
Restoring an NDB backup to a later version of NDB Cluster
Restoring to More Nodes Than the Original

--restore-privilege-tables

Section 18.5.12, “Distributed Privileges Using Shared Grant Tables”
Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”

--result-file

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

--retries

Section 18.4.10, “`ndb_desc` — Describe NDB Tables”

--rewrite-database
Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”

--rhost
Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”

--rollback
Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”

--routines
Section 7.4.5.3, “Dumping Stored Programs”
Section 7.4.5.4, “Dumping Table Definitions and Content Separately”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 2.11.4, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

--rowid
Section 18.4.23, “ndb_select_all — Print Rows from an NDB Table”

--rows
Section 4.6.12, “mysql_find_rows — Extract SQL Statements from Files”
Section 18.4.7, “ndb_config — Extract NDB Cluster Configuration Information”

--rpl-recovery-rank
Section 1.3, “What Is New in MySQL 5.6”

--rpm
Section 4.4.3, “mysql_install_db — Initialize MySQL Data Directory”

S

[index top]

-S
Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.4.1, “comp_err — Compile MySQL Error Message File”
Section 4.2.1, “Invoking MySQL Programs”
Section 7.6.4, “MyISAM Table Optimization”
Section 4.5.1.2, “mysql Client Commands”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.10, “mysqlhotcopy — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.4.25, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”
Section 4.6.3.4, “Other myisamchk Options”

-s
Section 7.6.2, “How to Check MyISAM Tables for Errors”

Section 7.6.3, “How to Repair MyISAM Tables”
Section 4.6.1, “innochecksum — Offline InnoDB File Checksum Utility”
Section 2.2, “Installing MySQL on Unix/Linux Using Generic Binaries”
Section 16.2.2.1, “memcached Command-Line Options”
Section 4.7.3, “my_print_defaults — Display Options from Option Files”
Section 4.6.2, “myisam_ftdump — Display Full-Text Index information”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.6.16, “mysql_zap — Kill Processes That Match a Pattern”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Section 18.4.25, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”
Section 4.8.1, “perror — Display MySQL Error Message Information”
Section 4.8.2, “replace — A String-Replacement Utility”
Section 4.7.4, “resolve_stack_dump — Resolve Numeric Stack Trace Dump to Symbols”
Section 4.8.3, “resolveip — Resolve Host name to IP Address or Vice Versa”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”

--safe-mode

Section 1.3, “What Is New in MySQL 5.6”

--safe-recover

Section 4.6.3.1, “myisamchk General Options”
Section 4.6.3.6, “myisamchk Memory Usage”
Section 4.6.3.3, “myisamchk Repair Options”

--safe-updates

Section 4.5.1.2, “mysql Client Commands”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.6, “mysql Client Tips”
Section 5.1.7, “Server System Variables”

--safe-user-create

Section 5.1.6, “Server Command Options”

--savequeries

Section 18.4.27, “ndb_size.pl — NDBCLUSTER Size Requirement Estimator”

--secure-auth

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.4.1.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”

Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 6.1.2.4, “Password Hashing in MySQL”

--select-limit

Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.6, “mysql Client Tips”

--server-id

Section 18.6.2, “General Requirements for NDB Cluster Replication”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 13.7.5.34, “SHOW SLAVE HOSTS Statement”

--server-id-bits

MySQL Server Options for NDB Cluster
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
NDB Cluster System Variables

--server-log-file

Section 18.4.25, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”

--server-name

Section 18.4.25, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”

--server-public-key-path

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.5.1.1, “mysql Client Options”
Section 6.4.1.4, “SHA-256 Pluggable Authentication”

service-startup-timeout

Section 4.3.3, “mysql.server — MySQL Server Startup Script”

--set-auto-increment

Section 4.6.3.4, “Other myisamchk Options”

--set-charset

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.4, “mysqldump — A Database Backup Program”

--set-collation

Section 4.6.3.3, “myisamchk Repair Options”

--set-gtid-purged

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 17.1.3.3, “Using GTIDs for Failover and Scaleout”

--shared-memory-base-name

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 5.7, “Running Multiple MySQL Instances on One Machine”
Section 5.7.4, “Using Client Programs in a Multiple-Server Environment”

--short-form

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”

--show

Section 4.7.3, “my_print_defaults — Display Options from Option Files”

--show-slave-auth-info

Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.1.4.2, “Replication Source Options and Variables”
Section 13.7.5.34, “SHOW SLAVE HOSTS Statement”

--show-table-type

Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”

--show-temp-status

Section 18.4.26, “ndb_show_tables — Display List of NDB Tables”

--show-warnings

Section 4.5.1.1, “mysql Client Options”

--shutdown-timeout

Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”

--sigint-ignore

Section 4.5.1.1, “mysql Client Options”
Section 4.10, “Unix Signal Handling in MySQL”

--silent

Section 4.6.3.1, “myisamchk General Options”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 4.8.1, “perror — Display MySQL Error Message Information”
Section 4.8.3, “resolveip — Resolve Host name to IP Address or Vice Versa”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”

--single-transaction

Section 7.2, “Database Backup Methods”
Section 7.3.1, “Establishing a Backup Policy”
Section 14.18.1, “InnoDB Backup”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 14.8.10, “Purge Configuration”

--single-user

Section 18.4.28, “ndb_waiter — Wait for NDB Cluster to Reach a Given Status”

--skip

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.2.2.4, “Program Option Modifiers”
Section 5.1.6, “Server Command Options”

--skip-add-drop-table

Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-add-locks

Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-auto-rehash

Section 4.5.1.1, “mysql Client Options”
Section 14.21.3, “Troubleshooting InnoDB Data Dictionary Operations”

--skip-broken-objects

Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”

--skip-character-set-client-handshake

Section A.11, “MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 10.10.7.1, “The cp932 Character Set”

--skip-column-names

Section 4.5.1.1, “mysql Client Options”

--skip-comments

Section 4.5.1.1, “mysql Client Options”
Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-config-cache

Section 18.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”

--skip-database

Section 4.5.3, “mysqlcheck — A Table Maintenance Program”

--skip-disable-keys

Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-dump-date

Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-engine_name

Section 13.7.5.17, “SHOW ENGINES Statement”
Section 21.7, “The INFORMATION_SCHEMA ENGINES Table”

--skip-events

Section 7.4.5.3, “Dumping Stored Programs”

--skip-extended-insert

Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-external-locking

Section 8.11.5, “External Locking”
Section 8.14.3, “General Thread States”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 8.12.1, “System Factors”
Section B.3.3.3, “What to Do If MySQL Keeps Crashing”

--skip-federated

Section 17.3.3, “Using Replication with Different Source and Replica Storage Engines”

--skip-grant-tables

Section 13.7.3.1, “CREATE FUNCTION Statement for User-Defined Functions”
Section 18.5.12, “Distributed Privileges Using Shared Grant Tables”
Section 20.4.2, “Event Scheduler Configuration”
Section 13.7.6.3, “FLUSH Statement”
Section 13.7.3.3, “INSTALL PLUGIN Statement”
Section 5.5.1, “Installing and Uninstalling Plugins”
Section 5.6.1, “Installing and Uninstalling User-Defined Functions”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 6.2.11, “Pluggable Authentication”
Resetting the Root Password: Generic Instructions
Section 5.1.6, “Server Command Options”
Section 5.3, “The mysql System Database”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 4.2.2.1, “Using Options on the Command Line”
Section 6.2.8, “When Privilege Changes Take Effect”

--skip-gtids

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”

--skip-host-cache

Section 5.1.11.2, “DNS Lookups and the Host Cache”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”

--skip-innodb

Section 14.1.3, “Checking InnoDB Availability”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 5.5.1, “Installing and Uninstalling Plugins”
Section A.14, “MySQL 5.6 FAQ: Replication”
Section 5.1.6, “Server Command Options”
Section 14.1.5, “Turning Off InnoDB”
Section 1.3, “What Is New in MySQL 5.6”

--skip-innodb-adaptive-hash-index

Section 14.14, “InnoDB Startup Options and System Variables”

--skip-innodb-checksums

Section 14.14, “InnoDB Startup Options and System Variables”

--skip-innodb-doublewrite

Section 14.14, “InnoDB Startup Options and System Variables”

--skip-kill-mysqld

Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

--skip-line-numbers

Section 4.5.1.1, “mysql Client Options”

--skip-lock-tables

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--skip-log-bin

Section 17.1.4.4, “Binary Log Options and Variables”

--skip-name-resolve

Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”

--skip-named-commands

Section 4.5.1.1, “mysql Client Options”

--skip-ndbcluster

MySQL Server Options for NDB Cluster
Section 18.3.2.5, “NDB Cluster mysqld Option and Variable Reference”

--skip-new

Section 5.8.1, “Debugging a MySQL Server”
Section 13.7.2.4, “OPTIMIZE TABLE Statement”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”

--skip-nodegroup

Section 18.4.13, “`ndb_error_reporter` — NDB Error-Reporting Utility”

--skip-opt

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--skip-pager

Section 4.5.1.1, “mysql Client Options”

--skip-partition

Chapter 19, *Partitioning*
Section 5.1.6, “Server Command Options”

--skip-password

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”

Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”

--skip-plugin-innodb-file-per-table

Section 5.1.6, “Server Command Options”

--skip-plugin_name

Section 5.5.1, “Installing and Uninstalling Plugins”

--skip-quick

Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-quote-names

Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-reconnect

Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.6, “mysql Client Tips”

--skip-routines

Section 7.4.5.3, “Dumping Stored Programs”

--skip-safe-updates

Section 4.5.1.1, “mysql Client Options”

--skip-secure-auth

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”

--skip-set-charset

Section 4.5.4, “mysqldump — A Database Backup Program”

--skip-show-database

Section 6.2.2, “Privileges Provided by MySQL”
Section 5.1.6, “Server Command Options”
Section 13.7.5.15, “SHOW DATABASES Statement”
Section 1.8.5, “Supporters of MySQL”

--skip-slave-start

Section 17.1.1.9, “Introducing Additional Replicas to an Existing Replication Environment”
Section 18.6.9, “NDB Cluster Backups With NDB Cluster Replication”
Section 18.6.5, “Preparing the NDB Cluster for Replication”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.3.8, “Setting Up Replication to Use Encrypted Connections”
Section 17.1.3.2, “Setting Up Replication Using GTIDs”
Section 17.1.1.8, “Setting Up Replication with Existing Data”

Section 13.4.2.5, “START SLAVE Statement”
Section 18.6.6, “Starting NDB Cluster Replication (Single Replication Channel)”
Section 17.4.4, “Troubleshooting Replication”
Section 17.4.3, “Upgrading a Replication Setup”

--skip-ssl

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 5.1.6, “Server Command Options”

--skip-stack-trace

Section 5.8.1.4, “Debugging mysqld under gdb”
Section 5.1.6, “Server Command Options”

--skip-super-large-pages

Section 8.12.4.2, “Enabling Large Page Support”
Section 5.1.6, “Server Command Options”

--skip-symbolic-links

Section 13.1.17, “CREATE TABLE Statement”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 8.12.3.3, “Using Symbolic Links for Databases on Windows”
Section 8.12.3.2, “Using Symbolic Links for MyISAM Tables on Unix”

--skip-syslog

Section 5.4.2.3, “Error Logging to the System Log”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”

--skip-table-check

Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”

--skip-thread-priority

Section 1.3, “What Is New in MySQL 5.6”

--skip-triggers

Section 7.4.5.3, “Dumping Stored Programs”
Section 4.5.4, “`mysqldump` — A Database Backup Program”

--skip-tz-utc

Section 4.5.4, “`mysqldump` — A Database Backup Program”

--skip-unknown-objects

Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”

--skip-use-db

Section 4.6.12, “`mysql_find_rows` — Extract SQL Statements from Files”

--skip-version-check

Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”

--skip-warn

Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”

--skip-write-binlog

Section 11.2.5, “2-Digit YEAR(2) Limitations and Migrating to 4-Digit YEAR”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

--skip_grant_tables

Section 4.2.2.1, “Using Options on the Command Line”

--slave-skip-errors

Section 18.6.8, “Implementing Failover with NDB Cluster Replication”
Section 17.4.1.27, “Replica Errors During Replication”
Section 17.1.4.3, “Replica Server Options and Variables”

--slave-sql-verify-checksum

Section 17.1.4.4, “Binary Log Options and Variables”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 1.3, “What Is New in MySQL 5.6”

--sleep

Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”

--slow-start-timeout

Section 5.1.6, “Server Command Options”

--slow_query_log

Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.4.5, “The Slow Query Log”

--slow_query_log_file

Section 5.7, “Running Multiple MySQL Instances on One Machine”
Section 5.4.5, “The Slow Query Log”

--socket

Section B.3.2.2, “Can't connect to [local] MySQL server”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 2.5.8.3, “Deploying MySQL on Windows and Other Non-Linux Platforms with Docker”
Section B.3.3.6, “How to Protect or Change the MySQL Unix Socket File”
Section 4.2.1, “Invoking MySQL Programs”
Section 4.5.1.1, “mysql Client Options”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.7.2, “`mysql_config` — Display Options for Compiling Clients”
Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.6.11, “`mysql_convert_table_format` — Convert Tables to Use a Given Storage Engine”
Section 4.6.14, “`mysql_setpermission` — Interactively Set Permissions in Grant Tables”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”

Section 18.4.27, “ndb_size.pl — NDBCLUSTER Size Requirement Estimator”
Section 5.7, “Running Multiple MySQL Instances on One Machine”
Section 5.7.3, “Running Multiple MySQL Instances on Unix”
Section 5.1.6, “Server Command Options”
Section 2.3.4.8, “Testing The MySQL Installation”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 5.7.4, “Using Client Programs in a Multiple-Server Environment”

socket

Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”
Section 4.2.2.2, “Using Option Files”

--sort-index

Section 7.6.4, “MyISAM Table Optimization”
Section 4.6.3.4, “Other myisamchk Options”

--sort-records

Section 7.6.4, “MyISAM Table Optimization”
Section 4.6.3.4, “Other myisamchk Options”

--sort-recover

Section 4.6.3.1, “myisamchk General Options”
Section 4.6.3.6, “myisamchk Memory Usage”
Section 4.6.3.3, “myisamchk Repair Options”

--spassword

Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”

--sporadic-binlog-dump-fail

Section 17.1.4.4, “Binary Log Options and Variables”

--sql-mode

Chapter 12, *Functions and Operators*
Section A.3, “MySQL 5.6 FAQ: Server SQL Mode”
Section 5.1.6, “Server Command Options”
Section 5.1.10, “Server SQL Modes”

sql-mode

Section 5.1.10, “Server SQL Modes”

--srcdir

Section 4.4.3, “mysql_install_db — Initialize MySQL Data Directory”

--ssl

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”

Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”

--ssl*

Section 4.5.1.1, “mysql Client Options”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”

--ssl-ca

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.3.3.1, “Creating SSL Certificates and Keys Using openssl”
Section 13.7.1.4, “GRANT Statement”
Section 5.1.6, “Server Command Options”
Section 6.3.4, “SSL Library-Dependent Capabilities”

--ssl-capath

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 13.7.1.4, “GRANT Statement”
Section 5.1.6, “Server Command Options”
Section 17.3.8, “Setting Up Replication to Use Encrypted Connections”
Section 6.3.4, “SSL Library-Dependent Capabilities”

--ssl-cert

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.3.3.1, “Creating SSL Certificates and Keys Using openssl”
Section 13.7.1.4, “GRANT Statement”

--ssl-cipher

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”
Section 6.3.4, “SSL Library-Dependent Capabilities”

--ssl-crl

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.3.4, “SSL Library-Dependent Capabilities”

--ssl-crlpath

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.3.4, “SSL Library-Dependent Capabilities”

--ssl-key

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”

Section 6.3.3.1, “Creating SSL Certificates and Keys Using openssl”
Section 13.7.1.4, “GRANT Statement”

--ssl-mode

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”

--ssl-verify-server-cert

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”

--ssl-xxx

Section 13.4.2.1, “CHANGE MASTER TO Statement”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 2.9.6, “Configuring SSL Library Support”

--staging-tries

Section 18.4.15, “ndb_move_data — NDB Data Copy Utility”

--standalone

Section 5.8.1.2, “Creating Trace Files”
Section 5.1.6, “Server Command Options”
Section 2.3.4.5, “Starting MySQL from the Windows Command Line”

--start-datetime

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 7.5.1, “Point-in-Time Recovery Using Event Times”

--start-position

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 7.5.2, “Point-in-Time Recovery Using Event Positions”

--start_row

Section 4.6.12, “mysql_find_rows — Extract SQL Statements from Files”

--state-file

Section 4.4.1, “comp_err — Compile MySQL Error Message File”

--stats

Section 4.6.2, “myisam_ftdump — Display Full-Text Index information”

--status

Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”

--stop-datetime

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 7.5.1, “Point-in-Time Recovery Using Event Times”

--stop-never

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.6.8.4, “Specifying the mysqlbinlog Server ID”

Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

--stop-never-slave-server-id

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.6.8.4, “Specifying the mysqlbinlog Server ID”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

--stop-position

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 7.5.2, “Point-in-Time Recovery Using Event Positions”

--suffix

Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”

--super-large-pages

Section 8.12.4.2, “Enabling Large Page Support”
Section 5.1.6, “Server Command Options”

--superuser

Section 4.6.7, “`mysqlaccess` — Client for Checking Access Privileges”

--symbolic-links

Section 5.1.6, “Server Command Options”

--symbols-file

Section 4.7.4, “`resolve_stack_dump` — Resolve Numeric Stack Trace Dump to Symbols”

--sys-*

Section 18.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

--sys-check

Section 18.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

--sys-create

Section 18.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

--sys-create-if-not-exist

Section 18.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

sys-create-if-not-exist

Section 18.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

--sys-create-if-not-valid

Section 18.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

--sys-drop

Section 18.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

--sys-skip-events

Section 18.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

--sys-skip-tables

Section 18.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”

SYSCONFDIR
Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”
Section 4.2.2.2, “Using Option Files”

--sysdate-is-now
Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 12.7, “Date and Time Functions”
Section 17.4.1.15, “Replication and System Functions”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”

--syslog
Section 5.4.2.3, “Error Logging to the System Log”
Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”

--syslog-tag
Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”

--system
Section 18.4.7, “ndb_config — Extract NDB Cluster Configuration Information”

T

[index top]

-T
Section 4.4.1, “comp_err — Compile MySQL Error Message File”
Section 4.6.3.2, “myisamchk Check Options”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Section 5.1.6, “Server Command Options”

-t
Section 16.2.2.1, “memcached Command-Line Options”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.6.16, “mysql_zap — Kill Processes That Match a Pattern”
Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 18.4.9, “ndb_delete_all — Delete All Rows from an NDB Table”
Section 18.4.10, “ndb_desc — Describe NDB Tables”
Section 18.4.5, “ndb_mgm — The NDB Cluster Management Client”
Section 18.4.21, “ndb_redo_log_reader — Check and Print Content of Cluster Redo Log”
Section 18.4.23, “ndb_select_all — Print Rows from an NDB Table”
Section 18.4.26, “ndb_show_tables — Display List of NDB Tables”
Section 18.4.28, “ndb_waiter — Wait for NDB Cluster to Reach a Given Status”
Section 5.1.6, “Server Command Options”

--tab
Section 7.1, “Backup and Recovery Types”
Section 7.2, “Database Backup Methods”
Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Section 7.4, “Using mysqldump for Backups”

--table
Section 4.5.1.1, “mysql Client Options”
Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”
Section 18.4.10, “ndb_desc — Describe NDB Tables”

--table-cache
Section 1.3, “What Is New in MySQL 5.6”

--tables
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”

--tc-heuristic-recover
Section 5.1.6, “Server Command Options”

--tcp-ip
Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”

--tee
Section 4.5.1.2, “mysql Client Commands”
Section 4.5.1.1, “mysql Client Options”

--temp-pool
Section 5.1.6, “Server Command Options”

--test
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”

Text
Section 1.1, “About This Manual”

--thread_cache_size
Section 5.8.1.4, “Debugging mysqld under gdb”

--thread_handling
Section 1.3, “What Is New in MySQL 5.6”

--timeout
Section 18.4.28, “ndb_waiter — Wait for NDB Cluster to Reach a Given Status”

--timezone
Section 5.1.13, “MySQL Server Time Zone Support”
Section 4.3.2, “mysqld_safe — MySQL Server Startup Script”
Section 5.1.7, “Server System Variables”
Section B.3.3.7, “Time Zone Problems”

--tls-version

Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”

--tmpdir

Section B.3.2.12, “Can't create/write to file”
Section 4.6.3.6, “myisamchk Memory Usage”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”
Section 5.7, “Running Multiple MySQL Instances on One Machine”
Section 5.1.6, “Server Command Options”
Section 2.3.4.7, “Starting MySQL as a Windows Service”
Section B.3.3.5, “Where MySQL Stores Temporary Files”

tmpdir

Section 2.3, “Installing MySQL on Microsoft Windows”

--to-last-log

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.6.8.4, “Specifying the mysqlbinlog Server ID”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

--transaction-isolation

Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 13.3.6, “SET TRANSACTION Statement”
Section 14.7.2.1, “Transaction Isolation Levels”

--transaction-read-only

Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 13.3.6, “SET TRANSACTION Statement”

--transactional

Section 18.4.9, “`ndb_delete_all` — Delete All Rows from an NDB Table”

--triggers

Section 7.4.5.3, “Dumping Stored Programs”
Section 4.5.4, “`mysqldump` — A Database Backup Program”

--try-reconnect

Section 18.4.5, “`ndb_mgm` — The NDB Cluster Management Client”

--tupscan

Section 18.4.9, “`ndb_delete_all` — Delete All Rows from an NDB Table”
Section 18.4.23, “`ndb_select_all` — Print Rows from an NDB Table”

--type

Section 4.6.11, “`mysql_convert_table_format` — Convert Tables to Use a Given Storage Engine”
Section 18.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”
Section 18.4.26, “`ndb_show_tables` — Display List of NDB Tables”

--tz-utc

Section 4.5.4, “`mysqldump` — A Database Backup Program”

U

[index top]

-U

Section 16.2.2.1, “memcached Command-Line Options”
Section 4.6.3.2, “myisamchk Check Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”

-u

Section 6.2.1, “Account User Names and Passwords”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 2.12, “Downgrading MySQL”
Section 4.2.1, “Invoking MySQL Programs”
Section 16.2.2.1, “memcached Command-Line Options”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.6.4, “myisamlog — Display MyISAM Log File Contents”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.10, “mysqlhotcopy — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.4.10, “ndb_desc — Describe NDB Tables”
Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Section 18.4.26, “ndb_show_tables — Display List of NDB Tables”
Section 5.1.6, “Server Command Options”
Section 2.3.4.8, “Testing The MySQL Installation”
Section 2.10.3, “Testing the Server”
Section 2.11, “Upgrading MySQL”
Section 2.3.6, “Windows Postinstallation Procedures”

--unbuffered

Section 4.5.1.1, “mysql Client Options”

--unpack

Section 15.2.3, “MyISAM Table Storage Formats”
Section 4.6.3.3, “myisamchk Repair Options”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”

--unqualified

Section 18.4.10, “ndb_desc — Describe NDB Tables”
Section 18.4.26, “ndb_show_tables — Display List of NDB Tables”

--update

Section 18.4.14, “ndb_index_stat — NDB Index Statistics Utility”

--update-state

Section 7.6.3, “How to Repair MyISAM Tables”

Section 4.6.3.2, “myisamchk Check Options”
Section 15.2, “The MyISAM Storage Engine”

--upgrade

Section 2.3.3.4, “MySQL Installer Product Catalog and Dashboard”

--upgrade-system-tables

Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”

--usage

Section 18.4.7, “`ndb_config` — Extract NDB Cluster Configuration Information”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”

--use-frm

Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”

--use-https

Section 18.2.8.1, “NDB Cluster Auto-Installer Requirements”
Section 18.4.25, “`ndb_setup.py` — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”

--use-threads

Section 4.5.5, “`mysqlimport` — A Data Import Program”

--useHexFormat

Section 18.4.23, “`ndb_select_all` — Print Rows from an NDB Table”

--user

Section 6.2.1, “Account User Names and Passwords”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 7.3, “Example Backup and Recovery Strategy”
Section B.3.2.17, “File Not Found and Similar Errors”
Section 6.1.5, “How to Run MySQL as a Normal User”
Section 2.10.1, “Initializing the Data Directory”
Section 4.2.1, “Invoking MySQL Programs”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.6.11, “`mysql_convert_table_format` — Convert Tables to Use a Given Storage Engine”
Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”
Section 4.6.14, “`mysql_setpermission` — Interactively Set Permissions in Grant Tables”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.6.7, “`mysqlaccess` — Client for Checking Access Privileges”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 4.3.2, “`mysqld_safe` — MySQL Server Startup Script”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”
Section 18.5.16.3, “NDB Cluster and MySQL Security Procedures”
Section 18.4.27, “`ndb_size.pl` — NDBCLUSTER Size Requirement Estimator”

Section 4.2.2.6, “Option Defaults, Options Expecting Values, and the = Sign”
Resetting the Root Password: Unix and Unix-Like Systems
Section 5.1.6, “Server Command Options”
Section 6.4.1.8, “Socket Peer-Credential Pluggable Authentication”
Section 4.2.2, “Specifying Program Options”
Section 2.10.2, “Starting the Server”
Section 6.4.1.9, “Test Pluggable Authentication”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”
Section 4.2.2.2, “Using Option Files”

user

Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”
Section 4.2.2.2, “Using Option Files”

V

[index top]

-V

Section 4.4.1, “comp_err — Compile MySQL Error Message File”
Section 4.7.3, “my_print_defaults — Display Options from Option Files”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.4, “myisamlog — Display MyISAM Log File Contents”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”
Section 4.4.4, “mysql_plugin — Configure MySQL Server Plugins”
Section 4.6.15, “mysql_waitpid — Kill Process and Wait for Its Termination”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 4.8.1, “perror — Display MySQL Error Message Information”
Section 4.8.2, “replace — A String-Replacement Utility”
Section 4.7.4, “resolve_stack_dump — Resolve Numeric Stack Trace Dump to Symbols”
Section 4.8.3, “resolveip — Resolve Host name to IP Address or Vice Versa”
Section 5.1.6, “Server Command Options”
Section 4.2.2.1, “Using Options on the Command Line”

-v

Section 17.1.4.4, “Binary Log Options and Variables”
Section 7.6.2, “How to Check MyISAM Tables for Errors”
Section 4.6.1, “innochecksum — Offline InnoDB File Checksum Utility”
Section 16.2.2.1, “memcached Command-Line Options”
Section 16.2.2.9, “memcached Logs”
Section 4.7.3, “my_print_defaults — Display Options from Option Files”
Section 4.6.2, “myisam_ftdump — Display Full-Text Index information”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.4, “myisamlog — Display MyISAM Log File Contents”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”

Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”
Section 4.4.4, “mysql_plugin — Configure MySQL Server Plugins”
Section 4.6.15, “mysql_waitpid — Kill Process and Wait for Its Termination”
Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8.2, “mysqlbinlog Row Event Display”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.4.6, “ndb_blob_tool — Check and Repair BLOB and TEXT columns of NDB Cluster Tables”
Section 18.4.14, “ndb_index_stat — NDB Index Statistics Utility”
Section 18.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”
Section 18.4.17, “ndb_print_file — Print NDB Disk Data File Contents”
Section 18.4.1, “ndbd — The NDB Cluster Data Node Daemon”
Section 4.6.3.5, “Obtaining Table Information with myisamchk”
Section 4.8.1, “perror — Display MySQL Error Message Information”
Section 4.8.2, “replace — A String-Replacement Utility”
Section 5.1.6, “Server Command Options”
Section 4.2.2.1, “Using Options on the Command Line”

--validate-password

Section 6.4.3.1, “Password Validation Plugin Installation”
Section 6.4.3.2, “Password Validation Plugin Options and Variables”

--var_name

Section 14.14, “InnoDB Startup Options and System Variables”
Section 4.6.3.1, “myisamchk General Options”
Section 5.1.6, “Server Command Options”

--variable

Section 4.7.2, “mysql_config — Display Options for Compiling Clients”

--verbose

Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.1.4.4, “Binary Log Options and Variables”
Section 4.5.1.5, “Executing SQL Statements from a Text File”
Section 4.7.3, “my_print_defaults — Display Options from Option Files”
Section 4.6.2, “myisam_ftdump — Display Full-Text Index information”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”
Section 4.6.11, “mysql_convert_table_format — Convert Tables to Use a Given Storage Engine”
Section 4.4.3, “mysql_install_db — Initialize MySQL Data Directory”
Section 4.4.4, “mysql_plugin — Configure MySQL Server Plugins”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.6.15, “mysql_waitpid — Kill Process and Wait for Its Termination”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8.2, “mysqlbinlog Row Event Display”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.6.9, “mysqldumpslow — Summarize Slow Query Log Files”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.4.6, “ndb_blob_tool — Check and Repair BLOB and TEXT columns of NDB Cluster Tables”
Section 18.4.14, “ndb_index_stat — NDB Index Statistics Utility”
Section 18.4.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”
Section 18.4.15, “ndb_move_data — NDB Data Copy Utility”
Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Section 18.4.1, “ndbd — The NDB Cluster Data Node Daemon”
Section 4.6.3.4, “Other myisamchk Options”
Section 4.8.1, “perror — Display MySQL Error Message Information”
Section 5.1.6, “Server Command Options”
Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”
Section 17.1.2.2, “Usage of Row-Based Logging and Replication”
Section 4.2.2.2, “Using Option Files”
Section 4.2.2.1, “Using Options on the Command Line”

--verify-binlog-checksum

Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”

--version

Section 4.4.1, “comp_err — Compile MySQL Error Message File”
Section 4.7.3, “my_print_defaults — Display Options from Option Files”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.7.2, “mysql_config — Display Options for Compiling Clients”
Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”
Section 4.6.11, “mysql_convert_table_format — Convert Tables to Use a Given Storage Engine”
Section 4.4.4, “mysql_plugin — Configure MySQL Server Plugins”
Section 4.6.15, “mysql_waitpid — Kill Process and Wait for Its Termination”
Section 4.6.7, “mysqlaccess — Client for Checking Access Privileges”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.6.8, “mysqlbinlog — Utility for Processing Binary Log Files”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.3.4, “mysqld_multi — Manage Multiple MySQL Servers”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 18.4.29, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”
Section 4.8.1, “perror — Display MySQL Error Message Information”
Section 4.7.4, “resolve_stack_dump — Resolve Numeric Stack Trace Dump to Symbols”
Section 4.8.3, “resolveip — Resolve Host name to IP Address or Vice Versa”
Section 5.1.6, “Server Command Options”
Section 4.2.2.1, “Using Options on the Command Line”

--version-check

Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”

--vertical

Section 1.6, “How to Report Bugs or Problems”

Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”

W

[index top]

-W

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.5, “mysqlimport — A Data Import Program”
Section 4.5.6, “mysqlshow — Display Database, Table, and Column Information”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 5.1.6, “Server Command Options”

-w

Section 4.6.3.1, “myisamchk General Options”
Section 4.6.4, “myisamlog — Display MyISAM Log File Contents”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 18.4.28, “ndb_waiter — Wait for NDB Cluster to Reach a Given Status”

--wait

Section 4.6.3.1, “myisamchk General Options”
Section 4.6.5, “myisampack — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”

--wait-nodes

Section 18.4.28, “ndb_waiter — Wait for NDB Cluster to Reach a Given Status”

--warn

Section 4.6.6, “mysql_config_editor — MySQL Configuration Utility”

--where

Section 4.5.4, “mysqldump — A Database Backup Program”

--windows

Section 4.4.3, “mysql_install_db — Initialize MySQL Data Directory”

--with-libevent

Section 16.2.1, “Installing memcached”

WITH_BUNDLED_MEMCACHED

Section 2.9.7, “MySQL Source-Configuration Options”

WITH_CLASSPATH

Section 18.2.1.4, “Building NDB Cluster from Source on Linux”

Section 18.2.2.2, “Compiling and Installing NDB Cluster from Source on Windows”

WITH_DEBUG

Section 14.14, “InnoDB Startup Options and System Variables”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.5, “`myisampack` — Generate Compressed, Read-Only MyISAM Tables”
Section 4.5.1.1, “mysql Client Options”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.6.6, “`mysql_config_editor` — MySQL Configuration Utility”
Section 4.6.7, “`mysqlaccess` — Client for Checking Access Privileges”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.6.9, “`mysqldumpslow` — Summarize Slow Query Log Files”
Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”
Section 13.7.5.16, “SHOW ENGINE Statement”

WITH_EDITLINE

Section 2.9.7, “MySQL Source-Configuration Options”

WITH_GMOCK

Section 2.9.7, “MySQL Source-Configuration Options”

WITH_LIBEDIT

Section 2.9.7, “MySQL Source-Configuration Options”

WITH_NDB_JAVA

Section 18.2.1.4, “Building NDB Cluster from Source on Linux”
Section 18.2.2.2, “Compiling and Installing NDB Cluster from Source on Windows”

WITH_NDBCLUSTER

Section 18.2.1.4, “Building NDB Cluster from Source on Linux”
Section 18.2.2.2, “Compiling and Installing NDB Cluster from Source on Windows”

WITH_NDBCLUSTER_STORAGE_ENGINE

Section 18.2.1.4, “Building NDB Cluster from Source on Linux”
Section 18.2.2.2, “Compiling and Installing NDB Cluster from Source on Windows”
Section 2.9.7, “MySQL Source-Configuration Options”

WITH_NUMA

Section 2.9.7, “MySQL Source-Configuration Options”

WITH_PERFSCHEMA_STORAGE_ENGINE

Section 22.2, “Performance Schema Build Configuration”

WITH_SSL

Section 2.9.6, “Configuring SSL Library Support”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 2.9.2, “Source Installation Prerequisites”

WITH_ZLIB

Section 2.9.7, “MySQL Source-Configuration Options”

--write-binlog

Section 4.4.7, “mysql_upgrade — Check and Upgrade MySQL Tables”
Section 4.5.3, “mysqlcheck — A Table Maintenance Program”
Section 17.1.3.4, “Restrictions on Replication with GTIDs”

X

[index top]

-X

Section 4.5.1.2, “mysql Client Commands”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.4, “mysqldump — A Database Backup Program”

-x

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.7, “mysqlslap — A Load Emulation Client”
Section 18.4.23, “ndb_select_all — Print Rows from an NDB Table”

--xml

Section 13.2.7, “LOAD XML Statement”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 18.4.7, “ndb_config — Extract NDB Cluster Configuration Information”
Section 12.12, “XML Functions”

Y

[index top]

-Y

Section 4.5.4, “mysqldump — A Database Backup Program”

-y

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 4.5.7, “mysqlslap — A Load Emulation Client”

Z

[index top]

-z

Section 18.4.22, “ndb_restore — Restore an NDB Cluster Backup”
Section 18.4.23, “ndb_select_all — Print Rows from an NDB Table”

Privileges Index

A | C | D | E | F | G | I | L | N | P | R | S | T | U

A

[index top]

ALL

Section 13.7.1.4, “GRANT Statement”

Section 6.2.2, “Privileges Provided by MySQL”

ALL PRIVILEGES

Section 6.2.2, “Privileges Provided by MySQL”

ALTER

Section 13.1.1, “ALTER DATABASE Statement”
Section 13.1.7, “ALTER TABLE Statement”
Section 19.3.3, “Exchanging Partitions and Subpartitions with Tables”
Section 13.7.1.4, “GRANT Statement”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.1.32, “RENAME TABLE Statement”

ALTER ROUTINE

Section 13.1.3, “ALTER FUNCTION Statement”
Section 13.1.5, “ALTER PROCEDURE Statement”
Section 17.1.4.4, “Binary Log Options and Variables”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 13.1.26, “DROP PROCEDURE and DROP FUNCTION Statements”
Section 13.7.1.4, “GRANT Statement”
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.1.7, “Server System Variables”
Section 20.7, “Stored Program Binary Logging”
Section 20.2.2, “Stored Routines and MySQL Privileges”

C

[index top]

CREATE

Section 13.1.7, “ALTER TABLE Statement”
Section 13.1.10, “CREATE DATABASE Statement”
Section 13.1.17, “CREATE TABLE Statement”
Section 19.3.3, “Exchanging Partitions and Subpartitions with Tables”
Section 13.7.1.4, “GRANT Statement”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.1.32, “RENAME TABLE Statement”

CREATE ROUTINE

Section 17.1.4.4, “Binary Log Options and Variables”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 13.7.1.4, “GRANT Statement”
Section A.4, “MySQL 5.6 FAQ: Stored Procedures and Functions”
Section 6.2.2, “Privileges Provided by MySQL”
Section 20.7, “Stored Program Binary Logging”
Section 20.2.2, “Stored Routines and MySQL Privileges”

CREATE TABLESPACE

Section 13.7.1.4, “GRANT Statement”
Section 6.2.2, “Privileges Provided by MySQL”

CREATE TEMPORARY TABLES

Section 13.1.17.2, “CREATE TEMPORARY TABLE Statement”
Section 13.7.1.4, “GRANT Statement”
Section 6.2.2, “Privileges Provided by MySQL”

CREATE USER

Section 6.2.7, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 13.7.1.1, “ALTER USER Statement”
Section 6.2.9, “Assigning Account Passwords”
Section 13.7.1.2, “CREATE USER Statement”
Section 13.7.1.3, “DROP USER Statement”
Section 13.7.1.4, “GRANT Statement”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.1.5, “RENAME USER Statement”
Section 13.7.1.6, “REVOKE Statement”

CREATE VIEW

Section 13.1.9, “ALTER VIEW Statement”
Section 13.1.20, “CREATE VIEW Statement”
Section 13.7.1.4, “GRANT Statement”
Section 6.2.2, “Privileges Provided by MySQL”
Section 20.9, “Restrictions on Views”

D

[index top]

DELETE

Section 6.2.6, “Access Control, Stage 2: Request Verification”
Section 13.1.17, “CREATE TABLE Statement”
Section 13.2.2, “DELETE Statement”
Section 13.7.3.2, “DROP FUNCTION Statement for User-Defined Functions”
Section 13.7.1.3, “DROP USER Statement”
Section 13.1.17.5, “FOREIGN KEY Constraints”
Section 13.7.1.4, “GRANT Statement”
Section 5.5.1, “Installing and Uninstalling Plugins”
Section 5.6.1, “Installing and Uninstalling User-Defined Functions”
Section 18.5.16.2, “NDB Cluster and MySQL Privileges”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.2.8, “REPLACE Statement”
Section 15.7, “The MERGE Storage Engine”
Section 22.12.2.4, “The setup_objects Table”
Section 13.7.3.4, “UNINSTALL PLUGIN Statement”

DROP

Section 6.2, “Access Control and Account Management”
Section 13.1.7, “ALTER TABLE Statement”
Section 13.1.9, “ALTER VIEW Statement”
Section 13.1.20, “CREATE VIEW Statement”
Section 5.1.11.2, “DNS Lookups and the Host Cache”
Section 13.1.21, “DROP DATABASE Statement”
Section 13.1.28, “DROP TABLE Statement”
Section 13.1.31, “DROP VIEW Statement”
Section 19.3.3, “Exchanging Partitions and Subpartitions with Tables”
Section 13.7.1.4, “GRANT Statement”
Section 19.3.1, “Management of RANGE and LIST Partitions”
Section 6.5.1, “MySQL Enterprise Encryption Installation”
Section 22.11, “Performance Schema General Table Characteristics”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.1.32, “RENAME TABLE Statement”
Section 22.12.10.1, “The host_cache Table”
Section 13.1.33, “TRUNCATE TABLE Statement”

E

[index top]

EVENT

Section 13.1.2, “ALTER EVENT Statement”
Section 13.1.11, “CREATE EVENT Statement”
Section 13.1.22, “DROP EVENT Statement”
Section 20.4.1, “Event Scheduler Overview”
Section 20.4.3, “Event Syntax”
Section 13.7.1.4, “GRANT Statement”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.5.9, “SHOW CREATE EVENT Statement”
Section 13.7.5.19, “SHOW EVENTS Statement”
Section 20.4.6, “The Event Scheduler and MySQL Privileges”

EXECUTE

Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 13.1.26, “DROP PROCEDURE and DROP FUNCTION Statements”
Section 13.7.1.4, “GRANT Statement”
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.1.7, “Server System Variables”
Section 20.6, “Stored Object Access Control”
Section 20.2.2, “Stored Routines and MySQL Privileges”
Section 6.4.5.3, “Using MySQL Enterprise Firewall”

F

[index top]

FILE

Section 13.1.17, “CREATE TABLE Statement”
Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 13.7.1.4, “GRANT Statement”
Section 6.2.3, “Grant Tables”
Section 13.2.6, “LOAD DATA Statement”
Section 13.2.7, “LOAD XML Statement”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 6.5.2, “MySQL Enterprise Encryption Usage and Examples”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.2.9.1, “SELECT ... INTO Statement”
Section 5.1.7, “Server System Variables”
Section 12.8, “String Functions and Operators”
Section 11.3.4, “The BLOB and TEXT Types”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”

G

[index top]

GRANT OPTION

Section 13.7.1.4, “GRANT Statement”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.1.6, “REVOKE Statement”

Section 21.6, “The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table”
Section 21.21, “The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table”
Section 21.26, “The INFORMATION_SCHEMA TABLE_PRIVILEGES Table”
Section 21.28, “The INFORMATION_SCHEMA USER_PRIVILEGES Table”

I

[index top]

INDEX

Section 13.1.7, “ALTER TABLE Statement”
Section 13.7.1.4, “GRANT Statement”
Section 6.2.2, “Privileges Provided by MySQL”

INSERT

Section 6.2.6, “Access Control, Stage 2: Request Verification”
Section 13.1.7, “ALTER TABLE Statement”
Section 13.7.2.1, “ANALYZE TABLE Statement”
Section 6.2.9, “Assigning Account Passwords”
Section 13.7.3.1, “CREATE FUNCTION Statement for User-Defined Functions”
Section 13.7.1.2, “CREATE USER Statement”
Section 13.1.20, “CREATE VIEW Statement”
Section 19.3.3, “Exchanging Partitions and Subpartitions with Tables”
Section 13.1.17.5, “FOREIGN KEY Constraints”
Section 13.7.1.4, “GRANT Statement”
Section 13.2.5, “INSERT Statement”
Section 13.7.3.3, “INSTALL PLUGIN Statement”
Section 5.5.1, “Installing and Uninstalling Plugins”
Section 5.6.1, “Installing and Uninstalling User-Defined Functions”
Section 6.5.1, “MySQL Enterprise Encryption Installation”
Section 13.7.2.4, “OPTIMIZE TABLE Statement”
Section 15.11.1, “Pluggable Storage Engine Architecture”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.1.32, “RENAME TABLE Statement”
Section 13.7.2.5, “REPAIR TABLE Statement”
Section 13.2.8, “REPLACE Statement”
Section 5.1.6, “Server Command Options”
Section 20.6, “Stored Object Access Control”
Section 20.4.6, “The Event Scheduler and MySQL Privileges”
Section 22.12.2.4, “The setup_objects Table”

L

[index top]

LOCK TABLES

Section 13.7.6.3, “FLUSH Statement”
Section 13.7.1.4, “GRANT Statement”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”
Section 6.2.2, “Privileges Provided by MySQL”

N

[index top]

NDB_STORED_USER

Restoring an NDB backup to a previous version of NDB Cluster

P

[index top]

PROCESS

Section 8.14.1, “Accessing the Process List”
Section 6.2.7, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 5.1.11.1, “Connection Interfaces”
Section 14.17.2, “Enabling InnoDB Monitors”
Section 20.4.2, “Event Scheduler Configuration”
Section 13.7.1.4, “GRANT Statement”
Section 21.1, “Introduction”
Section 13.7.6.4, “KILL Statement”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 18.5.9, “MySQL Server Usage for NDB Cluster”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.5.16, “SHOW ENGINE Statement”
Section 13.7.5.30, “SHOW PROCESSLIST Statement”
Section 21.30.1, “The INFORMATION_SCHEMA INNODB_BUFFER_PAGE Table”
Section 21.30.2, “The INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU Table”
Section 21.30.3, “The INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS Table”
Section 21.30.4, “The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables”
Section 21.30.6, “The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables”
Section 21.30.5, “The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables”
Section 21.30.7, “The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table”
Section 21.30.8, “The INFORMATION_SCHEMA INNODB_FT_CONFIG Table”
Section 21.30.9, “The INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD Table”
Section 21.30.10, “The INFORMATION_SCHEMA INNODB_FT_DELETED Table”
Section 21.30.11, “The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table”
Section 21.30.12, “The INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE Table”
Section 21.30.14, “The INFORMATION_SCHEMA INNODB_LOCK_WAITS Table”
Section 21.30.13, “The INFORMATION_SCHEMA INNODB_LOCKS Table”
Section 21.30.15, “The INFORMATION_SCHEMA INNODB_METRICS Table”
Section 21.30.16, “The INFORMATION_SCHEMA INNODB_SYS_COLUMNS Table”
Section 21.30.17, “The INFORMATION_SCHEMA INNODB_SYS_DATAFILES Table”
Section 21.30.18, “The INFORMATION_SCHEMA INNODB_SYS_FIELDS Table”
Section 21.30.19, “The INFORMATION_SCHEMA INNODB_SYS_FOREIGN Table”
Section 21.30.20, “The INFORMATION_SCHEMA INNODB_SYS_FOREIGN_COLS Table”
Section 21.30.21, “The INFORMATION_SCHEMA INNODB_SYS_INDEXES Table”
Section 21.30.22, “The INFORMATION_SCHEMA INNODB_SYS_TABLES Table”
Section 21.30.23, “The INFORMATION_SCHEMA INNODB_SYS_TABLESPACES Table”
Section 21.30.24, “The INFORMATION_SCHEMA INNODB_SYS_TABLESTATS View”
Section 21.30.25, “The INFORMATION_SCHEMA INNODB_TRX Table”
Section 21.16, “The INFORMATION_SCHEMA PROCESSLIST Table”
Section 22.12.10.3, “The threads Table”
Section B.3.2.6, “Too many connections”

PROXY

Section 13.7.1.4, “GRANT Statement”
Section 6.2.3, “Grant Tables”
Section 6.4.1.6, “PAM Pluggable Authentication”

Section 6.2.2, “Privileges Provided by MySQL”
Section 6.2.12, “Proxy Users”
Section 2.10.4, “Securing the Initial MySQL Accounts”
Section 22.12.10.1, “The host_cache Table”
Section 6.4.1.7, “Windows Pluggable Authentication”

PROXY ... WITH GRANT OPTION

Section 6.2.12, “Proxy Users”

R

[index top]

REFERENCES

Section 13.1.17.5, “FOREIGN KEY Constraints”
Section 13.7.1.4, “GRANT Statement”
Section 6.2.2, “Privileges Provided by MySQL”

RELOAD

Section 6.2.6, “Access Control, Stage 2: Request Verification”
Section 6.2.7, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 5.1.11.2, “DNS Lookups and the Host Cache”
Section 12.14, “Encryption and Compression Functions”
Section 13.7.6.3, “FLUSH Statement”
Section 13.7.1.4, “GRANT Statement”
Section 6.2.3, “Grant Tables”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.6.6, “RESET Statement”
Section 5.4.7, “Server Log Maintenance”

REPLICATION CLIENT

Section 13.7.1.4, “GRANT Statement”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.5.2, “SHOW BINARY LOGS Statement”
Section 13.7.5.24, “SHOW MASTER STATUS Statement”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”

REPLICATION SLAVE

Section 17.1.1.3, “Creating a User for Replication”
Section 13.7.1.4, “GRANT Statement”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.3.8, “Setting Up Replication to Use Encrypted Connections”

S

[index top]

SELECT

Section 6.2, “Access Control and Account Management”
Section 6.2.6, “Access Control, Stage 2: Request Verification”
Section 13.7.2.1, “ANALYZE TABLE Statement”
Section 13.7.2.3, “CHECKSUM TABLE Statement”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”

Section 13.1.17.3, “CREATE TABLE ... LIKE Statement”
Section 13.1.17, “CREATE TABLE Statement”
Section 13.1.19, “CREATE TRIGGER Statement”
Section 13.1.20, “CREATE VIEW Statement”
Section 13.2.2, “DELETE Statement”
Section 13.7.6.3, “FLUSH Statement”
Section 13.1.17.5, “FOREIGN KEY Constraints”
Section 13.7.1.4, “GRANT Statement”
Section 13.2.5, “INSERT Statement”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 6.4.5.4, “MySQL Enterprise Firewall Reference”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”
Section 18.5.16.2, “NDB Cluster and MySQL Privileges”
Section 13.7.2.4, “OPTIMIZE TABLE Statement”
Section 22.11, “Performance Schema General Table Characteristics”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.2.5, “REPAIR TABLE Statement”
Section 20.9, “Restrictions on Views”
Section 13.7.5.14, “SHOW CREATE VIEW Statement”
Section 13.7.5.22, “SHOW GRANTS Statement”
Section 20.6, “Stored Object Access Control”
Section 20.4.6, “The Event Scheduler and MySQL Privileges”
Section 15.7, “The MERGE Storage Engine”
Section 22.12.10.3, “The threads Table”
Section 20.3.1, “Trigger Syntax and Examples”
Section 13.2.11, “UPDATE Statement”

SHOW DATABASES

Section 13.7.1.4, “GRANT Statement”
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.1.7, “Server System Variables”
Section 13.7.5.15, “SHOW DATABASES Statement”

SHOW VIEW

Section 13.8.2, “EXPLAIN Statement”
Section 13.7.1.4, “GRANT Statement”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 6.2.2, “Privileges Provided by MySQL”
Section 20.9, “Restrictions on Views”
Section 13.7.5.14, “SHOW CREATE VIEW Statement”
Section 21.29, “The INFORMATION_SCHEMA VIEWS Table”

SHUTDOWN

Section 6.2.6, “Access Control, Stage 2: Request Verification”
Section 13.7.1.4, “GRANT Statement”
Section 6.2.3, “Grant Tables”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.1.3.2, “Setting Up Replication Using GTIDs”
Section 5.1.15, “The Server Shutdown Process”

SUPER

Section 13.7.1, “Account Management Statements”
Section 13.1.3, “ALTER FUNCTION Statement”
Section 13.1.6, “ALTER SERVER Statement”
Section 13.7.1.1, “ALTER USER Statement”

Section 13.1.9, “ALTER VIEW Statement”
Section 6.2.9, “Assigning Account Passwords”
Section 17.1.4.4, “Binary Log Options and Variables”
Section 13.7.6.1, “BINLOG Statement”
Section 13.4.2.1, “CHANGE MASTER TO Statement”
Section 10.5, “Configuring Application Character Set and Collation”
Section 5.1.11.1, “Connection Interfaces”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 13.1.16, “CREATE SERVER Statement”
Section 13.1.19, “CREATE TRIGGER Statement”
Section 13.7.1.2, “CREATE USER Statement”
Section 5.1.11.2, “DNS Lookups and the Host Cache”
Section 13.1.27, “DROP SERVER Statement”
Section 13.7.1.3, “DROP USER Statement”
Section 12.14, “Encryption and Compression Functions”
Section 13.7.1.4, “GRANT Statement”
Section 17.1.1, “How to Set Up Replication”
Section 13.7.6.4, “KILL Statement”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section A.4, “MySQL 5.6 FAQ: Stored Procedures and Functions”
Section 6.4.5.4, “MySQL Enterprise Firewall Reference”
Section 5.1.13, “MySQL Server Time Zone Support”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.1.5, “RENAME USER Statement”
Section 13.7.1.6, “REVOKE Statement”
Section 17.3.9.2, “Semisynchronous Replication Installation and Configuration”
Section 5.1.10, “Server SQL Modes”
Section 5.1.7, “Server System Variables”
Section 13.7.1.7, “SET PASSWORD Statement”
Section 13.3.6, “SET TRANSACTION Statement”
Section 17.1.3.2, “Setting Up Replication Using GTIDs”
Section 13.7.5.2, “SHOW BINARY LOGS Statement”
Section 13.7.5.22, “SHOW GRANTS Statement”
Section 13.7.5.24, “SHOW MASTER STATUS Statement”
Section 13.7.5.30, “SHOW PROCESSLIST Statement”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”
Section 13.4.2.5, “START SLAVE Statement”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”
Section 13.4.2.6, “STOP SLAVE Statement”
Section 20.6, “Stored Object Access Control”
Section 20.7, “Stored Program Binary Logging”
Section 5.1.8.1, “System Variable Privileges”
Section B.3.2.6, “Too many connections”
Section 6.4.5.3, “Using MySQL Enterprise Firewall”

T

[index top]

TRIGGER

Section 13.1.19, “CREATE TRIGGER Statement”
Section 13.1.30, “DROP TRIGGER Statement”
Section 13.7.1.4, “GRANT Statement”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.5.13, “SHOW CREATE TRIGGER Statement”
Section 13.7.5.39, “SHOW TRIGGERS Statement”

Section 21.27, “The INFORMATION_SCHEMA TRIGGERS Table”

U

[index top]

UPDATE

Section 13.7.1.1, “ALTER USER Statement”
Section 6.2.9, “Assigning Account Passwords”
Section 13.1.17, “CREATE TABLE Statement”
Section 13.1.19, “CREATE TRIGGER Statement”
Section 13.1.17.5, “FOREIGN KEY Constraints”
Section 13.7.1.4, “GRANT Statement”
Section 13.2.5, “INSERT Statement”
Section 18.5.16.2, “NDB Cluster and MySQL Privileges”
Section 22.11, “Performance Schema General Table Characteristics”
Section 22.4, “Performance Schema Runtime Configuration”
Section 22.12.2, “Performance Schema Setup Tables”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.1.5, “RENAME USER Statement”
Section 13.7.1.6, “REVOKE Statement”
Section 13.7.1.7, “SET PASSWORD Statement”
Section 20.6, “Stored Object Access Control”
Section 15.7, “The MERGE Storage Engine”
Section 22.12.2.4, “The setup_objects Table”
Section 20.3.1, “Trigger Syntax and Examples”
Section 13.2.11, “UPDATE Statement”

USAGE

Section 13.7.1.4, “GRANT Statement”
Section 6.2.2, “Privileges Provided by MySQL”

SQL Modes Index

A | D | E | H | I | M | N | O | P | R | S | T

A

[index top]

ALLOW_INVALID_DATES

Section 11.2, “Date and Time Data Types”
Section 12.7, “Date and Time Functions”
Section B.3.4.2, “Problems Using DATE Columns”
Section 5.1.10, “Server SQL Modes”
Section 11.2.2, “The DATE, DATETIME, and TIMESTAMP Types”

ANSI

Section 9.2.5, “Function Name Parsing and Resolution”
Section 5.1.10, “Server SQL Modes”
Section 13.7.5.14, “SHOW CREATE VIEW Statement”
Section 21.29, “The INFORMATION_SCHEMA VIEWS Table”

ANSI_QUOTES

Section 13.1.17.5, “FOREIGN KEY Constraints”
Section 4.5.1.6, “mysql Client Tips”

Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 4.5.4, “mysqldump — A Database Backup Program”
Section 9.2, “Schema Object Names”
Section 5.1.10, “Server SQL Modes”
Section 9.1.1, “String Literals”

D

[index top]

DB2

Section 5.1.10, “Server SQL Modes”

E

[index top]

ERROR_FOR_DIVISION_BY_ZERO

Section 12.21.3, “Expression Handling”
Section 12.21.5, “Precision Math Examples”
Section 5.1.10, “Server SQL Modes”
Section 1.3, “What Is New in MySQL 5.6”

H

[index top]

HIGH_NOT_PRECEDENCE

Section 9.5, “Expressions”
Section 12.4.1, “Operator Precedence”
Section 5.1.10, “Server SQL Modes”

I

[index top]

IGNORE_SPACE

Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 9.2.5, “Function Name Parsing and Resolution”
Section 4.5.1.1, “mysql Client Options”
Section 5.1.10, “Server SQL Modes”

M

[index top]

MAXDB

Section 11.2.1, “Date and Time Data Type Syntax”
Section 5.1.10, “Server SQL Modes”
Section 11.2.2, “The DATE, DATETIME, and TIMESTAMP Types”

MSSQL

Section 5.1.10, “Server SQL Modes”

MYSQL323

Section 5.1.10, “Server SQL Modes”

MYSQL40
Section 5.1.10, “Server SQL Modes”

N

[index top]

NO_AUTO_CREATE_USER
Section 13.7.1.4, “GRANT Statement”
Section 5.1.10, “Server SQL Modes”

NO_AUTO_VALUE_ON_ZERO
Section 13.1.17, “CREATE TABLE Statement”
Section 11.1.6, “Numeric Type Attributes”
Section 5.1.10, “Server SQL Modes”
Section 3.6.9, “Using AUTO_INCREMENT”

NO_BACKSLASH_ESCAPES
Section 5.1.10, “Server SQL Modes”
Section 12.8.1, “String Comparison Functions and Operators”
Section 9.1.1, “String Literals”

NO_DIR_IN_CREATE
Section 13.1.17, “CREATE TABLE Statement”
Section 17.4.1.10, “Replication and DIRECTORY Table Options”
Section 17.4.1.35, “Replication and Variables”
Section 5.1.10, “Server SQL Modes”
Section 19.2.6, “Subpartitioning”
Section 5.4.4, “The Binary Log”

NO_ENGINE_SUBSTITUTION
Section 13.1.7, “ALTER TABLE Statement”
Section 13.1.17, “CREATE TABLE Statement”
Section 5.5.1, “Installing and Uninstalling Plugins”
Section A.3, “MySQL 5.6 FAQ: Server SQL Mode”
Section 5.1.6, “Server Command Options”
Section 5.1.10, “Server SQL Modes”
Section 5.1.7, “Server System Variables”
Section 15.1, “Setting the Storage Engine”
Section 5.1.2.2, “Using a Sample Default Server Configuration File”
Section 17.3.3, “Using Replication with Different Source and Replica Storage Engines”

NO_FIELD_OPTIONS
Section 5.1.10, “Server SQL Modes”

NO_KEY_OPTIONS
Section 5.1.10, “Server SQL Modes”

NO_TABLE_OPTIONS
Section 5.1.10, “Server SQL Modes”

NO_UNSIGNED_SUBTRACTION
Section 12.6.1, “Arithmetic Operators”
Section 12.11, “Cast Functions and Operators”
Section 11.1.1, “Numeric Data Type Syntax”
Section 11.1.7, “Out-of-Range and Overflow Handling”

Section 19.6, “Restrictions and Limitations on Partitioning”
Section 5.1.10, “Server SQL Modes”

NO_ZERO_DATE

Section 11.2.6, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”
Section 12.11, “Cast Functions and Operators”
Section 13.1.17, “CREATE TABLE Statement”
Section 11.2, “Date and Time Data Types”
Section 12.7, “Date and Time Functions”
Section B.3.4.2, “Problems Using DATE Columns”
Section 5.1.10, “Server SQL Modes”
Section 5.1.7, “Server System Variables”
Section 11.2.2, “The DATE, DATETIME, and TIMESTAMP Types”
Section 1.3, “What Is New in MySQL 5.6”

NO_ZERO_IN_DATE

Section 13.1.17, “CREATE TABLE Statement”
Section 11.2, “Date and Time Data Types”
Section B.3.4.2, “Problems Using DATE Columns”
Section 5.1.10, “Server SQL Modes”
Section 1.3, “What Is New in MySQL 5.6”

O

[index top]

ONLY_FULL_GROUP_BY

Section 3.3.4.8, “Counting Rows”
Section 12.19.2, “GROUP BY Modifiers”
Section 12.19.3, “MySQL Handling of GROUP BY”
Section 5.1.10, “Server SQL Modes”

ORACLE

Section 5.1.10, “Server SQL Modes”

P

[index top]

PAD_CHAR_TO_FULL_LENGTH

Section 5.1.10, “Server SQL Modes”
Section 11.3.1, “String Data Type Syntax”
Section 11.3.2, “The CHAR and VARCHAR Types”

PIPES_AS_CONCAT

Section 9.5, “Expressions”
Section 12.4.3, “Logical Operators”
Section 12.4.1, “Operator Precedence”
Section 5.1.10, “Server SQL Modes”

POSTGRESQL

Section 5.1.10, “Server SQL Modes”

R

[index top]

REAL_AS_FLOAT
Section 11.1.1, “Numeric Data Type Syntax”
Section 11.1, “Numeric Data Types”
Section 5.1.10, “Server SQL Modes”

S

[index top]

STRICT_ALL_TABLES
Section 1.7.3.3, “Constraints on Invalid Data”
Section 12.21.3, “Expression Handling”
Section A.3, “MySQL 5.6 FAQ: Server SQL Mode”
Section 5.1.10, “Server SQL Modes”
Section 17.4.3, “Upgrading a Replication Setup”
Section 1.3, “What Is New in MySQL 5.6”

STRICT_TRANS_TABLES
Section 1.7.3.3, “Constraints on Invalid Data”
Section 12.21.3, “Expression Handling”
Section A.3, “MySQL 5.6 FAQ: Server SQL Mode”
Section 5.1.10, “Server SQL Modes”
Section 17.4.3, “Upgrading a Replication Setup”
Section 5.1.2.2, “Using a Sample Default Server Configuration File”
Section 1.3, “What Is New in MySQL 5.6”

T

[index top]

TRADITIONAL
Section 11.2.6, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”
Section 12.21.3, “Expression Handling”
Section 13.2.6, “LOAD DATA Statement”
Section A.3, “MySQL 5.6 FAQ: Server SQL Mode”
Section 5.1.10, “Server SQL Modes”
Section 5.1.7, “Server System Variables”

Statement/Syntax Index

A | B | C | D | E | F | G | H | I | K | L | O | P | R | S | T | U | W | X

A

[index top]

ALTER DATABASE
Section 13.1.1, “ALTER DATABASE Statement”
Section 10.5, “Configuring Application Character Set and Collation”
Section 10.3.3, “Database Character Set and Collation”
Section 17.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 17.2.3, “How Servers Evaluate Replication Filtering Rules”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 17.1.4.3, “Replica Server Options and Variables”

ALTER EVENT

Section 13.1.2, “ALTER EVENT Statement”
Section 13.1.11, “CREATE EVENT Statement”
Section 20.4.4, “Event Metadata”
Section 20.4.1, “Event Scheduler Overview”
Section 20.4.3, “Event Syntax”
Section 12.16, “Information Functions”
Section 17.4.1.8, “Replication of CURRENT_USER()”
Section 17.4.1.16, “Replication of Invoked Features”
Section 20.8, “Restrictions on Stored Programs”
Section 13.7.5.19, “SHOW EVENTS Statement”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 20.6, “Stored Object Access Control”
Section 20.7, “Stored Program Binary Logging”
Section 20.4.6, “The Event Scheduler and MySQL Privileges”
Section 21.8, “The INFORMATION_SCHEMA EVENTS Table”

ALTER EVENT event_name ENABLE

Section 17.4.1.16, “Replication of Invoked Features”

ALTER FUNCTION

Section 13.1.3, “ALTER FUNCTION Statement”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 20.7, “Stored Program Binary Logging”
Section 20.2.1, “Stored Routine Syntax”

ALTER LOGFILE GROUP

Section 13.1.4, “ALTER LOGFILE GROUP Statement”
Section 18.3.3.6, “Defining NDB Cluster Data Nodes”
Section 18.1.7.8, “Issues Exclusive to NDB Cluster”
Section 18.5.10.1, “NDB Cluster Disk Data Objects”
Section 21.31.1, “The INFORMATION_SCHEMA FILES Table”

ALTER ONLINE TABLE

Section 18.5.7.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”

ALTER ONLINE TABLE ... REORGANIZE PARTITION

Section 18.5.7.2, “Adding NDB Cluster Data Nodes Online: Basic procedure”
Section 18.5.7.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 18.5.7.1, “Adding NDB Cluster Data Nodes Online: General Issues”
Section 18.5.1, “Commands in the NDB Cluster Management Client”

ALTER PROCEDURE

Section 13.1.5, “ALTER PROCEDURE Statement”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 20.7, “Stored Program Binary Logging”
Section 20.2.1, “Stored Routine Syntax”

ALTER SCHEMA

Section 13.1.1, “ALTER DATABASE Statement”

ALTER SERVER

Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.4.1.5, “Replication of CREATE SERVER, ALTER SERVER, and DROP SERVER”

Section 13.3.3, “Statements That Cause an Implicit Commit”

ALTER TABLE

Section 11.2.5, “2-Digit YEAR(2) Limitations and Migrating to 4-Digit YEAR”
Section 18.5.7.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 13.1.7.2, “ALTER TABLE Examples”
Section 13.1.7.1, “ALTER TABLE Partition Operations”
Section 13.1.7, “ALTER TABLE Statement”
Section 14.6.1.6, “AUTO_INCREMENT Handling in InnoDB”
Section 17.1.4.4, “Binary Log Options and Variables”
Section 2.11.3, “Changes in MySQL 5.6”
Section 13.7.2.2, “CHECK TABLE Statement”
Section 10.3.5, “Column Character Set and Collation”
Section 10.7, “Column Character Set Conversion”
Section 8.3.4, “Column Indexes”
Section 14.8.11, “Configuring Optimizer Statistics for InnoDB”
Configuring Optimizer Statistics Parameters for Individual Tables
Section 14.8.12, “Configuring the Merge Threshold for Index Pages”
Section 14.7.2.3, “Consistent Nonlocking Reads”
Section 14.20.5.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”
Section 10.9.8, “Converting Between 3-Byte and 4-Byte Unicode Character Sets”
Section 14.6.1.5, “Converting Tables from MyISAM to InnoDB”
Section 13.1.13, “CREATE INDEX Statement”
Section 13.1.17, “CREATE TABLE Statement”
Section 3.3.2, “Creating a Table”
Section 14.6.1.1, “Creating InnoDB Tables”
Section 11.4.4, “Creating Spatial Columns”
Section 11.4.8, “Creating Spatial Indexes”
Section 14.6.1.2, “Creating Tables Externally”
Section 18.3.3.6, “Defining NDB Cluster Data Nodes”
Section 14.12.4, “Defragmenting a Table”
Section 13.1.24, “DROP INDEX Statement”
Section 14.9.2, “Enabling Compression for a Table”
Section 14.8.11.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”
Section 19.3.3, “Exchanging Partitions and Subpartitions with Tables”
Section 8.8.2, “EXPLAIN Output Format”
Section 15.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 14.6.3.2, “File-Per-Table Tablespaces”
Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”
Section 14.21.2, “Forcing InnoDB Recovery”
Section 1.7.3.2, “FOREIGN KEY Constraints”
Section 13.1.17.5, “FOREIGN KEY Constraints”
Section 12.10, “Full-Text Search Functions”
Section 8.14.3, “General Thread States”
Section 13.7.1.4, “GRANT Statement”
Section 14.9.5, “How Compression Works for InnoDB Tables”
Section B.3.3.4, “How MySQL Handles a Full Disk”
Section 8.10.3.1, “How the Query Cache Operates”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 12.16, “Information Functions”
Section 18.2.3, “Initial Configuration of NDB Cluster”
Section 8.3.7, “InnoDB and MyISAM Index Statistics Collection”
Section 14.13, “InnoDB and Online DDL”
Section 14.6.2.3, “InnoDB FULLTEXT Indexes”
Section 14.11, “InnoDB Row Formats”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 13.2.5.3, “INSERT DELAYED Statement”
Section 18.1.7.8, “Issues Exclusive to NDB Cluster”

Section 13.7.6.4, “KILL Statement”
Section B.3.7, “Known Issues in MySQL”
Section 18.1.7.10, “Limitations Relating to Multiple NDB Cluster Nodes”
Section 18.1.7.2, “Limits and Differences of NDB Cluster from Standard MySQL Limits”
Section 8.4.6, “Limits on Table Size”
Section 18.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 19.3.4, “Maintenance of Partitions”
Section 19.3.2, “Management of HASH and KEY Partitions”
Section 19.3.1, “Management of RANGE and LIST Partitions”
Section 15.7.2, “MERGE Table Problems”
Section 14.6.1.4, “Moving or Copying InnoDB Tables”
Section 15.2.1, “MyISAM Startup Options”
Section 15.2.3, “MyISAM Table Storage Formats”
Section 4.6.3.1, “myisamchk General Options”
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
Section 1.7.1, “MySQL Extensions to Standard SQL”
MySQL Glossary
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 18.3.3.1, “NDB Cluster Configuration: Basic Example”
Section 18.5.10.1, “NDB Cluster Disk Data Objects”
Section 18.2.5, “NDB Cluster Example with Tables and Data”
Section 18.6.4, “NDB Cluster Replication Schema and Tables”
NDB Cluster System Variables
Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 18.1.7.1, “Noncompliance with SQL Syntax in NDB Cluster”
Section 14.13.5, “Online DDL Failure Conditions”
Section 14.13.6, “Online DDL Limitations”
Section 14.13.1, “Online DDL Operations”
Section 14.13.2, “Online DDL Performance and Concurrency”
Section 18.5.11, “Online Operations with ALTER TABLE in NDB Cluster”
Section 13.7.2.4, “OPTIMIZE TABLE Statement”
Section 11.1.7, “Out-of-Range and Overflow Handling”
Section 19.1, “Overview of Partitioning in MySQL”
Section 14.9.1, “Overview of Table Compression”
Section 19.3, “Partition Management”
Section 19.6.4, “Partitioning and Locking”
Section 19.6.1, “Partitioning Keys, Primary Keys, and Unique Keys”
Section 19.6.2, “Partitioning Limitations Relating to Storage Engines”
Section 6.2.2, “Privileges Provided by MySQL”
Section B.3.6.1, “Problems with ALTER TABLE”
Section 19.2.3.1, “RANGE COLUMNS partitioning”
Section 19.2.1, “RANGE Partitioning”
Section 2.11.10, “Rebuilding or Repairing Tables or Indexes”
Section 13.1.32, “RENAME TABLE Statement”
Section 17.4.1.1, “Replication and AUTO_INCREMENT”
Section 17.4.1.25, “Replication and Reserved Words”
Section 17.2.2.2, “Replication Metadata Repositories”
Replication with More Columns on Source or Replica
Section 19.6, “Restrictions and Limitations on Partitioning”
Section 20.9, “Restrictions on Views”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.6, “Server Command Options”
Section 5.1.10, “Server SQL Modes”
Section 5.1.7, “Server System Variables”
Section 5.4.4.2, “Setting The Binary Log Format”
Section 15.1, “Setting the Storage Engine”
Section 13.7.5.16, “SHOW ENGINE Statement”

Section 13.7.5.23, “SHOW INDEX Statement”
Section 13.7.5.41, “SHOW WARNINGS Statement”
Section 13.1.17.6, “Silent Column Specification Changes”
Section 14.13.4, “Simplifying DDL Statements with Online DDL”
Section 14.9.7, “SQL Compression Syntax Warnings and Errors”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 11.3.1, “String Data Type Syntax”
Section 10.3.4, “Table Character Set and Collation”
Section B.3.6.2, “TEMPORARY Table Problems”
Section 5.4.6, “The DDL Log”
Section 21.14, “The INFORMATION_SCHEMA PARTITIONS Table”
Section 21.22, “The INFORMATION_SCHEMA STATISTICS Table”
Section 15.3, “The MEMORY Storage Engine”
Section 15.2, “The MyISAM Storage Engine”
Section 5.4.5, “The Slow Query Log”
Section 14.21.3, “Troubleshooting InnoDB Data Dictionary Operations”
Section 18.1.7.6, “Unsupported or Missing Features in NDB Cluster”
Section 3.6.9, “Using AUTO_INCREMENT”
Section 17.3.3, “Using Replication with Different Source and Replica Storage Engines”
Section 8.12.3.2, “Using Symbolic Links for MyISAM Tables on Unix”
Section 14.10.2, “Verifying File Format Compatibility”
Section 1.3, “What Is New in MySQL 5.6”
Section B.3.3.3, “What to Do If MySQL Keeps Crashing”

ALTER TABLE ... ADD FOREIGN KEY

Section 13.1.7, “ALTER TABLE Statement”

ALTER TABLE ... ADD PARTITION

Section 19.3.1, “Management of RANGE and LIST Partitions”

ALTER TABLE ... ALGORITHM=COPY

Section 13.1.7, “ALTER TABLE Statement”
Section 13.1.17.5, “FOREIGN KEY Constraints”

ALTER TABLE ... ALGORITHM=INPLACE

Section 13.1.7, “ALTER TABLE Statement”
Section 13.1.17.5, “FOREIGN KEY Constraints”
Section 14.13.6, “Online DDL Limitations”

ALTER TABLE ... DISABLE KEYS

Section 13.2.6, “LOAD DATA Statement”

ALTER TABLE ... DISCARD TABLESPACE

Section 14.6.1.3, “Importing InnoDB Tables”
MySQL Glossary

ALTER TABLE ... DROP FOREIGN KEY

Section 13.1.7, “ALTER TABLE Statement”

ALTER TABLE ... DROP PARTITION

Section 17.4.1.23, “Replication and Partitioning”

ALTER TABLE ... ENABLE KEYS

Section 13.2.6, “LOAD DATA Statement”

ALTER TABLE ... ENGINE = MEMORY

Section 17.4.1.20, “Replication and MEMORY Tables”

ALTER TABLE ... ENGINE = NDB

Section 18.5.12, “Distributed Privileges Using Shared Grant Tables”

ALTER TABLE ... ENGINE=...

Section 1.3, “What Is New in MySQL 5.6”

ALTER TABLE ... ENGINE=INNODB

Section 2.11.3, “Changes in MySQL 5.6”
Section 1.3, “What Is New in MySQL 5.6”

ALTER TABLE ... ENGINE=InnoDB

Section 17.2.2.2, “Replication Metadata Repositories”

ALTER TABLE ... EXCHANGE PARTITION

Section 13.1.7.1, “ALTER TABLE Partition Operations”
Section 19.3.3, “Exchanging Partitions and Subpartitions with Tables”
Section 19.6.4, “Partitioning and Locking”
Section 1.3, “What Is New in MySQL 5.6”

ALTER TABLE ... FORCE

Section 13.7.2.4, “OPTIMIZE TABLE Statement”
Section 1.3, “What Is New in MySQL 5.6”

ALTER TABLE ... IMPORT TABLESPACE

Section 2.11.3, “Changes in MySQL 5.6”
Section 14.6.1.3, “Importing InnoDB Tables”
Section 14.6.1.4, “Moving or Copying InnoDB Tables”
MySQL Glossary

ALTER TABLE ... OPTIMIZE PARTITION

Section 19.3.4, “Maintenance of Partitions”
Section 19.6.2, “Partitioning Limitations Relating to Storage Engines”

ALTER TABLE ... PARTITION BY

Section 19.6.1, “Partitioning Keys, Primary Keys, and Unique Keys”

ALTER TABLE ... PARTITION BY ...

Section 19.3.1, “Management of RANGE and LIST Partitions”
Section 19.6, “Restrictions and Limitations on Partitioning”

ALTER TABLE ... RENAME

Section 8.12.3.2, “Using Symbolic Links for MyISAM Tables on Unix”

ALTER TABLE ... REORGANIZE PARTITION

Section 18.5.7.1, “Adding NDB Cluster Data Nodes Online: General Issues”

ALTER TABLE ... REPAIR PARTITION

Section 19.3.4, “Maintenance of Partitions”

ALTER TABLE ... TRUNCATE PARTITION

Section 19.3.4, “Maintenance of Partitions”
Section 19.3, “Partition Management”
Section 19.6.4, “Partitioning and Locking”

ALTER TABLE EXCHANGE PARTITION

Section 19.3.3, “Exchanging Partitions and Subpartitions with Tables”

ALTER TABLE mysql.ndb_apply_status ENGINE=MyISAM

Section 18.6.3, “Known Issues in NDB Cluster Replication”

ALTER TABLE t TRUNCATE PARTITION ()

Section 13.2.2, “DELETE Statement”

ALTER TABLE t3 DROP PARTITION p2

Section 5.4.6, “The DDL Log”

ALTER TABLE table_name ENGINE=InnoDB;

Section 14.1.4, “Testing and Benchmarking with InnoDB”

ALTER TABLE table_name REORGANIZE PARTITION

Section 18.5.7.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”

ALTER TABLE tbl_name ENGINE=INNODB

Section 13.1.7, “ALTER TABLE Statement”
Section 14.12.4, “Defragmenting a Table”

ALTER TABLE tbl_name FORCE

Section 13.1.7, “ALTER TABLE Statement”
Section 14.12.4, “Defragmenting a Table”

ALTER TABLE tbl_name TABLESPACE tablespace_name

MySQL Glossary

ALTER TABLESPACE

Section 13.1.8, “ALTER TABLESPACE Statement”
Section 13.1.18, “CREATE TABLESPACE Statement”
Section 18.3.3.6, “Defining NDB Cluster Data Nodes”
Section 18.1.7.8, “Issues Exclusive to NDB Cluster”
Section 18.5.10.1, “NDB Cluster Disk Data Objects”
Section 21.31.1, “The INFORMATION_SCHEMA FILES Table”

ALTER USER

Section 13.7.1.1, “ALTER USER Statement”
Section 6.2.3, “Grant Tables”
Section 6.4.1.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”
Section 6.2.2, “Privileges Provided by MySQL”
Section 6.2.10, “Server Handling of Expired Passwords”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 1.3, “What Is New in MySQL 5.6”

ALTER USER ... PASSWORD EXPIRE

Section 6.2.10, “Server Handling of Expired Passwords”

ALTER VIEW

Section 13.1.9, “ALTER VIEW Statement”
Section 12.16, “Information Functions”
Section 17.4.1.8, “Replication of CURRENT_USER()”
Section 20.8, “Restrictions on Stored Programs”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 20.6, “Stored Object Access Control”

Section 20.5.2, “View Processing Algorithms”
Section 20.5.1, “View Syntax”

ANALYZE TABLE

Section 13.1.7, “ALTER TABLE Statement”
Section 13.7.2.1, “ANALYZE TABLE Statement”
Configuring Automatic Statistics Calculation for Persistent Optimizer Statistics
Section 14.8.11.2, “Configuring Non-Persistent Optimizer Statistics Parameters”
Section 14.8.11, “Configuring Optimizer Statistics for InnoDB”
Configuring Optimizer Statistics Parameters for Individual Tables
Configuring the Number of Sampled Pages for InnoDB Optimizer Statistics
Section 13.1.13, “CREATE INDEX Statement”
Section 13.1.17, “CREATE TABLE Statement”
Section 14.8.11.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”
Section 8.8.2, “EXPLAIN Output Format”
Section 13.8.2, “EXPLAIN Statement”
Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”
Section 8.14.3, “General Thread States”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Including Delete-marked Records in Persistent Statistics Calculations
Section 8.3.7, “InnoDB and MyISAM Index Statistics Collection”
InnoDB Persistent Statistics Tables
InnoDB Persistent Statistics Tables Example
Section 14.14, “InnoDB Startup Options and System Variables”
Section B.3.7, “Known Issues in MySQL”
Section 19.3.4, “Maintenance of Partitions”
Section 15.7.2, “MERGE Table Problems”
Section 7.6, “MyISAM Table Maintenance and Crash Recovery”
Section 4.6.3.1, “myisamchk General Options”
Section 1.7.1, “MySQL Extensions to Standard SQL”
MySQL Glossary
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 18.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”
Section 8.6.1, “Optimizing MyISAM Queries”
Section 8.8.1, “Optimizing Queries with EXPLAIN”
Section 8.2.1, “Optimizing SELECT Statements”
Section 6.2.2, “Privileges Provided by MySQL”
Section 8.2.1.2, “Range Optimization”
Section 17.4.1.13, “Replication and FLUSH”
Section 19.6, “Restrictions and Limitations on Partitioning”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 13.7.5.23, “SHOW INDEX Statement”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 21.30.24, “The INFORMATION_SCHEMA INNODB_SYS_TABLESTATS View”
Section 21.22, “The INFORMATION_SCHEMA STATISTICS Table”
Section 5.4.5, “The Slow Query Log”

B

[index top]

BEGIN

Section 14.7.2.2, “autocommit, Commit, and Rollback”
Section 13.6.1, “BEGIN ... END Compound Statement”
Section 18.6.11, “NDB Cluster Replication Conflict Resolution”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.4.1.32, “Replication and Transactions”

Section 20.8, “Restrictions on Stored Programs”
Section 5.1.7, “Server System Variables”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 20.7, “Stored Program Binary Logging”

BEGIN ... END

Section 13.6.1, “BEGIN ... END Compound Statement”
Section 13.6.5.1, “CASE Statement”
Section 13.6, “Compound Statements”
Section 13.1.19, “CREATE TRIGGER Statement”
Section 13.6.6.1, “Cursor CLOSE Statement”
Section 13.6.6.3, “Cursor FETCH Statement”
Section 13.6.7.2, “DECLARE ... HANDLER Statement”
Section 13.6.3, “DECLARE Statement”
Section 20.1, “Defining Stored Programs”
Section 20.4.1, “Event Scheduler Overview”
Section 13.6.5.4, “LEAVE Statement”
Section 13.6.4.1, “Local Variable DECLARE Statement”
Section 13.6.4.2, “Local Variable Scope and Resolution”
Section 20.8, “Restrictions on Stored Programs”
Section 13.6.7.6, “Scope Rules for Handlers”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”
Section 13.6.2, “Statement Labels”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 20.3.1, “Trigger Syntax and Examples”

BINLOG

Section 13.7.6.1, “BINLOG Statement”
Section 4.6.8.2, “mysqlbinlog Row Event Display”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 6.2.2, “Privileges Provided by MySQL”

C

[index top]

CACHE INDEX

Section 13.7.6.2, “CACHE INDEX Statement”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 8.10.2.4, “Index Preloading”
Section 13.7.6.5, “LOAD INDEX INTO CACHE Statement”
Section 8.10.2.2, “Multiple Key Caches”
Section 19.6, “Restrictions and Limitations on Partitioning”
Section 13.3.3, “Statements That Cause an Implicit Commit”

CALL

Section 13.2.1, “CALL Statement”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 13.5, “Prepared Statements”
Section 20.6, “Stored Object Access Control”
Chapter 20, *Stored Objects*
Section 20.7, “Stored Program Binary Logging”
Section 20.2.1, “Stored Routine Syntax”
Section 20.3.1, “Trigger Syntax and Examples”

CALL p()

Section 13.6.7.4, “RESIGNAL Statement”

CALL stored_procedure()

Section 19.6.4, “Partitioning and Locking”

CASE

Section 8.10.4, “Caching of Prepared Statements and Stored Programs”
Section 13.6.5.1, “CASE Statement”
Section 12.5, “Flow Control Functions”
Section 13.6.5, “Flow Control Statements”

CHANGE MASTER TO

Section 6.2.9, “Assigning Account Passwords”
Section 17.3.1.2, “Backing Up Raw Data from a Replica”
Section 13.4.2.1, “CHANGE MASTER TO Statement”
Section 17.1.5.1, “Checking Replication Status”
Section 17.1.1.5, “Creating a Data Snapshot Using mysqldump”
Section 17.3.10, “Delayed Replication”
Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 13.7.1.4, “GRANT Statement”
Section 17.1.3.1, “GTID Concepts”
Section 18.6.8, “Implementing Failover with NDB Cluster Replication”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 18.6.9, “NDB Cluster Backups With NDB Cluster Replication”
Section 18.6.10, “NDB Cluster Replication: Bidirectional and Circular Replication”
Section 18.6.5, “Preparing the NDB Cluster for Replication”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.1.4, “Replication and Binary Logging Options and Variables”
Section 17.4.1.26, “Replication and Source or Replica Shutdowns”
Section 17.1, “Replication Configuration”
Section 17.2.2.2, “Replication Metadata Repositories”
Section 8.14.9, “Replication Replica Connection Thread States”
Section 8.14.7, “Replication Replica I/O Thread States”
Section 8.14.8, “Replication Replica SQL Thread States”
Section 13.4.2.3, “RESET SLAVE Statement”
Section 5.1.9, “Server Status Variables”
Section 17.1.1.10, “Setting the Source Configuration on the Replica”
Section 17.3.8, “Setting Up Replication to Use Encrypted Connections”
Section 17.1.3.2, “Setting Up Replication Using GTIDs”
Section 17.1.1.8, “Setting Up Replication with Existing Data”
Section 17.1.1.7, “Setting Up Replication with New Source and Replicas”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”
Section 13.4.2.5, “START SLAVE Statement”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 17.3.7, “Switching Sources During Failover”
Section 14.20.6, “The InnoDB memcached Plugin and Replication”
Section 17.1.2.2, “Usage of Row-Based Logging and Replication”
Section 17.1.3.3, “Using GTIDs for Failover and Scaleout”
Section 1.3, “What Is New in MySQL 5.6”

CHANGE REPLICATION FILTER REPLICATE_REWRITE_DB

Section 17.1.4.3, “Replica Server Options and Variables”

CHECK TABLE

Section 11.2.5, “2-Digit YEAR(2) Limitations and Migrating to 4-Digit YEAR”
Section 13.1.7.1, “ALTER TABLE Partition Operations”

Section 2.11.3, “Changes in MySQL 5.6”
Section 13.7.2.2, “CHECK TABLE Statement”
Section 15.2.4.1, “Corrupted MyISAM Tables”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 13.1.20, “CREATE VIEW Statement”
Section 8.11.5, “External Locking”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 1.6, “How to Report Bugs or Problems”
Section 4.6.1, “`innochecksum` — Offline InnoDB File Checksum Utility”
Section 14.18.2, “InnoDB Recovery”
Section 14.21, “InnoDB Troubleshooting”
Section 19.3.4, “Maintenance of Partitions”
Section 7.6, “MyISAM Table Maintenance and Crash Recovery”
Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”
Section A.6, “MySQL 5.6 FAQ: Views”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section B.3.2.8, “MySQL server has gone away”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 15.2.4.2, “Problems from Tables Not Being Closed Properly”
Section 2.11.10, “Rebuilding or Repairing Tables or Indexes”
Section 15.4.1, “Repairing and Checking CSV Tables”
Restoring an NDB backup to a later version of NDB Cluster
Section 19.6, “Restrictions and Limitations on Partitioning”
Section 13.6.6.5, “Restrictions on Server-Side Cursors”
Section 20.8, “Restrictions on Stored Programs”
Section 20.9, “Restrictions on Views”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 15.5, “The ARCHIVE Storage Engine”
Section 15.7, “The MERGE Storage Engine”
Section 5.4.5, “The Slow Query Log”

CHECK TABLE ... EXTENDED

Section 13.7.2.2, “CHECK TABLE Statement”

CHECK TABLE ... FOR UPGRADE

Section 2.11.3, “Changes in MySQL 5.6”
Section 13.7.2.2, “CHECK TABLE Statement”
Section 13.7.2.5, “REPAIR TABLE Statement”

CHECK TABLE QUICK

Section 13.7.2.2, “CHECK TABLE Statement”

CHECKSUM TABLE

Section 2.11.3, “Changes in MySQL 5.6”
Section 13.7.2.3, “CHECKSUM TABLE Statement”
Section 13.1.17, “CREATE TABLE Statement”
Section 17.4.1.4, “Replication and CHECKSUM TABLE”

CHECKSUM TABLE ... QUICK

Section 13.7.2.3, “CHECKSUM TABLE Statement”

COMMIT

Section 14.7.2.2, “autocommit, Commit, and Rollback”
Section 8.5.5, “Bulk Data Loading for InnoDB Tables”
Section 14.6.1.5, “Converting Tables from MyISAM to InnoDB”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 14.2, “InnoDB and the ACID Model”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
NDB Cluster System Variables
Section 17.1.1.4, “Obtaining the Replication Source Binary Log Coordinates”
Section 8.5.3, “Optimizing InnoDB Read-Only Transactions”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.4.1.32, “Replication and Transactions”
Section 13.3.4, “SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Statements”
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 20.7, “Stored Program Binary Logging”
Section 5.4.4, “The Binary Log”
Section 13.3, “Transactional and Locking Statements”
Section 20.3.1, “Trigger Syntax and Examples”

CREATE DATABASE

Section 7.1, “Backup and Recovery Types”
Section 10.5, “Configuring Application Character Set and Collation”
Section 7.4.5.2, “Copy a Database from one Server to Another”
Section 13.1.10, “CREATE DATABASE Statement”
Section 10.3.3, “Database Character Set and Collation”
Section 7.4.1, “Dumping Data in SQL Format with mysqldump”
Section 17.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 17.2.3, “How Servers Evaluate Replication Filtering Rules”
Section 9.2.3, “Identifier Case Sensitivity”
Section 18.1.7.8, “Issues Exclusive to NDB Cluster”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 18.6.9, “NDB Cluster Backups With NDB Cluster Replication”
Section 22.6, “Performance Schema Instrument Naming Conventions”
Section 7.4.2, “Reloading SQL-Format Backups”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 10.3.2, “Server Character Set and Collation”
Section 13.7.5.8, “SHOW CREATE DATABASE Statement”
Section 13.3.3, “Statements That Cause an Implicit Commit”

CREATE DATABASE dbx

Section 17.2.3, “How Servers Evaluate Replication Filtering Rules”

CREATE DATABASE IF NOT EXISTS

Section 17.4.1.6, “Replication of CREATE ... IF NOT EXISTS Statements”

CREATE EVENT

Section 13.1.2, “ALTER EVENT Statement”
Section 13.1.11, “CREATE EVENT Statement”
Section 20.4.4, “Event Metadata”
Section 20.4.3, “Event Syntax”

Section 9.5, “Expressions”
Section 12.16, “Information Functions”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 17.4.1.8, “Replication of CURRENT_USER()”
Section 17.4.1.16, “Replication of Invoked Features”
Section 20.8, “Restrictions on Stored Programs”
Section 5.1.6, “Server Command Options”
Section 13.7.5.9, “SHOW CREATE EVENT Statement”
Section 13.7.5.19, “SHOW EVENTS Statement”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Chapter 20, *Stored Objects*
Section 20.7, “Stored Program Binary Logging”
Section 20.4.6, “The Event Scheduler and MySQL Privileges”
Section 21.8, “The INFORMATION_SCHEMA EVENTS Table”
Section 5.3, “The mysql System Database”

CREATE EVENT IF NOT EXISTS

Section 17.4.1.6, “Replication of CREATE ... IF NOT EXISTS Statements”

CREATE FULLTEXT INDEX

Section 8.5.5, “Bulk Data Loading for InnoDB Tables”

CREATE FUNCTION

Section 13.1.3, “ALTER FUNCTION Statement”
Section 1.8.1, “Contributors to MySQL”
Section 13.1.12, “CREATE FUNCTION Statement”
Section 13.7.3.1, “CREATE FUNCTION Statement for User-Defined Functions”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 13.7.3.2, “DROP FUNCTION Statement for User-Defined Functions”
Section 9.2.5, “Function Name Parsing and Resolution”
Section 12.16, “Information Functions”
Section 5.6.1, “Installing and Uninstalling User-Defined Functions”
Section 6.5.1, “MySQL Enterprise Encryption Installation”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 5.6.2, “Obtaining User-Defined Function Information”
Section 17.4.1.8, “Replication of CURRENT_USER()”
Section 17.4.1.16, “Replication of Invoked Features”
Section 20.8, “Restrictions on Stored Programs”
Section 5.1.6, “Server Command Options”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Chapter 20, *Stored Objects*
Section 20.7, “Stored Program Binary Logging”
Section 20.2.1, “Stored Routine Syntax”
Section 5.3, “The mysql System Database”
Section 2.11.9, “Upgrade Troubleshooting”

CREATE INDEX

Section 8.3.4, “Column Indexes”
Section 14.8.12, “Configuring the Merge Threshold for Index Pages”
Section 14.20.5.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”
Section 13.1.13, “CREATE INDEX Statement”
Section 13.1.17, “CREATE TABLE Statement”
Section 11.4.8, “Creating Spatial Indexes”
Section 14.9.2, “Enabling Compression for a Table”
Section 12.10, “Full-Text Search Functions”
Section 14.9.5, “How Compression Works for InnoDB Tables”
Section 14.6.2.3, “InnoDB FULLTEXT Indexes”

Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 14.13.1, “Online DDL Operations”
Section 18.5.11, “Online Operations with ALTER TABLE in NDB Cluster”
Section 8.7, “Optimizing for MEMORY Tables”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 13.7.5.23, “SHOW INDEX Statement”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 21.22, “The INFORMATION_SCHEMA STATISTICS Table”
Section 5.4.5, “The Slow Query Log”
Section 18.1.7.6, “Unsupported or Missing Features in NDB Cluster”

CREATE LOGFILE GROUP

Section 13.1.4, “ALTER LOGFILE GROUP Statement”
Section 13.1.14, “CREATE LOGFILE GROUP Statement”
Section 13.1.18, “CREATE TABLESPACE Statement”
Section 18.3.3.6, “Defining NDB Cluster Data Nodes”
Section 18.1.7.8, “Issues Exclusive to NDB Cluster”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 18.5.10.1, “NDB Cluster Disk Data Objects”
Section 21.31.1, “The INFORMATION_SCHEMA FILES Table”
Section 18.5.14.20, “The ndbinfo resources Table”

CREATE OR REPLACE VIEW

Section 13.1.20, “CREATE VIEW Statement”
Section 20.9, “Restrictions on Views”

CREATE PROCEDURE

Section 13.1.5, “ALTER PROCEDURE Statement”
Section 13.2.1, “CALL Statement”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 12.16, “Information Functions”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 17.4.1.8, “Replication of CURRENT_USER()”
Section 17.4.1.16, “Replication of Invoked Features”
Section 20.8, “Restrictions on Stored Programs”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Chapter 20, *Stored Objects*
Section 20.7, “Stored Program Binary Logging”
Section 20.2.1, “Stored Routine Syntax”

CREATE SCHEMA

Section 13.1.10, “CREATE DATABASE Statement”
Section 18.1.7.8, “Issues Exclusive to NDB Cluster”
Section 18.6.9, “NDB Cluster Backups With NDB Cluster Replication”

CREATE SERVER

Section 13.1.6, “ALTER SERVER Statement”
Section 15.8.2.2, “Creating a FEDERATED Table Using CREATE SERVER”
Section 15.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 13.7.6.3, “FLUSH Statement”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 8.12.4.1, “How MySQL Uses Memory”
Section 15.8.2, “How to Create FEDERATED Tables”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.4.1.5, “Replication of CREATE SERVER, ALTER SERVER, and DROP SERVER”

Section 13.3.3, “Statements That Cause an Implicit Commit”

CREATE TABLE

Section 13.1.7.2, “ALTER TABLE Examples”
Section 13.1.7.1, “ALTER TABLE Partition Operations”
Section 13.1.7, “ALTER TABLE Statement”
Chapter 15, *Alternative Storage Engines*
Section 14.6.1.6, “AUTO_INCREMENT Handling in InnoDB”
Section 7.1, “Backup and Recovery Types”
Section 14.1.2, “Best Practices for InnoDB Tables”
Section 17.1.4.4, “Binary Log Options and Variables”
Section 2.11.3, “Changes in MySQL 5.6”
Section 10.3.5, “Column Character Set and Collation”
Section 8.3.4, “Column Indexes”
Section 14.8.11, “Configuring Optimizer Statistics for InnoDB”
Configuring Optimizer Statistics Parameters for Individual Tables
Section 14.8.12, “Configuring the Merge Threshold for Index Pages”
Section 14.6.1.5, “Converting Tables from MyISAM to InnoDB”
Section 13.1.11, “CREATE EVENT Statement”
Section 13.1.13, “CREATE INDEX Statement”
Section 13.1.16, “CREATE SERVER Statement”
Section 13.1.17.3, “CREATE TABLE ... LIKE Statement”
Section 13.1.17.4, “CREATE TABLE ... SELECT Statement”
Section 13.1.17, “CREATE TABLE Statement”
Section 13.1.17.2, “CREATE TEMPORARY TABLE Statement”
Section 15.8.2.1, “Creating a FEDERATED Table Using CONNECTION”
Section 3.3.2, “Creating a Table”
Section 14.6.1.1, “Creating InnoDB Tables”
Section 11.4.4, “Creating Spatial Columns”
Section 11.4.8, “Creating Spatial Indexes”
Section 7.2, “Database Backup Methods”
Section 10.3.3, “Database Character Set and Collation”
Section 18.3.3.6, “Defining NDB Cluster Data Nodes”
Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 14.9.2, “Enabling Compression for a Table”
Section 14.17.2, “Enabling InnoDB Monitors”
Section B.2, “Error Information Interfaces”
Section 14.8.11.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”
Section 14.6.3.2, “File-Per-Table Tablespaces”
Section 1.7.3.2, “FOREIGN KEY Constraints”
Section 13.1.17.5, “FOREIGN KEY Constraints”
Section 12.10, “Full-Text Search Functions”
Section 3.4, “Getting Information About Databases and Tables”
Section 19.2.4, “HASH Partitioning”
Section 13.8.3, “HELP Statement”
Section 14.9.5, “How Compression Works for InnoDB Tables”
Section 19.2.7, “How MySQL Partitioning Handles NULL”
Section 8.12.4.1, “How MySQL Uses Memory”
Section 9.2.3, “Identifier Case Sensitivity”
Section 14.6.1.3, “Importing InnoDB Tables”
Section 12.16, “Information Functions”
Section 18.2.3, “Initial Configuration of NDB Cluster”
Section 14.19, “InnoDB and MySQL Replication”
Section 14.6.2.3, “InnoDB FULLTEXT Indexes”
Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”
Section 14.11, “InnoDB Row Formats”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.21, “InnoDB Troubleshooting”

Section 17.2.3.3, “Interactions Between Replication Filtering Options”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 14.1, “Introduction to InnoDB”
Section 18.1.7.8, “Issues Exclusive to NDB Cluster”
Section 19.2.5, “KEY Partitioning”
Section 18.1.7.5, “Limits Associated with Database Objects in NDB Cluster”
Section 8.4.6, “Limits on Table Size”
Section 18.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 19.2.2, “LIST Partitioning”
Section 13.2.7, “LOAD XML Statement”
Section 3.3.3, “Loading Data into a Table”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 19.3.1, “Management of RANGE and LIST Partitions”
Section 15.2.3, “MyISAM Table Storage Formats”
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
Section 4.5.1.1, “mysql Client Options”
Section 1.7.1, “MySQL Extensions to Standard SQL”
MySQL Glossary
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 18.5.13, “NDB API Statistics Counters and Variables”
Section 18.3.3.1, “NDB Cluster Configuration: Basic Example”
Section 18.5.10.1, “NDB Cluster Disk Data Objects”
NDB Cluster System Variables
Section 18.4.6, “`ndb_blob_tool` — Check and Repair BLOB and TEXT columns of NDB Cluster Tables”
Section 18.1.7.1, “Noncompliance with SQL Syntax in NDB Cluster”
Section 14.13.1, “Online DDL Operations”
Section 18.5.11, “Online Operations with ALTER TABLE in NDB Cluster”
Section 8.4.1, “Optimizing Data Size”
Section 8.5.7, “Optimizing InnoDB DDL Operations”
Section 19.1, “Overview of Partitioning in MySQL”
Section 14.9.1, “Overview of Table Compression”
Section 19.3, “Partition Management”
Section 19.6.1, “Partitioning Keys, Primary Keys, and Unique Keys”
Section 19.6.3, “Partitioning Limitations Relating to Functions”
Section 19.6.2, “Partitioning Limitations Relating to Storage Engines”
Section 19.2, “Partitioning Types”
Section 6.2.2, “Privileges Provided by MySQL”
Section 19.2.3.1, “RANGE COLUMNS partitioning”
Section 19.2.1, “RANGE Partitioning”
Section 7.4.4, “Reloading Delimited-Text Format Backups”
Section 13.2.8, “REPLACE Statement”
Section 17.4.1.1, “Replication and AUTO_INCREMENT”
Section 17.4.1.3, “Replication and Character Sets”
Section 17.4.1.10, “Replication and DIRECTORY Table Options”
Section 17.4.1.14, “Replication and Fractional Seconds Support”
Section 17.4.1.15, “Replication and System Functions”
Section 17.4.1.7, “Replication of CREATE TABLE ... SELECT Statements”
Replication with More Columns on Source or Replica
Restoring an NDB backup to a previous version of NDB Cluster
Section 19.6, “Restrictions and Limitations on Partitioning”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.6, “Server Command Options”
Section 5.1.10, “Server SQL Modes”
Section 5.1.7, “Server System Variables”
Section 5.4.4.2, “Setting The Binary Log Format”
Section 15.1, “Setting the Storage Engine”

Section 13.7.5.6, “SHOW COLUMNS Statement”
Section 13.7.5.12, “SHOW CREATE TABLE Statement”
Section 13.7.5.16, “SHOW ENGINE Statement”
Section 13.7.5.23, “SHOW INDEX Statement”
Section 13.7.5.37, “SHOW TABLE STATUS Statement”
Section 13.7.5.41, “SHOW WARNINGS Statement”
Section 13.1.17.6, “Silent Column Specification Changes”
Section 14.9.7, “SQL Compression Syntax Warnings and Errors”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 11.3.1, “String Data Type Syntax”
Section 19.2.6, “Subpartitioning”
Section 10.3.4, “Table Character Set and Collation”
Section 14.1.4, “Testing and Benchmarking with InnoDB”
Section 15.5, “The ARCHIVE Storage Engine”
Section 11.3.5, “The ENUM Type”
Section 21.14, “The INFORMATION_SCHEMA PARTITIONS Table”
Section 21.22, “The INFORMATION_SCHEMA STATISTICS Table”
Section 21.23, “The INFORMATION_SCHEMA TABLES Table”
Section 15.3, “The MEMORY Storage Engine”
Section 15.2, “The MyISAM Storage Engine”
Section 13.2.10.1, “The Subquery as Scalar Operand”
Section 14.21.3, “Troubleshooting InnoDB Data Dictionary Operations”
Section 13.1.33, “TRUNCATE TABLE Statement”
Section 13.7.3.4, “UNINSTALL PLUGIN Statement”
Section 18.1.7.6, “Unsupported or Missing Features in NDB Cluster”
Section 3.6.9, “Using AUTO_INCREMENT”
Section 3.3.4.9, “Using More Than one Table”
Section 7.4, “Using mysqldump for Backups”
Section 17.3.3, “Using Replication with Different Source and Replica Storage Engines”
Section 8.12.3, “Using Symbolic Links”
Section 8.12.3.2, “Using Symbolic Links for MyISAM Tables on Unix”
Section 1.3, “What Is New in MySQL 5.6”
Section 2.3.7, “Windows Platform Restrictions”

CREATE TABLE ... KEY ()

Section 2.11.3, “Changes in MySQL 5.6”

CREATE TABLE ... LIKE

Section 13.1.17.3, “CREATE TABLE ... LIKE Statement”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 17.4.1.1, “Replication and AUTO_INCREMENT”
Section 15.7, “The MERGE Storage Engine”

CREATE TABLE ... ROW_FORMAT=COMPRESSED

Section 2.11.3, “Changes in MySQL 5.6”

CREATE TABLE ... SELECT

Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 12.11, “Cast Functions and Operators”
Section 14.7.2.3, “Consistent Nonlocking Reads”
Section 13.1.17.4, “CREATE TABLE ... SELECT Statement”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section B.3.7, “Known Issues in MySQL”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 17.4.1.7, “Replication of CREATE TABLE ... SELECT Statements”
Section 17.1.3.4, “Restrictions on Replication with GTIDs”
Section 1.7.2.1, “SELECT INTO TABLE Differences”

Section 13.2.9, “SELECT Statement”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 20.7, “Stored Program Binary Logging”

CREATE TABLE ... SELECT ...

Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 19.3.1, “Management of RANGE and LIST Partitions”

CREATE TABLE IF NOT EXISTS

Section 13.1.17.4, “CREATE TABLE ... SELECT Statement”
Section 17.4.1.6, “Replication of CREATE ... IF NOT EXISTS Statements”

CREATE TABLE IF NOT EXISTS ... LIKE

Section 17.4.1.6, “Replication of CREATE ... IF NOT EXISTS Statements”

CREATE TABLE IF NOT EXISTS ... SELECT

Section 13.1.17.4, “CREATE TABLE ... SELECT Statement”
Section 17.4.1.6, “Replication of CREATE ... IF NOT EXISTS Statements”

CREATE TABLE new_table SELECT ... FROM old_table ...

Section 13.1.17.4, “CREATE TABLE ... SELECT Statement”
Section 13.2.9, “SELECT Statement”

CREATE TABLE tbl_name ... TABLESPACE tablespace_name

MySQL Glossary

CREATE TABLE...AS SELECT

Section 8.2.1, “Optimizing SELECT Statements”

CREATE TABLESPACE

Section 13.1.8, “ALTER TABLESPACE Statement”
Section 13.1.17, “CREATE TABLE Statement”
Section 13.1.18, “CREATE TABLESPACE Statement”
Section 18.3.3.6, “Defining NDB Cluster Data Nodes”
Section 13.1.29, “DROP TABLESPACE Statement”
Section 18.1.7.8, “Issues Exclusive to NDB Cluster”
MySQL Glossary
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 18.5.10.1, “NDB Cluster Disk Data Objects”
Section 21.31.1, “The INFORMATION_SCHEMA FILES Table”

CREATE TEMPORARY TABLE

Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 13.7.1.4, “GRANT Statement”
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.1.3.4, “Restrictions on Replication with GTIDs”
Section 5.1.7, “Server System Variables”
Section 15.1, “Setting the Storage Engine”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section B.3.6.2, “TEMPORARY Table Problems”

CREATE TRIGGER

Section 13.1.19, “CREATE TRIGGER Statement”
Section 12.16, “Information Functions”

Section A.5, “MySQL 5.6 FAQ: Triggers”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”
Section 17.4.1.8, “Replication of CURRENT_USER()”
Section 17.4.1.16, “Replication of Invoked Features”
Section 20.8, “Restrictions on Stored Programs”
Section 13.7.5.13, “SHOW CREATE TRIGGER Statement”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Chapter 20, *Stored Objects*
Section 20.7, “Stored Program Binary Logging”
Section 20.3.1, “Trigger Syntax and Examples”

CREATE UNDO TABLESPACE

MySQL Glossary

CREATE USER

Section 6.2, “Access Control and Account Management”
Section 6.2.1, “Account User Names and Passwords”
Section 6.2.7, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 6.2.9, “Assigning Account Passwords”
Section 5.1.12.3, “Connecting Using the IPv6 Local Host Address”
Section 13.7.1.2, “CREATE USER Statement”
Section 17.1.1.3, “Creating a User for Replication”
Section 12.14, “Encryption and Compression Functions”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 13.7.6.3, “FLUSH Statement”
Section 13.7.1.4, “GRANT Statement”
Section 6.2.3, “Grant Tables”
Section 8.12.4.1, “How MySQL Uses Memory”
Section 5.1.12, “IPv6 Support”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 18.5.16.2, “NDB Cluster and MySQL Privileges”
Section 6.4.1.6, “PAM Pluggable Authentication”
Section 6.1.2.4, “Password Hashing in MySQL”
Section 6.1.2.3, “Passwords and Logging”
Section 6.2.2, “Privileges Provided by MySQL”
Section 2.10.1.1, “Problems Running mysql_install_db”
Section 6.2.12, “Proxy Users”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 6.4.1.4, “SHA-256 Pluggable Authentication”
Section 6.2.4, “Specifying Account Names”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 6.4.3, “The Password Validation Plugin”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 6.4.5.3, “Using MySQL Enterprise Firewall”
Section 1.3, “What Is New in MySQL 5.6”
Section 6.4.1.7, “Windows Pluggable Authentication”

CREATE VIEW

Section 13.1.9, “ALTER VIEW Statement”
Section 13.1.20, “CREATE VIEW Statement”
Section 8.14.3, “General Thread States”
Section 9.2.1, “Identifier Length Limits”
Section 12.16, “Information Functions”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 19.6.4, “Partitioning and Locking”

Section 6.2.2, “Privileges Provided by MySQL”
Section 17.4.1.8, “Replication of CURRENT_USER()”
Section 20.9, “Restrictions on Views”
Section 13.7.5.14, “SHOW CREATE VIEW Statement”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Chapter 20, *Stored Objects*
Section 21.29, “The INFORMATION_SCHEMA VIEWS Table”
Section 20.5.3, “Updatable and Insertable Views”
Section 20.5.2, “View Processing Algorithms”
Section 20.5.1, “View Syntax”

D

[index top]

DEALLOCATE PREPARE

Section 13.5.3, “DEALLOCATE PREPARE Statement”
Section 13.5.1, “PREPARE Statement”
Section 13.5, “Prepared Statements”
Section 20.8, “Restrictions on Stored Programs”
Section 5.1.9, “Server Status Variables”

DECLARE

Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 13.6.3, “DECLARE Statement”
Section 13.6.7.3, “GET DIAGNOSTICS Statement”
Section 13.6.7.5, “SIGNAL Statement”
Section 13.6.4, “Variables in Stored Programs”

DECLARE ... CONDITION

Section 13.6.7, “Condition Handling”
Section 13.6.7.1, “DECLARE ... CONDITION Statement”
Section 13.6.7.2, “DECLARE ... HANDLER Statement”
Section 13.6.7.5, “SIGNAL Statement”

DECLARE ... HANDLER

Section 13.6.7, “Condition Handling”
Section 13.6.7.1, “DECLARE ... CONDITION Statement”
Section 13.6.7.2, “DECLARE ... HANDLER Statement”
Section 13.6.7.5, “SIGNAL Statement”

DELETE

Section 6.2, “Access Control and Account Management”
Section 14.20.5.5, “Adapting DML Statements to memcached Operations”
Section 6.2.7, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 13.1.7.1, “ALTER TABLE Partition Operations”
Section 14.1.2, “Best Practices for InnoDB Tables”
Section 17.1.4.4, “Binary Log Options and Variables”
Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 14.5.2, “Change Buffer”
Section 14.9.6, “Compression for OLTP Workloads”
Section 14.7.2.3, “Consistent Nonlocking Reads”
Section 14.6.1.5, “Converting Tables from MyISAM to InnoDB”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 13.1.19, “CREATE TRIGGER Statement”
Section 13.1.20, “CREATE VIEW Statement”

Section 13.2.2, “DELETE Statement”
Section B.3.4.6, “Deleting Rows from Related Tables”
Section 8.8.2, “EXPLAIN Output Format”
Section 13.8.2, “EXPLAIN Statement”
Section 8.8.3, “Extended EXPLAIN Output Format”
Section 15.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 14.21.2, “Forcing InnoDB Recovery”
Section 13.1.17.5, “FOREIGN KEY Constraints”
Section 12.10.5, “Full-Text Restrictions”
Chapter 12, *Functions and Operators*
Section 8.14.3, “General Thread States”
Section 13.7.1.4, “GRANT Statement”
Section 6.2.3, “Grant Tables”
Section 8.10.3.1, “How the Query Cache Operates”
Section 12.16, “Information Functions”
Section 14.19, “InnoDB and MySQL Replication”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 8.11.1, “Internal Locking Methods”
Section 21.1, “Introduction”
Section 13.2.9.2, “JOIN Clause”
Section 9.3, “Keywords and Reserved Words”
Section 13.7.6.4, “KILL Statement”
Section B.3.7, “Known Issues in MySQL”
Section 18.1.7.2, “Limits and Differences of NDB Cluster from Standard MySQL Limits”
Section 18.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 19.2.2, “LIST Partitioning”
Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 19.3.1, “Management of RANGE and LIST Partitions”
Section 15.7.2, “MERGE Table Problems”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.6, “mysql Client Tips”
Section 1.7.1, “MySQL Extensions to Standard SQL”
MySQL Glossary
Section 18.5.16.3, “NDB Cluster and MySQL Security Procedures”
Section 18.5.10.1, “NDB Cluster Disk Data Objects”
Section 18.4.9, “`ndb_delete_all` — Delete All Rows from an NDB Table”
Section 14.13.1, “Online DDL Operations”
Section 13.7.2.4, “OPTIMIZE TABLE Statement”
Section 8.2.4, “Optimizing Data Change Statements”
Section 8.8.1, “Optimizing Queries with EXPLAIN”
Section 8.2.1, “Optimizing SELECT Statements”
Section 8.2.2, “Optimizing Subqueries and Derived Tables”
Section 19.1, “Overview of Partitioning in MySQL”
Section 19.4, “Partition Pruning”
Section 19.5, “Partition Selection”
Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 6.2.2, “Privileges Provided by MySQL”
Section 14.8.10, “Purge Configuration”
Section 8.14.5, “Query Cache Thread States”
Section 19.2.1, “RANGE Partitioning”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.4.1.17, “Replication and LIMIT”
Section 17.4.1.20, “Replication and MEMORY Tables”
Section 17.4.1.22, “Replication and the Query Optimizer”
Section 13.2.10.11, “Rewriting Subqueries as Joins”
Section 3.3.4.1, “Selecting All Data”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”

Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”
Statement Probes
Section 13.2.10, “Subqueries”
Section 13.2.10.9, “Subquery Errors”
Section 8.11.2, “Table Locking Issues”
Section 15.5, “The ARCHIVE Storage Engine”
Section 5.4.4, “The Binary Log”
Section 15.6, “The BLACKHOLE Storage Engine”
Section 21.30.24, “The INFORMATION_SCHEMA INNODB_SYS_TABLESTATS View”
Section 21.29, “The INFORMATION_SCHEMA VIEWS Table”
Section 1.2.2, “The Main Features of MySQL”
Section 15.3, “The MEMORY Storage Engine”
Section 15.7, “The MERGE Storage Engine”
Section 14.7.2.1, “Transaction Isolation Levels”
Section 20.3.1, “Trigger Syntax and Examples”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 13.1.33, “TRUNCATE TABLE Statement”
Section 14.6.7, “Undo Logs”
Section 20.5.3, “Updatable and Insertable Views”
Section 17.1.2.2, “Usage of Row-Based Logging and Replication”
Section 1.3, “What Is New in MySQL 5.6”
Section 6.2.8, “When Privilege Changes Take Effect”
Section 8.2.1.1, “WHERE Clause Optimization”

DELETE FROM ... WHERE ...

Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”

DELETE FROM t1,t2

Statement Probes

DESCRIBE

Section 3.3.2, “Creating a Table”
Section 13.8.1, “DESCRIBE Statement”
Section 13.8.2, “EXPLAIN Statement”
Section 21.35, “Extensions to SHOW Statements”
Section 3.4, “Getting Information About Databases and Tables”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 13.7.5.6, “SHOW COLUMNS Statement”
Section 13.1.17.6, “Silent Column Specification Changes”
Section 3.6.6, “Using Foreign Keys”
Section 10.2.2, “UTF-8 for Metadata”

DO

Section 13.1.2, “ALTER EVENT Statement”
Section 13.1.11, “CREATE EVENT Statement”
Section 13.2.3, “DO Statement”
Section 12.15, “Locking Functions”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 19.6.4, “Partitioning and Locking”
Section 20.8, “Restrictions on Stored Programs”
Section 20.7, “Stored Program Binary Logging”
Section 13.2.10, “Subqueries”
Section 21.8, “The INFORMATION_SCHEMA EVENTS Table”

DROP DATABASE

Section 13.1.21, “DROP DATABASE Statement”
Section 7.4.1, “Dumping Data in SQL Format with mysqldump”
Section 17.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 17.2.3, “How Servers Evaluate Replication Filtering Rules”
Section 8.10.3.1, “How the Query Cache Operates”
Section 18.1.7.8, “Issues Exclusive to NDB Cluster”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 7.5, “Point-in-Time (Incremental) Recovery Using the Binary Log”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 2.3.7, “Windows Platform Restrictions”

DROP DATABASE IF EXISTS

Section 17.4.1.11, “Replication of DROP ... IF EXISTS Statements”

DROP EVENT

Section 20.4.3, “Event Syntax”
Section 17.4.1.16, “Replication of Invoked Features”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 20.7, “Stored Program Binary Logging”
Section 20.4.6, “The Event Scheduler and MySQL Privileges”

DROP FUNCTION

Section 13.1.3, “ALTER FUNCTION Statement”
Section 1.8.1, “Contributors to MySQL”
Section 13.7.3.1, “CREATE FUNCTION Statement for User-Defined Functions”
Section 13.1.23, “DROP FUNCTION Statement”
Section 13.7.3.2, “DROP FUNCTION Statement for User-Defined Functions”
Section 13.1.26, “DROP PROCEDURE and DROP FUNCTION Statements”
Section 9.2.5, “Function Name Parsing and Resolution”
Section 5.6.1, “Installing and Uninstalling User-Defined Functions”
Section 6.5.1, “MySQL Enterprise Encryption Installation”
Section 17.4.1.16, “Replication of Invoked Features”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 20.7, “Stored Program Binary Logging”
Section 20.2.1, “Stored Routine Syntax”
Section 2.11.9, “Upgrade Troubleshooting”

DROP INDEX

Section 13.1.7, “ALTER TABLE Statement”
Section 11.4.8, “Creating Spatial Indexes”
Section 13.1.24, “DROP INDEX Statement”
Section 1.7.1, “MySQL Extensions to Standard SQL”
MySQL Glossary
Section 14.13.1, “Online DDL Operations”
Section 18.5.11, “Online Operations with ALTER TABLE in NDB Cluster”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 5.4.5, “The Slow Query Log”

DROP LOGFILE GROUP

Section 13.1.25, “DROP LOGFILE GROUP Statement”

Section 18.1.7.8, “Issues Exclusive to NDB Cluster”

DROP PROCEDURE

Section 13.1.5, “ALTER PROCEDURE Statement”
Section 17.4.1.16, “Replication of Invoked Features”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 20.7, “Stored Program Binary Logging”
Section 20.2.1, “Stored Routine Syntax”

DROP SCHEMA

Section 13.1.21, “DROP DATABASE Statement”
Section 18.1.7.8, “Issues Exclusive to NDB Cluster”
Section 5.1.7, “Server System Variables”

DROP SERVER

Section 13.7.6.3, “FLUSH Statement”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 8.12.4.1, “How MySQL Uses Memory”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.4.1.5, “Replication of CREATE SERVER, ALTER SERVER, and DROP SERVER”
Section 13.3.3, “Statements That Cause an Implicit Commit”

DROP TABLE

Section 18.5.7.1, “Adding NDB Cluster Data Nodes Online: General Issues”
Section 13.1.7, “ALTER TABLE Statement”
Section 6.4.4.3, “Audit Log File Formats”
Section 18.5.1, “Commands in the NDB Cluster Management Client”
Section 14.7.2.3, “Consistent Nonlocking Reads”
Section 13.1.17.2, “CREATE TEMPORARY TABLE Statement”
Section 13.1.19, “CREATE TRIGGER Statement”
Section 13.1.28, “DROP TABLE Statement”
Section 15.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 14.6.3.2, “File-Per-Table Tablespaces”
Section 14.21.2, “Forcing InnoDB Recovery”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 8.10.3.1, “How the Query Cache Operates”
Section 12.16, “Information Functions”
Section 18.1.7.8, “Issues Exclusive to NDB Cluster”
Section 18.1.7.2, “Limits and Differences of NDB Cluster from Standard MySQL Limits”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 15.7.2, “MERGE Table Problems”
Section 4.5.1.1, “mysql Client Options”
Section 1.7.1, “MySQL Extensions to Standard SQL”
MySQL Glossary
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 18.4.11, “`ndb_drop_index` — Drop Index from an NDB Table”
Section 18.4.12, “`ndb_drop_table` — Drop an NDB Table”
Section 8.5.7, “Optimizing InnoDB DDL Operations”
Section 18.6.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”
Section 6.2.2, “Privileges Provided by MySQL”
Section 20.9, “Restrictions on Views”
Section 13.6.7.6, “Scope Rules for Handlers”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 5.4.4.2, “Setting The Binary Log Format”
Section 13.6.7.5, “SIGNAL Statement”

Section 13.4.2.5, “START SLAVE Statement”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 5.4.6, “The DDL Log”
Section 15.3, “The MEMORY Storage Engine”
Section 15.7, “The MERGE Storage Engine”
Section 13.6.7.7, “The MySQL Diagnostics Area”
Section 14.21.3, “Troubleshooting InnoDB Data Dictionary Operations”
Section 13.1.33, “TRUNCATE TABLE Statement”
Section 13.7.3.4, “UNINSTALL PLUGIN Statement”

DROP TABLE IF EXISTS

Section 17.4.1.11, “Replication of DROP ... IF EXISTS Statements”

DROP TABLE IF EXISTS mysql.user mysql.db mysql.tables_priv mysql.columns_priv mysql.procs_priv

Section 18.5.12, “Distributed Privileges Using Shared Grant Tables”

DROP TABLESPACE

Section 13.1.29, “DROP TABLESPACE Statement”
Section 18.1.7.8, “Issues Exclusive to NDB Cluster”

DROP TEMPORARY TABLE

Section 17.1.3.4, “Restrictions on Replication with GTIDs”

DROP TEMPORARY TABLE IF EXISTS

Section 17.1.2.2, “Usage of Row-Based Logging and Replication”

DROP TRIGGER

Section 13.1.30, “DROP TRIGGER Statement”
Section A.5, “MySQL 5.6 FAQ: Triggers”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 17.4.1.16, “Replication of Invoked Features”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 20.3.1, “Trigger Syntax and Examples”

DROP USER

Section 6.2.1, “Account User Names and Passwords”
Section 6.2.7, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 13.7.1.3, “DROP USER Statement”
Section 13.7.6.3, “FLUSH Statement”
Section 13.7.1.4, “GRANT Statement”
Section 8.12.4.1, “How MySQL Uses Memory”
Section 12.16, “Information Functions”
Section 18.5.16.2, “NDB Cluster and MySQL Privileges”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.4.1.8, “Replication of CURRENT_USER()”
Section 13.7.1.6, “REVOKE Statement”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 20.6, “Stored Object Access Control”
Section 20.4.6, “The Event Scheduler and MySQL Privileges”

DROP VIEW

Section 13.1.31, “DROP VIEW Statement”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 20.9, “Restrictions on Views”
Section 13.3.3, “Statements That Cause an Implicit Commit”

Section 20.5.1, “View Syntax”

DROP VIEW IF EXISTS

Section 17.4.1.11, “Replication of DROP ... IF EXISTS Statements”

E

[index top]

EXECUTE

Section 13.2.1, “CALL Statement”
Section 13.5.2, “EXECUTE Statement”
Section 13.5.1, “PREPARE Statement”
Section 13.5, “Prepared Statements”
Section 20.8, “Restrictions on Stored Programs”
Section 20.9, “Restrictions on Views”
Section 5.1.9, “Server Status Variables”

EXPLAIN

Section 13.1.7, “ALTER TABLE Statement”
Section 8.2.1.19, “Avoiding Full Table Scans”
Section 8.2.1.11, “Block Nested-Loop and Batched Key Access Joins”
Section 8.3.4, “Column Indexes”
Configuring the Number of Sampled Pages for InnoDB Optimizer Statistics
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 5.8.1, “Debugging a MySQL Server”
Section 13.2.10.8, “Derived Tables”
Section 13.8.1, “DESCRIBE Statement”
Section 8.2.1.15, “DISTINCT Optimization”
Section 8.2.1.4, “Engine Condition Pushdown Optimization”
Section 8.8.2, “EXPLAIN Output Format”
Section 13.8.2, “EXPLAIN Statement”
Section 8.8.3, “Extended EXPLAIN Output Format”
Section 8.2.1.14, “GROUP BY Optimization”
Section 8.2.1.5, “Index Condition Pushdown Optimization”
Section 8.9.3, “Index Hints”
Section 8.2.1.3, “Index Merge Optimization”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 21.1, “Introduction”
Section 8.2.1.12, “IS NULL Optimization”
Section 8.2.1.10, “Multi-Range Read Optimization”
Chapter 22, *MySQL Performance Schema*
NDB Cluster Status Variables
NDB Cluster System Variables
Section 19.3.5, “Obtaining Information About Partitions”
Section B.3.5, “Optimizer-Related Issues”
Section 8.2.2.4, “Optimizing Derived Tables”
Section 8.2.3, “Optimizing INFORMATION_SCHEMA Queries”
Section 8.8.1, “Optimizing Queries with EXPLAIN”
Section 8.2.1, “Optimizing SELECT Statements”
Section 13.2.10.10, “Optimizing Subqueries”
Section 8.2.2.2, “Optimizing Subqueries with Materialization”
Section 8.2.2.1, “Optimizing Subqueries with Semijoin Transformations”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”
Section 8.2.1.13, “ORDER BY Optimization”
Section 19.4, “Partition Pruning”
Section 6.2.2, “Privileges Provided by MySQL”

Section 8.2.1.2, “Range Optimization”
Section 20.8, “Restrictions on Stored Programs”
Section 13.2.9, “SELECT Statement”
Section 13.7.5.41, “SHOW WARNINGS Statement”
Section B.3.4.7, “Solving Problems with No Matching Rows”
Section 1.2.2, “The Main Features of MySQL”
Section 8.8, “Understanding the Query Execution Plan”
Section 8.3.9, “Use of Index Extensions”
Section 5.8.1.6, “Using Server Logs to Find Causes of Errors in mysqld”
Section 11.4.9, “Using Spatial Indexes”
Section 8.3.6, “Verifying Index Usage”
Section 1.3, “What Is New in MySQL 5.6”

EXPLAIN ... SELECT

Section 19.3.5, “Obtaining Information About Partitions”

EXPLAIN EXTENDED

Section 8.8.2, “EXPLAIN Output Format”
Section 13.8.2, “EXPLAIN Statement”
Section 8.8.1, “Optimizing Queries with EXPLAIN”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”
Section 13.7.5.41, “SHOW WARNINGS Statement”

EXPLAIN PARTITIONS

Section 13.8.2, “EXPLAIN Statement”
Section 19.3.5, “Obtaining Information About Partitions”
Section 8.8.1, “Optimizing Queries with EXPLAIN”

EXPLAIN PARTITIONS SELECT

Section 19.3.5, “Obtaining Information About Partitions”

EXPLAIN PARTITIONS SELECT COUNT()

Section 19.2.1, “RANGE Partitioning”

EXPLAIN SELECT

Section 13.2.10.8, “Derived Tables”
Section 8.8.2, “EXPLAIN Output Format”
Section 14.7.5.3, “How to Minimize and Handle Deadlocks”
Section 1.6, “How to Report Bugs or Problems”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 19.3.5, “Obtaining Information About Partitions”

EXPLAIN tbl_name

Section 8.8.1, “Optimizing Queries with EXPLAIN”

F

[index top]

FETCH

Section 13.6.6.2, “Cursor DECLARE Statement”
Section 13.6.6.3, “Cursor FETCH Statement”
Section 20.8, “Restrictions on Stored Programs”

FETCH ... INTO var_list

Section 13.6.4, “Variables in Stored Programs”

FLUSH

Section 7.3.1, “Establishing a Backup Policy”
Section 13.7.6.3, “FLUSH Statement”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 13.7.1.4, “GRANT Statement”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.4.1.13, “Replication and FLUSH”
Section 13.7.6.6, “RESET Statement”
Section 20.8, “Restrictions on Stored Programs”
Section 2.10.4, “Securing the Initial MySQL Accounts”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 4.10, “Unix Signal Handling in MySQL”

FLUSH BINARY LOGS

Section 13.7.6.3, “FLUSH Statement”
Section 5.4.7, “Server Log Maintenance”

FLUSH DES_KEY_FILE

Section 12.14, “Encryption and Compression Functions”
Section 13.7.6.3, “FLUSH Statement”

FLUSH ENGINE LOGS

Section 13.7.6.3, “FLUSH Statement”

FLUSH ERROR LOGS

Section 5.4.2.5, “Error Log File Flushing and Renaming”
Section 13.7.6.3, “FLUSH Statement”

FLUSH GENERAL LOGS

Section 13.7.6.3, “FLUSH Statement”

FLUSH HOSTS

Section 5.1.11.2, “DNS Lookups and the Host Cache”
Section 13.7.6.3, “FLUSH Statement”

FLUSH LOGS

Section 7.3.3, “Backup Strategy Summary”
Section 7.2, “Database Backup Methods”
Section 17.1.3.5, “Disabling GTID Transactions”
Section 5.4.2.5, “Error Log File Flushing and Renaming”
Section 7.3.1, “Establishing a Backup Policy”
Section 13.7.6.3, “FLUSH Statement”
Section 5.4, “MySQL Server Logs”
Section 17.4.1.13, “Replication and FLUSH”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.4.7, “Server Log Maintenance”
Section 5.1.9, “Server Status Variables”
Section 17.1.3.2, “Setting Up Replication Using GTIDs”
Section 17.2.2.1, “The Relay Log”
Section 17.1.3.3, “Using GTIDs for Failover and Scaleout”

FLUSH PRIVILEGES

Section 1.1, “About This Manual”

Section 18.5.12, “Distributed Privileges Using Shared Grant Tables”
Section 13.7.6.3, “FLUSH Statement”
Section 6.2.3, “Grant Tables”
Section 8.12.4.1, “How MySQL Uses Memory”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 18.5.16.3, “NDB Cluster and MySQL Security Procedures”
Section 17.4.1.13, “Replication and FLUSH”
Section 5.1.6, “Server Command Options”
Section 6.2.13, “Setting Account Resource Limits”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 6.2.8, “When Privilege Changes Take Effect”

FLUSH QUERY CACHE

Section 13.7.6.3, “FLUSH Statement”
Section 8.10.3.4, “Query Cache Status and Maintenance”

FLUSH RELAY LOGS

Section 13.7.6.3, “FLUSH Statement”

FLUSH SLOW LOGS

Section 13.7.6.3, “FLUSH Statement”

FLUSH STATUS

Section 13.7.6.3, “FLUSH Statement”
Section 5.1.9, “Server Status Variables”
Section 8.3.9, “Use of Index Extensions”

FLUSH TABLE

Section 13.7.6.3, “FLUSH Statement”
Section 1.3, “What Is New in MySQL 5.6”

FLUSH TABLES

Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 8.10.4, “Caching of Prepared Statements and Stored Programs”
Section 13.7.6.3, “FLUSH Statement”
Section 8.14.3, “General Thread States”
Section 13.2.4, “HANDLER Statement”
Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section 8.12.4.1, “How MySQL Uses Memory”
Section 13.2.5.3, “INSERT DELAYED Statement”
Section 15.7.2, “MERGE Table Problems”
Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”
Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”
Section 17.1.1.4, “Obtaining the Replication Source Binary Log Coordinates”
Section 15.2.4.2, “Problems from Tables Not Being Closed Properly”
Section 8.10.3.4, “Query Cache Status and Maintenance”
Section 17.4.1.13, “Replication and FLUSH”
Section 19.6, “Restrictions and Limitations on Partitioning”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”
Section 8.3.9, “Use of Index Extensions”

FLUSH TABLES ... FOR EXPORT

Section 14.6.1.2, “Creating Tables Externally”
Section 13.7.6.3, “FLUSH Statement”

Section 14.6.1.3, “Importing InnoDB Tables”
MySQL Glossary

FLUSH TABLES ...FOR EXPORT

Section 13.7.6.3, “FLUSH Statement”

FLUSH TABLES tbl_list WITH READ LOCK

Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”

FLUSH TABLES tbl_name ...

Section 13.7.6.3, “FLUSH Statement”

FLUSH TABLES tbl_name ... FOR EXPORT

Section 13.7.6.3, “FLUSH Statement”

FLUSH TABLES tbl_name ... WITH READ LOCK

Section 13.7.6.3, “FLUSH Statement”

FLUSH TABLES tbl_name WITH READ LOCK

Section 13.2.4, “HANDLER Statement”

FLUSH TABLES WITH READ LOCK

Section 17.1.1.5, “Creating a Data Snapshot Using mysqldump”
Section 7.2, “Database Backup Methods”
Section 7.3.1, “Establishing a Backup Policy”
Section 13.7.6.3, “FLUSH Statement”
Section 8.14.3, “General Thread States”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
MySQL Glossary
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 17.1.1.4, “Obtaining the Replication Source Binary Log Coordinates”
Section 17.4.1.13, “Replication and FLUSH”
Section 13.3.7.3, “Restrictions on XA Transactions”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.7, “Server System Variables”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”
Section 13.3.3, “Statements That Cause an Implicit Commit”

FLUSH USER_RESOURCES

Section 13.7.6.3, “FLUSH Statement”
Section 6.2.13, “Setting Account Resource Limits”

G

[index top]

GET DIAGNOSTICS

Section 13.6.7, “Condition Handling”
Section B.2, “Error Information Interfaces”
Section 13.6.7.3, “GET DIAGNOSTICS Statement”
Section 13.6.7.4, “RESIGNAL Statement”
Section 13.6.7.9, “Restrictions on Condition Handling”
Section 20.8, “Restrictions on Stored Programs”
Section 5.1.7, “Server System Variables”
Section 13.7.5.41, “SHOW WARNINGS Statement”

Section 13.6.7.5, “SIGNAL Statement”
Section 13.6.7.7, “The MySQL Diagnostics Area”
Section 1.3, “What Is New in MySQL 5.6”

GRANT

Section 6.2, “Access Control and Account Management”
Section 6.2.6, “Access Control, Stage 2: Request Verification”
Section 6.2.1, “Account User Names and Passwords”
Section 6.2.7, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 6.2.9, “Assigning Account Passwords”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 14.8.2, “Configuring InnoDB for Read-Only Operation”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 5.1.12.3, “Connecting Using the IPv6 Local Host Address”
Section 13.7.1.2, “CREATE USER Statement”
Section 17.1.1.3, “Creating a User for Replication”
Section 12.14, “Encryption and Compression Functions”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 13.7.6.3, “FLUSH Statement”
Section 13.7.1.4, “GRANT Statement”
Section 6.2.3, “Grant Tables”
Section 8.12.4.1, “How MySQL Uses Memory”
Section 12.16, “Information Functions”
Section 5.1.12, “IPv6 Support”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section A.14, “MySQL 5.6 FAQ: Replication”
MySQL Glossary
Section 18.5.16.2, “NDB Cluster and MySQL Privileges”
Section 8.2.5, “Optimizing Database Privileges”
Section 6.4.1.6, “PAM Pluggable Authentication”
Section 6.1.2.4, “Password Hashing in MySQL”
Section 6.1.2.3, “Passwords and Logging”
Section 18.6.5, “Preparing the NDB Cluster for Replication”
Section 6.2.2, “Privileges Provided by MySQL”
Section 2.10.1.1, “Problems Running mysql_install_db”
Section 6.2.12, “Proxy Users”
Section 17.4.1.13, “Replication and FLUSH”
Section 17.4.1.8, “Replication of CURRENT_USER()”
Section 17.4.1.21, “Replication of the mysql System Database”
Section 13.7.1.6, “REVOKE Statement”
Section 6.1.1, “Security Guidelines”
Section 5.1.6, “Server Command Options”
Section 5.1.10, “Server SQL Modes”
Section 5.1.7, “Server System Variables”
Section 6.2.13, “Setting Account Resource Limits”
Section 13.7.5.22, “SHOW GRANTS Statement”
Section 6.2.4, “Specifying Account Names”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 20.4.6, “The Event Scheduler and MySQL Privileges”
Section 6.4.3, “The Password Validation Plugin”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 6.3, “Using Encrypted Connections”
Section 6.4.5.3, “Using MySQL Enterprise Firewall”
Section 1.3, “What Is New in MySQL 5.6”
Section 6.2.8, “When Privilege Changes Take Effect”
Section 6.4.1.7, “Windows Pluggable Authentication”

GRANT ALL
Section 13.7.1.4, “GRANT Statement”

GRANT EVENT
Section 20.4.6, “The Event Scheduler and MySQL Privileges”

GRANT USAGE
Section 6.2.9, “Assigning Account Passwords”
Section 13.7.1.4, “GRANT Statement”
Section 6.2.13, “Setting Account Resource Limits”

GROUP BY
Section 14.1.1, “Benefits of Using InnoDB Tables”

H

[index top]

HANDLER
Section 15.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 13.7.6.3, “FLUSH Statement”
Section A.4, “MySQL 5.6 FAQ: Stored Procedures and Functions”
Section 1.7, “MySQL Standards Compliance”
Section 19.6, “Restrictions and Limitations on Partitioning”
Section 5.1.7, “Server System Variables”

HANDLER ... CLOSE
Section 13.7.5.25, “SHOW OPEN TABLES Statement”

HANDLER ... OPEN
Section 13.7.5.25, “SHOW OPEN TABLES Statement”

HANDLER ... READ
Section 20.8, “Restrictions on Stored Programs”

HANDLER OPEN
Section 13.2.4, “HANDLER Statement”
Section 13.1.33, “TRUNCATE TABLE Statement”

HELP
Section 13.8.3, “HELP Statement”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 5.1.14, “Server-Side Help Support”

I

[index top]

IF
Section 8.10.4, “Caching of Prepared Statements and Stored Programs”
Section 13.6.7.2, “DECLARE ... HANDLER Statement”
Section 12.5, “Flow Control Functions”
Section 13.6.5, “Flow Control Statements”

Section 13.6.5.2, “IF Statement”

INSERT

Section 6.2, “Access Control and Account Management”
Section 6.2.6, “Access Control, Stage 2: Request Verification”
Section 6.2.7, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 14.6.1.6, “AUTO_INCREMENT Handling in InnoDB”
Section 14.7.2.2, “autocommit, Commit, and Rollback”
Section 7.1, “Backup and Recovery Types”
Section 14.1.2, “Best Practices for InnoDB Tables”
Section 8.5.5, “Bulk Data Loading for InnoDB Tables”
Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 8.10.4, “Caching of Prepared Statements and Stored Programs”
Section 14.5.2, “Change Buffer”
Section 2.11.3, “Changes in MySQL 5.6”
Section 10.7, “Column Character Set Conversion”
Section 14.9.6, “Compression for OLTP Workloads”
Section 8.11.3, “Concurrent Inserts”
Section 1.7.3.3, “Constraints on Invalid Data”
Section 14.6.1.5, “Converting Tables from MyISAM to InnoDB”
Section 13.1.13, “CREATE INDEX Statement”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 13.1.17.4, “CREATE TABLE ... SELECT Statement”
Section 13.1.17.2, “CREATE TEMPORARY TABLE Statement”
Section 13.1.19, “CREATE TRIGGER Statement”
Section 13.1.20, “CREATE VIEW Statement”
Section 15.8.2.1, “Creating a FEDERATED Table Using CONNECTION”
Section 11.5, “Data Type Default Values”
Section 11.2.1, “Date and Time Data Type Syntax”
Section 13.6.7.2, “DECLARE ... HANDLER Statement”
Section 13.2.2, “DELETE Statement”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 7.3.1, “Establishing a Backup Policy”
Section 8.8.2, “EXPLAIN Output Format”
Section 13.8.2, “EXPLAIN Statement”
Section 12.21.3, “Expression Handling”
Section 8.8.3, “Extended EXPLAIN Output Format”
Section 15.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 14.21.2, “Forcing InnoDB Recovery”
Section 13.1.17.5, “FOREIGN KEY Constraints”
Section 12.10.5, “Full-Text Restrictions”
Section 8.14.3, “General Thread States”
Section 13.7.1.4, “GRANT Statement”
Section 6.2.3, “Grant Tables”
Section 8.10.3.1, “How the Query Cache Operates”
Section 12.16, “Information Functions”
Section 14.7.1, “InnoDB Locking”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 13.2.5.2, “INSERT ... ON DUPLICATE KEY UPDATE Statement”
Section 13.2.5.1, “INSERT ... SELECT Statement”
Section 13.2.5.3, “INSERT DELAYED Statement”
Section 13.2.5, “INSERT Statement”
Section 17.2.3.3, “Interactions Between Replication Filtering Options”
Section 8.11.1, “Internal Locking Methods”
Section 21.1, “Introduction”
Section 18.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 19.2.2, “LIST Partitioning”

Section 13.2.6, “LOAD DATA Statement”
Section 3.3.3, “Loading Data into a Table”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 19.3.1, “Management of RANGE and LIST Partitions”
Section 15.7.2, “MERGE Table Problems”
Section 8.11.4, “Metadata Locking”
Section 12.20, “Miscellaneous Functions”
Section A.1, “MySQL 5.6 FAQ: General”
Section A.11, “MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section A.5, “MySQL 5.6 FAQ: Triggers”
Section A.6, “MySQL 5.6 FAQ: Views”
Section 4.5.1.1, “mysql Client Options”
Section 6.5.2, “MySQL Enterprise Encryption Usage and Examples”
Section 1.7.1, “MySQL Extensions to Standard SQL”
MySQL Glossary
Section B.3.2.8, “MySQL server has gone away”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 18.5.13, “NDB API Statistics Counters and Variables”
Section 18.5.10.1, “NDB Cluster Disk Data Objects”
Section 18.2.5, “NDB Cluster Example with Tables and Data”
Section 18.6.11, “NDB Cluster Replication Conflict Resolution”
Section 14.13.1, “Online DDL Operations”
Section 13.7.2.4, “OPTIMIZE TABLE Statement”
Section 8.2.4, “Optimizing Data Change Statements”
Section 8.2.4.1, “Optimizing INSERT Statements”
Section 8.6.1, “Optimizing MyISAM Queries”
Section 8.8.1, “Optimizing Queries with EXPLAIN”
Section 11.1.7, “Out-of-Range and Overflow Handling”
Section 19.1, “Overview of Partitioning in MySQL”
Section 19.4, “Partition Pruning”
Section 19.5, “Partition Selection”
Section 19.6.4, “Partitioning and Locking”
Section 6.1.2.3, “Passwords and Logging”
Section 22.12.6, “Performance Schema Statement Event Tables”
Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 11.4.5, “Populating Spatial Columns”
Section 1.7.3.1, “PRIMARY KEY and UNIQUE Index Constraints”
Section 6.2.2, “Privileges Provided by MySQL”
Section 14.8.10, “Purge Configuration”
Section 8.14.5, “Query Cache Thread States”
Section 19.2.1, “RANGE Partitioning”
Section 13.2.8, “REPLACE Statement”
Section 17.4.1.27, “Replica Errors During Replication”
Section 17.4.1.1, “Replication and AUTO_INCREMENT”
Section 17.4.1.28, “Replication and Server SQL Mode”
Section 17.4.1.15, “Replication and System Functions”
Section 17.4.1.35, “Replication and Variables”
Section 17.1.4.2, “Replication Source Options and Variables”
Section 19.6, “Restrictions and Limitations on Partitioning”
Section 20.8, “Restrictions on Stored Programs”
Row-Level Probes
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.10, “Server SQL Modes”
Section 5.1.7, “Server System Variables”
Section 13.7.5.28, “SHOW PROCEDURE CODE Statement”

Section 13.7.5.41, “SHOW WARNINGS Statement”
Statement Probes
Section 20.7, “Stored Program Binary Logging”
Section 13.2.10, “Subqueries”
Section 8.11.2, “Table Locking Issues”
Section 15.5, “The ARCHIVE Storage Engine”
Section 10.8.5, “The binary Collation Compared to _bin Collations”
Section 5.4.4, “The Binary Log”
Section 15.6, “The BLACKHOLE Storage Engine”
Section 21.29, “The INFORMATION_SCHEMA VIEWS Table”
Section 1.2.2, “The Main Features of MySQL”
Section 15.7, “The MERGE Storage Engine”
Section 15.2, “The MyISAM Storage Engine”
Section 8.10.3, “The MySQL Query Cache”
Section 5.1.15, “The Server Shutdown Process”
Section 20.3.1, “Trigger Syntax and Examples”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 14.20.8, “Troubleshooting the InnoDB memcached Plugin”
Section 14.6.7, “Undo Logs”
Section 20.5.3, “Updatable and Insertable Views”
Section 13.2.11, “UPDATE Statement”
Section 14.15.2.1, “Using InnoDB Transaction and Locking Information”
Section 20.3, “Using Triggers”
Section 1.3, “What Is New in MySQL 5.6”
Section 6.2.8, “When Privilege Changes Take Effect”

INSERT ... ()

Statement Probes

INSERT ... ON DUPLICATE KEY UPDATE

Section 14.6.1.6, “AUTO_INCREMENT Handling in InnoDB”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 15.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 12.16, “Information Functions”
Section 13.2.5.2, “INSERT ... ON DUPLICATE KEY UPDATE Statement”
Section 13.2.5.3, “INSERT DELAYED Statement”
Section 13.2.5, “INSERT Statement”
Section 18.6.3, “Known Issues in NDB Cluster Replication”
Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 15.7.2, “MERGE Table Problems”
Section 12.20, “Miscellaneous Functions”
MySQL Glossary
Section 19.6.4, “Partitioning and Locking”

INSERT ... SELECT

Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 14.6.1.6, “AUTO_INCREMENT Handling in InnoDB”
Section 8.11.3, “Concurrent Inserts”
Section 13.1.17.4, “CREATE TABLE ... SELECT Statement”
Section 13.2.5.2, “INSERT ... ON DUPLICATE KEY UPDATE Statement”
Section 13.2.5.1, “INSERT ... SELECT Statement”
Section 13.2.5.3, “INSERT DELAYED Statement”
Section 13.2.5, “INSERT Statement”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section B.3.7, “Known Issues in MySQL”
Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”
NDB Cluster System Variables

Section 19.5, “Partition Selection”
Section 19.6.4, “Partitioning and Locking”
Section 17.4.1.17, “Replication and LIMIT”
Section 5.1.7, “Server System Variables”
Statement Probes
Section 5.4.4, “The Binary Log”

INSERT ... SELECT ON DUPLICATE KEY UPDATE

Section 13.2.5.2, “INSERT ... ON DUPLICATE KEY UPDATE Statement”
Section 13.2.5.1, “INSERT ... SELECT Statement”

INSERT ... SET

Section 13.2.5, “INSERT Statement”

INSERT ... VALUES

Section 13.2.5, “INSERT Statement”

INSERT DELAYED

Section 13.1.7, “ALTER TABLE Statement”
Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 8.14.4, “Delayed-Insert Thread States”
Section 13.2.5.3, “INSERT DELAYED Statement”
Section 13.2.5, “INSERT Statement”
Section 13.7.6.4, “KILL Statement”
Section B.3.7, “Known Issues in MySQL”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 15.7.2, “MERGE Table Problems”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 8.6.1, “Optimizing MyISAM Queries”
Section 19.6.4, “Partitioning and Locking”
Section 19.6, “Restrictions and Limitations on Partitioning”
Section 20.8, “Restrictions on Stored Programs”
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”
Section 8.11.2, “Table Locking Issues”
Section 15.5, “The ARCHIVE Storage Engine”
Section 15.3, “The MEMORY Storage Engine”
Section 22.12.10.3, “The threads Table”
Section 20.5.3, “Updatable and Insertable Views”

INSERT IGNORE

Section 1.7.3.3, “Constraints on Invalid Data”
Section 13.1.17.4, “CREATE TABLE ... SELECT Statement”
Section 1.7.3.4, “ENUM and SET Constraints”
Section 12.16, “Information Functions”
Section 13.2.5, “INSERT Statement”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 5.1.10, “Server SQL Modes”

INSERT IGNORE ... SELECT

Section 13.2.5.1, “INSERT ... SELECT Statement”

INSERT INTO ... SELECT

Section 6.2.6, “Access Control, Stage 2: Request Verification”
Section 14.7.2.3, “Consistent Nonlocking Reads”

Section 1.7.3.3, “Constraints on Invalid Data”
Section 13.1.11, “CREATE EVENT Statement”
Section 13.2.5, “INSERT Statement”
Section 1.7.2.1, “SELECT INTO TABLE Differences”
Section 15.3, “The MEMORY Storage Engine”

INSERT INTO ... SELECT FROM memory_table

Section 17.4.1.20, “Replication and MEMORY Tables”

INSERT INTO...SELECT

Section 8.2.1, “Optimizing SELECT Statements”

INSTALL PLUGIN

Section 6.4.4.6, “Audit Log Reference”
Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 13.7.6.3, “FLUSH Statement”
Section 8.12.4.1, “How MySQL Uses Memory”
Section 14.20.2, “InnoDB memcached Architecture”
Section 13.7.3.3, “INSTALL PLUGIN Statement”
Section 5.5.1, “Installing and Uninstalling Plugins”
Section 6.4.4.1, “Installing MySQL Enterprise Audit”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”
Section 5.5.2, “Obtaining Server Plugin Information”
Section 6.4.1.6, “PAM Pluggable Authentication”
Section 6.4.3.1, “Password Validation Plugin Installation”
Section 6.4.3.2, “Password Validation Plugin Options and Variables”
Section 15.11.1, “Pluggable Storage Engine Architecture”
Section 17.3.9.1, “Semisynchronous Replication Administrative Interface”
Section 17.3.9.2, “Semisynchronous Replication Installation and Configuration”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 14.20.3, “Setting Up the InnoDB memcached Plugin”
Section 13.7.5.26, “SHOW PLUGINS Statement”
Section 6.4.1.8, “Socket Peer-Credential Pluggable Authentication”
Section 6.4.1.9, “Test Pluggable Authentication”
Section 21.15, “The INFORMATION_SCHEMA PLUGINS Table”
Section 5.3, “The mysql System Database”
Section 14.20.8, “Troubleshooting the InnoDB memcached Plugin”
Section 13.7.3.4, “UNINSTALL PLUGIN Statement”
Section 6.4.1.7, “Windows Pluggable Authentication”

ITERATE

Section 13.6.7.2, “DECLARE ... HANDLER Statement”
Section 13.6.5, “Flow Control Statements”
Section 13.6.5.3, “ITERATE Statement”
Section 13.6.2, “Statement Labels”

K

[index top]

KILL

Section 8.14, “Examining Server Thread (Process) Information”
Section 8.14.3, “General Thread States”
Section 13.7.1.4, “GRANT Statement”

Section 13.7.6.4, “KILL Statement”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section B.3.2.8, “MySQL server has gone away”
Section 4.6.16, “`mysql_zap` — Kill Processes That Match a Pattern”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.5.30, “SHOW PROCESSLIST Statement”

KILL CONNECTION

Section 13.7.6.4, “KILL Statement”
Section 13.4.2.6, “STOP SLAVE Statement”
Section 5.1.15, “The Server Shutdown Process”

KILL QUERY

Section 13.7.6.4, “KILL Statement”
Section 12.20, “Miscellaneous Functions”
Section 13.4.2.6, “STOP SLAVE Statement”
Section 5.1.15, “The Server Shutdown Process”

L

[index top]

LEAVE

Section 13.6.7.2, “DECLARE ... HANDLER Statement”
Section 13.6.5, “Flow Control Statements”
Section 13.6.5.4, “LEAVE Statement”
Section 13.6.5.5, “LOOP Statement”
Section 20.8, “Restrictions on Stored Programs”
Section 13.6.5.7, “RETURN Statement”
Section 13.6.2, “Statement Labels”

LOAD DATA

Section 6.4.4.3, “Audit Log File Formats”
Section 6.4.4.7, “Audit Log Restrictions”
Section 14.6.1.6, “AUTO_INCREMENT Handling in InnoDB”
Section 17.3.1.2, “Backing Up Raw Data from a Replica”
Section 7.1, “Backup and Recovery Types”
Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 8.11.3, “Concurrent Inserts”
Section 13.1.19, “CREATE TRIGGER Statement”
Section 7.2, “Database Backup Methods”
Section 10.3.3, “Database Character Set and Collation”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 13.1.17.5, “FOREIGN KEY Constraints”
Section B.3.3.4, “How MySQL Handles a Full Disk”
Section 12.16, “Information Functions”
Section B.3.7, “Known Issues in MySQL”
Section 18.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 13.2.6, “LOAD DATA Statement”
Section 13.2.7, “LOAD XML Statement”
Section 3.3.3, “Loading Data into a Table”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 15.2.1, “MyISAM Startup Options”
Section 4.5.1.1, “mysql Client Options”
Section 1.7.1, “MySQL Extensions to Standard SQL”

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 18.4.26, “`ndb_show_tables` — Display List of NDB Tables”
Section 9.1.7, “NULL Values”
Section 8.2.4.1, “Optimizing INSERT Statements”
Section 11.1.7, “Out-of-Range and Overflow Handling”
Section 4.1, “Overview of MySQL Programs”
Section 19.1, “Overview of Partitioning in MySQL”
Section 19.5, “Partition Selection”
Section 19.6.4, “Partitioning and Locking”
Section 18.5.5, “Performing a Rolling Restart of an NDB Cluster”
Section 6.2.2, “Privileges Provided by MySQL”
Section B.3.4.3, “Problems with NULL Values”
Section 7.4.4, “Reloading Delimited-Text Format Backups”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.4.1.18, “Replication and LOAD DATA”
Section 17.4.2, “Replication Compatibility Between MySQL Versions”
Section 8.14.8, “Replication Replica SQL Thread States”
Section 19.6, “Restrictions and Limitations on Partitioning”
Section 10.11, “Restrictions on Character Sets”
Section 20.8, “Restrictions on Stored Programs”
Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”
Section 13.2.9.1, “SELECT ... INTO Statement”
Section 3.3.4.1, “Selecting All Data”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 13.7.5.41, “SHOW WARNINGS Statement”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 13.2.10, “Subqueries”
Section 11.3.5, “The ENUM Type”
Section 15.3, “The MEMORY Storage Engine”
Section 13.2.10.1, “The Subquery as Scalar Operand”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 9.4, “User-Defined Variables”
Section 20.3, “Using Triggers”
Section 1.3, “What Is New in MySQL 5.6”
Section B.3.3.5, “Where MySQL Stores Temporary Files”
Section 2.3.7, “Windows Platform Restrictions”

LOAD DATA ... REPLACE

Section 13.2.8, “REPLACE Statement”

LOAD DATA LOCAL

Section 13.2.6, “LOAD DATA Statement”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”
Section 5.1.7, “Server System Variables”

LOAD INDEX INTO CACHE

Section 13.7.6.2, “CACHE INDEX Statement”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 8.10.2.4, “Index Preloading”
Section 13.7.6.5, “LOAD INDEX INTO CACHE Statement”
Section 19.6, “Restrictions and Limitations on Partitioning”
Section 13.3.3, “Statements That Cause an Implicit Commit”

LOAD INDEX INTO CACHE ... IGNORE LEAVES

Section 13.7.6.5, “LOAD INDEX INTO CACHE Statement”

LOAD XML

Section 13.2.7, “LOAD XML Statement”
Section 19.1, “Overview of Partitioning in MySQL”
Section 19.5, “Partition Selection”
Section 1.3, “What Is New in MySQL 5.6”

LOAD XML LOCAL

Section 13.2.7, “LOAD XML Statement”

LOCK TABLE

Section 8.11.3, “Concurrent Inserts”
Section 8.14.3, “General Thread States”
Section B.3.6.1, “Problems with ALTER TABLE”

LOCK TABLES

Section 14.1.2, “Best Practices for InnoDB Tables”
Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 13.1.10, “CREATE DATABASE Statement”
Section 13.1.17.3, “CREATE TABLE ... LIKE Statement”
Section 13.1.19, “CREATE TRIGGER Statement”
Section 14.7.5.2, “Deadlock Detection”
Section 14.7.5, “Deadlocks in InnoDB”
Section 13.7.6.3, “FLUSH Statement”
Section 8.14.3, “General Thread States”
Section 13.7.1.4, “GRANT Statement”
Section 14.7.5.3, “How to Minimize and Handle Deadlocks”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 8.11.1, “Internal Locking Methods”
Section 18.1.7.10, “Limitations Relating to Multiple NDB Cluster Nodes”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 15.7.2, “MERGE Table Problems”
Section 8.11.4, “Metadata Locking”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.6.10, “`mysqlhotcopy` — A Database Backup Program”
Section 19.6.4, “Partitioning and Locking”
Section 6.2.2, “Privileges Provided by MySQL”
Section 15.2.4.2, “Problems from Tables Not Being Closed Properly”
Section 20.8, “Restrictions on Stored Programs”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.7, “Server System Variables”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 8.12.1, “System Factors”
Section 8.11.2, “Table Locking Issues”

LOCK TABLES ... READ

Section 13.7.6.3, “FLUSH Statement”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”

LOCK TABLES ... WRITE

Section 14.7.1, “InnoDB Locking”

Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”

LOOP

Section 13.6.5, “Flow Control Statements”
Section 13.6.5.3, “ITERATE Statement”
Section 13.6.5.4, “LEAVE Statement”
Section 13.6.5.5, “LOOP Statement”
Section 13.6.2, “Statement Labels”

O

[index top]

OPTIMIZE TABLE

Section 18.5.7.2, “Adding NDB Cluster Data Nodes Online: Basic procedure”
Section 18.5.7.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”
Section 5.8.1, “Debugging a MySQL Server”
Section 13.2.2, “DELETE Statement”
Section 15.2.3.2, “Dynamic Table Characteristics”
Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”
Section 8.14.3, “General Thread States”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section B.3.3.4, “How MySQL Handles a Full Disk”
Section 14.6.1.3, “Importing InnoDB Tables”
Section 14.15.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 13.7.6.4, “KILL Statement”
Section B.3.7, “Known Issues in MySQL”
Section 18.1.7.2, “Limits and Differences of NDB Cluster from Standard MySQL Limits”
Section 19.3.4, “Maintenance of Partitions”
Section 15.7.2, “MERGE Table Problems”
Section 7.6, “MyISAM Table Maintenance and Crash Recovery”
Section 7.6.4, “MyISAM Table Optimization”
Section 4.6.3.1, “myisamchk General Options”
Section 1.7.1, “MySQL Extensions to Standard SQL”
MySQL Server Options for NDB Cluster
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 18.5.10.3, “NDB Cluster Disk Data Storage Requirements”
Section 14.13.6, “Online DDL Limitations”
Section 18.5.11, “Online Operations with ALTER TABLE in NDB Cluster”
Section 13.7.2.4, “OPTIMIZE TABLE Statement”
Section 8.6.1, “Optimizing MyISAM Queries”
Section 8.2.4.2, “Optimizing UPDATE Statements”
Section 8.2.6, “Other Optimization Tips”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.4.1.13, “Replication and FLUSH”
Section 19.6, “Restrictions and Limitations on Partitioning”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 15.2.3.1, “Static (Fixed-Length) Table Characteristics”
Section 15.5, “The ARCHIVE Storage Engine”
Section 21.30.7, “The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table”
Section 21.30.8, “The INFORMATION_SCHEMA INNODB_FT_CONFIG Table”
Section 21.30.10, “The INFORMATION_SCHEMA INNODB_FT_DELETED Table”
Section 21.30.11, “The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table”

Section 21.30.12, “The INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE Table”
Section 5.1.15, “The Server Shutdown Process”
Section 5.4.5, “The Slow Query Log”
Section 8.12.3.2, “Using Symbolic Links for MyISAM Tables on Unix”
Section 1.3, “What Is New in MySQL 5.6”

ORDER BY

Section 14.1.1, “Benefits of Using InnoDB Tables”

P

[index top]

PREPARE

Section 8.10.4, “Caching of Prepared Statements and Stored Programs”
Section 13.2.1, “CALL Statement”
Section 13.5.3, “DEALLOCATE PREPARE Statement”
Section 13.5.2, “EXECUTE Statement”
Section 9.2.3, “Identifier Case Sensitivity”
Section 8.11.4, “Metadata Locking”
Section 13.5.1, “PREPARE Statement”
Section 13.5, “Prepared Statements”
Section 20.8, “Restrictions on Stored Programs”
Section 20.9, “Restrictions on Views”
Section 5.1.9, “Server Status Variables”

PURGE BINARY LOGS

Section 17.1.4.4, “Binary Log Options and Variables”
Section 7.3.1, “Establishing a Backup Policy”
Section 13.7.1.4, “GRANT Statement”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.4.1.1, “PURGE BINARY LOGS Statement”
Section 13.4.1.2, “RESET MASTER Statement”
Section 5.4.7, “Server Log Maintenance”
Section 5.4.4, “The Binary Log”
Section 17.1.3.3, “Using GTIDs for Failover and Scaleout”

R

[index top]

RELEASE SAVEPOINT

Section 13.3.4, “SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Statements”

RENAME TABLE

Section 13.1.7, “ALTER TABLE Statement”
Section 13.2.2, “DELETE Statement”
Section 8.14.3, “General Thread States”
Section 9.2.3, “Identifier Case Sensitivity”
Section 8.11.4, “Metadata Locking”
Section 14.6.1.4, “Moving or Copying InnoDB Tables”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 14.13.1, “Online DDL Operations”
Section 13.1.32, “RENAME TABLE Statement”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”

Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 8.12.3.2, “Using Symbolic Links for MyISAM Tables on Unix”

RENAME USER

Section 13.7.1.4, “GRANT Statement”
Section 12.16, “Information Functions”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.1.5, “RENAME USER Statement”
Section 17.4.1.8, “Replication of CURRENT_USER()”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 20.6, “Stored Object Access Control”
Section 20.4.6, “The Event Scheduler and MySQL Privileges”
Section 6.2.8, “When Privilege Changes Take Effect”

REPAIR TABLE

Section 11.2.5, “2-Digit YEAR(2) Limitations and Migrating to 4-Digit YEAR”
Section 13.1.7.1, “ALTER TABLE Partition Operations”
Section 13.1.7, “ALTER TABLE Statement”
Section 2.11.3, “Changes in MySQL 5.6”
Section 13.7.2.2, “CHECK TABLE Statement”
Section 15.2.4.1, “Corrupted MyISAM Tables”
Section 7.2, “Database Backup Methods”
Section 8.11.5, “External Locking”
Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”
Section 8.14.3, “General Thread States”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 16.1.3, “Handling MySQL Recovery with ZFS”
Section B.3.3.4, “How MySQL Handles a Full Disk”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 1.6, “How to Report Bugs or Problems”
Section 13.7.6.4, “KILL Statement”
Section B.3.7, “Known Issues in MySQL”
Section 13.2.6, “LOAD DATA Statement”
Section 19.3.4, “Maintenance of Partitions”
Section 15.7.2, “MERGE Table Problems”
Section 15.2.1, “MyISAM Startup Options”
Section 7.6, “MyISAM Table Maintenance and Crash Recovery”
Section 4.6.3.1, “myisamchk General Options”
Section 4.6.3, “`myisamchk` — MyISAM Table-Maintenance Utility”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 8.6.3, “Optimizing REPAIR TABLE Statements”
Section 6.2.2, “Privileges Provided by MySQL”
Section 15.2.4.2, “Problems from Tables Not Being Closed Properly”
Section B.3.6.1, “Problems with ALTER TABLE”
Section 2.11.10, “Rebuilding or Repairing Tables or Indexes”
Section 13.7.2.5, “REPAIR TABLE Statement”
Section 15.4.1, “Repairing and Checking CSV Tables”
Section 17.4.1.13, “Replication and FLUSH”
Section 17.4.1.24, “Replication and REPAIR TABLE”
Restoring an NDB backup to a later version of NDB Cluster
Section 19.6, “Restrictions and Limitations on Partitioning”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”
Section 13.3.3, “Statements That Cause an Implicit Commit”

Section 15.5, “The ARCHIVE Storage Engine”
Section 5.1.15, “The Server Shutdown Process”
Section 5.4.5, “The Slow Query Log”
Section 8.12.3.2, “Using Symbolic Links for MyISAM Tables on Unix”

REPEAT

Section 13.6.7.2, “DECLARE ... HANDLER Statement”
Section 20.1, “Defining Stored Programs”
Section 13.6.5, “Flow Control Statements”
Section 13.6.5.3, “ITERATE Statement”
Section 13.6.5.4, “LEAVE Statement”
Section 13.6.5.6, “REPEAT Statement”
Section 13.6.2, “Statement Labels”

REPLACE

Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 14.6.1.6, “AUTO_INCREMENT Handling in InnoDB”
Section 13.1.17.4, “CREATE TABLE ... SELECT Statement”
Section 13.1.19, “CREATE TRIGGER Statement”
Section 11.5, “Data Type Default Values”
Section 8.8.2, “EXPLAIN Output Format”
Section 13.8.2, “EXPLAIN Statement”
Section 8.8.3, “Extended EXPLAIN Output Format”
Section 12.16, “Information Functions”
Section 13.2.5.3, “INSERT DELAYED Statement”
Section 13.2.5, “INSERT Statement”
Section B.3.7, “Known Issues in MySQL”
Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 15.7.2, “MERGE Table Problems”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section B.3.2.8, “MySQL server has gone away”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 8.8.1, “Optimizing Queries with EXPLAIN”
Section 19.1, “Overview of Partitioning in MySQL”
Section 19.5, “Partition Selection”
Section 19.6.4, “Partitioning and Locking”
Section 13.2.8, “REPLACE Statement”
Section 19.6, “Restrictions and Limitations on Partitioning”
Section 13.2.10, “Subqueries”
Section 15.5, “The ARCHIVE Storage Engine”
Section 1.2.2, “The Main Features of MySQL”
Section 18.5.14.19, “The ndbinfo operations_per_fragment Table”
Section 13.2.11, “UPDATE Statement”
Section 1.3, “What Is New in MySQL 5.6”

REPLACE ... SELECT

Section 14.6.1.6, “AUTO_INCREMENT Handling in InnoDB”
Section B.3.7, “Known Issues in MySQL”

RESET

Section 13.7.6.3, “FLUSH Statement”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.6.6, “RESET Statement”
Section 13.3.3, “Statements That Cause an Implicit Commit”

RESET MASTER

Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 18.6.3, “Known Issues in NDB Cluster Replication”
Section 13.4.1.2, “RESET MASTER Statement”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”
Section 17.3.7, “Switching Sources During Failover”
Section 5.4.4, “The Binary Log”
Section 1.3, “What Is New in MySQL 5.6”

RESET QUERY CACHE

Section 8.14.5, “Query Cache Thread States”

RESET SLAVE

Section 13.4.2.1, “CHANGE MASTER TO Statement”
Section 17.3.10, “Delayed Replication”
Section 18.6.3, “Known Issues in NDB Cluster Replication”
NDB Cluster System Variables
Section 17.1.4, “Replication and Binary Logging Options and Variables”
Section 17.2.2.2, “Replication Metadata Repositories”
Section 13.4.1.2, “RESET MASTER Statement”
Section 13.4.2.3, “RESET SLAVE Statement”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 1.3, “What Is New in MySQL 5.6”

RESET SLAVE ALL

Section 13.4.2.1, “CHANGE MASTER TO Statement”

RESIGNAL

Section 13.6.7, “Condition Handling”
Section 13.6.7.8, “Condition Handling and OUT or INOUT Parameters”
Section 13.6.7.1, “DECLARE ... CONDITION Statement”
Section 13.6.7.2, “DECLARE ... HANDLER Statement”
Section 13.6.7.4, “RESIGNAL Statement”
Section 13.6.7.9, “Restrictions on Condition Handling”
Section 20.8, “Restrictions on Stored Programs”
Section 13.6.7.6, “Scope Rules for Handlers”
Section 13.6.7.5, “SIGNAL Statement”
Section 13.6.7.7, “The MySQL Diagnostics Area”

RETURN

Section 8.10.4, “Caching of Prepared Statements and Stored Programs”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 13.6.5, “Flow Control Statements”
Section 13.6.5.5, “LOOP Statement”
Section 20.8, “Restrictions on Stored Programs”
Section 13.6.5.7, “RETURN Statement”
Section 13.6.7.5, “SIGNAL Statement”

REVOKE

Section 6.2, “Access Control and Account Management”
Section 6.2.1, “Account User Names and Passwords”
Section 6.2.7, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 14.8.2, “Configuring InnoDB for Read-Only Operation”
Section 13.7.6.3, “FLUSH Statement”
Section 13.7.1.4, “GRANT Statement”

Section 6.2.3, “Grant Tables”
Section 8.12.4.1, “How MySQL Uses Memory”
Section 12.16, “Information Functions”
Section 5.1.12, “IPv6 Support”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section A.14, “MySQL 5.6 FAQ: Replication”
Section 1.7.2, “MySQL Differences from Standard SQL”
MySQL Glossary
Section 18.5.16.2, “NDB Cluster and MySQL Privileges”
Section 6.2.2, “Privileges Provided by MySQL”
Section 2.10.1.1, “Problems Running mysql_install_db”
Section 6.2.12, “Proxy Users”
Section 17.4.1.8, “Replication of CURRENT_USER()”
Section 17.4.1.21, “Replication of the mysql System Database”
Section 13.7.1.6, “REVOKE Statement”
Section 6.1.1, “Security Guidelines”
Section 5.1.7, “Server System Variables”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 20.4.6, “The Event Scheduler and MySQL Privileges”
Section 6.2.8, “When Privilege Changes Take Effect”

REVOKE ALL PRIVILEGES

Section 13.7.1.4, “GRANT Statement”
Section 6.2.2, “Privileges Provided by MySQL”

ROLLBACK

Section 14.7.2.2, “autocommit, Commit, and Rollback”
Section 14.6.1.5, “Converting Tables from MyISAM to InnoDB”
Section 12.16, “Information Functions”
Section 14.2, “InnoDB and the ACID Model”
Section 14.21.4, “InnoDB Error Handling”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.4.1.32, “Replication and Transactions”
Section B.3.4.5, “Rollback Failure for Nontransactional Tables”
Section 13.3.4, “SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Statements”
Section 5.1.7, “Server System Variables”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”
Section 13.3.2, “Statements That Cannot Be Rolled Back”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 20.7, “Stored Program Binary Logging”
Section 5.4.4, “The Binary Log”
Section 13.3, “Transactional and Locking Statements”
Section 20.3.1, “Trigger Syntax and Examples”

ROLLBACK TO SAVEPOINT

Section 13.3.4, “SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Statements”

ROLLBACK to SAVEPOINT

Section 20.3.1, “Trigger Syntax and Examples”

S

[index top]

SAVEPOINT

Section 13.3.4, “SAVEPOINT, ROLLBACK TO SAVEPOINT, and RELEASE SAVEPOINT Statements”

SELECT

Section 1.1, “About This Manual”
Section 6.2, “Access Control and Account Management”
Section 12.19.1, “Aggregate Function Descriptions”
Section 13.1.7, “ALTER TABLE Statement”
Section 13.1.9, “ALTER VIEW Statement”
Section 12.4.4, “Assignment Operators”
Section 6.4.4.3, “Audit Log File Formats”
Section 6.4.4.5, “Audit Log Filtering”
Section 14.6.1.6, “AUTO_INCREMENT Handling in InnoDB”
Section 14.7.2.2, “autocommit, Commit, and Rollback”
Section 17.1.4.4, “Binary Log Options and Variables”
Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 8.10.4, “Caching of Prepared Statements and Stored Programs”
Section 12.4.2, “Comparison Functions and Operators”
Section 8.3.8, “Comparison of B-Tree and Hash Indexes”
Section 8.11.3, “Concurrent Inserts”
Section 10.4, “Connection Character Sets and Collations”
Section 14.7.2.3, “Consistent Nonlocking Reads”
Section 14.6.1.5, “Converting Tables from MyISAM to InnoDB”
Section 13.1.11, “CREATE EVENT Statement”
Section 13.1.13, “CREATE INDEX Statement”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 13.1.17.4, “CREATE TABLE ... SELECT Statement”
Section 13.1.17, “CREATE TABLE Statement”
Section 13.1.17.2, “CREATE TEMPORARY TABLE Statement”
Section 13.1.20, “CREATE VIEW Statement”
Section 15.8.2.1, “Creating a FEDERATED Table Using CONNECTION”
Section 3.3.1, “Creating and Selecting a Database”
Section 13.6.6.2, “Cursor DECLARE Statement”
Section 13.6.6.3, “Cursor FETCH Statement”
Section 13.2.2, “DELETE Statement”
Section 13.2.10.8, “Derived Tables”
Section 8.4.3.2, “Disadvantages of Creating Many Tables in the Same Database”
Section 5.1.11.2, “DNS Lookups and the Host Cache”
Section 13.2.3, “DO Statement”
Section 3.2, “Entering Queries”
Section 20.4.2, “Event Scheduler Configuration”
Section 10.8.6, “Examples of the Effect of Collation”
Section 8.8.2, “EXPLAIN Output Format”
Section 13.8.2, “EXPLAIN Statement”
Section 8.8.3, “Extended EXPLAIN Output Format”
Section 15.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 14.21.2, “Forcing InnoDB Recovery”
Section 8.2.1.17, “Function Call Optimization”
Chapter 12, *Functions and Operators*
Section 8.14.3, “General Thread States”
Section 13.7.1.4, “GRANT Statement”
Section 13.2.4, “HANDLER Statement”
Section 19.2.7, “How MySQL Partitioning Handles NULL”
Section 8.10.3.1, “How the Query Cache Operates”
Section 14.7.5.3, “How to Minimize and Handle Deadlocks”
Section 1.6, “How to Report Bugs or Problems”
Section 8.9.3, “Index Hints”
Section 12.16, “Information Functions”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 13.2.5.2, “INSERT ... ON DUPLICATE KEY UPDATE Statement”

Section 13.2.5.1, “INSERT ... SELECT Statement”
Section 13.2.5.3, “INSERT DELAYED Statement”
Section 13.2.5, “INSERT Statement”
Section 8.11.1, “Internal Locking Methods”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 21.1, “Introduction”
Section 13.2.9.2, “JOIN Clause”
Section 9.3, “Keywords and Reserved Words”
Section 13.7.6.4, “KILL Statement”
Section B.3.7, “Known Issues in MySQL”
Section 18.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 13.2.7, “LOAD XML Statement”
Section 13.6.4.2, “Local Variable Scope and Resolution”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 14.7.2.4, “Locking Reads”
Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 19.3.1, “Management of RANGE and LIST Partitions”
Section 15.7.2, “MERGE Table Problems”
Section 8.3.5, “Multiple-Column Indexes”
Section 7.6.4, “MyISAM Table Optimization”
Section A.11, “MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section A.14, “MySQL 5.6 FAQ: Replication”
Section A.4, “MySQL 5.6 FAQ: Stored Procedures and Functions”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.6, “mysql Client Tips”
Section 6.5.2, “MySQL Enterprise Encryption Usage and Examples”
Section 1.7.1, “MySQL Extensions to Standard SQL”
MySQL Glossary
Chapter 22, *MySQL Performance Schema*
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”
Section 12.10.1, “Natural Language Full-Text Searches”
Section 18.5.10.1, “NDB Cluster Disk Data Objects”
Section 18.2.5, “NDB Cluster Example with Tables and Data”
Section 18.6.11, “NDB Cluster Replication Conflict Resolution”
Section 18.6.4, “NDB Cluster Replication Schema and Tables”
NDB Cluster Status Variables
NDB Cluster System Variables
Section 18.4.23, “`ndb_select_all` — Print Rows from an NDB Table”
Section 18.5.14, “ndbinfo: The NDB Cluster Information Database”
Section 19.3.5, “Obtaining Information About Partitions”
Section 14.13.2, “Online DDL Performance and Concurrency”
Section 8.3, “Optimization and Indexes”
Section B.3.5, “Optimizer-Related Issues”
Section 8.2.2.4, “Optimizing Derived Tables”
Section 8.5.3, “Optimizing InnoDB Read-Only Transactions”
Section 8.5.2, “Optimizing InnoDB Transaction Management”
Section 8.6.1, “Optimizing MyISAM Queries”
Section 8.8.1, “Optimizing Queries with EXPLAIN”
Section 8.2.1, “Optimizing SELECT Statements”
Section 8.2.2.1, “Optimizing Subqueries with Semijoin Transformations”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”
Section 8.2.4.2, “Optimizing UPDATE Statements”
Section 4.6.3.4, “Other myisamchk Options”
Section 19.4, “Partition Pruning”
Section 19.5, “Partition Selection”

Section 19.6.4, “Partitioning and Locking”
Section 22.6, “Performance Schema Instrument Naming Conventions”
Section 14.7.4, “Phantom Rows”
Section 6.2.2, “Privileges Provided by MySQL”
Section B.3.4.2, “Problems Using DATE Columns”
Section B.3.4.8, “Problems with Floating-Point Values”
Section 14.8.10, “Purge Configuration”
Section 8.10.3.2, “Query Cache SELECT Options”
Section 8.10.3.4, “Query Cache Status and Maintenance”
Section 8.14.5, “Query Cache Thread States”
Section 18.5.15, “Quick Reference: NDB Cluster SQL Statements”
Section 19.2.3.1, “RANGE COLUMNS partitioning”
Section 8.2.1.2, “Range Optimization”
Section 13.2.8, “REPLACE Statement”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.2, “Replication Implementation”
Section 17.4.1.6, “Replication of CREATE ... IF NOT EXISTS Statements”
Section 17.4.1.16, “Replication of Invoked Features”
Section 17.1.4.2, “Replication Source Options and Variables”
Section 20.8, “Restrictions on Stored Programs”
Section 3.3.4, “Retrieving Information from a Table”
Section 3.6.7, “Searching on Two Keys”
Section 2.10.4, “Securing the Initial MySQL Accounts”
Section 13.2.9.1, “SELECT ... INTO Statement”
Section 13.2.9, “SELECT Statement”
Section 3.3.4.1, “Selecting All Data”
Section 3.3.4.2, “Selecting Particular Rows”
Section 5.1.10, “Server SQL Modes”
Section 5.1.7, “Server System Variables”
Section 13.7.4.1, “SET Syntax for Variable Assignment”
Section 13.7.5.3, “SHOW BINLOG EVENTS Statement”
Section 13.7.5.11, “SHOW CREATE PROCEDURE Statement”
Section 13.7.5.14, “SHOW CREATE VIEW Statement”
Section 13.7.5.18, “SHOW ERRORS Statement”
Section 13.7.5.28, “SHOW PROCEDURE CODE Statement”
Section 13.7.5.29, “SHOW PROCEDURE STATUS Statement”
Section 13.7.5.30, “SHOW PROCESSLIST Statement”
Section 13.7.5.33, “SHOW RELAYLOG EVENTS Statement”
Section 13.7.5, “SHOW Statements”
Section 13.7.5.40, “SHOW VARIABLES Statement”
Section 13.7.5.41, “SHOW WARNINGS Statement”
Section B.3.4.7, “Solving Problems with No Matching Rows”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”
Statement Probes
Section 20.7, “Stored Program Binary Logging”
Section 20.2.1, “Stored Routine Syntax”
Section 9.1.1, “String Literals”
Section 13.2.10, “Subqueries”
Section 13.2.10.6, “Subqueries with EXISTS or NOT EXISTS”
Section 13.2.10.9, “Subquery Errors”
Section 8.11.2, “Table Locking Issues”
Section 15.5, “The ARCHIVE Storage Engine”
Section 5.4.4, “The Binary Log”
Section 11.3.5, “The ENUM Type”
Section 22.12.10.1, “The host_cache Table”
Section 21.5, “The INFORMATION_SCHEMA COLUMNS Table”
Section 21.8, “The INFORMATION_SCHEMA EVENTS Table”

Section 21.10, “The INFORMATION_SCHEMA GLOBAL_VARIABLES and SESSION_VARIABLES Tables”
Section 21.30.25, “The INFORMATION_SCHEMA INNODB_TRX Table”
Section 21.16, “The INFORMATION_SCHEMA PROCESSLIST Table”
Section 21.19, “The INFORMATION_SCHEMA ROUTINES Table”
Section 21.29, “The INFORMATION_SCHEMA VIEWS Table”
Section 1.2.2, “The Main Features of MySQL”
Section 15.7, “The MERGE Storage Engine”
Section 8.10.3, “The MySQL Query Cache”
Section 18.5.14.18, “The ndbinfo nodes Table”
Section 13.2.10.1, “The Subquery as Scalar Operand”
Section 22.12.10.3, “The threads Table”
Section 14.7.2.1, “Transaction Isolation Levels”
Section 20.3.1, “Trigger Syntax and Examples”
Section 12.3, “Type Conversion in Expression Evaluation”
Section 13.2.9.3, “UNION Clause”
Section 13.2.11, “UPDATE Statement”
Section 9.4, “User-Defined Variables”
Section 14.15.2.1, “Using InnoDB Transaction and Locking Information”
Section 4.2.2.1, “Using Options on the Command Line”
Section 8.4.2.4, “Using PROCEDURE ANALYSE”
Section 5.8.1.6, “Using Server Logs to Find Causes of Errors in mysqld”
Section 11.4.9, “Using Spatial Indexes”
Section 10.2.2, “UTF-8 for Metadata”
Section 20.5.1, “View Syntax”
Section 1.3, “What Is New in MySQL 5.6”
Section 8.2.1.1, “WHERE Clause Optimization”
Section B.3.3.5, “Where MySQL Stores Temporary Files”

SELECT *

Section 11.3.4, “The BLOB and TEXT Types”

SELECT * FROM

Section 18.5.14.17, “The ndbinfo memory_per_fragment Table”

SELECT * FROM t PARTITION ()

Section 19.1, “Overview of Partitioning in MySQL”

SELECT * INTO OUTFILE 'file_name' FROM tbl_name

Section 7.2, “Database Backup Methods”

SELECT ... FOR UPDATE

Section 14.1.2, “Best Practices for InnoDB Tables”
Section 14.7.5, “Deadlocks in InnoDB”
Section 14.7.5.3, “How to Minimize and Handle Deadlocks”
Section 14.7.1, “InnoDB Locking”
Section 14.7.2.4, “Locking Reads”
Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”

SELECT ... FROM

Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”

SELECT ... INTO

Section 13.1.11, “CREATE EVENT Statement”
Section 13.6.4.2, “Local Variable Scope and Resolution”
Section 17.4.1.15, “Replication and System Functions”
Section 13.2.9.1, “SELECT ... INTO Statement”

Section 1.7.2.1, “SELECT INTO TABLE Differences”
Section 13.2.9, “SELECT Statement”

SELECT ... INTO DUMPFILE

Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 5.1.7, “Server System Variables”

SELECT ... INTO OUTFILE

Section 1.1, “About This Manual”
Section 7.1, “Backup and Recovery Types”
Section 7.4.3, “Dumping Data in Delimited-Text Format with mysqldump”
Section 14.21.2, “Forcing InnoDB Recovery”
Section 13.2.6, “LOAD DATA Statement”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 9.1.7, “NULL Values”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.2.9.1, “SELECT ... INTO Statement”
Section 1.7.2.1, “SELECT INTO TABLE Differences”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 2.3.7, “Windows Platform Restrictions”

SELECT ... INTO OUTFILE 'file_name'

Section 13.2.9.1, “SELECT ... INTO Statement”

SELECT ... INTO var_list

Section 20.8, “Restrictions on Stored Programs”
Section 13.6.4, “Variables in Stored Programs”

SELECT ... LOCK IN SHARE MODE

Section 14.7.1, “InnoDB Locking”
Section 18.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 14.7.2.4, “Locking Reads”
Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 14.7.2.1, “Transaction Isolation Levels”

SELECT ... PARTITION

Section 1.3, “What Is New in MySQL 5.6”

SELECT DISTINCT

Configuring the Number of Sampled Pages for InnoDB Optimizer Statistics
Section 8.14.3, “General Thread States”
Section 8.2.2.1, “Optimizing Subqueries with Semijoin Transformations”

SET

Section 13.7.1.1, “ALTER USER Statement”
Section 12.4.4, “Assignment Operators”
Section 17.1.4.4, “Binary Log Options and Variables”
Section 14.8.8, “Configuring InnoDB I/O Capacity”
Section 14.8.11.2, “Configuring Non-Persistent Optimizer Statistics Parameters”
Section 10.4, “Connection Character Sets and Collations”
Section 20.1, “Defining Stored Programs”
Section 20.4.2, “Event Scheduler Configuration”
Chapter 12, *Functions and Operators*
Section 12.16, “Information Functions”
Section 14.14, “InnoDB Startup Options and System Variables”

Section 4.5.1.6, “mysql Client Tips”
Section 6.5.2, “MySQL Enterprise Encryption Usage and Examples”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 4.6.12, “`mysql_find_rows` — Extract SQL Statements from Files”
Section 18.6.11, “NDB Cluster Replication Conflict Resolution”
Section 12.4, “Operators”
Section 19.6.4, “Partitioning and Locking”
Section 8.10.3.3, “Query Cache Configuration”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.1.4.2, “Replication Source Options and Variables”
Section 13.4.2.3, “RESET SLAVE Statement”
Section 5.1.6, “Server Command Options”
Section 6.2.10, “Server Handling of Expired Passwords”
Section 5.1.10, “Server SQL Modes”
Section 5.1.7, “Server System Variables”
Section 13.7.4, “SET Statements”
Section 13.7.4.1, “SET Syntax for Variable Assignment”
Section 13.3.6, “SET TRANSACTION Statement”
Section 13.7.5.40, “SHOW VARIABLES Statement”
Section 12.1, “SQL Function and Operator Reference”
Section 20.7, “Stored Program Binary Logging”
Section 13.2.10, “Subqueries”
Section 5.1.8.1, “System Variable Privileges”
Section 13.6.7.7, “The MySQL Diagnostics Area”
Section 5.4.5, “The Slow Query Log”
Section 20.3.1, “Trigger Syntax and Examples”
Section 9.4, “User-Defined Variables”
Section 4.2.2.1, “Using Options on the Command Line”
Section 4.2.2.5, “Using Options to Set Program Variables”
Section 5.1.8, “Using System Variables”
Section 13.6.4, “Variables in Stored Programs”
Section 1.3, “What Is New in MySQL 5.6”

SET @@GLOBAL.gtid_purged

Section 4.5.4, “`mysqldump` — A Database Backup Program”

SET @@GLOBAL.ndb_slave_conflict_role = 'NONE'

NDB Cluster System Variables

SET autocommit

Section 8.5.5, “Bulk Data Loading for InnoDB Tables”
Section 13.3, “Transactional and Locking Statements”

SET autocommit = 0

Section 17.3.9, “Semisynchronous Replication”

SET CHARACTER SET

Section 10.4, “Connection Character Sets and Collations”
Section 13.7.4.2, “SET CHARACTER SET Statement”
Section 13.7.4, “SET Statements”
Section 10.9, “Unicode Support”

SET CHARACTER SET 'charset_name'

Section 10.4, “Connection Character Sets and Collations”

SET CHARACTER SET charset_name

Section 10.4, “Connection Character Sets and Collations”

SET GLOBAL

Section 14.5.2, “Change Buffer”
Section 14.8.3.3, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”
Section 14.8.8, “Configuring InnoDB I/O Capacity”
Section 14.8.9, “Configuring Spin Lock Polling”
Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 14.6.3.2, “File-Per-Table Tablespaces”
Section 13.7.1.4, “GRANT Statement”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.8.3.2, “Making the Buffer Pool Scan Resistant”
Section 8.10.2.2, “Multiple Key Caches”
Section 17.3.9.2, “Semisynchronous Replication Installation and Configuration”

SET GLOBAL sql_slave_skip_counter

Section 13.4.2.4, “SET GLOBAL sql_slave_skip_counter Statement”

SET NAMES

Section 10.3.6, “Character String Literal Character Set and Collation”
Section 10.5, “Configuring Application Character Set and Collation”
Section 10.4, “Connection Character Sets and Collations”
Section 10.6, “Error Message Character Set”
Section 13.2.6, “LOAD DATA Statement”
Section A.11, “MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 4.5.1.2, “mysql Client Commands”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 13.7.4.3, “SET NAMES Statement”
Section 13.7.4, “SET Statements”
Section 12.3, “Type Conversion in Expression Evaluation”
Section 10.9, “Unicode Support”
Section 10.2.2, “UTF-8 for Metadata”

SET NAMES 'charset_name'

Section 10.4, “Connection Character Sets and Collations”

SET NAMES 'cp1251'

Section 10.4, “Connection Character Sets and Collations”

SET NAMES charset_name

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”

SET NAMES default_character_set

Section 4.5.4, “`mysqldump` — A Database Backup Program”

SET PASSWORD

Section 13.7.1.1, “ALTER USER Statement”
Section 6.2.9, “Assigning Account Passwords”
Section 2.11.3, “Changes in MySQL 5.6”
Section B.3.2.4, “Client does not support authentication protocol”
Section 13.7.1.2, “CREATE USER Statement”
Section 6.1.2.1, “End-User Guidelines for Password Security”
Section 6.2.3, “Grant Tables”
Section 12.16, “Information Functions”
Section 2.5.5, “Installing MySQL on Linux Using RPM Packages from Oracle”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”
Section 6.1.2.4, “Password Hashing in MySQL”

Section 6.1.2.3, “Passwords and Logging”
Section 17.4.1.35, “Replication and Variables”
Section 17.4.1.8, “Replication of CURRENT_USER()”
Resetting the Root Password: Generic Instructions
Section 2.10.4, “Securing the Initial MySQL Accounts”
Section 6.2.10, “Server Handling of Expired Passwords”
Section 13.7.1.7, “SET PASSWORD Statement”
Section 13.7.4, “SET Statements”
Section 6.4.1.4, “SHA-256 Pluggable Authentication”
Section 6.2.4, “Specifying Account Names”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 6.4.3, “The Password Validation Plugin”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 1.3, “What Is New in MySQL 5.6”
Section 6.2.8, “When Privilege Changes Take Effect”

SET PASSWORD ... = PASSWORD()

Section 6.2.9, “Assigning Account Passwords”

SET SESSION

Section 5.1.8.1, “System Variable Privileges”

SET SESSION TRANSACTION ISOLATION LEVEL

Section 5.1.7, “Server System Variables”

SET SESSION TRANSACTION {READ WRITE | READ ONLY}

Section 5.1.7, “Server System Variables”

SET sql_log_bin = 0

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”

SET sql_log_bin=OFF

Section 5.4.4, “The Binary Log”

SET sql_mode='modes'

Section A.3, “MySQL 5.6 FAQ: Server SQL Mode”

SET TIMESTAMP = value

Section 8.14.1, “Accessing the Process List”

SET TRANSACTION

Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 13.3.6, “SET TRANSACTION Statement”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”
Section 14.7.2.1, “Transaction Isolation Levels”

SET TRANSACTION ISOLATION LEVEL

Section 5.1.7, “Server System Variables”
Section 13.7.4, “SET Statements”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”

SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED

Section 14.20.6, “The InnoDB memcached Plugin and Replication”

SET TRANSACTION {READ WRITE | READ ONLY}

Section 5.1.7, “Server System Variables”

SET var_name = value

Section 13.7.4, “SET Statements”

SHOW

Section 13.1.11, “CREATE EVENT Statement”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 3.3, “Creating and Using a Database”
Section 13.6.6.2, “Cursor DECLARE Statement”
Section 21.35, “Extensions to SHOW Statements”
Section 21.1, “Introduction”
Section 9.2.4, “Mapping of Identifiers to File Names”
Section A.14, “MySQL 5.6 FAQ: Replication”
Section 1.7.1, “MySQL Extensions to Standard SQL”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 22.1, “Performance Schema Quick Start”
Section 20.8, “Restrictions on Stored Programs”
Section 13.7.5.6, “SHOW COLUMNS Statement”
Section 13.7.5, “SHOW Statements”
Section 13.7.5.38, “SHOW TABLES Statement”
Section 13.4.1, “SQL Statements for Controlling Replication Source Servers”
Section 5.4.4, “The Binary Log”
Section 1.2.2, “The Main Features of MySQL”
Section 10.2.2, “UTF-8 for Metadata”

SHOW AUTHORS

Section 13.7.5.1, “SHOW AUTHORS Statement”

SHOW BINARY LOGS

Section 6.2.2, “Privileges Provided by MySQL”
Section 13.4.1.1, “PURGE BINARY LOGS Statement”
Section 13.7.5.2, “SHOW BINARY LOGS Statement”
Section 13.4.1, “SQL Statements for Controlling Replication Source Servers”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

SHOW BINLOG EVENTS

Section 17.1.3.1, “GTID Concepts”
Section 18.6.4, “NDB Cluster Replication Schema and Tables”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.6.6.5, “Restrictions on Server-Side Cursors”
Section 13.7.5.3, “SHOW BINLOG EVENTS Statement”
Section 13.4.1, “SQL Statements for Controlling Replication Source Servers”
Section 13.4.2.5, “START SLAVE Statement”

SHOW CHARACTER SET

Section 13.1.1, “ALTER DATABASE Statement”
Section 10.3.8, “Character Set Introducers”
Section 10.2, “Character Sets and Collations in MySQL”
Section 10.3.6, “Character String Literal Character Set and Collation”
Section 10.3.5, “Column Character Set and Collation”
Section 13.1.10, “CREATE DATABASE Statement”
Section 10.3.3, “Database Character Set and Collation”
Section 21.35, “Extensions to SHOW Statements”
Section A.11, “MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”

Section 13.7.5.4, “SHOW CHARACTER SET Statement”
Section 10.10, “Supported Character Sets and Collations”
Section 10.3.4, “Table Character Set and Collation”
Section 21.2, “The INFORMATION_SCHEMA CHARACTER_SETS Table”

SHOW COLLATION

Section 13.1.1, “ALTER DATABASE Statement”
Section 10.15, “Character Set Configuration”
Section 10.2, “Character Sets and Collations in MySQL”
Section 10.14.2, “Choosing a Collation ID”
Section 13.1.10, “CREATE DATABASE Statement”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 13.7.5.5, “SHOW COLLATION Statement”
Section 21.4, “The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table”
Section 21.3, “The INFORMATION_SCHEMA COLLATIONS Table”

SHOW COLUMNS

Section 13.8.2, “EXPLAIN Statement”
Section 21.35, “Extensions to SHOW Statements”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 22.1, “Performance Schema Quick Start”
Section 13.7.5.6, “SHOW COLUMNS Statement”
Section 21.5, “The INFORMATION_SCHEMA COLUMNS Table”
Section 21.30.1, “The INFORMATION_SCHEMA INNODB_BUFFER_PAGE Table”
Section 21.30.2, “The INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU Table”
Section 21.30.3, “The INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS Table”
Section 21.30.4, “The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables”
Section 21.30.6, “The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables”
Section 21.30.5, “The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables”
Section 21.30.7, “The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table”
Section 21.30.8, “The INFORMATION_SCHEMA INNODB_FT_CONFIG Table”
Section 21.30.9, “The INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD Table”
Section 21.30.10, “The INFORMATION_SCHEMA INNODB_FT_DELETED Table”
Section 21.30.11, “The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table”
Section 21.30.12, “The INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE Table”
Section 21.30.14, “The INFORMATION_SCHEMA INNODB_LOCK_WAITS Table”
Section 21.30.13, “The INFORMATION_SCHEMA INNODB_LOCKS Table”
Section 21.30.15, “The INFORMATION_SCHEMA INNODB_METRICS Table”
Section 21.30.16, “The INFORMATION_SCHEMA INNODB_SYS_COLUMNS Table”
Section 21.30.17, “The INFORMATION_SCHEMA INNODB_SYS_DATAFILES Table”
Section 21.30.18, “The INFORMATION_SCHEMA INNODB_SYS_FIELDS Table”
Section 21.30.19, “The INFORMATION_SCHEMA INNODB_SYS_FOREIGN Table”
Section 21.30.20, “The INFORMATION_SCHEMA INNODB_SYS_FOREIGN_COLS Table”
Section 21.30.21, “The INFORMATION_SCHEMA INNODB_SYS_INDEXES Table”
Section 21.30.22, “The INFORMATION_SCHEMA INNODB_SYS_TABLES Table”
Section 21.30.23, “The INFORMATION_SCHEMA INNODB_SYS_TABLESPACES Table”
Section 21.30.24, “The INFORMATION_SCHEMA INNODB_SYS_TABLESTATS View”
Section 21.30.25, “The INFORMATION_SCHEMA INNODB_TRX Table”

SHOW COLUMNS FROM tbl_name LIKE 'enum_col'

Section 11.3.5, “The ENUM Type”

SHOW CONTRIBUTORS

Section 13.7.5.7, “SHOW CONTRIBUTORS Statement”

SHOW COUNT()

Section 13.7.5.18, “SHOW ERRORS Statement”
Section 13.7.5.41, “SHOW WARNINGS Statement”

SHOW CREATE DATABASE

Section 5.1.7, “Server System Variables”
Section 13.7.5.8, “SHOW CREATE DATABASE Statement”

SHOW CREATE EVENT

Section 20.4.4, “Event Metadata”
Section 13.7.5.19, “SHOW EVENTS Statement”
Section 20.4.6, “The Event Scheduler and MySQL Privileges”

SHOW CREATE FUNCTION

Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 1.6, “How to Report Bugs or Problems”
Section A.4, “MySQL 5.6 FAQ: Stored Procedures and Functions”
Section 13.7.5.11, “SHOW CREATE PROCEDURE Statement”
Section 20.2.3, “Stored Routine Metadata”

SHOW CREATE PROCEDURE

Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 1.6, “How to Report Bugs or Problems”
Section A.4, “MySQL 5.6 FAQ: Stored Procedures and Functions”
Section 13.7.5.10, “SHOW CREATE FUNCTION Statement”
Section 20.2.3, “Stored Routine Metadata”

SHOW CREATE SCHEMA

Section 13.7.5.8, “SHOW CREATE DATABASE Statement”

SHOW CREATE TABLE

Section 13.1.7.1, “ALTER TABLE Partition Operations”
Section 13.1.7, “ALTER TABLE Statement”
Section 2.11.3, “Changes in MySQL 5.6”
Section 14.8.12, “Configuring the Merge Threshold for Index Pages”
Section 13.1.17, “CREATE TABLE Statement”
Section 11.5, “Data Type Default Values”
Section 2.12.3, “Downgrade Notes”
Section 13.8.2, “EXPLAIN Statement”
Section 13.1.17.5, “FOREIGN KEY Constraints”
Section 3.4, “Getting Information About Databases and Tables”
Section 15.8.2, “How to Create FEDERATED Tables”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 14.6.1.3, “Importing InnoDB Tables”
Section 19.2.5, “KEY Partitioning”
Section 19.3.1, “Management of RANGE and LIST Partitions”
Section 18.5.10.1, “NDB Cluster Disk Data Objects”
NDB Cluster System Variables
Section 18.1.7.1, “Noncompliance with SQL Syntax in NDB Cluster”
Section 19.3.5, “Obtaining Information About Partitions”
Section 22.1, “Performance Schema Quick Start”
Section 2.11.10, “Rebuilding or Repairing Tables or Indexes”
Section 5.1.10, “Server SQL Modes”
Section 5.1.7, “Server System Variables”
Section 13.7.5.6, “SHOW COLUMNS Statement”
Section 13.7.5.12, “SHOW CREATE TABLE Statement”

Section 13.1.17.6, “Silent Column Specification Changes”
Section 3.6.6, “Using Foreign Keys”

SHOW CREATE TRIGGER

Section 13.7.5.13, “SHOW CREATE TRIGGER Statement”
Section 20.3.2, “Trigger Metadata”

SHOW CREATE VIEW

Section 13.7.1.4, “GRANT Statement”
Section 6.2.2, “Privileges Provided by MySQL”
Section 20.9, “Restrictions on Views”
Section 13.7.5.14, “SHOW CREATE VIEW Statement”
Section 21.29, “The INFORMATION_SCHEMA VIEWS Table”
Section 20.5.5, “View Metadata”

SHOW DATABASES

Section 13.1.10, “CREATE DATABASE Statement”
Section 3.3, “Creating and Using a Database”
Section 21.35, “Extensions to SHOW Statements”
Section 3.4, “Getting Information About Databases and Tables”
Section 13.7.1.4, “GRANT Statement”
Section 6.2.3, “Grant Tables”
Section 9.2.3, “Identifier Case Sensitivity”
Section 21.1, “Introduction”
Section 18.5.16.2, “NDB Cluster and MySQL Privileges”
Section 18.5.14, “ndbinfo: The NDB Cluster Information Database”
Section 22.2, “Performance Schema Build Configuration”
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 13.7.5.15, “SHOW DATABASES Statement”
Section 21.20, “The INFORMATION_SCHEMA SCHEMATA Table”

SHOW ENGINE

Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.5.16, “SHOW ENGINE Statement”

SHOW ENGINE INNODB MUTEX

Section 13.7.5.16, “SHOW ENGINE Statement”
Section 1.3, “What Is New in MySQL 5.6”

SHOW ENGINE INNODB STATUS

Section 14.5.3, “Adaptive Hash Index”
Section 14.5.1, “Buffer Pool”
Section 14.8.2, “Configuring InnoDB for Read-Only Operation”
Section 14.6.1.5, “Converting Tables from MyISAM to InnoDB”
Section 14.7.5, “Deadlocks in InnoDB”
Section 14.17.2, “Enabling InnoDB Monitors”
Section B.2, “Error Information Interfaces”
Section 13.1.17.5, “FOREIGN KEY Constraints”
Section 14.7.5.3, “How to Minimize and Handle Deadlocks”
Section 14.15.5, “InnoDB INFORMATION_SCHEMA Buffer Pool Tables”
Section 14.15.6, “InnoDB INFORMATION_SCHEMA Metrics Table”
Section 14.15.3, “InnoDB INFORMATION_SCHEMA System Tables”
Section 14.7.1, “InnoDB Locking”
Section 14.17.3, “InnoDB Standard Monitor and Lock Monitor Output”
Section 14.14, “InnoDB Startup Options and System Variables”

Section 14.6.1.4, “Moving or Copying InnoDB Tables”
MySQL Glossary
Section 8.5.8, “Optimizing InnoDB Disk I/O”
Section 8.5.3, “Optimizing InnoDB Read-Only Transactions”
Section 14.8.10, “Purge Configuration”
Section 13.7.5.16, “SHOW ENGINE Statement”
Section 21.30.3, “The INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS Table”

SHOW ENGINE NDB STATUS

Section 18.2.2.3, “Initial Startup of NDB Cluster on Windows”
Section 18.5, “Management of NDB Cluster”
Section 18.6.4, “NDB Cluster Replication Schema and Tables”
Section 18.5.15, “Quick Reference: NDB Cluster SQL Statements”

SHOW ENGINE NDBCLUSTER STATUS

MySQL Server Options for NDB Cluster
Section 18.5.15, “Quick Reference: NDB Cluster SQL Statements”

SHOW ENGINE PERFORMANCE_SCHEMA STATUS

Section 22.7, “Performance Schema Status Monitoring”
Section 13.7.5.16, “SHOW ENGINE Statement”

SHOW ENGINES

Chapter 15, *Alternative Storage Engines*
Section 14.1.3, “Checking InnoDB Availability”
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
Section 18.5.9, “MySQL Server Usage for NDB Cluster”
Section 18.5.14, “ndbinfo: The NDB Cluster Information Database”
Section 22.2, “Performance Schema Build Configuration”
Section 22.1, “Performance Schema Quick Start”
Section 18.5.15, “Quick Reference: NDB Cluster SQL Statements”
Section 2.3.4.3, “Selecting a MySQL Server Type”
Section 5.1.7, “Server System Variables”
Section 13.7.5.17, “SHOW ENGINES Statement”
Section 15.5, “The ARCHIVE Storage Engine”
Section 15.6, “The BLACKHOLE Storage Engine”
Section 21.7, “The INFORMATION_SCHEMA ENGINES Table”

SHOW ERRORS

Section B.2, “Error Information Interfaces”
Section 13.6.7.3, “GET DIAGNOSTICS Statement”
Section 13.6.7.4, “RESIGNAL Statement”
Section 5.1.7, “Server System Variables”
Section 13.7.5.18, “SHOW ERRORS Statement”
Section 13.7.5.41, “SHOW WARNINGS Statement”
Section 13.6.7.5, “SIGNAL Statement”
Section 13.6.7.7, “The MySQL Diagnostics Area”

SHOW EVENTS

Section 20.4.4, “Event Metadata”
Section 17.4.1.16, “Replication of Invoked Features”
Section 13.7.5.19, “SHOW EVENTS Statement”
Section 20.4.6, “The Event Scheduler and MySQL Privileges”
Section 21.8, “The INFORMATION_SCHEMA EVENTS Table”

SHOW FULL COLUMNS

Section 13.1.17, “CREATE TABLE Statement”

Section 21.6, “The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table”

SHOW FULL PROCESSLIST

Section 8.14.1, “Accessing the Process List”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 14.13.2, “Online DDL Performance and Concurrency”

SHOW FULL TABLES

Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”

SHOW FUNCTION CODE

Section 13.7.5.28, “SHOW PROCEDURE CODE Statement”
Section 20.2.3, “Stored Routine Metadata”

SHOW FUNCTION STATUS

Section 13.7.5.29, “SHOW PROCEDURE STATUS Statement”
Section 20.2.3, “Stored Routine Metadata”

SHOW GLOBAL STATUS

NDB Cluster Status Variables
Section 5.1.7, “Server System Variables”
Section 21.9, “The INFORMATION_SCHEMA GLOBAL_STATUS and SESSION_STATUS Tables”

SHOW GLOBAL VARIABLES

Section 21.10, “The INFORMATION_SCHEMA GLOBAL_VARIABLES and SESSION_VARIABLES Tables”

SHOW GRANTS

Section 6.2, “Access Control and Account Management”
Section 6.2.7, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 13.7.1.4, “GRANT Statement”
Section 6.2.3, “Grant Tables”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.1.6, “REVOKE Statement”
Section 6.1.1, “Security Guidelines”
Section 13.7.5.22, “SHOW GRANTS Statement”
Section 13.7.5.27, “SHOW PRIVILEGES Statement”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”

SHOW INDEX

Section 13.7.2.1, “ANALYZE TABLE Statement”
Section 14.8.11.2, “Configuring Non-Persistent Optimizer Statistics Parameters”
Section 14.8.12, “Configuring the Merge Threshold for Index Pages”
Section 8.8.2, “EXPLAIN Output Format”
Section 13.8.2, “EXPLAIN Statement”
Section 8.9.3, “Index Hints”
Section 8.3.7, “InnoDB and MyISAM Index Statistics Collection”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 18.4.14, “`ndb_index_stat` — NDB Index Statistics Utility”
Section 4.6.3.4, “Other myisamchk Options”
Section 13.7.5.6, “SHOW COLUMNS Statement”
Section 13.7.5.23, “SHOW INDEX Statement”
Section 21.22, “The INFORMATION_SCHEMA STATISTICS Table”
Section 21.25, “The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table”

SHOW MASTER LOGS

Section 13.7.5.2, “SHOW BINARY LOGS Statement”

SHOW MASTER STATUS

Section 17.1.1.5, “Creating a Data Snapshot Using mysqldump”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 17.1.3.1, “GTID Concepts”
Section 17.4.5, “How to Report Replication Bugs or Problems”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 18.6.9, “NDB Cluster Backups With NDB Cluster Replication”
Section 17.1.1.4, “Obtaining the Replication Source Binary Log Coordinates”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”
Section 13.4.1, “SQL Statements for Controlling Replication Source Servers”
Section 14.20.6, “The InnoDB memcached Plugin and Replication”
Section 17.4.4, “Troubleshooting Replication”

SHOW OPEN TABLES

Section 13.7.5.25, “SHOW OPEN TABLES Statement”

SHOW PLUGINS

Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 13.7.3.3, “INSTALL PLUGIN Statement”
Section 5.5.1, “Installing and Uninstalling Plugins”
Section 6.4.4.1, “Installing MySQL Enterprise Audit”
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
MySQL Server Options for NDB Cluster
Section 18.5.14, “ndbinfo: The NDB Cluster Information Database”
Section 5.5.2, “Obtaining Server Plugin Information”
Section 6.4.1.6, “PAM Pluggable Authentication”
Chapter 19, *Partitioning*
Section 6.4.3.1, “Password Validation Plugin Installation”
Section 17.3.9.2, “Semisynchronous Replication Installation and Configuration”
Section 13.7.5.26, “SHOW PLUGINS Statement”
Section 6.4.1.8, “Socket Peer-Credential Pluggable Authentication”
Section 6.4.1.9, “Test Pluggable Authentication”
Section 21.31.2, “The INFORMATION_SCHEMA ndb_transid_mysql_connection_map Table”
Section 21.15, “The INFORMATION_SCHEMA PLUGINS Table”
Section 5.5.3.2, “Thread Pool Installation”
Section 1.3, “What Is New in MySQL 5.6”
Section 6.4.1.7, “Windows Pluggable Authentication”

SHOW PRIVILEGES

Section 13.7.5.27, “SHOW PRIVILEGES Statement”

SHOW PROCEDURE CODE

Section 13.7.5.20, “SHOW FUNCTION CODE Statement”
Section 20.2.3, “Stored Routine Metadata”

SHOW PROCEDURE STATUS

Section 13.7.5.21, “SHOW FUNCTION STATUS Statement”
Section 20.2.3, “Stored Routine Metadata”

SHOW PROCESSLIST

Section 8.14.1, “Accessing the Process List”
Section 13.4.2.1, “CHANGE MASTER TO Statement”
Section 17.1.5.1, “Checking Replication Status”
Command Probes
Section 5.1.11.1, “Connection Interfaces”

Connection Probes
Section 17.3.10, “Delayed Replication”
Section 20.4.2, “Event Scheduler Configuration”
Section 13.7.1.4, “GRANT Statement”
Section 12.16, “Information Functions”
Section 14.21.4, “InnoDB Error Handling”
Section 13.7.6.4, “KILL Statement”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 17.2.1.1, “Monitoring Replication Main Threads”
Section A.14, “MySQL 5.6 FAQ: Replication”
Section 18.5.9, “MySQL Server Usage for NDB Cluster”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 22.6, “Performance Schema Instrument Naming Conventions”
Section 22.12.5, “Performance Schema Stage Event Tables”
Section 6.2.2, “Privileges Provided by MySQL”
Query Execution Probes
Query Probes
Section 17.2.1, “Replication Threads”
Section 13.7.5.30, “SHOW PROCESSLIST Statement”
Section 13.7.5.31, “SHOW PROFILE Statement”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”
Section 13.4.2.5, “START SLAVE Statement”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”
Section 17.3.7, “Switching Sources During Failover”
Section 21.31.2, “The INFORMATION_SCHEMA ndb_transid_mysql_connection_map Table”
Section 21.16, “The INFORMATION_SCHEMA PROCESSLIST Table”
Section 18.5.14.22, “The ndbinfo server_operations Table”
Section 18.5.14.23, “The ndbinfo server_transactions Table”
Section 22.12.10.3, “The threads Table”
Section B.3.2.6, “Too many connections”
Section 17.4.4, “Troubleshooting Replication”

SHOW PROFILE

Section 8.14.3, “General Thread States”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 22.18.1, “Query Profiling Using Performance Schema”
Section 5.1.7, “Server System Variables”
Section 13.7.5.31, “SHOW PROFILE Statement”
Section 13.7.5.32, “SHOW PROFILES Statement”
Section 21.17, “The INFORMATION_SCHEMA PROFILING Table”

SHOW PROFILES

Section 2.9.7, “MySQL Source-Configuration Options”
Section 22.18.1, “Query Profiling Using Performance Schema”
Section 5.1.7, “Server System Variables”
Section 13.7.5.31, “SHOW PROFILE Statement”
Section 13.7.5.32, “SHOW PROFILES Statement”
Section 21.17, “The INFORMATION_SCHEMA PROFILING Table”

SHOW RELAYLOG EVENTS

Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.5.3, “SHOW BINLOG EVENTS Statement”
Section 13.7.5.33, “SHOW RELAYLOG EVENTS Statement”
Section 13.4.2, “SQL Statements for Controlling Replica Servers”

SHOW REPLICA | SLAVE STATUS

Section A.14, “MySQL 5.6 FAQ: Replication”

SHOW SCHEMAS

Section 13.7.5.15, “SHOW DATABASES Statement”

SHOW SESSION STATUS

NDB Cluster Status Variables
Section 21.9, “The INFORMATION_SCHEMA GLOBAL_STATUS and SESSION_STATUS Tables”

SHOW SESSION VARIABLES

Section 21.10, “The INFORMATION_SCHEMA GLOBAL_VARIABLES and SESSION_VARIABLES Tables”

SHOW SLAVE HOSTS

Section 17.1.5.1, “Checking Replication Status”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.1.4, “Replication and Binary Logging Options and Variables”
Section 17.1.4.2, “Replication Source Options and Variables”
Section 13.4.1, “SQL Statements for Controlling Replication Source Servers”

SHOW SLAVE STATUS

Section 13.4.2.1, “CHANGE MASTER TO Statement”
Section 17.1.5.1, “Checking Replication Status”
Section 17.3.10, “Delayed Replication”
Section B.2, “Error Information Interfaces”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 17.1.3.1, “GTID Concepts”
Section 17.4.5, “How to Report Replication Bugs or Problems”
Section 18.6.8, “Implementing Failover with NDB Cluster Replication”
Section 17.1.1.9, “Introducing Additional Replicas to an Existing Replication Environment”
Section 18.6.3, “Known Issues in NDB Cluster Replication”
Section 17.2.1.1, “Monitoring Replication Main Threads”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.4.1.1, “PURGE BINARY LOGS Statement”
Section 17.4.1.27, “Replica Errors During Replication”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.1.4, “Replication and Binary Logging Options and Variables”
Section 17.2.2.2, “Replication Metadata Repositories”
Section 8.14.7, “Replication Replica I/O Thread States”
Section 17.2.1, “Replication Threads”
Section 17.3.8, “Setting Up Replication to Use Encrypted Connections”
Section 13.7.5.24, “SHOW MASTER STATUS Statement”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”
Section 13.4.2, “SQL Statements for Controlling Replica Servers”
Section 13.4.2.5, “START SLAVE Statement”
Section 17.4.4, “Troubleshooting Replication”

SHOW STATUS

Section 17.1.5.1, “Checking Replication Status”
Section 18.3.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”
Section 13.2.5.3, “INSERT DELAYED Statement”
Section 18.5, “Management of NDB Cluster”
MySQL Server Options for NDB Cluster
Section 18.5.13, “NDB API Statistics Counters and Variables”
Section 18.6, “NDB Cluster Replication”
Section 8.10.3.4, “Query Cache Status and Maintenance”
Section 18.5.15, “Quick Reference: NDB Cluster SQL Statements”

Section 17.4.1.29, “Replication and Temporary Tables”
Section 17.4.1.30, “Replication Retries and Timeouts”
Section 20.8, “Restrictions on Stored Programs”
Section 17.3.9.3, “Semisynchronous Replication Monitoring”
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”
Section 13.7.5.36, “SHOW STATUS Statement”
Section 8.3.9, “Use of Index Extensions”

SHOW STATUS LIKE 'perf%'

Section 22.7, “Performance Schema Status Monitoring”

SHOW TABLE STATUS

Section 12.19.1, “Aggregate Function Descriptions”
Section 14.6.1.6, “AUTO_INCREMENT Handling in InnoDB”
Section 14.8.11.2, “Configuring Non-Persistent Optimizer Statistics Parameters”
Section 13.1.17, “CREATE TABLE Statement”
Section 14.6.1.1, “Creating InnoDB Tables”
Section 13.8.2, “EXPLAIN Statement”
Section 14.12.2, “File Space Management”
Section 14.23, “InnoDB Restrictions and Limitations”
Section 14.11, “InnoDB Row Formats”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 19.3.5, “Obtaining Information About Partitions”
Section 13.7.5.6, “SHOW COLUMNS Statement”
Section 13.7.5.12, “SHOW CREATE TABLE Statement”
Section 13.7.5.37, “SHOW TABLE STATUS Statement”
Section 15.5, “The ARCHIVE Storage Engine”
Section 21.23, “The INFORMATION_SCHEMA TABLES Table”

SHOW TABLES

Section 3.3.2, “Creating a Table”
Section 21.35, “Extensions to SHOW Statements”
Section 9.2.3, “Identifier Case Sensitivity”
Section 14.15, “InnoDB INFORMATION_SCHEMA Tables”
Section 21.1, “Introduction”
Section 9.2.4, “Mapping of Identifiers to File Names”
MySQL Glossary
Section 18.6.10, “NDB Cluster Replication: Bidirectional and Circular Replication”
Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 18.5.14, “ndbinfo: The NDB Cluster Information Database”
Section 5.1.6, “Server Command Options”
Section 13.7.5.37, “SHOW TABLE STATUS Statement”
Section 13.7.5.38, “SHOW TABLES Statement”
Section B.3.2.15, “Table 'tbl_name' doesn't exist”
Section B.3.6.2, “TEMPORARY Table Problems”
Section 21.23, “The INFORMATION_SCHEMA TABLES Table”
Section 6.4.5.3, “Using MySQL Enterprise Firewall”

SHOW TABLES FROM some_ndb_database

Section 18.5.16.2, “NDB Cluster and MySQL Privileges”

SHOW TRIGGERS

Section A.5, “MySQL 5.6 FAQ: Triggers”
Section 13.7.5.39, “SHOW TRIGGERS Statement”
Section 21.27, “The INFORMATION_SCHEMA TRIGGERS Table”

Section 20.3.2, “Trigger Metadata”

SHOW VARIABLES

Section 6.4.4.5, “Audit Log Filtering”
Section 20.4.2, “Event Scheduler Configuration”
Section 14.14, “InnoDB Startup Options and System Variables”
Section A.11, “MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 22.3, “Performance Schema Startup Configuration”
Section 22.18.1, “Query Profiling Using Performance Schema”
Section 18.5.15, “Quick Reference: NDB Cluster SQL Statements”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 5.7, “Running Multiple MySQL Instances on One Machine”
Section 17.3.9.3, “Semisynchronous Replication Monitoring”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 13.7.4.1, “SET Syntax for Variable Assignment”
Section 13.7.5.40, “SHOW VARIABLES Statement”
Section 18.6.6, “Starting NDB Cluster Replication (Single Replication Channel)”
Section 5.1.8, “Using System Variables”

SHOW WARNINGS

Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 13.1.7, “ALTER TABLE Statement”
Section 2.11.3, “Changes in MySQL 5.6”
Section 10.14.4.3, “Diagnostics During Index.xml Parsing”
Section 13.1.26, “DROP PROCEDURE and DROP FUNCTION Statements”
Section 13.1.28, “DROP TABLE Statement”
Section B.2, “Error Information Interfaces”
Section 8.8.2, “EXPLAIN Output Format”
Section 13.8.2, “EXPLAIN Statement”
Section 8.8.3, “Extended EXPLAIN Output Format”
Section 9.2.5, “Function Name Parsing and Resolution”
Section 13.6.7.3, “GET DIAGNOSTICS Statement”
Section 13.2.6, “LOAD DATA Statement”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 8.8.1, “Optimizing Queries with EXPLAIN”
Section 8.2.2.2, “Optimizing Subqueries with Materialization”
Section 8.2.2.1, “Optimizing Subqueries with Semijoin Transformations”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”
Section 1.7.3.1, “PRIMARY KEY and UNIQUE Index Constraints”
Section 12.21.4, “Rounding Behavior”
Section 5.1.7, “Server System Variables”
Section 13.7.5.18, “SHOW ERRORS Statement”
Section 13.7.5.41, “SHOW WARNINGS Statement”
Section 13.6.7.5, “SIGNAL Statement”
Section 13.6.7.7, “The MySQL Diagnostics Area”

SIGNAL

Section 13.6.7, “Condition Handling”
Section 13.6.7.1, “DECLARE ... CONDITION Statement”
Section 13.6.7.2, “DECLARE ... HANDLER Statement”
Section 12.16, “Information Functions”
Section 13.6.7.4, “RESIGNAL Statement”
Section 13.6.7.9, “Restrictions on Condition Handling”
Section 20.8, “Restrictions on Stored Programs”
Section 13.6.7.6, “Scope Rules for Handlers”
Section 13.6.7.5, “SIGNAL Statement”

Section 13.6.7.7, “The MySQL Diagnostics Area”

START SLAVE

Section 13.4.2.1, “CHANGE MASTER TO Statement”
Section 17.3.10, “Delayed Replication”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 17.1.3.1, “GTID Concepts”
Section 18.6.8, “Implementing Failover with NDB Cluster Replication”
Section 17.1.1.9, “Introducing Additional Replicas to an Existing Replication Environment”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 18.6.10, “NDB Cluster Replication: Bidirectional and Circular Replication”
Section 18.4.22, “`ndb_restore` — Restore an NDB Cluster Backup”
Section 6.1.2.3, “Passwords and Logging”
Section 17.1.5.2, “Pausing Replication on the Slave”
Section 17.4.1.27, “Replica Errors During Replication”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.3.5, “Replicating Different Databases to Different Replicas”
Section 17.1.4, “Replication and Binary Logging Options and Variables”
Section 17.2.1, “Replication Threads”
Section 13.4.2.3, “RESET SLAVE Statement”
Section 17.3.9.2, “Semisynchronous Replication Installation and Configuration”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”
Section 13.4.2.5, “START SLAVE Statement”
Section 18.6.6, “Starting NDB Cluster Replication (Single Replication Channel)”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 13.4.2.6, “STOP SLAVE Statement”
Section 17.3.7, “Switching Sources During Failover”
Section 17.4.4, “Troubleshooting Replication”
Section 17.1.3.3, “Using GTIDs for Failover and Scaleout”
Section 18.6.7, “Using Two Replication Channels for NDB Cluster Replication”
Section 1.3, “What Is New in MySQL 5.6”

START SLAVE UNTIL

Section 17.1.4.3, “Replica Server Options and Variables”

START SLAVE UNTIL SQL_AFTER_MTS_GAPS

Section 17.3.2, “Handling an Unexpected Halt of a Replica Server”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 13.4.2.5, “START SLAVE Statement”

START TRANSACTION

Section 14.7.2.2, “autocommit, Commit, and Rollback”
Section 13.6.1, “BEGIN ... END Compound Statement”
Section 13.7.6.3, “FLUSH Statement”
Section 14.7.5.3, “How to Minimize and Handle Deadlocks”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 14.7.2.4, “Locking Reads”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 8.5.3, “Optimizing InnoDB Read-Only Transactions”
Section 20.8, “Restrictions on Stored Programs”
Section 17.3.9, “Semisynchronous Replication”
Section 5.1.7, “Server System Variables”
Section 13.3.6, “SET TRANSACTION Statement”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 13.3, “Transactional and Locking Statements”
Section 20.3.1, “Trigger Syntax and Examples”

Section 13.3.7.2, “XA Transaction States”

START TRANSACTION READ ONLY

MySQL Glossary
Section 8.5.3, “Optimizing InnoDB Read-Only Transactions”
Section 1.3, “What Is New in MySQL 5.6”

START TRANSACTION WITH CONSISTENT SNAPSHOT

Section 14.7.2.3, “Consistent Nonlocking Reads”

STATS_PERSISTENT=0

Section 14.8.11.2, “Configuring Non-Persistent Optimizer Statistics Parameters”

STATS_PERSISTENT=1

Section 14.8.11.1, “Configuring Persistent Optimizer Statistics Parameters”

STOP SLAVE

Section 13.4.2.1, “CHANGE MASTER TO Statement”
Section 17.1.5.1, “Checking Replication Status”
Section 17.3.10, “Delayed Replication”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 17.1.1.9, “Introducing Additional Replicas to an Existing Replication Environment”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 17.1.5.2, “Pausing Replication on the Slave”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.1.4, “Replication and Binary Logging Options and Variables”
Section 13.4.1.2, “RESET MASTER Statement”
Section 13.4.2.3, “RESET SLAVE Statement”
Section 17.3.9.2, “Semisynchronous Replication Installation and Configuration”
Section 13.4.2.5, “START SLAVE Statement”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 13.4.2.6, “STOP SLAVE Statement”
Section 17.3.7, “Switching Sources During Failover”
Section 17.1.2.2, “Usage of Row-Based Logging and Replication”
Section 1.3, “What Is New in MySQL 5.6”

STOP SLAVE SQL_THREAD

Section 17.1.2.2, “Usage of Row-Based Logging and Replication”

T

[index top]

TRUNCATE TABLE

Section 14.20.5.5, “Adapting DML Statements to memcached Operations”
Section 18.5.7.1, “Adding NDB Cluster Data Nodes Online: General Issues”
Section 18.5.1, “Commands in the NDB Cluster Management Client”
Section 15.2.3.3, “Compressed Table Characteristics”
Section 14.6.1.5, “Converting Tables from MyISAM to InnoDB”
Section 13.1.19, “CREATE TRIGGER Statement”
Section 13.2.2, “DELETE Statement”
Section 5.1.11.2, “DNS Lookups and the Host Cache”
Section 22.4.3, “Event Pre-Filtering”
Section 15.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 22.12.9.5, “File I/O Summary Tables”
Section 14.6.3.2, “File-Per-Table Tablespaces”
Section 13.2.4, “HANDLER Statement”

Section 8.10.3.1, “How the Query Cache Operates”
Section 14.20.7, “InnoDB memcached Plugin Internals”
Section 18.1.7.2, “Limits and Differences of NDB Cluster from Standard MySQL Limits”
Section 18.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 19.3.4, “Maintenance of Partitions”
Section 19.3.1, “Management of RANGE and LIST Partitions”
Section 15.7.2, “MERGE Table Problems”
MySQL Glossary
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 18.4.9, “`ndb_delete_all` — Delete All Rows from an NDB Table”
Section 22.12.9.4, “Object Wait Summary Table”
Section 8.5.7, “Optimizing InnoDB DDL Operations”
Section 22.12.7, “Performance Schema Connection Tables”
Section 22.4.1, “Performance Schema Event Timing”
Section 22.11, “Performance Schema General Table Characteristics”
Section 22.12.9, “Performance Schema Summary Tables”
Section 18.6.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.4.1.20, “Replication and MEMORY Tables”
Section 17.4.1.34, “Replication and TRUNCATE TABLE”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.7, “Server System Variables”
Section 22.12.9.7, “Socket Summary Tables”
Section 22.12.9.2, “Stage Summary Tables”
Section 22.12.9.3, “Statement Summary Tables”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 22.12.7.1, “The accounts Table”
Section 22.12.3.1, “The cond_instances Table”
Section 22.12.5.1, “The events_stages_current Table”
Section 22.12.5.2, “The events_stages_history Table”
Section 22.12.5.3, “The events_stages_history_long Table”
Section 22.12.6.1, “The events_statements_current Table”
Section 22.12.6.2, “The events_statements_history Table”
Section 22.12.6.3, “The events_statements_history_long Table”
Section 22.12.4.1, “The events_waits_current Table”
Section 22.12.4.2, “The events_waits_history Table”
Section 22.12.4.3, “The events_waits_history_long Table”
Section 22.12.3.2, “The file_instances Table”
Section 22.12.10.1, “The host_cache Table”
Section 22.12.7.2, “The hosts Table”
Section 21.30.21, “The INFORMATION_SCHEMA INNODB_SYS_INDEXES Table”
Section 21.30.22, “The INFORMATION_SCHEMA INNODB_SYS_TABLES Table”
Section 14.20.6, “The InnoDB memcached Plugin and Replication”
Section 15.3, “The MEMORY Storage Engine”
Section 22.12.3.3, “The mutex_instances Table”
Section 22.12.10.2, “The performance_timers Table”
Section 22.12.3.4, “The rwlock_instances Table”
Section 22.12.8.1, “The session_account_connect_attrs Table”
Section 22.12.8.2, “The session_connect_attrs Table”
Section 22.12.2.1, “The setup_actors Table”
Section 22.12.2.2, “The setup_consumers Table”
Section 22.12.2.3, “The setup_instruments Table”
Section 22.12.2.4, “The setup_objects Table”
Section 22.12.2.5, “The setup_timers Table”
Section 22.12.3.5, “The socket_instances Table”
The table_io_waits_summary_by_index_usage Table

The table_io_waits_summary_by_table Table
The table_lock_waits_summary_by_table Table
Section 22.12.10.3, “The threads Table”
Section 22.12.7.3, “The users Table”
Section 13.1.33, “TRUNCATE TABLE Statement”
Section 22.12.9.1, “Wait Event Summary Tables”

U

[index top]

UNINSTALL PLUGIN

Section 13.7.6.3, “FLUSH Statement”
Section 8.12.4.1, “How MySQL Uses Memory”
Section 13.7.3.3, “INSTALL PLUGIN Statement”
Section 5.5.1, “Installing and Uninstalling Plugins”
Section 4.4.4, “`mysql_plugin` — Configure MySQL Server Plugins”
Section 6.4.1.6, “PAM Pluggable Authentication”
Section 22.17, “Performance Schema and Plugins”
Section 15.11.1, “Pluggable Storage Engine Architecture”
Section 13.7.5.26, “SHOW PLUGINS Statement”
Section 6.4.1.8, “Socket Peer-Credential Pluggable Authentication”
Section 6.4.1.9, “Test Pluggable Authentication”
Section 21.15, “The INFORMATION_SCHEMA PLUGINS Table”
Section 13.7.3.4, “UNINSTALL PLUGIN Statement”
Section 6.4.1.7, “Windows Pluggable Authentication”

UNION

Section 12.8.3, “Character Set and Collation of Function Results”
Section 13.1.17, “CREATE TABLE Statement”
Section 13.1.20, “CREATE VIEW Statement”
Section 8.8.2, “EXPLAIN Output Format”
Section 12.16, “Information Functions”
Section 13.2.5.2, “INSERT ... ON DUPLICATE KEY UPDATE Statement”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 11.1.6, “Numeric Type Attributes”
Section 8.2.2.1, “Optimizing Subqueries with Semijoin Transformations”
Section 8.2.1.2, “Range Optimization”
Section 3.6.7, “Searching on Two Keys”
Section 13.2.9.1, “SELECT ... INTO Statement”
Section 13.2.9, “SELECT Statement”
Section 5.1.9, “Server Status Variables”
Section 13.2.10, “Subqueries”
Section 15.7, “The MERGE Storage Engine”
Section 13.2.9.3, “UNION Clause”
Section 20.5.3, “Updatable and Insertable Views”
Section 8.4.2.4, “Using PROCEDURE ANALYSE”
Section 20.5.2, “View Processing Algorithms”
Section 20.5.1, “View Syntax”
Section 12.12, “XML Functions”

UNION ALL

Section 12.16, “Information Functions”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 13.2.9.3, “UNION Clause”
Section 20.5.3, “Updatable and Insertable Views”
Section 20.5.2, “View Processing Algorithms”

UNION DISTINCT

Section 13.2.9.3, “UNION Clause”

UNLOCK TABLES

Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 7.2, “Database Backup Methods”
Section 13.7.6.3, “FLUSH Statement”
Section 14.7.5.3, “How to Minimize and Handle Deadlocks”
Section 14.6.1.3, “Importing InnoDB Tables”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 20.8, “Restrictions on Stored Programs”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”
Section 13.3.3, “Statements That Cause an Implicit Commit”
Section 8.12.1, “System Factors”

UPDATE

Section 6.2, “Access Control and Account Management”
Section 6.2.6, “Access Control, Stage 2: Request Verification”
Section 6.2.7, “Adding Accounts, Assigning Privileges, and Dropping Accounts”
Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 12.4.4, “Assignment Operators”
Section 14.6.1.6, “AUTO_INCREMENT Handling in InnoDB”
Section 14.1.2, “Best Practices for InnoDB Tables”
Section 17.1.4.4, “Binary Log Options and Variables”
Section 8.5.5, “Bulk Data Loading for InnoDB Tables”
Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 8.10.4, “Caching of Prepared Statements and Stored Programs”
Section 14.5.2, “Change Buffer”
Section 13.7.2.2, “CHECK TABLE Statement”
Section 10.7, “Column Character Set Conversion”
Section 14.9.6, “Compression for OLTP Workloads”
Section 14.8.12, “Configuring the Merge Threshold for Index Pages”
Section 14.7.2.3, “Consistent Nonlocking Reads”
Section 1.7.3.3, “Constraints on Invalid Data”
Section 14.6.1.5, “Converting Tables from MyISAM to InnoDB”
Section 13.1.17.2, “CREATE TEMPORARY TABLE Statement”
Section 13.1.19, “CREATE TRIGGER Statement”
Section 13.1.20, “CREATE VIEW Statement”
Section 15.8.2.1, “Creating a FEDERATED Table Using CONNECTION”
Section 11.5, “Data Type Default Values”
Section 11.2.1, “Date and Time Data Type Syntax”
Section 14.7.5, “Deadlocks in InnoDB”
Section 8.8.2, “EXPLAIN Output Format”
Section 13.8.2, “EXPLAIN Statement”
Section 8.8.3, “Extended EXPLAIN Output Format”
Section 15.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 14.21.2, “Forcing InnoDB Recovery”
Section 1.7.2.3, “FOREIGN KEY Constraint Differences”
Section 13.1.17.5, “FOREIGN KEY Constraints”
Section 12.10.5, “Full-Text Restrictions”
Section 8.2.1.17, “Function Call Optimization”
Chapter 12, *Functions and Operators*
Section 8.14.3, “General Thread States”
Section 13.7.1.4, “GRANT Statement”
Section 6.2.3, “Grant Tables”

Section 8.10.3.1, “How the Query Cache Operates”
Section 8.9.3, “Index Hints”
Section 12.16, “Information Functions”
Section 14.7.1, “InnoDB Locking”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 13.2.5.2, “INSERT ... ON DUPLICATE KEY UPDATE Statement”
Section 13.2.5, “INSERT Statement”
Section 17.2.3.3, “Interactions Between Replication Filtering Options”
Section 8.11.1, “Internal Locking Methods”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 21.1, “Introduction”
Section 13.2.9.2, “JOIN Clause”
Section 13.7.6.4, “KILL Statement”
Section B.3.7, “Known Issues in MySQL”
Section 18.6.3, “Known Issues in NDB Cluster Replication”
Section 13.2.6, “LOAD DATA Statement”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 14.7.2.4, “Locking Reads”
Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 12.20, “Miscellaneous Functions”
Section A.4, “MySQL 5.6 FAQ: Stored Procedures and Functions”
Section 4.5.1.1, “mysql Client Options”
Section 4.5.1.6, “mysql Client Tips”
Section 1.7.1, “MySQL Extensions to Standard SQL”
MySQL Glossary
Section 4.6.8.2, “mysqlbinlog Row Event Display”
Section 18.5.10.1, “NDB Cluster Disk Data Objects”
Section 18.6.11, “NDB Cluster Replication Conflict Resolution”
Section 14.13.1, “Online DDL Operations”
Section 12.4, “Operators”
Section 13.7.2.4, “OPTIMIZE TABLE Statement”
Section 8.2.4, “Optimizing Data Change Statements”
Section 8.8.1, “Optimizing Queries with EXPLAIN”
Section 8.2.2, “Optimizing Subqueries and Derived Tables”
Section 11.1.7, “Out-of-Range and Overflow Handling”
Section 19.1, “Overview of Partitioning in MySQL”
Section 19.4, “Partition Pruning”
Section 19.5, “Partition Selection”
Section 19.6.4, “Partitioning and Locking”
Section 6.1.2.3, “Passwords and Logging”
Section 1.7.3.1, “PRIMARY KEY and UNIQUE Index Constraints”
Section 6.2.2, “Privileges Provided by MySQL”
Section B.3.4.2, “Problems Using DATE Columns”
Section 14.8.10, “Purge Configuration”
Section 17.4.1.27, “Replica Errors During Replication”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.4.1.17, “Replication and LIMIT”
Section 17.4.1.22, “Replication and the Query Optimizer”
Section 19.6, “Restrictions and Limitations on Partitioning”
Section 13.2.10.11, “Rewriting Subqueries as Joins”
Section 2.10.4, “Securing the Initial MySQL Accounts”
Section 3.3.4.1, “Selecting All Data”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.10, “Server SQL Modes”
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”
Section 13.7.5.41, “SHOW WARNINGS Statement”

Section 12.1, “SQL Function and Operator Reference”
Statement Probes
Section 13.2.10, “Subqueries”
Section 13.2.10.9, “Subquery Errors”
Section 8.11.2, “Table Locking Issues”
Section 15.5, “The ARCHIVE Storage Engine”
Section 10.8.5, “The binary Collation Compared to _bin Collations”
Section 5.4.4, “The Binary Log”
Section 15.6, “The BLACKHOLE Storage Engine”
Section 21.30.24, “The INFORMATION_SCHEMA INNODB_SYS_TABLESTATS View”
Section 21.29, “The INFORMATION_SCHEMA VIEWS Table”
Section 1.2.2, “The Main Features of MySQL”
Section 15.7, “The MERGE Storage Engine”
Section 15.2, “The MyISAM Storage Engine”
Section 5.1.15, “The Server Shutdown Process”
Section 14.7.2.1, “Transaction Isolation Levels”
Section 20.3.1, “Trigger Syntax and Examples”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 14.6.7, “Undo Logs”
Section 20.5.3, “Updatable and Insertable Views”
Section 1.7.2.2, “UPDATE Differences”
Section 13.2.11, “UPDATE Statement”
Section 17.1.2.2, “Usage of Row-Based Logging and Replication”
Section 1.3, “What Is New in MySQL 5.6”
Section 6.2.8, “When Privilege Changes Take Effect”
Section 8.2.1.1, “WHERE Clause Optimization”

UPDATE ... ()

Section 14.7.2.3, “Consistent Nonlocking Reads”

UPDATE ... WHERE

Section 14.7.5, “Deadlocks in InnoDB”

UPDATE ... WHERE ...

Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”

UPDATE IGNORE

Section 5.1.10, “Server SQL Modes”
Section 13.2.11, “UPDATE Statement”

UPDATE t1,t2 ...

Statement Probes

USE

Section 17.1.4.4, “Binary Log Options and Variables”
Section 7.4.5.2, “Copy a Database from one Server to Another”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 3.3.1, “Creating and Selecting a Database”
Section 3.3, “Creating and Using a Database”
Section 7.4.1, “Dumping Data in SQL Format with mysqldump”
Section 17.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 17.2.3.3, “Interactions Between Replication Filtering Options”
Section 21.1, “Introduction”
Section 4.5.1.1, “mysql Client Options”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 18.5.14, “ndbinfo: The NDB Cluster Information Database”

Section 7.4.2, “Reloading SQL-Format Backups”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 20.2.1, “Stored Routine Syntax”
Section 13.8.4, “USE Statement”

USE db2

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”

USE db_name

Section 4.5.1.1, “mysql Client Options”

USE test

Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”

W

[index top]

WHERE

Section 14.1.1, “Benefits of Using InnoDB Tables”

WHILE

Section 13.6.5, “Flow Control Statements”
Section 13.6.5.3, “ITERATE Statement”
Section 13.6.5.4, “LEAVE Statement”
Section 13.6.2, “Statement Labels”
Section 13.6.5.8, “WHILE Statement”

X

[index top]

XA COMMIT

Section 2.12.4, “Downgrading Binary and Package-based Installations on Unix/Linux”
Section 5.1.7, “Server System Variables”
Section 2.11.4, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 13.3.7.2, “XA Transaction States”

XA END

Section 13.3.7.3, “Restrictions on XA Transactions”
Section 13.3.7.1, “XA Transaction SQL Statements”
Section 13.3.7.2, “XA Transaction States”

XA PREPARE

Section 13.3.7.2, “XA Transaction States”

XA RECOVER

Section 2.12.4, “Downgrading Binary and Package-based Installations on Unix/Linux”
Section 2.11.4, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 13.3.7.1, “XA Transaction SQL Statements”
Section 13.3.7.2, “XA Transaction States”

XA ROLLBACK

Section 2.12.4, “Downgrading Binary and Package-based Installations on Unix/Linux”
Section 5.1.7, “Server System Variables”

Section 2.11.4, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 13.3.7.2, “XA Transaction States”

XA START

Section 13.3.7.3, “Restrictions on XA Transactions”
Section 13.3.7.1, “XA Transaction SQL Statements”
Section 13.3.7.2, “XA Transaction States”

XA START xid

Section 13.3.7.1, “XA Transaction SQL Statements”

Status Variable Index

A | B | C | D | F | H | I | K | L | M | N | O | P | Q | R | S | T | U | V

A

[index top]

Aborted_clients

Section B.3.2.10, “Communication Errors and Aborted Connections”
Section 5.1.9, “Server Status Variables”

Aborted_connects

Section B.3.2.10, “Communication Errors and Aborted Connections”
Section 5.1.9, “Server Status Variables”

Audit_log_current_size

Section 6.4.4.6, “Audit Log Reference”

Audit_log_event_max_drop_size

Section 6.4.4.6, “Audit Log Reference”

Audit_log_events

Section 6.4.4.6, “Audit Log Reference”

Audit_log_events_filtered

Section 6.4.4.6, “Audit Log Reference”

Audit_log_events_lost

Section 6.4.4.6, “Audit Log Reference”

Audit_log_events_written

Section 6.4.4.6, “Audit Log Reference”

Audit_log_total_size

Section 6.4.4.6, “Audit Log Reference”

Audit_log_write_waits

Section 6.4.4.6, “Audit Log Reference”

B

[index top]

Binlog_cache_disk_use
Section 17.1.4.4, “Binary Log Options and Variables”
Section 5.1.9, “Server Status Variables”
Section 5.4.4, “The Binary Log”

Binlog_cache_use
Section 17.1.4.4, “Binary Log Options and Variables”
Section 5.1.9, “Server Status Variables”
Section 5.4.4, “The Binary Log”

Binlog_stmt_cache_disk_use
Section 17.1.4.4, “Binary Log Options and Variables”
Section 5.1.9, “Server Status Variables”

Binlog_stmt_cache_use
Section 17.1.4.4, “Binary Log Options and Variables”
Section 5.1.9, “Server Status Variables”

Bytes_received
Section 5.1.9, “Server Status Variables”

Bytes_sent
Section 5.1.9, “Server Status Variables”

C

[index top]

Com_flush
Section 5.1.9, “Server Status Variables”

Com_stmt_reprepare
Section 8.10.4, “Caching of Prepared Statements and Stored Programs”

Compression
Section 5.1.9, “Server Status Variables”

Connection_control_delay_generated
Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 6.4.2.2, “Connection-Control System and Status Variables”

Connection_errors_accept
Section 5.1.9, “Server Status Variables”

Connection_errors_internal
Section 5.1.9, “Server Status Variables”

Connection_errors_max_connections
Section 5.1.11.1, “Connection Interfaces”
Section 5.1.9, “Server Status Variables”

Connection_errors_peer_address
Section 5.1.9, “Server Status Variables”

Connection_errors_select
Section 5.1.9, “Server Status Variables”

Connection_errors_tcpwrap
Section 5.1.9, “Server Status Variables”

Connection_errors_xxx
Section 5.1.11.2, “DNS Lookups and the Host Cache”
Section 5.1.9, “Server Status Variables”
Section 1.3, “What Is New in MySQL 5.6”

Connections
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”

Created_tmp_disk_tables
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”
Section 22.12.6.1, “The events_statements_current Table”

Created_tmp_files
Section 5.1.9, “Server Status Variables”

Created_tmp_tables
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”
Section 13.7.5.36, “SHOW STATUS Statement”
Section 22.12.6.1, “The events_statements_current Table”

D

[index top]

Delayed_errors
Section 5.1.9, “Server Status Variables”

Delayed_insert_threads
Section 13.2.5.3, “INSERT DELAYED Statement”
Section 5.1.9, “Server Status Variables”

Delayed_writes
Section 13.2.5.3, “INSERT DELAYED Statement”
Section 5.1.9, “Server Status Variables”

F

[index top]

Firewall_access_denied
Section 6.4.5.4, “MySQL Enterprise Firewall Reference”

Firewall_access_granted
Section 6.4.5.4, “MySQL Enterprise Firewall Reference”
Section 6.4.5.3, “Using MySQL Enterprise Firewall”

Firewall_access_suspicious
Section 6.4.5.4, “MySQL Enterprise Firewall Reference”

Firewall_cached_entries
Section 6.4.5.4, “MySQL Enterprise Firewall Reference”

Flush_commands
Section 5.1.9, “Server Status Variables”

H

[index top]

Handler_commit
Section 5.1.9, “Server Status Variables”

Handler_delete
Section 5.1.9, “Server Status Variables”

Handler_discover
NDB Cluster Status Variables

Handler_external_lock
Section 5.1.9, “Server Status Variables”

Handler_mrr_init
Section 5.1.9, “Server Status Variables”

Handler_prepare
Section 5.1.9, “Server Status Variables”

Handler_read_first
Section 5.1.9, “Server Status Variables”

Handler_read_key
Section 5.1.9, “Server Status Variables”

Handler_read_last
Section 5.1.9, “Server Status Variables”

Handler_read_next
Section 5.1.9, “Server Status Variables”
Section 8.3.9, “Use of Index Extensions”

Handler_read_prev
Section 5.1.9, “Server Status Variables”

Handler_read_rnd
Section 5.1.9, “Server Status Variables”

Handler_read_rnd_next
Section 5.1.9, “Server Status Variables”

Handler_rollback
Section 5.1.9, “Server Status Variables”

Handler_savepoint
Section 5.1.9, “Server Status Variables”

Handler_savepoint_rollback
Section 5.1.9, “Server Status Variables”

Handler_update
Section 5.1.9, “Server Status Variables”

Handler_write
Section 5.1.9, “Server Status Variables”

I

[index top]

Innodb_available_undo_logs
Section 5.1.9, “Server Status Variables”

Innodb_buffer_pool_bytes_data
Section 5.1.9, “Server Status Variables”

Innodb_buffer_pool_bytes_dirty
Section 5.1.9, “Server Status Variables”

Innodb_buffer_pool_dump_status
Section 5.1.9, “Server Status Variables”

Innodb_buffer_pool_load_status
Section 5.1.9, “Server Status Variables”

Innodb_buffer_pool_pages_data
Section 5.1.9, “Server Status Variables”

Innodb_buffer_pool_pages_dirty
Section 5.1.9, “Server Status Variables”

Innodb_buffer_pool_pages_flushed
Section 5.1.9, “Server Status Variables”

Innodb_buffer_pool_pages_free
Section 5.1.9, “Server Status Variables”

Innodb_buffer_pool_pages_latched
Section 5.1.9, “Server Status Variables”

Innodb_buffer_pool_pages_misc
Section 5.1.9, “Server Status Variables”

Innodb_buffer_pool_pages_total
Section 5.1.9, “Server Status Variables”

Innodb_buffer_pool_read_ahead
Section 14.8.3.3, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”

Section 14.14, “InnoDB Startup Options and System Variables”
Section 5.1.9, “Server Status Variables”

Innodb_buffer_pool_read_ahead_evicted

Section 14.8.3.3, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 5.1.9, “Server Status Variables”

Innodb_buffer_pool_read_ahead_rnd

Section 14.8.3.3, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”
Section 5.1.9, “Server Status Variables”

Innodb_buffer_pool_read_requests

Section 5.1.9, “Server Status Variables”

Innodb_buffer_pool_reads

Section 5.1.9, “Server Status Variables”

Innodb_buffer_pool_wait_free

Section 5.1.9, “Server Status Variables”

Innodb_buffer_pool_write_requests

Section 5.1.9, “Server Status Variables”

Innodb_data_fsyncs

Section 14.14, “InnoDB Startup Options and System Variables”
Section 5.1.9, “Server Status Variables”

Innodb_data_pending_fsyncs

Section 5.1.9, “Server Status Variables”

Innodb_data_pending_reads

Section 5.1.9, “Server Status Variables”

Innodb_data_pending_writes

Section 5.1.9, “Server Status Variables”

Innodb_data_read

Section 5.1.9, “Server Status Variables”

Innodb_data_reads

Section 5.1.9, “Server Status Variables”

Innodb_data_writes

Section 5.1.9, “Server Status Variables”

Innodb_data_written

Section 5.1.9, “Server Status Variables”

Innodb_dblwr_pages_written

Section 5.1.9, “Server Status Variables”

Innodb_dblwr_writes

Section 5.1.9, “Server Status Variables”

Innodb_have_atomic_builtins
Section 5.1.9, “Server Status Variables”

Innodb_log_waits
Section 5.1.9, “Server Status Variables”

Innodb_log_write_requests
Section 5.1.9, “Server Status Variables”

Innodb_log_writes
Section 5.1.9, “Server Status Variables”

Innodb_num_open_files
Section 5.1.9, “Server Status Variables”

Innodb_os_log_fsyncs
Section 5.1.9, “Server Status Variables”

Innodb_os_log_pending_fsyncs
Section 5.1.9, “Server Status Variables”

Innodb_os_log_pending_writes
Section 5.1.9, “Server Status Variables”

Innodb_os_log_written
Section 14.14, “InnoDB Startup Options and System Variables”
Section 5.1.9, “Server Status Variables”

Innodb_page_size
Section 5.1.9, “Server Status Variables”

Innodb_pages_created
Section 5.1.9, “Server Status Variables”

Innodb_pages_read
Section 5.1.9, “Server Status Variables”

Innodb_pages_written
Section 5.1.9, “Server Status Variables”

Innodb_row_lock_current_waits
Section 5.1.9, “Server Status Variables”

Innodb_row_lock_time
Section 5.1.9, “Server Status Variables”

Innodb_row_lock_time_avg
Section 5.1.9, “Server Status Variables”

Innodb_row_lock_time_max
Section 5.1.9, “Server Status Variables”

Innodb_row_lock_waits
Section 5.1.9, “Server Status Variables”

Innodb_rows_deleted
Section 5.1.9, “Server Status Variables”

Innodb_rows_inserted
Section 5.1.9, “Server Status Variables”

Innodb_rows_read
Section 5.1.9, “Server Status Variables”

Innodb_rows_updated
Section 5.1.9, “Server Status Variables”

Innodb_truncated_status_writes
Section 5.1.9, “Server Status Variables”

K

[index top]

Key_blocks_not_flushed
Section 5.1.9, “Server Status Variables”

Key_blocks_unused
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”

Key_blocks_used
Section 5.1.9, “Server Status Variables”

Key_read_requests
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”

Key_reads
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”

Key_write_requests
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”

Key_writes
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”

L

[index top]

Last_query_cost
Section 5.1.9, “Server Status Variables”

Last_query_partial_plans
Section 5.1.9, “Server Status Variables”

M

[index top]

Max_used_connections
Section 13.7.6.3, “FLUSH Statement”
Section 5.1.9, “Server Status Variables”

N

[index top]

Ndb_api_adaptive_send_deferred_count
NDB Cluster Status Variables

Ndb_api_adaptive_send_deferred_count_session
NDB Cluster Status Variables

Ndb_api_adaptive_send_deferred_count_slave
NDB Cluster Status Variables

Ndb_api_adaptive_send_forced_count
NDB Cluster Status Variables

Ndb_api_adaptive_send_forced_count_session
NDB Cluster Status Variables

Ndb_api_adaptive_send_forced_count_slave
NDB Cluster Status Variables

Ndb_api_adaptive_send_unforced_count
NDB Cluster Status Variables

Ndb_api_adaptive_send_unforced_count_session
NDB Cluster Status Variables

Ndb_api_adaptive_send_unforced_count_slave
NDB Cluster Status Variables

Ndb_api_bytes_received_count
Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_bytes_received_count_session
Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_bytes_received_count_slave
Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_bytes_sent_count
Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_bytes_sent_count_session

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_bytes_sent_count_slave

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_event_bytes_count

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_event_bytes_count_injector

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_event_data_count

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_event_data_count_injector

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_event_nondata_count

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_event_nondata_count_injector

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_pk_op_count

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_pk_op_count_session

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_pk_op_count_slave

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_pruned_scan_count

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_pruned_scan_count_session

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_pruned_scan_count_slave

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_range_scan_count

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_range_scan_count_session

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_range_scan_count_slave

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_read_row_count

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_read_row_count_session

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_read_row_count_slave

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_scan_batch_count

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_scan_batch_count_session

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_scan_batch_count_slave

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_table_scan_count

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_table_scan_count_session

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_table_scan_count_slave

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_abort_count

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_abort_count_session

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_abort_count_slave

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_close_count

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_close_count_session

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_close_count_slave

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_commit_count

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_commit_count_session

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_commit_count_slave

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_local_read_row_count

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_local_read_row_count_session

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_local_read_row_count_slave

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_start_count

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_start_count_session

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_trans_start_count_slave

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_uk_op_count

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_uk_op_count_session

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_uk_op_count_slave

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_exec_complete_count

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_exec_complete_count_session

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_exec_complete_count_slave

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_meta_request_count

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_meta_request_count_session

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_meta_request_count_slave

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_nanos_count

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_nanos_count_session

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_nanos_count_slave

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_scan_result_count

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_scan_result_count_session

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_api_wait_scan_result_count_slave

Section 18.5.13, “NDB API Statistics Counters and Variables”
NDB Cluster Status Variables

Ndb_cluster_node_id
NDB Cluster Status Variables

Ndb_config_from_host
NDB Cluster Status Variables

Ndb_config_from_port
NDB Cluster Status Variables

Ndb_conflict_fn_epoch
Section 18.6.11, “NDB Cluster Replication Conflict Resolution”
NDB Cluster Status Variables

Ndb_conflict_fn_epoch2
Section 18.6.11, “NDB Cluster Replication Conflict Resolution”
NDB Cluster Status Variables

Ndb_conflict_fn_epoch2_trans
Section 18.6.11, “NDB Cluster Replication Conflict Resolution”
NDB Cluster Status Variables

Ndb_conflict_fn_epoch_trans
Section 18.6.11, “NDB Cluster Replication Conflict Resolution”
NDB Cluster Status Variables

Ndb_conflict_fn_max
Section 18.6.11, “NDB Cluster Replication Conflict Resolution”
NDB Cluster Status Variables

Ndb_conflict_fn_max_del_win
NDB Cluster Status Variables

Ndb_conflict_fn_old
Section 18.6.11, “NDB Cluster Replication Conflict Resolution”
NDB Cluster Status Variables

Ndb_conflict_last_conflict_epoch
NDB Cluster Status Variables

Ndb_conflict_last_stable_epoch
NDB Cluster Status Variables

Ndb_conflict_reflected_op_discard_count
Section 18.6.11, “NDB Cluster Replication Conflict Resolution”
NDB Cluster Status Variables

Ndb_conflict_reflected_op_prepare_count
Section 18.6.11, “NDB Cluster Replication Conflict Resolution”
NDB Cluster Status Variables

Ndb_conflict_refresh_op_count
NDB Cluster Status Variables

Ndb_conflict_trans_conflict_commit_count
NDB Cluster Status Variables

Ndb_conflict_trans_detect_iter_count

NDB Cluster Status Variables

Ndb_conflict_trans_reject_count

NDB Cluster Status Variables

Ndb_conflict_trans_row_conflict_count

NDB Cluster Status Variables

Ndb_conflict_trans_row_reject_count

Section 18.6.11, “NDB Cluster Replication Conflict Resolution”
NDB Cluster Status Variables

Ndb_epoch_delete_delete_count

NDB Cluster Status Variables

Ndb_execute_count

NDB Cluster Status Variables

Ndb_last_commit_epoch_server

NDB Cluster Status Variables

Ndb_last_commit_epoch_session

NDB Cluster Status Variables

Ndb_number_of_data_nodes

NDB Cluster Status Variables

Ndb_pruned_scan_count

NDB Cluster Status Variables

Ndb_pushed_queries_defined

NDB Cluster Status Variables
NDB Cluster System Variables

Ndb_pushed_queries_dropped

NDB Cluster Status Variables
NDB Cluster System Variables

Ndb_pushed_queries_executed

NDB Cluster Status Variables
NDB Cluster System Variables

Ndb_pushed_reads

NDB Cluster Status Variables
NDB Cluster System Variables

Ndb_scan_count

NDB Cluster Status Variables

Ndb_slave_max_replicated_epoch

NDB Cluster Status Variables

Not_flushed_delayed_rows

Section 13.2.5.3, “INSERT DELAYED Statement”

Section 5.1.9, “Server Status Variables”

O

[index top]

Open_files
Section 5.1.9, “Server Status Variables”

Open_streams
Section 5.1.9, “Server Status Variables”

Open_table_definitions
Section 5.1.9, “Server Status Variables”

Open_tables
Section 5.1.9, “Server Status Variables”

Opened_files
Section 5.1.9, “Server Status Variables”

Opened_table_definitions
Section 5.1.9, “Server Status Variables”

Opened_tables
Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”

P

[index top]

Performance_schema_accounts_lost
Section 22.16, “Performance Schema Status Variables”

Performance_schema_cond_classes_lost
Section 22.16, “Performance Schema Status Variables”

Performance_schema_cond_instances_lost
Section 22.16, “Performance Schema Status Variables”

Performance_schema_digest_lost
Section 22.16, “Performance Schema Status Variables”
Section 22.15, “Performance Schema System Variables”

Performance_schema_file_classes_lost
Section 22.16, “Performance Schema Status Variables”

Performance_schema_file_handles_lost
Section 22.16, “Performance Schema Status Variables”

Performance_schema_file_instances_lost
Section 22.16, “Performance Schema Status Variables”

Performance_schema_hosts_lost
Section 22.16, “Performance Schema Status Variables”

Performance_schema_locker_lost
Section 22.16, “Performance Schema Status Variables”

Performance_schema_mutex_classes_lost
Section 22.7, “Performance Schema Status Monitoring”
Section 22.16, “Performance Schema Status Variables”

Performance_schema_mutex_instances_lost
Section 22.7, “Performance Schema Status Monitoring”
Section 22.16, “Performance Schema Status Variables”

Performance_schema_rwlock_classes_lost
Section 22.16, “Performance Schema Status Variables”

Performance_schema_rwlock_instances_lost
Section 22.16, “Performance Schema Status Variables”

Performance_schema_session_connect_attrs_lost
Section 22.12.8, “Performance Schema Connection Attribute Tables”
Section 22.16, “Performance Schema Status Variables”
Section 22.15, “Performance Schema System Variables”

Performance_schema_socket_classes_lost
Section 22.16, “Performance Schema Status Variables”

Performance_schema_socket_instances_lost
Section 22.16, “Performance Schema Status Variables”

Performance_schema_stage_classes_lost
Section 22.16, “Performance Schema Status Variables”

Performance_schema_statement_classes_lost
Section 22.16, “Performance Schema Status Variables”

Performance_schema_table_handles_lost
Section 22.16, “Performance Schema Status Variables”

Performance_schema_table_instances_lost
Section 22.16, “Performance Schema Status Variables”

Performance_schema_thread_classes_lost
Section 22.16, “Performance Schema Status Variables”

Performance_schema_thread_instances_lost
Section 22.16, “Performance Schema Status Variables”
Section 22.15, “Performance Schema System Variables”

Performance_schema_users_lost
Section 22.16, “Performance Schema Status Variables”

Prepared_stmt_count
Section 5.1.9, “Server Status Variables”

Q

[index top]

Qcache_free_blocks
Section 8.10.3.3, “Query Cache Configuration”
Section 8.10.3.4, “Query Cache Status and Maintenance”
Section 5.1.9, “Server Status Variables”

Qcache_free_memory
Section 5.1.9, “Server Status Variables”

Qcache_hits
Section 8.10.3.1, “How the Query Cache Operates”
Section 5.1.9, “Server Status Variables”

Qcache_inserts
Section 5.1.9, “Server Status Variables”

Qcache_lowmem_prunes
Section 8.10.3.3, “Query Cache Configuration”
Section 8.10.3.4, “Query Cache Status and Maintenance”
Section 5.1.9, “Server Status Variables”

Qcache_not_cached
Section 5.1.9, “Server Status Variables”

Qcache_queries_in_cache
Section 8.10.3.3, “Query Cache Configuration”
Section 5.1.9, “Server Status Variables”

Qcache_total_blocks
Section 8.10.3.3, “Query Cache Configuration”
Section 8.10.3.4, “Query Cache Status and Maintenance”
Section 5.1.9, “Server Status Variables”

Queries
Section 5.1.9, “Server Status Variables”

Questions
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 5.1.9, “Server Status Variables”

R

[index top]

Rpl_semi_sync_master_clients
Section 17.3.9.1, “Semisynchronous Replication Administrative Interface”
Section 17.3.9.3, “Semisynchronous Replication Monitoring”
Section 5.1.9, “Server Status Variables”

Rpl_semi_sync_master_net_avg_wait_time
Section 5.1.9, “Server Status Variables”

Rpl_semi_sync_master_net_wait_time
Section 5.1.9, “Server Status Variables”

Rpl_semi_sync_master_net_waits
Section 5.1.9, “Server Status Variables”

Rpl_semi_sync_master_no_times
Section 5.1.9, “Server Status Variables”

Rpl_semi_sync_master_no_tx
Section 17.3.9.1, “Semisynchronous Replication Administrative Interface”
Section 17.3.9.3, “Semisynchronous Replication Monitoring”
Section 5.1.9, “Server Status Variables”

Rpl_semi_sync_master_status
Section 17.3.9.1, “Semisynchronous Replication Administrative Interface”
Section 17.3.9.3, “Semisynchronous Replication Monitoring”
Section 5.1.9, “Server Status Variables”

Rpl_semi_sync_master_timefunc_failures
Section 5.1.9, “Server Status Variables”

Rpl_semi_sync_master_tx_avg_wait_time
Section 5.1.9, “Server Status Variables”

Rpl_semi_sync_master_tx_wait_time
Section 5.1.9, “Server Status Variables”

Rpl_semi_sync_master_tx_waits
Section 5.1.9, “Server Status Variables”

Rpl_semi_sync_master_wait_pos_backtraverse
Section 5.1.9, “Server Status Variables”

Rpl_semi_sync_master_wait_sessions
Section 5.1.9, “Server Status Variables”

Rpl_semi_sync_master_yes_tx
Section 17.3.9.1, “Semisynchronous Replication Administrative Interface”
Section 17.3.9.3, “Semisynchronous Replication Monitoring”
Section 5.1.9, “Server Status Variables”

Rpl_semi_sync_slave_status
Section 17.3.9.1, “Semisynchronous Replication Administrative Interface”
Section 17.3.9.3, “Semisynchronous Replication Monitoring”
Section 5.1.9, “Server Status Variables”

Rsa_public_key
Section 5.1.9, “Server Status Variables”
Section 6.4.1.4, “SHA-256 Pluggable Authentication”
Section 6.3.4, “SSL Library-Dependent Capabilities”

S

[index top]

Select_full_join

Section 5.1.9, “Server Status Variables”
Section 22.12.6.1, “The events_statements_current Table”

Select_full_range_join

Section 5.1.9, “Server Status Variables”
Section 22.12.6.1, “The events_statements_current Table”

Select_range

Section 5.1.9, “Server Status Variables”
Section 22.12.6.1, “The events_statements_current Table”

Select_range_check

Section 5.1.9, “Server Status Variables”
Section 22.12.6.1, “The events_statements_current Table”

Select_scan

Section 5.1.9, “Server Status Variables”
Section 22.12.6.1, “The events_statements_current Table”

Slave_heartbeat_period

Section 5.1.9, “Server Status Variables”

Slave_last_heartbeat

Section 17.1.5.1, “Checking Replication Status”
Section 5.1.9, “Server Status Variables”

Slave_open_temp_tables

Section 17.4.1.29, “Replication and Temporary Tables”
Section 5.1.9, “Server Status Variables”

Slave_received_heartbeats

Section 5.1.9, “Server Status Variables”

Slave_retried_transactions

Section 5.1.9, “Server Status Variables”

Slave_rows_last_search_algorithm_used

Section 5.1.9, “Server Status Variables”

Slave_running

Section 5.1.9, “Server Status Variables”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”

Slow_launch_threads

Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”

Slow_queries

Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”

Sort_merge_passes

Section 8.2.1.13, “ORDER BY Optimization”

Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”
Section 22.12.6.1, “The events_statements_current Table”

Sort_range

Section 5.1.9, “Server Status Variables”
Section 22.12.6.1, “The events_statements_current Table”

Sort_rows

Section 5.1.9, “Server Status Variables”
Section 22.12.6.1, “The events_statements_current Table”

Sort_scan

Section 5.1.9, “Server Status Variables”
Section 22.12.6.1, “The events_statements_current Table”

Ssl_accept_renegotiates

Section 5.1.9, “Server Status Variables”

Ssl_accepts

Section 5.1.9, “Server Status Variables”

Ssl_callback_cache_hits

Section 5.1.9, “Server Status Variables”

Ssl_cipher

Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”
Section 5.1.9, “Server Status Variables”

Ssl_cipher_list

Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”
Section 5.1.9, “Server Status Variables”

Ssl_client_connects

Section 5.1.9, “Server Status Variables”

Ssl_connect_renegotiates

Section 5.1.9, “Server Status Variables”

Ssl_ctx_verify_depth

Section 5.1.9, “Server Status Variables”

Ssl_ctx_verify_mode

Section 5.1.9, “Server Status Variables”

Ssl_default_timeout

Section 5.1.9, “Server Status Variables”

Ssl_finished_accepts

Section 5.1.9, “Server Status Variables”

Ssl_finished_connects

Section 5.1.9, “Server Status Variables”

Ssl_server_not_after
Section 5.1.9, “Server Status Variables”

Ssl_server_not_before
Section 5.1.9, “Server Status Variables”

Ssl_session_cache_hits
Section 5.1.9, “Server Status Variables”

Ssl_session_cache_misses
Section 5.1.9, “Server Status Variables”

Ssl_session_cache_mode
Section 5.1.9, “Server Status Variables”

Ssl_session_cache_overflows
Section 5.1.9, “Server Status Variables”

Ssl_session_cache_size
Section 5.1.9, “Server Status Variables”

Ssl_session_cache_timeouts
Section 5.1.9, “Server Status Variables”

Ssl_sessions_reused
Section 5.1.9, “Server Status Variables”

Ssl_used_session_cache_entries
Section 5.1.9, “Server Status Variables”

Ssl_verify_depth
Section 5.1.9, “Server Status Variables”

Ssl_verify_mode
Section 5.1.9, “Server Status Variables”

Ssl_version
Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”
Section 5.1.9, “Server Status Variables”

T

[index top]

Table_locks_immediate
Section 8.11.1, “Internal Locking Methods”
Section 5.1.9, “Server Status Variables”

Table_locks_waited
Section 8.11.1, “Internal Locking Methods”
Section 5.1.9, “Server Status Variables”

Table_open_cache_hits
Section 5.1.9, “Server Status Variables”

Table_open_cache_misses
Section 5.1.9, “Server Status Variables”

Table_open_cache_overflows
Section 5.1.9, “Server Status Variables”

Tc_log_max_pages_used
Section 5.1.9, “Server Status Variables”

Tc_log_page_size
Section 5.1.9, “Server Status Variables”

Tc_log_page_waits
Section 5.1.9, “Server Status Variables”

Threads_cached
Section 5.1.11.1, “Connection Interfaces”
Section 5.1.9, “Server Status Variables”

Threads_connected
Section 5.1.9, “Server Status Variables”

Threads_created
Section 5.1.11.1, “Connection Interfaces”
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”

Threads_running
Section A.15, “MySQL 5.6 FAQ: MySQL Enterprise Thread Pool”
Section 5.1.9, “Server Status Variables”

U

[index top]

Uptime
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 5.1.9, “Server Status Variables”

Uptime_since_flush_status
Section 5.1.9, “Server Status Variables”

V

[index top]

validate_password_dictionary_file_last_parsed
Section 6.4.3.2, “Password Validation Plugin Options and Variables”

validate_password_dictionary_file_words_count
Section 6.4.3.2, “Password Validation Plugin Options and Variables”

System Variable Index

A | B | C | D | E | F | G | H | I | J | K | L | M | N | O | P | Q | R | S | T | U | V | W

A

[index top]

audit_log_buffer_size
Section 6.4.4.4, “Audit Log Logging Control”
Section 6.4.4.6, “Audit Log Reference”

audit_log_connection_policy
Section 6.4.4.5, “Audit Log Filtering”
Section 6.4.4.6, “Audit Log Reference”

audit_log_current_session
Section 6.4.4.6, “Audit Log Reference”

audit_log_exclude_accounts
Section 6.4.4.5, “Audit Log Filtering”
Section 6.4.4.6, “Audit Log Reference”

audit_log_file
Section 6.4.4.4, “Audit Log Logging Control”
Section 6.4.4.6, “Audit Log Reference”
Section 2.5.8.2, “More Topics on Deploying MySQL Server with Docker”
Section 6.4.4, “MySQL Enterprise Audit”
Section 6.4.4.2, “MySQL Enterprise Audit Security Considerations”

audit_log_flush
Section 6.4.4.4, “Audit Log Logging Control”
Section 6.4.4.6, “Audit Log Reference”

audit_log_format
Section 6.4.4.3, “Audit Log File Formats”
Section 6.4.4.4, “Audit Log Logging Control”
Section 6.4.4.6, “Audit Log Reference”
Section 6.4.4, “MySQL Enterprise Audit”

audit_log_include_accounts
Section 6.4.4.5, “Audit Log Filtering”
Section 6.4.4.6, “Audit Log Reference”

audit_log_policy
Section 6.4.4.5, “Audit Log Filtering”
Section 6.4.4.6, “Audit Log Reference”
Section 5.1.8, “Using System Variables”

audit_log_rotate_on_size
Section 6.4.4.4, “Audit Log Logging Control”
Section 6.4.4.6, “Audit Log Reference”

audit_log_statement_policy
Section 6.4.4.5, “Audit Log Filtering”
Section 6.4.4.6, “Audit Log Reference”

audit_log_strategy
Section 6.4.4.4, “Audit Log Logging Control”

Section 6.4.4.6, “Audit Log Reference”

authentication_windows_log_level

Section 5.1.7, “Server System Variables”
Section 6.4.1.7, “Windows Pluggable Authentication”

authentication_windows_use_principal_name

Section 5.1.7, “Server System Variables”
Section 6.4.1.7, “Windows Pluggable Authentication”

auto_increment_increment

Section 14.6.1.6, “AUTO_INCREMENT Handling in InnoDB”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section A.1, “MySQL 5.6 FAQ: General”
Section 17.4.1.35, “Replication and Variables”
Section 17.1.4.2, “Replication Source Options and Variables”
Section 3.6.9, “Using AUTO_INCREMENT”

auto_increment_offset

Section 14.6.1.6, “AUTO_INCREMENT Handling in InnoDB”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section A.1, “MySQL 5.6 FAQ: General”
Section 17.4.1.35, “Replication and Variables”
Section 17.1.4.2, “Replication Source Options and Variables”
Section 3.6.9, “Using AUTO_INCREMENT”

AUTOCOMMIT

Section 17.4.1.32, “Replication and Transactions”

autocommit

Section 14.7.2.2, “autocommit, Commit, and Rollback”
Section 14.6.1.5, “Converting Tables from MyISAM to InnoDB”
Section 14.7.5.2, “Deadlock Detection”
Section 13.2.2, “DELETE Statement”
Section 14.7, “InnoDB Locking and Transaction Model”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 14.7.2.4, “Locking Reads”
Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 14.6.1.4, “Moving or Copying InnoDB Tables”
NDB Cluster System Variables
Section 8.5.3, “Optimizing InnoDB Read-Only Transactions”
Section 14.8.10, “Purge Configuration”
Section 17.4.1.32, “Replication and Transactions”
Section 17.1.3.4, “Restrictions on Replication with GTIDs”
Section 5.1.7, “Server System Variables”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”
Section 21.30.25, “The INFORMATION_SCHEMA INNODB_TRX Table”
Section 5.5.3.3, “Thread Pool Operation”
Section 14.7.2.1, “Transaction Isolation Levels”

automatic_sp_privileges

Section 13.1.5, “ALTER PROCEDURE Statement”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”

Section 20.2.2, “Stored Routines and MySQL Privileges”

avoid_temporal_upgrade

Section 2.11.3, “Changes in MySQL 5.6”
NDB Cluster System Variables
Section 5.1.7, “Server System Variables”

B

[index top]

back_log

Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
Section 5.1.7, “Server System Variables”

basedir

Section 13.7.3.3, “INSTALL PLUGIN Statement”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”

big_tables

Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 5.1.7, “Server System Variables”
Section 1.3, “What Is New in MySQL 5.6”

bind_address

Section B.3.2.2, “Can't connect to [local] MySQL server”
Section 5.1.12.2, “Configuring the MySQL Server to Permit IPv6 Connections”
Section 5.1.12.4, “Connecting Using IPv6 Nonlocal Host Addresses”
Section 5.1.12.3, “Connecting Using the IPv6 Local Host Address”
Section 5.1.12, “IPv6 Support”
Section 4.3.4, “`mysqld_multi` — Manage Multiple MySQL Servers”
Section 5.1.12.5, “Obtaining an IPv6 Address from a Broker”
Section 5.7, “Running Multiple MySQL Instances on One Machine”
Section 5.1.7, “Server System Variables”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”

binlog_cache_size

Section 17.1.4.4, “Binary Log Options and Variables”
Section 5.1.9, “Server Status Variables”
Section 5.4.4, “The Binary Log”

binlog_checksum

Section 17.1.4.4, “Binary Log Options and Variables”
Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
MySQL Glossary
Section 17.4.2, “Replication Compatibility Between MySQL Versions”
Section 5.4.4, “The Binary Log”
Section 1.3, “What Is New in MySQL 5.6”

binlog_direct_non_transactional_updates

Section 17.1.4.4, “Binary Log Options and Variables”
Section 17.4.1.32, “Replication and Transactions”

binlog_error_action

Section 17.1.4.4, “Binary Log Options and Variables”
Section 5.4.4, “The Binary Log”

binlog_format

Section 17.1.4.4, “Binary Log Options and Variables”
Section 2.11.3, “Changes in MySQL 5.6”
Section 12.7, “Date and Time Functions”
Section 17.1.2.3, “Determination of Safe and Unsafe Statements in Binary Logging”
Section 17.2.3.1, “Evaluation of Database-Level Replication and Binary Logging Options”
Section 17.2.3.2, “Evaluation of Table-Level Replication Options”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 12.16, “Information Functions”
Section 13.2.5.3, “INSERT DELAYED Statement”
Section 17.2.3.3, “Interactions Between Replication Filtering Options”
Section 12.15, “Locking Functions”
Section 5.4.4.4, “Logging Format for Changes to mysql Database Tables”
Section 12.6.2, “Mathematical Functions”
Section 12.20, “Miscellaneous Functions”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section A.14, “MySQL 5.6 FAQ: Replication”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.4.1.2, “Replication and BLACKHOLE Tables”
Section 17.4.1.18, “Replication and LOAD DATA”
Section 17.4.1.20, “Replication and MEMORY Tables”
Section 17.4.1.29, “Replication and Temporary Tables”
Section 17.4.1.32, “Replication and Transactions”
Section 17.1.2, “Replication Formats”
Section 17.4.1.21, “Replication of the mysql System Database”
Section 5.4.4.2, “Setting The Binary Log Format”
Section 20.7, “Stored Program Binary Logging”
Section 5.1.8.1, “System Variable Privileges”
Section 15.6, “The BLACKHOLE Storage Engine”
Section 5.4.3, “The General Query Log”
Section 14.20.6, “The InnoDB memcached Plugin and Replication”
Section 5.4.5, “The Slow Query Log”
Section 14.7.2.1, “Transaction Isolation Levels”
Section 17.4.3, “Upgrading a Replication Setup”
Section 17.1.2.2, “Usage of Row-Based Logging and Replication”

binlog_gtid_simple_recovery

Section 17.1.4.5, “Global Transaction ID Options and Variables”

binlog_max_flush_queue_time

Section 17.1.4.4, “Binary Log Options and Variables”

binlog_order_commits

Section 17.1.4.4, “Binary Log Options and Variables”

binlog_row_image

Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.1.4.4, “Binary Log Options and Variables”
Section 8.5.8, “Optimizing InnoDB Disk I/O”
Section 17.4.2, “Replication Compatibility Between MySQL Versions”

Section 1.3, “What Is New in MySQL 5.6”

binlog_rows_query_log_events

Section 17.1.2.1, “Advantages and Disadvantages of Statement-Based and Row-Based Replication”
Section 17.4.2, “Replication Compatibility Between MySQL Versions”

binlog_stmt_cache_size

Section 17.1.4.4, “Binary Log Options and Variables”
Section 5.1.9, “Server Status Variables”

binlogging_impossible_mode

Section 17.1.4.4, “Binary Log Options and Variables”

block_encryption_mode

Section 12.14, “Encryption and Compression Functions”
Section 5.1.7, “Server System Variables”

bulk_insert_buffer_size

Section 15.2.1, “MyISAM Startup Options”
Section 8.2.4.1, “Optimizing INSERT Statements”
Section 5.1.7, “Server System Variables”

C

[index top]

character_set_client

Section 10.15, “Character Set Configuration”
Section 10.4, “Connection Character Sets and Collations”
Section 13.2.6, “LOAD DATA Statement”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section A.11, “MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 17.4.1.35, “Replication and Variables”
Section 5.1.7, “Server System Variables”
Section 13.7.4.2, “SET CHARACTER SET Statement”
Section 13.7.4.3, “SET NAMES Statement”
Section 13.7.5.9, “SHOW CREATE EVENT Statement”
Section 13.7.5.11, “SHOW CREATE PROCEDURE Statement”
Section 13.7.5.13, “SHOW CREATE TRIGGER Statement”
Section 13.7.5.14, “SHOW CREATE VIEW Statement”
Section 13.7.5.19, “SHOW EVENTS Statement”
Section 13.7.5.29, “SHOW PROCEDURE STATUS Statement”
Section 13.7.5.39, “SHOW TRIGGERS Statement”
Section 5.4.4, “The Binary Log”
Section 21.8, “The INFORMATION_SCHEMA EVENTS Table”
Section 21.19, “The INFORMATION_SCHEMA ROUTINES Table”
Section 21.27, “The INFORMATION_SCHEMA TRIGGERS Table”
Section 21.29, “The INFORMATION_SCHEMA VIEWS Table”

character_set_connection

Section 12.11, “Cast Functions and Operators”
Section 12.8.3, “Character Set and Collation of Function Results”
Section 10.3.8, “Character Set Introducers”
Section 10.2.1, “Character Set Repertoire”
Section 10.3.6, “Character String Literal Character Set and Collation”
Section 10.8.4, “Collation Coercibility in Expressions”
Section 10.4, “Connection Character Sets and Collations”

Section 12.7, “Date and Time Functions”
Section 12.14, “Encryption and Compression Functions”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section A.11, “MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 10.16, “MySQL Server Locale Support”
Section 17.4.1.35, “Replication and Variables”
Section 5.1.7, “Server System Variables”
Section 13.7.4.2, “SET CHARACTER SET Statement”
Section 13.7.4.3, “SET NAMES Statement”
Section 9.1.1, “String Literals”
Section 12.3, “Type Conversion in Expression Evaluation”

character_set_database

Section 13.1.7, “ALTER TABLE Statement”
Section 10.4, “Connection Character Sets and Collations”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 10.3.3, “Database Character Set and Collation”
Section 13.2.6, “LOAD DATA Statement”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.4.1.35, “Replication and Variables”
Section 5.1.7, “Server System Variables”
Section 13.7.4.2, “SET CHARACTER SET Statement”

character_set_filesystem

Section 13.2.6, “LOAD DATA Statement”
Section 13.2.9.1, “SELECT ... INTO Statement”
Section 5.1.7, “Server System Variables”
Section 12.8, “String Functions and Operators”

character_set_results

Section 10.4, “Connection Character Sets and Collations”
Section 10.6, “Error Message Character Set”
Section A.11, “MySQL 5.6 FAQ: MySQL Chinese, Japanese, and Korean Character Sets”
Section 5.1.7, “Server System Variables”
Section 13.7.4.2, “SET CHARACTER SET Statement”
Section 13.7.4.3, “SET NAMES Statement”
Section 10.2.2, “UTF-8 for Metadata”

character_set_server

Section 10.15, “Character Set Configuration”
Section 10.4, “Connection Character Sets and Collations”
Section 10.3.3, “Database Character Set and Collation”
Section 12.10.4, “Full-Text Stopwords”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.4.1.3, “Replication and Character Sets”
Section 17.4.1.35, “Replication and Variables”
Section 10.3.2, “Server Character Set and Collation”
Section 5.1.7, “Server System Variables”

character_set_system

Section 10.15, “Character Set Configuration”
Section 5.1.7, “Server System Variables”
Section 10.2.2, “UTF-8 for Metadata”

character_sets_dir

Section 10.14.3, “Adding a Simple Collation to an 8-Bit Character Set”
Section 10.14.4.1, “Defining a UCA Collation Using LDML Syntax”

Section 5.1.7, “Server System Variables”

collation_connection

Section 12.11, “Cast Functions and Operators”
Section 12.8.3, “Character Set and Collation of Function Results”
Section 10.3.8, “Character Set Introducers”
Section 10.3.6, “Character String Literal Character Set and Collation”
Section 10.8.4, “Collation Coercibility in Expressions”
Section 10.4, “Connection Character Sets and Collations”
Section 12.7, “Date and Time Functions”
Section 12.14, “Encryption and Compression Functions”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.4.1.35, “Replication and Variables”
Section 5.1.7, “Server System Variables”
Section 13.7.4.3, “SET NAMES Statement”
Section 13.7.5.9, “SHOW CREATE EVENT Statement”
Section 13.7.5.11, “SHOW CREATE PROCEDURE Statement”
Section 13.7.5.13, “SHOW CREATE TRIGGER Statement”
Section 13.7.5.14, “SHOW CREATE VIEW Statement”
Section 13.7.5.19, “SHOW EVENTS Statement”
Section 13.7.5.29, “SHOW PROCEDURE STATUS Statement”
Section 13.7.5.39, “SHOW TRIGGERS Statement”
Section 5.4.4, “The Binary Log”
Section 21.8, “The INFORMATION_SCHEMA EVENTS Table”
Section 21.19, “The INFORMATION_SCHEMA ROUTINES Table”
Section 21.27, “The INFORMATION_SCHEMA TRIGGERS Table”
Section 21.29, “The INFORMATION_SCHEMA VIEWS Table”
Section 12.3, “Type Conversion in Expression Evaluation”

collation_database

Section 10.4, “Connection Character Sets and Collations”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 10.3.3, “Database Character Set and Collation”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.4.1.35, “Replication and Variables”
Section 5.1.7, “Server System Variables”
Section 5.4.4, “The Binary Log”

collation_server

Section 10.4, “Connection Character Sets and Collations”
Section 10.3.3, “Database Character Set and Collation”
Section 12.10.4, “Full-Text Stopwords”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.4.1.35, “Replication and Variables”
Section 10.3.2, “Server Character Set and Collation”
Section 5.1.7, “Server System Variables”
Section 5.4.4, “The Binary Log”

completion_type

Section 5.1.7, “Server System Variables”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”

concurrent_insert

Section 8.11.3, “Concurrent Inserts”
Section 8.11.1, “Internal Locking Methods”
Section 8.6.1, “Optimizing MyISAM Queries”
Section 5.1.6, “Server Command Options”

Section 5.1.7, “Server System Variables”

connect_timeout

Section B.3.2.10, “Communication Errors and Aborted Connections”
Section B.3.2.3, “Lost connection to MySQL server”
Section 5.1.7, “Server System Variables”

connection_control_failed_connections_threshold

Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 6.4.2.2, “Connection-Control System and Status Variables”
Section 21.33.1, “The INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS Table”

connection_control_max_connection_delay

Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 6.4.2.2, “Connection-Control System and Status Variables”

connection_control_min_connection_delay

Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 6.4.2.2, “Connection-Control System and Status Variables”

core_file

Section 5.1.7, “Server System Variables”

create_old_temporals

NDB Cluster System Variables

D

[index top]

daemon_memcached_enable_binlog

Section 14.14, “InnoDB Startup Options and System Variables”

daemon_memcached_engine_lib_name

Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.20.3, “Setting Up the InnoDB memcached Plugin”

daemon_memcached_engine_lib_path

Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.20.3, “Setting Up the InnoDB memcached Plugin”

daemon_memcached_option

Section 14.20.2, “InnoDB memcached Architecture”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.20.4, “Security Considerations for the InnoDB memcached Plugin”
Section 14.20.3, “Setting Up the InnoDB memcached Plugin”
Section 14.20.8, “Troubleshooting the InnoDB memcached Plugin”

daemon_memcached_r_batch_size

Section 14.20.2, “InnoDB memcached Architecture”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.20.5.6, “Performing DML and DDL Statements on the Underlying InnoDB Table”
Section 14.20.3, “Setting Up the InnoDB memcached Plugin”
Section 14.20.6, “The InnoDB memcached Plugin and Replication”
Section 14.20.5.3, “Tuning InnoDB memcached Plugin Performance”

daemon_memcached_w_batch_size

Section 14.20.5.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”
Section 14.20.2, “InnoDB memcached Architecture”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.20.5.6, “Performing DML and DDL Statements on the Underlying InnoDB Table”
Section 14.20.3, “Setting Up the InnoDB memcached Plugin”
Section 14.20.6, “The InnoDB memcached Plugin and Replication”
Section 14.20.5.3, “Tuning InnoDB memcached Plugin Performance”

datadir

Section 14.8.1, “InnoDB Startup Configuration”
Section 2.3, “Installing MySQL on Microsoft Windows”
MySQL Glossary
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”

date_format

Section 5.1.7, “Server System Variables”
Section 1.3, “What Is New in MySQL 5.6”

datetime_format

Section 5.1.7, “Server System Variables”
Section 1.3, “What Is New in MySQL 5.6”

debug

Section 5.1.7, “Server System Variables”
Section 5.8.3, “The DBUG Package”

debug_sync

Section 5.1.7, “Server System Variables”

default

Section 15.1, “Setting the Storage Engine”
Section 14.1.4, “Testing and Benchmarking with InnoDB”

default_storage_engine

Section 13.1.14, “CREATE LOGFILE GROUP Statement”
Section 5.5.1, “Installing and Uninstalling Plugins”
Section 17.4.1.35, “Replication and Variables”
Section 5.1.7, “Server System Variables”
Section 15.1, “Setting the Storage Engine”
Section 17.3.3, “Using Replication with Different Source and Replica Storage Engines”

default_tmp_storage_engine

Section 5.5.1, “Installing and Uninstalling Plugins”
Section 5.1.7, “Server System Variables”
Section 15.1, “Setting the Storage Engine”

default_week_format

Section 12.7, “Date and Time Functions”
Section 19.6.3, “Partitioning Limitations Relating to Functions”
Section 5.1.7, “Server System Variables”

delay_key_write

Section 13.1.17, “CREATE TABLE Statement”

Section 8.11.5, “External Locking”
Section A.14, “MySQL 5.6 FAQ: Replication”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section B.3.3.3, “What to Do If MySQL Keeps Crashing”

delayed_insert_limit

Section 13.2.5.3, “INSERT DELAYED Statement”
Section 5.1.7, “Server System Variables”

delayed_insert_timeout

Section 13.2.5.3, “INSERT DELAYED Statement”
Section 5.1.7, “Server System Variables”

delayed_queue_size

Section 13.2.5.3, “INSERT DELAYED Statement”
Section 5.1.7, “Server System Variables”

disconnect_on_expired_password

Section 6.2.10, “Server Handling of Expired Passwords”
Section 5.1.7, “Server System Variables”

div_precision_increment

Section 12.6.1, “Arithmetic Operators”
Section 5.1.7, “Server System Variables”

E

[index top]

end_markers_in_json

Section 5.1.7, “Server System Variables”

enforce_gtid_consistency

Section 17.1.3.5, “Disabling GTID Transactions”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 17.1.3.2, “Setting Up Replication Using GTIDs”
Section 1.3, “What Is New in MySQL 5.6”

eq_range_index_dive_limit

Section 8.2.1.2, “Range Optimization”
Section 5.1.7, “Server System Variables”

error_count

Section B.2, “Error Information Interfaces”
Section 5.1.7, “Server System Variables”
Section 13.7.5.18, “SHOW ERRORS Statement”
Section 13.6.7.7, “The MySQL Diagnostics Area”

event

Section 14.8.2, “Configuring InnoDB for Read-Only Operation”

event_scheduler

Section 20.4.2, “Event Scheduler Configuration”
Section 23.6.2, “Restrictions When Using the Embedded MySQL Server”

Section 5.1.7, “Server System Variables”
Section 20.4.6, “The Event Scheduler and MySQL Privileges”

expire_logs_days

Section 17.1.4.4, “Binary Log Options and Variables”
Section 13.4.1.1, “PURGE BINARY LOGS Statement”
Section 5.4.7, “Server Log Maintenance”

explicit_defaults_for_timestamp

Section 11.2.6, “Automatic Initialization and Updating for TIMESTAMP and DATETIME”
Section 2.11.3, “Changes in MySQL 5.6”
Section 11.5, “Data Type Default Values”
Section 11.2.1, “Date and Time Data Type Syntax”
Section 5.1.7, “Server System Variables”
Section 1.3, “What Is New in MySQL 5.6”

external_user

Section 6.4.4.3, “Audit Log File Formats”
Section 6.2.12, “Proxy Users”
Section 5.1.7, “Server System Variables”

F

[index top]

flush

Section 5.1.7, “Server System Variables”

flush_time

Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”

foreign_key_checks

Section 13.1.7, “ALTER TABLE Statement”
Section 13.1.17.5, “FOREIGN KEY Constraints”
Section 14.6.1.3, “Importing InnoDB Tables”
Section 5.4.4.3, “Mixed Binary Logging Format”
NDB Cluster System Variables
Section 14.13.1, “Online DDL Operations”
Section 17.4.1.35, “Replication and Variables”
Section 5.1.10, “Server SQL Modes”
Section 5.1.7, “Server System Variables”
Section 5.4.4, “The Binary Log”

ft_boolean_syntax

Section 12.10.2, “Boolean Full-Text Searches”
Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”
Section 5.1.7, “Server System Variables”

ft_max_word_len

Section 12.10.2, “Boolean Full-Text Searches”
Section 17.1.1.6, “Creating a Data Snapshot Using Raw Data Files”
Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”
Section 5.1.7, “Server System Variables”

ft_min_word_len
Section 12.10.2, “Boolean Full-Text Searches”
Section 17.1.1.6, “Creating a Data Snapshot Using Raw Data Files”
Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”
Section 12.10.1, “Natural Language Full-Text Searches”
Section 5.1.7, “Server System Variables”

ft_query_expansion_limit
Section 5.1.7, “Server System Variables”

ft_stopword_file
Section 12.10.2, “Boolean Full-Text Searches”
Section 17.1.1.6, “Creating a Data Snapshot Using Raw Data Files”
Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”
Section 12.10.4, “Full-Text Stopwords”
Section 12.10.1, “Natural Language Full-Text Searches”
Section 5.1.7, “Server System Variables”

G

[index top]

general_log
MySQL Glossary
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.7, “Server System Variables”
Section 5.4.3, “The General Query Log”
Section 1.3, “What Is New in MySQL 5.6”

general_log_file
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.7, “Server System Variables”
Section 5.4.3, “The General Query Log”
Section 1.3, “What Is New in MySQL 5.6”

group_concat_max_len
Section 12.19.1, “Aggregate Function Descriptions”
Section 5.1.7, “Server System Variables”

gtid_executed
Section 13.4.2.1, “CHANGE MASTER TO Statement”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 17.1.3.1, “GTID Concepts”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 13.4.1.2, “RESET MASTER Statement”
Section 17.1.3.4, “Restrictions on Replication with GTIDs”
Section 13.7.5.24, “SHOW MASTER STATUS Statement”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”
Section 17.1.3.3, “Using GTIDs for Failover and Scaleout”

gtid_mode
Section 13.4.2.1, “CHANGE MASTER TO Statement”
Section 17.1.3.5, “Disabling GTID Transactions”
Section 12.18, “Functions Used with Global Transaction IDs”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.4.2, “Replication Compatibility Between MySQL Versions”
Section 17.1.3.4, “Restrictions on Replication with GTIDs”
Section 17.1.3.2, “Setting Up Replication Using GTIDs”
Section 17.4.3, “Upgrading a Replication Setup”
Section 2.11.4, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”
Section 17.1.3.3, “Using GTIDs for Failover and Scaleout”
Section 1.3, “What Is New in MySQL 5.6”

gtid_next

Section 13.1.6, “ALTER SERVER Statement”
Section 13.7.2.1, “ANALYZE TABLE Statement”
Section 13.7.6.2, “CACHE INDEX Statement”
Section 13.4.2.1, “CHANGE MASTER TO Statement”
Section 13.7.2.2, “CHECK TABLE Statement”
Section 13.1.16, “CREATE SERVER Statement”
Section 13.1.27, “DROP SERVER Statement”
Section 13.7.6.3, “FLUSH Statement”
Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 17.1.3.1, “GTID Concepts”
Section 13.7.6.5, “LOAD INDEX INTO CACHE Statement”
Section 13.7.2.4, “OPTIMIZE TABLE Statement”
Section 13.7.2.5, “REPAIR TABLE Statement”
Section 13.7.6.6, “RESET Statement”
Section 13.4.2.5, “START SLAVE Statement”
Section 13.4.2.6, “STOP SLAVE Statement”

gtid_owned

Section 17.1.4.5, “Global Transaction ID Options and Variables”

gtid_purged

Section 17.1.4.5, “Global Transaction ID Options and Variables”
Section 17.1.3.1, “GTID Concepts”
Section 13.4.1.2, “RESET MASTER Statement”
Section 17.1.3.3, “Using GTIDs for Failover and Scaleout”

H

[index top]

have_compress

Section 5.1.7, “Server System Variables”

have_crypt

Section 5.1.7, “Server System Variables”

have_dynamic_loading

Section 17.3.9.2, “Semisynchronous Replication Installation and Configuration”
Section 5.1.7, “Server System Variables”

have_geometry

Section 5.1.7, “Server System Variables”

have_openssl

Section 5.1.7, “Server System Variables”

have_profiling
Section 5.1.7, “Server System Variables”
Section 1.3, “What Is New in MySQL 5.6”

have_query_cache
Section 8.10.3.3, “Query Cache Configuration”
Section 5.1.7, “Server System Variables”

have_rtree_keys
Section 5.1.7, “Server System Variables”

have_ssl
Section 2.9.6, “Configuring SSL Library Support”
Section 5.1.7, “Server System Variables”

have_symlink
Section 5.1.7, “Server System Variables”
Section 8.12.3.3, “Using Symbolic Links for Databases on Windows”
Section 8.12.3.2, “Using Symbolic Links for MyISAM Tables on Unix”

host_cache_size
Section 5.1.2.1, “Changes to Server Defaults”
Section 5.1.11.2, “DNS Lookups and the Host Cache”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 22.12.10.1, “The host_cache Table”
Section 1.3, “What Is New in MySQL 5.6”

hostname
Section 5.1.7, “Server System Variables”

I

[index top]

identity
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.4.1.35, “Replication and Variables”
Section 5.1.7, “Server System Variables”

ignore_db_dirs
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”

init_connect
Section 10.5, “Configuring Application Character Set and Collation”
Section 6.2.2, “Privileges Provided by MySQL”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 5.1.7, “Server System Variables”
Section 22.12.10.1, “The host_cache Table”

init_file
Section 8.10.2.2, “Multiple Key Caches”
Section 22.4, “Performance Schema Runtime Configuration”
Section 17.4.1.20, “Replication and MEMORY Tables”

Resetting the Root Password: Unix and Unix-Like Systems
Resetting the Root Password: Windows Systems
Section 5.1.7, “Server System Variables”
Section 15.3, “The MEMORY Storage Engine”

init_slave

Section 17.1.4.3, “Replica Server Options and Variables”

innodb

Section 14.8.2, “Configuring InnoDB for Read-Only Operation”
Section A.16, “MySQL 5.6 FAQ: InnoDB Change Buffer”
Section 1.3, “What Is New in MySQL 5.6”

innodb_adaptive_flushing

Section 14.8.3.4, “Configuring Buffer Pool Flushing”
Section 14.14, “InnoDB Startup Options and System Variables”

innodb_adaptive_flushing_lwm

Section 14.8.3.4, “Configuring Buffer Pool Flushing”
Section 14.14, “InnoDB Startup Options and System Variables”

innodb_adaptive_hash_index

Section 14.5.3, “Adaptive Hash Index”
Section 14.8.5, “Configuring Thread Concurrency for InnoDB”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 8.5.9, “Optimizing InnoDB Configuration Variables”
Section 13.1.33, “TRUNCATE TABLE Statement”

innodb_adaptive_max_sleep_delay

Section 14.8.5, “Configuring Thread Concurrency for InnoDB”
Section 14.14, “InnoDB Startup Options and System Variables”

innodb_additional_mem_pool_size

Section 14.8.4, “Configuring the Memory Allocator for InnoDB”
Section 14.8.1, “InnoDB Startup Configuration”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 1.3, “What Is New in MySQL 5.6”

innodb_api_bk_commit_interval

Section 14.20.5.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”
Section 14.20.2, “InnoDB memcached Architecture”
Section 14.14, “InnoDB Startup Options and System Variables”

innodb_api_disable_rowlock

Section 14.20.5.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”
Section 14.14, “InnoDB Startup Options and System Variables”

innodb_api_enable_binlog

Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.20.6, “The InnoDB memcached Plugin and Replication”

innodb_api_enable_mdl

Section 14.20.5.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”
Section 14.20.2, “InnoDB memcached Architecture”
Section 14.14, “InnoDB Startup Options and System Variables”

innodb_api_trx_level

Section 14.20.5.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”
Section 14.20.2, “InnoDB memcached Architecture”
Section 14.14, “InnoDB Startup Options and System Variables”

innodb_autoextend_increment

Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
Section 14.6.3.2, “File-Per-Table Tablespaces”
Section 14.8.1, “InnoDB Startup Configuration”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 14.6.3.1, “The System Tablespace”

innodb_autoinc_lock_mode

Section 14.6.1.6, “AUTO_INCREMENT Handling in InnoDB”
Section 8.5.5, “Bulk Data Loading for InnoDB Tables”
Section 12.16, “Information Functions”
Section 14.7.1, “InnoDB Locking”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”
MySQL Glossary

innodb_buffer_pool_dump_at_shutdown

Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary

innodb_buffer_pool_dump_now

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_buffer_pool_filename

Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.8.3.5, “Saving and Restoring the Buffer Pool State”

innodb_buffer_pool_instances

Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
Section 14.8.3.4, “Configuring Buffer Pool Flushing”
Section 14.8.3.1, “Configuring Multiple Buffer Pool Instances”
Section 8.12.4.1, “How MySQL Uses Memory”
Section 14.8.1, “InnoDB Startup Configuration”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 14.20.5.3, “Tuning InnoDB memcached Plugin Performance”

innodb_buffer_pool_load_abort

Section 14.14, “InnoDB Startup Options and System Variables”
Section 5.1.9, “Server Status Variables”

innodb_buffer_pool_load_at_startup

Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 5.1.9, “Server Status Variables”

innodb_buffer_pool_load_now

Section 14.14, “InnoDB Startup Options and System Variables”

Section 5.1.9, “Server Status Variables”

innodb_buffer_pool_size

Section 14.5.1, “Buffer Pool”
Section 14.9.6, “Compression for OLTP Workloads”
Section 14.8.3.1, “Configuring Multiple Buffer Pool Instances”
Section 14.6.1.5, “Converting Tables from MyISAM to InnoDB”
Section 8.12.4.1, “How MySQL Uses Memory”
Section 14.20.2, “InnoDB memcached Architecture”
Section 14.8.1, “InnoDB Startup Configuration”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 8.5.8, “Optimizing InnoDB Disk I/O”
Section 5.1.9, “Server Status Variables”
Section 14.20.5.3, “Tuning InnoDB memcached Plugin Performance”

innodb_change_buffer_max_size

Section 14.5.2, “Change Buffer”
Section 14.14, “InnoDB Startup Options and System Variables”
Section A.16, “MySQL 5.6 FAQ: InnoDB Change Buffer”
MySQL Glossary

innodb_change_buffering

Section 14.5.2, “Change Buffer”
Section 14.8.2, “Configuring InnoDB for Read-Only Operation”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 8.5.2, “Optimizing InnoDB Transaction Management”

innodb_change_buffering_debug

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_checksum_algorithm

Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 8.5.8, “Optimizing InnoDB Disk I/O”
Section 1.3, “What Is New in MySQL 5.6”

innodb_checksums

Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary

innodb_cmp_per_index_enabled

Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.9.4, “Monitoring InnoDB Table Compression at Runtime”
Section 21.30.6, “The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables”
Section 14.9.3, “Tuning Compression for InnoDB Tables”

innodb_commit_concurrency

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_compression_failure_threshold_pct

Section 14.9.6, “Compression for OLTP Workloads”

Section 14.9.5, “How Compression Works for InnoDB Tables”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 14.9.3, “Tuning Compression for InnoDB Tables”
Section 1.3, “What Is New in MySQL 5.6”

innodb_compression_level

Section 14.9.6, “Compression for OLTP Workloads”
Section 14.9.5, “How Compression Works for InnoDB Tables”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 14.9.3, “Tuning Compression for InnoDB Tables”
Section 1.3, “What Is New in MySQL 5.6”

innodb_compression_pad_pct_max

Section 14.9.6, “Compression for OLTP Workloads”
Section 14.9.5, “How Compression Works for InnoDB Tables”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 14.9.3, “Tuning Compression for InnoDB Tables”
Section 1.3, “What Is New in MySQL 5.6”

innodb_concurrency_tickets

Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
Section 14.8.5, “Configuring Thread Concurrency for InnoDB”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 8.5.9, “Optimizing InnoDB Configuration Variables”
Section 21.30.25, “The INFORMATION_SCHEMA INNODB_TRX Table”

innodb_data_file_path

Section 5.1.2.1, “Changes to Server Defaults”
Section 14.12.2, “File Space Management”
Section 14.8.1, “InnoDB Startup Configuration”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”
Section 14.6.3.1, “The System Tablespace”
Section 14.21.1, “Troubleshooting InnoDB I/O Problems”

innodb_data_home_dir

Section 14.8.1, “InnoDB Startup Configuration”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”
Section 14.21.1, “Troubleshooting InnoDB I/O Problems”

innodb_deadlock_detect

MySQL Glossary

innodb_default_row_format

MySQL Glossary

innodb_disable_sort_file_cache

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_doublewrite

Section 14.6.5, “Doublewrite Buffer”

Section 14.2, “InnoDB and the ACID Model”
Section 14.12.1, “InnoDB Disk I/O”
Section 14.8.1, “InnoDB Startup Configuration”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 14.20.5.3, “Tuning InnoDB memcached Plugin Performance”

innodb_fast_shutdown

Section 14.10.2.1, “Compatibility Check When InnoDB Is Started”
Section 2.12.4, “Downgrading Binary and Package-based Installations on Unix/Linux”
Section 14.18.2, “InnoDB Recovery”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 14.6.6, “Redo Log”
Section 5.1.15, “The Server Shutdown Process”
Section 2.11.4, “Upgrading MySQL Binary or Package-based Installations on Unix/Linux”

innodb_fil_make_page_dirty_debug

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_file_format

Section 14.10.2.2, “Compatibility Check When a Table Is Opened”
Section 14.10.2.1, “Compatibility Check When InnoDB Is Started”
Section 10.9.8, “Converting Between 3-Byte and 4-Byte Unicode Character Sets”
Section 14.6.1.5, “Converting Tables from MyISAM to InnoDB”
Section 13.1.17, “CREATE TABLE Statement”
Section 14.6.1.1, “Creating InnoDB Tables”
Section 14.9.2, “Enabling Compression for a Table”
Section 14.10.1, “Enabling File Formats”
Section 14.10.3, “Identifying the File Format in Use”
Section 14.11, “InnoDB Row Formats”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 14.9.7, “SQL Compression Syntax Warnings and Errors”

innodb_file_format_check

Section 14.10.2.2, “Compatibility Check When a Table Is Opened”
Section 14.10.2.1, “Compatibility Check When InnoDB Is Started”
Section 14.14, “InnoDB Startup Options and System Variables”

innodb_file_format_max

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_file_per_table

Section 14.1.2, “Best Practices for InnoDB Tables”
Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
Section 10.9.8, “Converting Between 3-Byte and 4-Byte Unicode Character Sets”
Section 14.6.1.5, “Converting Tables from MyISAM to InnoDB”
Section 13.1.17, “CREATE TABLE Statement”
Section 14.6.1.1, “Creating InnoDB Tables”
Section 14.6.1.2, “Creating Tables Externally”
Section 14.9.2, “Enabling Compression for a Table”
Section 14.10.1, “Enabling File Formats”
Section 14.12.2, “File Space Management”
Section 14.6.3.2, “File-Per-Table Tablespaces”
Section 13.1.17.1, “Files Created by CREATE TABLE”

Section 13.7.6.3, “FLUSH Statement”
Section 14.6.1.3, “Importing InnoDB Tables”
Section 14.2, “InnoDB and the ACID Model”
Section 14.6.2.3, “InnoDB FULLTEXT Indexes”
Section 14.11, “InnoDB Row Formats”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.17.4, “InnoDB Tablespace Monitor Output”
MySQL Glossary
Section 13.7.2.4, “OPTIMIZE TABLE Statement”
Section 14.12.5, “Reclaiming Disk Space with TRUNCATE TABLE”
Section 17.3.5, “Replicating Different Databases to Different Replicas”
Section 19.6, “Restrictions and Limitations on Partitioning”
Section 14.9.7, “SQL Compression Syntax Warnings and Errors”
Section 14.21.3, “Troubleshooting InnoDB Data Dictionary Operations”

innodb_flush_log_at_timeout

Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.5.4, “Log Buffer”

innodb_flush_log_at_trx_commit

Section 17.1.4.4, “Binary Log Options and Variables”
Section 14.2, “InnoDB and the ACID Model”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.5.4, “Log Buffer”
Section 8.5.2, “Optimizing InnoDB Transaction Management”
Section 17.4.1.26, “Replication and Source or Replica Shutdowns”
Section 14.20.5.3, “Tuning InnoDB memcached Plugin Performance”

innodb_flush_method

Section 14.6.3.2, “File-Per-Table Tablespaces”
Section 14.8.1, “InnoDB Startup Configuration”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 2.3, “Installing MySQL on Microsoft Windows”
Section 8.5.8, “Optimizing InnoDB Disk I/O”
Section 5.1.9, “Server Status Variables”
Section 14.20.5.3, “Tuning InnoDB memcached Plugin Performance”

innodb_flush_neighbors

Section 14.8.3.4, “Configuring Buffer Pool Flushing”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 8.5.8, “Optimizing InnoDB Disk I/O”

innodb_flushing_avg_loops

Section 14.8.3.4, “Configuring Buffer Pool Flushing”
Section 14.14, “InnoDB Startup Options and System Variables”

innodb_force_load_corrupted

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_force_recovery

Section 13.1.28, “DROP TABLE Statement”
Section 14.21.2, “Forcing InnoDB Recovery”
Section 1.6, “How to Report Bugs or Problems”
Section 14.18.2, “InnoDB Recovery”
Section 14.14, “InnoDB Startup Options and System Variables”

Section 8.5.2, “Optimizing InnoDB Transaction Management”
Section 2.11.10, “Rebuilding or Repairing Tables or Indexes”

innodb_ft_aux_table

Section 14.15.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 21.30.7, “The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table”
Section 21.30.8, “The INFORMATION_SCHEMA INNODB_FT_CONFIG Table”
Section 21.30.10, “The INFORMATION_SCHEMA INNODB_FT_DELETED Table”
Section 21.30.11, “The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table”
Section 21.30.12, “The INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE Table”

innodb_ft_cache_size

Section 14.6.2.3, “InnoDB FULLTEXT Indexes”
Section 14.15.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 21.30.11, “The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table”

innodb_ft_enable_diag_print

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_ft_enable_stopword

Section 12.10.2, “Boolean Full-Text Searches”
Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 12.10.1, “Natural Language Full-Text Searches”

innodb_ft_max_token_size

Section 12.10.2, “Boolean Full-Text Searches”
Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”
Section 12.10.4, “Full-Text Stopwords”
Section 14.14, “InnoDB Startup Options and System Variables”

innodb_ft_min_token_size

Section 12.10.2, “Boolean Full-Text Searches”
Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”
Section 12.10.4, “Full-Text Stopwords”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 12.10.1, “Natural Language Full-Text Searches”

innodb_ft_num_word_optimize

Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 13.7.2.4, “OPTIMIZE TABLE Statement”

innodb_ft_result_cache_limit

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_ft_server_stopword_table

Section 12.10.2, “Boolean Full-Text Searches”
Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”
Section 12.10.4, “Full-Text Stopwords”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 12.10.1, “Natural Language Full-Text Searches”
Section 21.30.9, “The INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD Table”

innodb_ft_sort_pll_degree

Section 14.6.2.3, “InnoDB FULLTEXT Indexes”
Section 14.14, “InnoDB Startup Options and System Variables”

innodb_ft_total_cache_size

Section 14.6.2.3, “InnoDB FULLTEXT Indexes”
Section 14.15.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 21.30.11, “The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table”

innodb_ft_user_stopword_table

Section 12.10.2, “Boolean Full-Text Searches”
Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”
Section 12.10.4, “Full-Text Stopwords”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 12.10.1, “Natural Language Full-Text Searches”
Section 21.30.9, “The INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD Table”

innodb_io_capacity

Section 14.8.3.4, “Configuring Buffer Pool Flushing”
Section 14.8.8, “Configuring InnoDB I/O Capacity”
Section 14.14, “InnoDB Startup Options and System Variables”
Section A.16, “MySQL 5.6 FAQ: InnoDB Change Buffer”
Section 8.5.8, “Optimizing InnoDB Disk I/O”

innodb_io_capacity_max

Section 14.8.3.4, “Configuring Buffer Pool Flushing”
Section 14.8.8, “Configuring InnoDB I/O Capacity”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 8.5.8, “Optimizing InnoDB Disk I/O”

innodb_large_prefix

Section 8.3.4, “Column Indexes”
Section 10.9.8, “Converting Between 3-Byte and 4-Byte Unicode Character Sets”
Section 13.1.13, “CREATE INDEX Statement”
Section 13.1.17, “CREATE TABLE Statement”
Section 2.12.3, “Downgrade Notes”
Section 14.10, “InnoDB File-Format Management”
Section 14.22, “InnoDB Limits”
Section 14.11, “InnoDB Row Formats”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary

innodb_limit_optimistic_insert_debug

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_lock_wait_timeout

Section 14.7.5.2, “Deadlock Detection”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.4.1.30, “Replication Retries and Timeouts”

innodb_locks_unsafe_for_binlog

Section 14.7.2.3, “Consistent Nonlocking Reads”

Section 14.7.1, “InnoDB Locking”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”
MySQL Glossary
Section 14.7.2.1, “Transaction Isolation Levels”

innodb_log_buffer_size

Section 14.8.1, “InnoDB Startup Configuration”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.5.4, “Log Buffer”
MySQL Glossary
Section 8.5.4, “Optimizing InnoDB Redo Logging”

innodb_log_checkpoint_now

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_log_compressed_pages

Section 14.9.6, “Compression for OLTP Workloads”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 8.5.8, “Optimizing InnoDB Disk I/O”
Section 1.3, “What Is New in MySQL 5.6”

innodb_log_file_size

Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
Section 14.8.3.4, “Configuring Buffer Pool Flushing”
Section 14.8.2, “Configuring InnoDB for Read-Only Operation”
Section 2.12.3, “Downgrade Notes”
Section 14.8.1, “InnoDB Startup Configuration”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”
Section 8.5.8, “Optimizing InnoDB Disk I/O”
Section 8.5.4, “Optimizing InnoDB Redo Logging”
Section 14.6.6, “Redo Log”
Section 5.1.8, “Using System Variables”
Section 1.3, “What Is New in MySQL 5.6”

innodb_log_files_in_group

Section 2.11.3, “Changes in MySQL 5.6”
Section 14.8.1, “InnoDB Startup Configuration”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 8.5.4, “Optimizing InnoDB Redo Logging”
Section 14.6.6, “Redo Log”
Section 1.3, “What Is New in MySQL 5.6”

innodb_log_group_home_dir

Section 14.8.1, “InnoDB Startup Configuration”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”

innodb_lru_scan_depth

Section 14.8.3.4, “Configuring Buffer Pool Flushing”
Section 14.14, “InnoDB Startup Options and System Variables”

innodb_max_dirty_pages_pct

Section 14.8.3.4, “Configuring Buffer Pool Flushing”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 8.5.8, “Optimizing InnoDB Disk I/O”

innodb_max_dirty_pages_pct_lwm

Section 14.8.3.4, “Configuring Buffer Pool Flushing”
Section 14.14, “InnoDB Startup Options and System Variables”

innodb_max_purge_lag

Section 14.3, “InnoDB Multi-Versioning”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 14.8.10, “Purge Configuration”

innodb_max_purge_lag_delay

Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.8.10, “Purge Configuration”

innodb_mirrored_log_groups

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_monitor_disable

Section 14.15.6, “InnoDB INFORMATION_SCHEMA Metrics Table”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 21.30.15, “The INFORMATION_SCHEMA INNODB_METRICS Table”

innodb_monitor_enable

Section 14.15.6, “InnoDB INFORMATION_SCHEMA Metrics Table”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 21.30.15, “The INFORMATION_SCHEMA INNODB_METRICS Table”

innodb_monitor_reset

Section 14.15.6, “InnoDB INFORMATION_SCHEMA Metrics Table”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 21.30.15, “The INFORMATION_SCHEMA INNODB_METRICS Table”

innodb_monitor_reset_all

Section 14.15.6, “InnoDB INFORMATION_SCHEMA Metrics Table”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 21.30.15, “The INFORMATION_SCHEMA INNODB_METRICS Table”

innodb_numa_interleave

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_old_blocks_pct

Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.8.3.2, “Making the Buffer Pool Scan Resistant”
MySQL Glossary

innodb_old_blocks_time

Section 14.5.1, “Buffer Pool”
Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
Section 14.14, “InnoDB Startup Options and System Variables”

Section 14.8.3.2, “Making the Buffer Pool Scan Resistant”

innodb_online_alter_log_max_size

Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 14.13.5, “Online DDL Failure Conditions”
Section 14.13.3, “Online DDL Space Requirements”

innodb_open_files

Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 5.1.7, “Server System Variables”

innodb_optimize_fulltext_only

Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”
Section 14.6.2.3, “InnoDB FULLTEXT Indexes”
Section 14.15.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 13.7.2.4, “OPTIMIZE TABLE Statement”
Section 21.30.12, “The INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE Table”

innodb_page_cleaners

MySQL Glossary

innodb_page_size

Section 14.6.1.5, “Converting Tables from MyISAM to InnoDB”
Section 13.1.17, “CREATE TABLE Statement”
Section 2.12.3, “Downgrade Notes”
Section 14.9.2, “Enabling Compression for a Table”
Section 14.8.11.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”
Section 14.12.2, “File Space Management”
Section 14.9.5, “How Compression Works for InnoDB Tables”
Section 14.6.1.3, “Importing InnoDB Tables”
Section 14.22, “InnoDB Limits”
Section 14.23, “InnoDB Restrictions and Limitations”
Section 14.8.1, “InnoDB Startup Configuration”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 8.4.7, “Limits on Table Column Count and Row Size”
MySQL Glossary
Section 8.5.8, “Optimizing InnoDB Disk I/O”
Section 14.6.2.2, “The Physical Structure of an InnoDB Index”
Section 14.6.3.1, “The System Tablespace”
Section 14.20.8, “Troubleshooting the InnoDB memcached Plugin”
Section 14.6.7, “Undo Logs”
Section 1.3, “What Is New in MySQL 5.6”

innodb_print_all_deadlocks

Section 14.6.1.5, “Converting Tables from MyISAM to InnoDB”
Section 14.7.5, “Deadlocks in InnoDB”
Section 14.7.5.3, “How to Minimize and Handle Deadlocks”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.21, “InnoDB Troubleshooting”

innodb_purge_batch_size

Section 14.14, “InnoDB Startup Options and System Variables”

Section 14.8.10, “Purge Configuration”

innodb_purge_threads

Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 14.8.10, “Purge Configuration”

innodb_random_read_ahead

Section 14.8.3.3, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary

innodb_read_ahead_threshold

Section 14.8.3.3, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”
Section 14.14, “InnoDB Startup Options and System Variables”

innodb_read_io_threads

Section 14.8.6, “Configuring the Number of Background InnoDB I/O Threads”
Section 14.17.3, “InnoDB Standard Monitor and Lock Monitor Output”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.8.7, “Using Asynchronous I/O on Linux”

innodb_read_only

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_replication_delay

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_rollback_on_timeout

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_rollback_segments

Section 14.8.1, “InnoDB Startup Configuration”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 5.1.9, “Server Status Variables”
Section 14.6.7, “Undo Logs”
Section 14.6.3.3, “Undo Tablespaces”

innodb_saved_page_number_debug

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_sort_buffer_size

Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 14.13.3, “Online DDL Space Requirements”

innodb_spin_wait_delay

Section 14.8.9, “Configuring Spin Lock Polling”
Section 14.14, “InnoDB Startup Options and System Variables”

innodb_stats_auto_recalc

Configuring Automatic Statistics Calculation for Persistent Optimizer Statistics
Section 14.8.11, “Configuring Optimizer Statistics for InnoDB”
Configuring Optimizer Statistics Parameters for Individual Tables

Section 13.1.17, “CREATE TABLE Statement”
InnoDB Persistent Statistics Tables Example
Section 14.14, “InnoDB Startup Options and System Variables”

innodb_stats_include_delete_marked

Including Delete-marked Records in Persistent Statistics Calculations
Section 14.14, “InnoDB Startup Options and System Variables”

innodb_stats_method

Section 8.3.7, “InnoDB and MyISAM Index Statistics Collection”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary

innodb_stats_on_metadata

Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
Section 14.8.11.2, “Configuring Non-Persistent Optimizer Statistics Parameters”
Section 14.14, “InnoDB Startup Options and System Variables”

innodb_stats_persistent

Section 13.7.2.1, “ANALYZE TABLE Statement”
Section 14.8.11.2, “Configuring Non-Persistent Optimizer Statistics Parameters”
Section 14.8.11, “Configuring Optimizer Statistics for InnoDB”
Configuring Optimizer Statistics Parameters for Individual Tables
Section 14.8.11.1, “Configuring Persistent Optimizer Statistics Parameters”
Section 13.1.13, “CREATE INDEX Statement”
Section 13.1.17, “CREATE TABLE Statement”
Section 14.8.11.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 8.5.10, “Optimizing InnoDB for Systems with Many Tables”

innodb_stats_persistent_sample_pages

Section 13.7.2.1, “ANALYZE TABLE Statement”
Configuring Optimizer Statistics Parameters for Individual Tables
Configuring the Number of Sampled Pages for InnoDB Optimizer Statistics
Section 14.8.11.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”
Section 14.14, “InnoDB Startup Options and System Variables”

innodb_stats_sample_pages

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_stats_transient_sample_pages

Section 13.7.2.1, “ANALYZE TABLE Statement”
Section 14.8.11.2, “Configuring Non-Persistent Optimizer Statistics Parameters”
Section 14.8.11.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”
Section 14.14, “InnoDB Startup Options and System Variables”

innodb_status_output

Section 14.17.2, “Enabling InnoDB Monitors”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 1.3, “What Is New in MySQL 5.6”

innodb_status_output_locks

Section 14.17.2, “Enabling InnoDB Monitors”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 1.3, “What Is New in MySQL 5.6”

innodb_strict_mode

Section 13.1.17, “CREATE TABLE Statement”
Section 14.9.2, “Enabling Compression for a Table”
Section 14.9.5, “How Compression Works for InnoDB Tables”
Section 14.11, “InnoDB Row Formats”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 8.4.7, “Limits on Table Column Count and Row Size”
MySQL Glossary
Section 5.1.10, “Server SQL Modes”
Section 14.9.7, “SQL Compression Syntax Warnings and Errors”

innodb_support_xa

Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 8.5.2, “Optimizing InnoDB Transaction Management”

innodb_sync_array_size

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_sync_spin_loops

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_table_locks

Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”

innodb_temp_data_file_path

MySQL Glossary

innodb_thread_concurrency

Section 14.8.5, “Configuring Thread Concurrency for InnoDB”
Section 14.17.3, “InnoDB Standard Monitor and Lock Monitor Output”
Section 14.14, “InnoDB Startup Options and System Variables”
Section A.15, “MySQL 5.6 FAQ: MySQL Enterprise Thread Pool”
Section 8.5.9, “Optimizing InnoDB Configuration Variables”

innodb_thread_sleep_delay

Section 14.8.5, “Configuring Thread Concurrency for InnoDB”
Section 14.14, “InnoDB Startup Options and System Variables”

innodb_tmpdir

Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.13.5, “Online DDL Failure Conditions”
Section 14.13.3, “Online DDL Space Requirements”

innodb_trx_purge_view_update_only_debug

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_trx_rseg_n_slots_debug

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_undo_directory

Section 14.8.2, “Configuring InnoDB for Read-Only Operation”
Section 14.8.1, “InnoDB Startup Configuration”

Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.6.3.3, “Undo Tablespaces”

innodb_undo_logs

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_undo_tablespaces

Section 14.8.2, “Configuring InnoDB for Read-Only Operation”
Section 14.8.1, “InnoDB Startup Configuration”
Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 14.6.3.3, “Undo Tablespaces”

innodb_use_native_aio

Section 14.14, “InnoDB Startup Options and System Variables”
MySQL Glossary
Section 8.5.8, “Optimizing InnoDB Disk I/O”
Section 14.8.7, “Using Asynchronous I/O on Linux”

innodb_use_sys_malloc

Section 14.8.4, “Configuring the Memory Allocator for InnoDB”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 1.3, “What Is New in MySQL 5.6”

innodb_version

Section 14.14, “InnoDB Startup Options and System Variables”

innodb_write_io_threads

Section 14.8.6, “Configuring the Number of Background InnoDB I/O Threads”
Section 14.17.3, “InnoDB Standard Monitor and Lock Monitor Output”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.8.7, “Using Asynchronous I/O on Linux”

insert_id

Section 15.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 5.1.7, “Server System Variables”

interactive_timeout

Section B.3.2.10, “Communication Errors and Aborted Connections”
Section 5.1.7, “Server System Variables”

J

[index top]

join_buffer_size

Section 8.2.1.11, “Block Nested-Loop and Batched Key Access Joins”
Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
Section 8.2.1.6, “Nested-Loop Join Algorithms”
Section 5.1.7, “Server System Variables”

K

[index top]

keep_files_on_create
Section 5.1.7, “Server System Variables”

key_buffer_size
Section 8.6.2, “Bulk Data Loading for MyISAM Tables”
Section 5.1.1, “Configuring the Server”
Section 14.6.1.5, “Converting Tables from MyISAM to InnoDB”
Section 8.8.4, “Estimating Query Performance”
Section 8.12.4.1, “How MySQL Uses Memory”
Section 7.6.3, “How to Repair MyISAM Tables”
Section 14.8.1, “InnoDB Startup Configuration”
Section B.3.7, “Known Issues in MySQL”
Section 8.10.2.2, “Multiple Key Caches”
Section 8.2.4.3, “Optimizing DELETE Statements”
Section 8.6.3, “Optimizing REPAIR TABLE Statements”
Section 8.10.2.6, “Restructuring a Key Cache”
Section 5.1.6, “Server Command Options”
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”
Section 5.1.8.3, “Structured System Variables”
Section 8.10.2, “The MyISAM Key Cache”

key_cache_age_threshold
Section 8.10.2.3, “Midpoint Insertion Strategy”
Section 5.1.7, “Server System Variables”
Section 5.1.8.3, “Structured System Variables”

key_cache_block_size
Section 8.10.2.5, “Key Cache Block Size”
Section 8.10.2.6, “Restructuring a Key Cache”
Section 5.1.7, “Server System Variables”
Section 5.1.8.3, “Structured System Variables”

key_cache_division_limit
Section 8.10.2.3, “Midpoint Insertion Strategy”
Section 5.1.7, “Server System Variables”
Section 5.1.8.3, “Structured System Variables”

L

[index top]

large_files_support
Section 19.6, “Restrictions and Limitations on Partitioning”
Section 5.1.7, “Server System Variables”

large_page_size
Section 5.1.7, “Server System Variables”

large_pages
Section 5.1.7, “Server System Variables”

last_insert_id
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.4.1.35, “Replication and Variables”

Section 5.1.7, “Server System Variables”

lc_messages

Section 5.1.7, “Server System Variables”
Section 10.12, “Setting the Error Message Language”

lc_messages_dir

Section 5.1.7, “Server System Variables”
Section 10.12, “Setting the Error Message Language”

lc_time_names

Section 12.7, “Date and Time Functions”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 10.16, “MySQL Server Locale Support”
Section 17.4.1.35, “Replication and Variables”
Section 5.1.7, “Server System Variables”
Section 12.8, “String Functions and Operators”

license

Section 5.1.7, “Server System Variables”

local

Section 13.2.7, “LOAD XML Statement”

local_infile

Section 13.2.6, “LOAD DATA Statement”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 6.1.6, “Security Considerations for LOAD DATA LOCAL”
Section 5.1.7, “Server System Variables”

lock_wait_timeout

Section 5.1.7, “Server System Variables”

locked_in_memory

Section 5.1.7, “Server System Variables”

log

Section 14.14, “InnoDB Startup Options and System Variables”
Section 14.20.6, “The InnoDB memcached Plugin and Replication”

log_bin

Section 17.1.4.4, “Binary Log Options and Variables”
NDB Cluster System Variables
Section 5.4.4, “The Binary Log”
Section 1.3, “What Is New in MySQL 5.6”

log_bin_basename

Section 17.1.4.4, “Binary Log Options and Variables”
Section 1.3, “What Is New in MySQL 5.6”

log_bin_index

Section 17.1.4.4, “Binary Log Options and Variables”

log_bin_trust_function_creators

Section 17.1.4.4, “Binary Log Options and Variables”

Section A.4, “MySQL 5.6 FAQ: Stored Procedures and Functions”
Section 20.7, “Stored Program Binary Logging”

log_bin_use_v

Section 17.1.4.4, “Binary Log Options and Variables”
MySQL Server Options for NDB Cluster
Section 18.6.11, “NDB Cluster Replication Conflict Resolution”
Section 17.4.2, “Replication Compatibility Between MySQL Versions”

log_error

Section 5.4.2.2, “Error Logging on Unix and Unix-Like Systems”
Section 5.4.2.1, “Error Logging on Windows”
Section 2.5.8.2, “More Topics on Deploying MySQL Server with Docker”
Section 5.1.7, “Server System Variables”

log_output

Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.7, “Server System Variables”
Section 5.4.3, “The General Query Log”
Section 5.4.5, “The Slow Query Log”

log_queries_not_using_indexes

Section 5.1.7, “Server System Variables”
Section 5.4.5, “The Slow Query Log”

log_slave_updates

Section 17.1.4.4, “Binary Log Options and Variables”
Section 17.3.6, “Improving Replication Performance”
Section 18.6.3, “Known Issues in NDB Cluster Replication”
MySQL Server Options for NDB Cluster
Section 18.6.10, “NDB Cluster Replication: Bidirectional and Circular Replication”
NDB Cluster System Variables
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.3.7, “Switching Sources During Failover”

log_slow_admin_statements

Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 5.4.5, “The Slow Query Log”

log_slow_slave_statements

Section 17.1.4.3, “Replica Server Options and Variables”
Section 5.4.5, “The Slow Query Log”

log_throttle_queries_not_using_indexes

Section 5.1.7, “Server System Variables”
Section 5.4.5, “The Slow Query Log”

log_warnings

Section 5.4.2.4, “Error Log Filtering”
Section 22.12.8, “Performance Schema Connection Attribute Tables”
Section 22.15, “Performance Schema System Variables”
Section 5.1.7, “Server System Variables”

long_query_time

Section 5.4, “MySQL Server Logs”

Section 4.5.2, “mysqladmin — A MySQL Server Administration Program”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”
Section 5.4.5, “The Slow Query Log”

low_priority_updates

Section A.14, “MySQL 5.6 FAQ: Replication”
Section 5.1.7, “Server System Variables”
Section 8.11.2, “Table Locking Issues”
Section 1.3, “What Is New in MySQL 5.6”

lower_case_file_system

Section 5.1.7, “Server System Variables”

lower_case_table_names

Advanced Options
Section 13.1.17.5, “FOREIGN KEY Constraints”
Section 13.7.1.4, “GRANT Statement”
Section 17.2.3, “How Servers Evaluate Replication Filtering Rules”
Section 1.6, “How to Report Bugs or Problems”
Section 9.2.3, “Identifier Case Sensitivity”
Section 14.6.1.3, “Importing InnoDB Tables”
Section 17.4.1.35, “Replication and Variables”
Section 13.7.1.6, “REVOKE Statement”
Section 5.1.7, “Server System Variables”
Section 13.7.5.38, “SHOW TABLES Statement”
Section 21.30.16, “The INFORMATION_SCHEMA INNODB_SYS_COLUMNS Table”
Section 21.30.22, “The INFORMATION_SCHEMA INNODB_SYS_TABLES Table”
Section 10.8.7, “Using Collation in INFORMATION_SCHEMA Searches”

M

[index top]

master_info_repository

Section 13.4.2.1, “CHANGE MASTER TO Statement”
Section 13.7.6.3, “FLUSH Statement”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.2.2.2, “Replication Metadata Repositories”
Section 17.1.1.8, “Setting Up Replication with Existing Data”
Section 1.3, “What Is New in MySQL 5.6”

master_verify_checksum

Section 17.1.4.4, “Binary Log Options and Variables”
MySQL Glossary
Section 5.4.4, “The Binary Log”
Section 1.3, “What Is New in MySQL 5.6”

max_allowed_packet

Section 12.19.1, “Aggregate Function Descriptions”
Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
Section B.3.2.10, “Communication Errors and Aborted Connections”
Section 12.4.2, “Comparison Functions and Operators”

Section B.3.4.6, “Deleting Rows from Related Tables”
Section 8.12.4.1, “How MySQL Uses Memory”
Section B.3.2.3, “Lost connection to MySQL server”
Section B.3.2.8, “MySQL server has gone away”
Section 18.6.11, “NDB Cluster Replication Conflict Resolution”
Section B.3.2.9, “Packet Too Large”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.4.1.19, “Replication and max_allowed_packet”
Section 5.1.7, “Server System Variables”
Section 12.8, “String Functions and Operators”
Section 11.3.4, “The BLOB and TEXT Types”
Section 1.3, “What Is New in MySQL 5.6”

max_binlog_cache_size

Section 17.1.4.4, “Binary Log Options and Variables”
Section 5.4.4, “The Binary Log”

max_binlog_size

Section 17.1.4.4, “Binary Log Options and Variables”
Section 5.4, “MySQL Server Logs”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 5.4.7, “Server Log Maintenance”
Section 5.4.4, “The Binary Log”
Section 17.2.2.1, “The Relay Log”

max_binlog_stmt_cache_size

Section 17.1.4.4, “Binary Log Options and Variables”

max_connect_errors

Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
Section 5.1.11.2, “DNS Lookups and the Host Cache”
Section 5.1.7, “Server System Variables”
Section 22.12.10.1, “The host_cache Table”
Section 1.3, “What Is New in MySQL 5.6”

max_connections

Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
Section 5.1.11.1, “Connection Interfaces”
Section 5.8.1.4, “Debugging mysqld under gdb”
Section B.3.2.17, “File Not Found and Similar Errors”
Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section 22.15, “Performance Schema System Variables”
Section 6.2.2, “Privileges Provided by MySQL”
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”
Section 5.5.3.3, “Thread Pool Operation”
Section B.3.2.6, “Too many connections”

max_delayed_threads

Section 5.1.7, “Server System Variables”

max_digest_length

Section 8.12.4.1, “How MySQL Uses Memory”
Section 6.4.5.4, “MySQL Enterprise Firewall Reference”

Section 22.10, “Performance Schema Statement Digests”
Section 22.15, “Performance Schema System Variables”
Section 5.1.7, “Server System Variables”
Section 22.12.6.1, “The events_statements_current Table”

max_error_count

Section 13.2.6, “LOAD DATA Statement”
Section 13.6.7.4, “RESIGNAL Statement”
Section 5.1.7, “Server System Variables”
Section 13.7.5.18, “SHOW ERRORS Statement”
Section 13.7.5.41, “SHOW WARNINGS Statement”
Section 13.6.7.7, “The MySQL Diagnostics Area”

max_heap_table_size

Section 8.12.4.1, “How MySQL Uses Memory”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 8.4.6, “Limits on Table Size”
Section 17.4.1.20, “Replication and MEMORY Tables”
Section 17.4.1.35, “Replication and Variables”
Section 13.6.6.5, “Restrictions on Server-Side Cursors”
Section 5.1.7, “Server System Variables”
Section 15.3, “The MEMORY Storage Engine”

max_insert_delayed_threads

Section 5.1.7, “Server System Variables”

max_join_size

Section 8.8.2, “EXPLAIN Output Format”
Section 4.5.1.6, “mysql Client Tips”
Section 5.1.7, “Server System Variables”
Section 13.7.4.1, “SET Syntax for Variable Assignment”
Section 1.3, “What Is New in MySQL 5.6”

max_length_for_sort_data

Section 8.2.1.13, “ORDER BY Optimization”
Section 5.1.7, “Server System Variables”

max_prepared_stmt_count

Section 8.10.4, “Caching of Prepared Statements and Stored Programs”
Section 13.5.3, “DEALLOCATE PREPARE Statement”
Section 13.5, “Prepared Statements”
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”

max_relay_log_size

Section 17.1.4.4, “Binary Log Options and Variables”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.2.2.1, “The Relay Log”

max_seeks_for_key

Section 13.7.2.1, “ANALYZE TABLE Statement”
Section 5.1.7, “Server System Variables”

max_sort_length

Section B.3.7, “Known Issues in MySQL”

Section 8.2.1.13, “ORDER BY Optimization”
Section 13.2.9, “SELECT Statement”
Section 5.1.7, “Server System Variables”
Section 11.3.4, “The BLOB and TEXT Types”

max_sp_recursion_depth

Section 5.1.7, “Server System Variables”
Section 20.2.1, “Stored Routine Syntax”

max_tmp_tables

Section 5.1.7, “Server System Variables”

max_user_connections

Section 13.7.6.3, “FLUSH Statement”
Section 13.7.1.4, “GRANT Statement”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 5.1.7, “Server System Variables”
Section 6.2.13, “Setting Account Resource Limits”

max_write_lock_count

Section 13.3.5, “LOCK TABLES and UNLOCK TABLES Statements”
Section 8.11.4, “Metadata Locking”
Section 5.1.7, “Server System Variables”
Section 8.11.2, “Table Locking Issues”

metadata_locks_cache_size

Section 5.1.7, “Server System Variables”

metadata_locks_hash_instances

Section 5.1.7, “Server System Variables”

min_examined_row_limit

Section 5.1.7, “Server System Variables”
Section 5.4.5, “The Slow Query Log”

multi_range_count

Section 5.1.7, “Server System Variables”

myisam_data_pointer_size

Section 13.1.17, “CREATE TABLE Statement”
Section 8.4.6, “Limits on Table Size”
Section 5.1.7, “Server System Variables”

myisam_max_sort_file_size

Section 15.2.1, “MyISAM Startup Options”
Section 8.6.3, “Optimizing REPAIR TABLE Statements”
Section 19.6, “Restrictions and Limitations on Partitioning”
Section 5.1.7, “Server System Variables”

myisam_mmap_size

Section 5.1.7, “Server System Variables”

myisam_recover_options

Section 15.2.1, “MyISAM Startup Options”

Section 8.6.1, “Optimizing MyISAM Queries”
Section 5.1.7, “Server System Variables”
Section 7.6.5, “Setting Up a MyISAM Table Maintenance Schedule”
Section B.3.2.18, “Table-Corruption Issues”
Section 15.2, “The MyISAM Storage Engine”
Section 5.8.1.6, “Using Server Logs to Find Causes of Errors in mysqld”

myisam_repair_threads

Section 5.1.7, “Server System Variables”

myisam_sort_buffer_size

Section 13.1.7, “ALTER TABLE Statement”
Section 15.2.1, “MyISAM Startup Options”
Section 4.6.3.1, “myisamchk General Options”
Section 8.6.3, “Optimizing REPAIR TABLE Statements”
Section 5.1.7, “Server System Variables”

myisam_stats_method

Section 8.3.7, “InnoDB and MyISAM Index Statistics Collection”
Section 5.1.7, “Server System Variables”

myisam_use_mmap

Section 8.12.4.1, “How MySQL Uses Memory”
Section 5.1.7, “Server System Variables”

mysql_firewall_max_query_size

Section 6.4.5.4, “MySQL Enterprise Firewall Reference”

mysql_firewall_mode

Section 6.4.5.4, “MySQL Enterprise Firewall Reference”
Section 6.4.5.3, “Using MySQL Enterprise Firewall”

mysql_firewall_trace

Section 6.4.5.4, “MySQL Enterprise Firewall Reference”
Section 6.4.5.3, “Using MySQL Enterprise Firewall”

N

[index top]

named_pipe

Section B.3.2.2, “Can't connect to [local] MySQL server”
Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”
Section 2.3.4.3, “Selecting a MySQL Server Type”
Section 5.1.7, “Server System Variables”
Section 1.2.2, “The Main Features of MySQL”

Type and Networking

named_pipe_full_access_group

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 4.2.5, “Connection Transport Protocols”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”
Section 5.1.7, “Server System Variables”
Section 1.3, “What Is New in MySQL 5.6”

ndb_autoincrement_prefetch_sz

NDB Cluster System Variables

ndb_cache_check_time

NDB Cluster System Variables

ndb_clear_apply_status

NDB Cluster System Variables
Section 13.4.2.3, “RESET SLAVE Statement”

ndb_deferred_constraints

NDB Cluster System Variables

ndb_distribution

NDB Cluster System Variables

ndb_eventbuffer_free_percent

NDB Cluster System Variables
Section 18.1.4.2, “What is New in NDB Cluster 7.4”

ndb_eventbuffer_max_alloc

NDB Cluster System Variables

ndb_extra_logging

NDB Cluster System Variables

ndb_force_send

NDB Cluster System Variables

ndb_index_stat_cache_entries

NDB Cluster System Variables

ndb_index_stat_enable

NDB Cluster System Variables

ndb_index_stat_option

NDB Cluster System Variables

ndb_index_stat_update_freq

NDB Cluster System Variables

ndb_join_pushdown

Section 8.8.2, “EXPLAIN Output Format”
NDB Cluster System Variables

ndb_log_apply_status

Section 18.6.10, “NDB Cluster Replication: Bidirectional and Circular Replication”
NDB Cluster System Variables

ndb_log_bin

NDB Cluster System Variables

ndb_log_binlog_index

NDB Cluster System Variables

ndb_log_empty_epochs

NDB Cluster System Variables

ndb_log_empty_update

NDB Cluster System Variables

ndb_log_exclusive_reads

MySQL Server Options for NDB Cluster
Section 18.6.11, “NDB Cluster Replication Conflict Resolution”
NDB Cluster System Variables
Section 18.1.4.2, “What is New in NDB Cluster 7.4”

ndb_log_orig

NDB Cluster System Variables

ndb_log_transaction_id

NDB Cluster System Variables

ndb_optimized_node_selection

NDB Cluster System Variables
Section 18.3.3.9, “NDB Cluster TCP/IP Connections”
Section 18.5.3.3, “Using CLUSTERLOG STATISTICS in the NDB Cluster Management Client”

ndb_recv_thread_activation_threshold

NDB Cluster System Variables

ndb_recv_thread_cpu_mask

NDB Cluster System Variables

ndb_report_thresh_binlog_epoch_slip

NDB Cluster System Variables

ndb_report_thresh_binlog_mem_usage

NDB Cluster System Variables

ndb_show_foreign_key_mock_tables

NDB Cluster System Variables

ndb_slave_conflict_role

Section 18.6.11, “NDB Cluster Replication Conflict Resolution”
NDB Cluster System Variables
Section 18.1.4, “What is New in MySQL NDB Cluster”
Section 18.1.4.2, “What is New in NDB Cluster 7.4”

ndb_table_no_logging

NDB Cluster System Variables

ndb_table_temporary

NDB Cluster System Variables

ndb_use_copying_alter_table

NDB Cluster System Variables

ndb_use_exact_count

NDB Cluster System Variables

ndb_use_transactions

NDB Cluster System Variables

ndb_version

NDB Cluster System Variables

ndb_version_string

NDB Cluster System Variables

ndbinfo_database

NDB Cluster System Variables

ndbinfo_max_bytes

NDB Cluster System Variables

ndbinfo_max_rows

NDB Cluster System Variables

ndbinfo_offline

NDB Cluster System Variables

ndbinfo_show_hidden

NDB Cluster System Variables
Section 18.5.14.4, “The ndbinfo cluster_operations Table”
Section 18.5.14.5, “The ndbinfo cluster_transactions Table”
Section 18.5.14.22, “The ndbinfo server_operations Table”
Section 18.5.14.23, “The ndbinfo server_transactions Table”

ndbinfo_table_prefix

NDB Cluster System Variables

ndbinfo_version

NDB Cluster System Variables

net_buffer_length

Section 8.12.4.1, “How MySQL Uses Memory”

Section 4.5.4, “mysqldump — A Database Backup Program”
Section 5.1.7, “Server System Variables”

net_read_timeout

Section B.3.2.3, “Lost connection to MySQL server”
Section 5.1.7, “Server System Variables”

net_retry_count

Section 5.1.7, “Server System Variables”

net_write_timeout

Section 5.1.7, “Server System Variables”

new

Section 19.6.2, “Partitioning Limitations Relating to Storage Engines”
Section 5.1.7, “Server System Variables”

O

[index top]

old

Section 8.9.3, “Index Hints”
Section 5.1.7, “Server System Variables”

old_alter_table

Section 13.1.7, “ALTER TABLE Statement”
Section 14.13, “InnoDB and Online DDL”
Section 14.13.2, “Online DDL Performance and Concurrency”
Section 13.7.2.4, “OPTIMIZE TABLE Statement”
Section 5.1.7, “Server System Variables”
Section 14.13.4, “Simplifying DDL Statements with Online DDL”

old_passwords

Section 13.7.1.1, “ALTER USER Statement”
Section 6.2.9, “Assigning Account Passwords”
Section 2.11.3, “Changes in MySQL 5.6”
Section B.3.2.4, “Client does not support authentication protocol”
Section 13.7.1.2, “CREATE USER Statement”
Section 12.14, “Encryption and Compression Functions”
Section 6.1.2.5, “Implications of Password Hashing Changes in MySQL 4.1 for Application Programs”
Section 6.1.2.4, “Password Hashing in MySQL”
Section 5.1.6, “Server Command Options”
Section 6.2.10, “Server Handling of Expired Passwords”
Section 5.1.7, “Server System Variables”
Section 13.7.1.7, “SET PASSWORD Statement”
Section 6.4.1.4, “SHA-256 Pluggable Authentication”
Section 1.3, “What Is New in MySQL 5.6”

open_files_limit

Section 5.1.2.1, “Changes to Server Defaults”
Section 5.1.11.1, “Connection Interfaces”
Section B.3.2.17, “File Not Found and Similar Errors”
Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 22.15, “Performance Schema System Variables”

Section 19.6, “Restrictions and Limitations on Partitioning”
Section 5.1.7, “Server System Variables”

optimizer_prune_level

Section 8.9.1, “Controlling Query Plan Evaluation”
Section 5.1.7, “Server System Variables”

optimizer_search_depth

Section 8.9.1, “Controlling Query Plan Evaluation”
Section 5.1.7, “Server System Variables”

optimizer_switch

Section 8.2.1.11, “Block Nested-Loop and Batched Key Access Joins”
Section 8.2.1.4, “Engine Condition Pushdown Optimization”
Section 8.2.1.5, “Index Condition Pushdown Optimization”
Section 8.2.1.3, “Index Merge Optimization”
Section 8.2.1.10, “Multi-Range Read Optimization”
Section 8.2.2.2, “Optimizing Subqueries with Materialization”
Section 8.2.2.1, “Optimizing Subqueries with Semijoin Transformations”
Section 8.2.2.3, “Optimizing Subqueries with the EXISTS Strategy”
Section 5.1.7, “Server System Variables”
Section 8.9.2, “Switchable Optimizations”
Section 13.2.11, “UPDATE Statement”
Section 8.3.9, “Use of Index Extensions”
Section 1.3, “What Is New in MySQL 5.6”

optimizer_trace

Section 5.1.7, “Server System Variables”
Section 21.12, “The INFORMATION_SCHEMA OPTIMIZER_TRACE Table”

optimizer_trace_features

Section 5.1.7, “Server System Variables”

optimizer_trace_limit

Section 5.1.7, “Server System Variables”

optimizer_trace_max_mem_size

Section 5.1.7, “Server System Variables”
Section 21.12, “The INFORMATION_SCHEMA OPTIMIZER_TRACE Table”

optimizer_trace_offset

Section 5.1.7, “Server System Variables”

P

[index top]

performance_schema

Section 5.1.2.1, “Changes to Server Defaults”
Section 22.1, “Performance Schema Quick Start”
Section 22.3, “Performance Schema Startup Configuration”
Section 22.15, “Performance Schema System Variables”

performance_schema_accounts_size

Section 22.15, “Performance Schema System Variables”

Section 22.12.7.1, “The accounts Table”

performance_schema_digests_size

Section 22.10, “Performance Schema Statement Digests”
Section 22.16, “Performance Schema Status Variables”
Section 22.15, “Performance Schema System Variables”
Section 22.12.9.3, “Statement Summary Tables”

performance_schema_events_stages_history_long_size

Section 22.15, “Performance Schema System Variables”
Section 22.12.5.3, “The events_stages_history_long Table”

performance_schema_events_stages_history_size

Section 22.15, “Performance Schema System Variables”
Section 22.18.1, “Query Profiling Using Performance Schema”
Section 22.12.5.2, “The events_stages_history Table”

performance_schema_events_statements_history_long_size

Section 22.15, “Performance Schema System Variables”
Section 22.12.6.3, “The events_statements_history_long Table”

performance_schema_events_statements_history_size

Section 22.15, “Performance Schema System Variables”
Section 22.18.1, “Query Profiling Using Performance Schema”
Section 22.12.6.2, “The events_statements_history Table”

performance_schema_events_waits_history_long_size

Section 5.1.2.1, “Changes to Server Defaults”
Section 22.15, “Performance Schema System Variables”
Section 22.12, “Performance Schema Table Descriptions”
Section 13.7.5.16, “SHOW ENGINE Statement”
Section 22.12.4.3, “The events_waits_history_long Table”

performance_schema_events_waits_history_size

Section 5.1.2.1, “Changes to Server Defaults”
Section 22.15, “Performance Schema System Variables”
Section 22.12, “Performance Schema Table Descriptions”
Section 13.7.5.16, “SHOW ENGINE Statement”
Section 22.12.4.2, “The events_waits_history Table”

performance_schema_hosts_size

Section 22.15, “Performance Schema System Variables”
Section 22.12.7.2, “The hosts Table”

performance_schema_max_cond_classes

Section 22.15, “Performance Schema System Variables”

performance_schema_max_cond_instances

Section 5.1.2.1, “Changes to Server Defaults”
Section 22.15, “Performance Schema System Variables”

performance_schema_max_digest_length

Section 22.10, “Performance Schema Statement Digests”
Section 22.15, “Performance Schema System Variables”
Section 5.1.7, “Server System Variables”

Section 22.12.6.1, “The events_statements_current Table”

performance_schema_max_file_classes

Section 22.15, “Performance Schema System Variables”

performance_schema_max_file_handles

Section 22.15, “Performance Schema System Variables”

performance_schema_max_file_instances

Section 5.1.2.1, “Changes to Server Defaults”
Section 22.15, “Performance Schema System Variables”

performance_schema_max_mutex_classes

Section 22.7, “Performance Schema Status Monitoring”
Section 22.15, “Performance Schema System Variables”

performance_schema_max_mutex_instances

Section 5.1.2.1, “Changes to Server Defaults”
Section 22.15, “Performance Schema System Variables”

performance_schema_max_rwlock_classes

Section 22.15, “Performance Schema System Variables”

performance_schema_max_rwlock_instances

Section 5.1.2.1, “Changes to Server Defaults”
Section 22.15, “Performance Schema System Variables”

performance_schema_max_socket_classes

Section 22.15, “Performance Schema System Variables”

performance_schema_max_socket_instances

Section 22.15, “Performance Schema System Variables”

performance_schema_max_stage_classes

Section 22.15, “Performance Schema System Variables”

performance_schema_max_statement_classes

Section 22.15, “Performance Schema System Variables”

performance_schema_max_table_handles

Section 5.1.2.1, “Changes to Server Defaults”
Section 22.15, “Performance Schema System Variables”

performance_schema_max_table_instances

Section 5.1.2.1, “Changes to Server Defaults”
Section 22.15, “Performance Schema System Variables”

performance_schema_max_thread_classes

Section 22.15, “Performance Schema System Variables”

performance_schema_max_thread_instances

Section 5.1.2.1, “Changes to Server Defaults”
Section 22.16, “Performance Schema Status Variables”

Section 22.15, “Performance Schema System Variables”
Section 13.7.5.16, “SHOW ENGINE Statement”

performance_schema_session_connect_attrs_size

Section 22.12.8, “Performance Schema Connection Attribute Tables”
Section 22.16, “Performance Schema Status Variables”
Section 22.15, “Performance Schema System Variables”

performance_schema_setup_actors_size

Section 22.15, “Performance Schema System Variables”
Section 22.12.2.1, “The setup_actors Table”

performance_schema_setup_objects_size

Section 22.15, “Performance Schema System Variables”
Section 22.12.2.4, “The setup_objects Table”

performance_schema_users_size

Section 22.15, “Performance Schema System Variables”
Section 22.12.7.3, “The users Table”

pid

Section 14.8.2, “Configuring InnoDB for Read-Only Operation”

pid_file

Section 5.1.7, “Server System Variables”

plugin_dir

Section 6.1.2.2, “Administrator Guidelines for Password Security”
Section 6.4.2.1, “Connection-Control Plugin Installation”
Section 13.7.3.1, “CREATE FUNCTION Statement for User-Defined Functions”
Section 13.7.3.3, “INSTALL PLUGIN Statement”
Section 5.5.1, “Installing and Uninstalling Plugins”
Section 6.4.4.1, “Installing MySQL Enterprise Audit”
Section 2.5.5, “Installing MySQL on Linux Using RPM Packages from Oracle”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 6.5.1, “MySQL Enterprise Encryption Installation”
Section 5.6.2, “Obtaining User-Defined Function Information”
Section 6.4.1.6, “PAM Pluggable Authentication”
Section 6.4.3.1, “Password Validation Plugin Installation”
Section 6.2.11, “Pluggable Authentication”
Section 15.11.1, “Pluggable Storage Engine Architecture”
Section 17.3.9.2, “Semisynchronous Replication Installation and Configuration”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 14.20.3, “Setting Up the InnoDB memcached Plugin”
Section 13.7.5.26, “SHOW PLUGINS Statement”
Section 6.4.1.8, “Socket Peer-Credential Pluggable Authentication”
Section 6.4.1.9, “Test Pluggable Authentication”
Section 21.15, “The INFORMATION_SCHEMA PLUGINS Table”
Section 5.5.3.2, “Thread Pool Installation”
Section 6.4.1.7, “Windows Pluggable Authentication”

port

Section B.3.2.2, “Can't connect to [local] MySQL server”
Section 5.1.7, “Server System Variables”
Section 6.6.5.1, “Setting the TCP Port Context for mysqld”

preload_buffer_size
Section 5.1.7, “Server System Variables”

profiling
Section 5.1.7, “Server System Variables”
Section 13.7.5.31, “SHOW PROFILE Statement”
Section 21.17, “The INFORMATION_SCHEMA PROFILING Table”
Section 1.3, “What Is New in MySQL 5.6”

profiling_history_size
Section 5.1.7, “Server System Variables”
Section 13.7.5.31, “SHOW PROFILE Statement”
Section 1.3, “What Is New in MySQL 5.6”

protocol_version
Section 5.1.7, “Server System Variables”

proxy_user
Section 6.2.12, “Proxy Users”
Section 5.1.7, “Server System Variables”

pseudo_slave_mode
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 5.1.7, “Server System Variables”

pseudo_thread_id
Section 12.16, “Information Functions”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.4.1.35, “Replication and Variables”
Section 5.1.7, “Server System Variables”

Q

[index top]

query_alloc_block_size
Section 5.1.7, “Server System Variables”

query_cache_limit
Section 8.10.3.3, “Query Cache Configuration”
Section 5.1.7, “Server System Variables”

query_cache_min_res_unit
Section 8.10.3.3, “Query Cache Configuration”
Section 5.1.7, “Server System Variables”

query_cache_size
Section 5.1.2.1, “Changes to Server Defaults”
Section 8.10.3.3, “Query Cache Configuration”
Section 5.1.7, “Server System Variables”
Section 8.10.3, “The MySQL Query Cache”

query_cache_type
Section 5.1.2.1, “Changes to Server Defaults”
Section 8.10.3.3, “Query Cache Configuration”

Section 8.10.3.2, “Query Cache SELECT Options”
Section 13.2.9, “SELECT Statement”
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”

query_cache_wlock_invalidate

Section 5.1.7, “Server System Variables”

query_prealloc_size

Section 5.1.7, “Server System Variables”

R

[index top]

rand_seed

Section 5.1.7, “Server System Variables”

range_alloc_block_size

Section 5.1.7, “Server System Variables”

read_buffer_size

Section 8.12.4.1, “How MySQL Uses Memory”
Section 8.6.3, “Optimizing REPAIR TABLE Statements”
Section 5.1.7, “Server System Variables”

read_only

Section 13.7.1, “Account Management Statements”
Section 13.7.1.1, “ALTER USER Statement”
Section 6.2.9, “Assigning Account Passwords”
Section 17.3.1.3, “Backing Up a Source or Replica by Making It Read Only”
Section 13.7.1.2, “CREATE USER Statement”
Section 17.1.3.5, “Disabling GTID Transactions”
Section 13.7.1.3, “DROP USER Statement”
Section 8.14.3, “General Thread States”
Section 13.7.1.4, “GRANT Statement”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.7.1.5, “RENAME USER Statement”
Section 17.4.1.35, “Replication and Variables”
Section 13.7.1.6, “REVOKE Statement”
Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.7, “Server System Variables”
Section 13.7.1.7, “SET PASSWORD Statement”
Section 17.1.3.2, “Setting Up Replication Using GTIDs”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”

read_rnd_buffer_size

Section 5.1.1, “Configuring the Server”
Section 8.12.4.1, “How MySQL Uses Memory”
Section 8.2.1.10, “Multi-Range Read Optimization”
Section 8.2.1.13, “ORDER BY Optimization”
Section 5.1.7, “Server System Variables”

relay_log

Section 17.3.6, “Improving Replication Performance”
Section 17.1.1.9, “Introducing Additional Replicas to an Existing Replication Environment”

Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.2.2.1, “The Relay Log”

relay_log_basename

Section 17.1.4.3, “Replica Server Options and Variables”
Section 1.3, “What Is New in MySQL 5.6”

relay_log_index

Section 17.1.1.9, “Introducing Additional Replicas to an Existing Replication Environment”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.2.2.1, “The Relay Log”

relay_log_info_file

Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.2.2.2, “Replication Metadata Repositories”

relay_log_info_repository

Section 13.7.6.3, “FLUSH Statement”
Section 17.3.2, “Handling an Unexpected Halt of a Replica Server”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.2.2.2, “Replication Metadata Repositories”
Section 1.3, “What Is New in MySQL 5.6”

relay_log_purge

Section 13.4.2.1, “CHANGE MASTER TO Statement”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”

relay_log_recovery

Section 17.3.2, “Handling an Unexpected Halt of a Replica Server”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 13.4.2.5, “START SLAVE Statement”

relay_log_space_limit

Section 17.1.4.3, “Replica Server Options and Variables”
Section 8.14.7, “Replication Replica I/O Thread States”

report_host

Section 17.1.4.3, “Replica Server Options and Variables”

report_password

Section 17.1.4.3, “Replica Server Options and Variables”

report_port

Section 17.1.4.3, “Replica Server Options and Variables”

report_user

Section 17.1.4.3, “Replica Server Options and Variables”

require_secure_transport

Section 4.2.5, “Connection Transport Protocols”

rpl_semi_sync_master_enabled

Section 17.1.4.2, “Replication Source Options and Variables”

Section 17.3.9.1, “Semisynchronous Replication Administrative Interface”
Section 17.3.9.2, “Semisynchronous Replication Installation and Configuration”
Section 17.3.9.3, “Semisynchronous Replication Monitoring”

rpl_semi_sync_master_timeout

Section 17.1.4.2, “Replication Source Options and Variables”
Section 17.3.9.1, “Semisynchronous Replication Administrative Interface”
Section 17.3.9.2, “Semisynchronous Replication Installation and Configuration”

rpl_semi_sync_master_trace_level

Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.1.4.2, “Replication Source Options and Variables”

rpl_semi_sync_master_wait_no_slave

Section 17.1.4.2, “Replication Source Options and Variables”

rpl_semi_sync_slave_enabled

Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.3.9.1, “Semisynchronous Replication Administrative Interface”
Section 17.3.9.2, “Semisynchronous Replication Installation and Configuration”

rpl_semi_sync_slave_trace_level

Section 17.1.4.3, “Replica Server Options and Variables”

rpl_stop_slave_timeout

Section 17.1.4.3, “Replica Server Options and Variables”
Section 13.4.2.6, “STOP SLAVE Statement”

S

[index top]

secure_auth

Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
Section 6.4.1.3, “Migrating Away from Pre-4.1 Password Hashing and the mysql_old_password Plugin”
Section 6.1.2.4, “Password Hashing in MySQL”
Section 5.1.7, “Server System Variables”
Section 1.3, “What Is New in MySQL 5.6”

secure_file_priv

Section 2.10.1, “Initializing the Data Directory”
Section 13.2.6, “LOAD DATA Statement”
Section 6.1.3, “Making MySQL Secure Against Attackers”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.2.9.1, “SELECT ... INTO Statement”
Section 6.6.4, “SELinux File Context”
Section 5.1.7, “Server System Variables”
Section 12.8, “String Functions and Operators”

server_id

Advanced Options
Section 6.4.4.3, “Audit Log File Formats”
Section 17.1.1.9, “Introducing Additional Replicas to an Existing Replication Environment”

Section 12.20, “Miscellaneous Functions”
MySQL Server Options for NDB Cluster
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 18.6.11, “NDB Cluster Replication Conflict Resolution”
NDB Cluster System Variables
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.1.4, “Replication and Binary Logging Options and Variables”
Section 17.1, “Replication Configuration”
Section 17.1.4.2, “Replication Source Options and Variables”
Section 17.1.1.2, “Setting the Replica Configuration”
Section 17.1.1.1, “Setting the Replication Source Configuration”
Section 17.1.1.8, “Setting Up Replication with Existing Data”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”
Section 17.4.4, “Troubleshooting Replication”
Section 17.1.2.2, “Usage of Row-Based Logging and Replication”

server_id_bits

MySQL Server Options for NDB Cluster
NDB Cluster System Variables

server_uuid

Section 17.1.3.1, “GTID Concepts”
Section 17.1.4, “Replication and Binary Logging Options and Variables”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”
Section 13.4.2.5, “START SLAVE Statement”

sha

Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”
Section 6.4.1.4, “SHA-256 Pluggable Authentication”
Section 6.3.4, “SSL Library-Dependent Capabilities”

shared_memory

Section 4.2.3, “Command Options for Connecting to the Server”
Section 4.2.4, “Connecting to the MySQL Server Using Command Options”
Section 4.5.1.1, “mysql Client Options”
Section 4.4.7, “`mysql_upgrade` — Check and Upgrade MySQL Tables”
Section 4.5.2, “`mysqladmin` — A MySQL Server Administration Program”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 4.5.3, “`mysqlcheck` — A Table Maintenance Program”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 4.5.5, “`mysqlimport` — A Data Import Program”
Section 4.5.6, “`mysqlshow` — Display Database, Table, and Column Information”
Section 4.5.7, “`mysqlslap` — A Load Emulation Client”
Section 5.1.7, “Server System Variables”
Section 5.7.2.1, “Starting Multiple MySQL Instances at the Windows Command Line”
Section 2.3.4.4, “Starting the Server for the First Time”
Section 1.2.2, “The Main Features of MySQL”
Type and Networking

shared_memory_base_name

Section 5.1.7, “Server System Variables”
Section 5.7.2.1, “Starting Multiple MySQL Instances at the Windows Command Line”

show_old_temporals

Section 2.11.3, “Changes in MySQL 5.6”
NDB Cluster System Variables

Section 5.1.7, “Server System Variables”

simplified_binlog_gtid_recovery

Section 17.1.4.5, “Global Transaction ID Options and Variables”

skip_external_locking

Section 8.11.5, “External Locking”
Section 5.1.7, “Server System Variables”

skip_name_resolve

Section 5.1.11.2, “DNS Lookups and the Host Cache”
Section 2.10.4, “Securing the Initial MySQL Accounts”
Section 5.1.7, “Server System Variables”
Section 2.3.4.8, “Testing The MySQL Installation”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”

skip_networking

Section B.3.2.2, “Can't connect to [local] MySQL server”
Section 5.1.11.2, “DNS Lookups and the Host Cache”
Section A.14, “MySQL 5.6 FAQ: Replication”
Section B.3.2.8, “MySQL server has gone away”
Section 6.2.11, “Pluggable Authentication”
Resetting the Root Password: Generic Instructions
Section 5.1.7, “Server System Variables”
Section 17.1.1.1, “Setting the Replication Source Configuration”
Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
Section 17.4.4, “Troubleshooting Replication”
Section 17.4.3, “Upgrading a Replication Setup”

skip_show_database

Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”

slave_allow_batching

NDB Cluster System Variables
Section 18.6.5, “Preparing the NDB Cluster for Replication”
Section 18.6.6, “Starting NDB Cluster Replication (Single Replication Channel)”

slave_checkpoint_group

Section 18.6.3, “Known Issues in NDB Cluster Replication”
Section 12.20, “Miscellaneous Functions”
Section 17.1.4.3, “Replica Server Options and Variables”

slave_checkpoint_period

Section 12.20, “Miscellaneous Functions”
Section 17.1.4.3, “Replica Server Options and Variables”

slave_compressed_protocol

Section 4.2.6, “Connection Compression Control”
Section 17.1.4.3, “Replica Server Options and Variables”

slave_exec_mode

Section 18.6.3, “Known Issues in NDB Cluster Replication”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.4.1.20, “Replication and MEMORY Tables”
Section 17.1.2.2, “Usage of Row-Based Logging and Replication”

slave_load_tmpdir

Section 17.3.1.2, “Backing Up Raw Data from a Replica”
Section 7.2, “Database Backup Methods”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section B.3.3.5, “Where MySQL Stores Temporary Files”

slave_max_allowed_packet

Section 17.1.4.3, “Replica Server Options and Variables”

slave_net_timeout

Section 13.4.2.1, “CHANGE MASTER TO Statement”
Section 17.1.5.1, “Checking Replication Status”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.4.1.26, “Replication and Source or Replica Shutdowns”
Section 8.14.7, “Replication Replica I/O Thread States”
Section 5.1.7, “Server System Variables”

slave_parallel_type

Section 17.2.1.2, “Monitoring Replication Applier Worker Threads”

slave_parallel_workers

Section 18.6.3, “Known Issues in NDB Cluster Replication”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 8.14.8, “Replication Replica SQL Thread States”
Section 17.2.1, “Replication Threads”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”
Section 13.4.2.5, “START SLAVE Statement”
Section 1.3, “What Is New in MySQL 5.6”

slave_pending_jobs_size_max

Section 17.2.1.2, “Monitoring Replication Applier Worker Threads”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 8.14.8, “Replication Replica SQL Thread States”

slave_rows_search_algorithms

Section 17.1.4.3, “Replica Server Options and Variables”
Section 5.1.9, “Server Status Variables”

slave_skip_errors

Section 17.1.4.3, “Replica Server Options and Variables”

slave_sql_verify_checksum

MySQL Glossary
Section 17.1.4.3, “Replica Server Options and Variables”
Section 5.4.4, “The Binary Log”

slave_transaction_retries

Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.4.1.30, “Replication Retries and Timeouts”

slave_type_conversions

Section 18.6.3, “Known Issues in NDB Cluster Replication”
Section 17.1.4.3, “Replica Server Options and Variables”

slow_launch_time

Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”

slow_query_log

Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.7, “Server System Variables”
Section 5.4.5, “The Slow Query Log”
Section 1.3, “What Is New in MySQL 5.6”

slow_query_log_file

Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.7, “Server System Variables”
Section 5.4.5, “The Slow Query Log”
Section 1.3, “What Is New in MySQL 5.6”

socket

Section 5.1.7, “Server System Variables”

sort_buffer_size

Section 7.6.3, “How to Repair MyISAM Tables”
Section 8.2.1.13, “ORDER BY Optimization”
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”
Section 13.7.4.1, “SET Syntax for Variable Assignment”
Section 1.3, “What Is New in MySQL 5.6”

sql_auto_is_null

Section 12.4.2, “Comparison Functions and Operators”
Section 13.1.17, “CREATE TABLE Statement”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.4.1.35, “Replication and Variables”
Section 5.1.7, “Server System Variables”
Section 5.4.4, “The Binary Log”

sql_big_selects

Section 5.1.7, “Server System Variables”

sql_buffer_result

Section 5.1.7, “Server System Variables”

sql_log_bin

Section 17.1.4.4, “Binary Log Options and Variables”
Section 18.1.7.8, “Issues Exclusive to NDB Cluster”
Section 18.6.3, “Known Issues in NDB Cluster Replication”
Section 4.6.8, “`mysqlbinlog` — Utility for Processing Binary Log Files”
Section 18.1.7.1, “Noncompliance with SQL Syntax in NDB Cluster”
Section 6.2.2, “Privileges Provided by MySQL”
Section 13.4.1.3, “SET sql_log_bin Statement”
Section 5.1.8.1, “System Variable Privileges”
Section 17.4.3, “Upgrading a Replication Setup”

sql_log_off

MySQL Glossary
Section 6.2.2, “Privileges Provided by MySQL”

Section 5.4.1, “Selecting General Query Log and Slow Query Log Output Destinations”
Section 5.1.7, “Server System Variables”
Section 5.4.3, “The General Query Log”

SQL_MODE

Section 14.13.1, “Online DDL Operations”

sql_mode

Section 14.1.2, “Best Practices for InnoDB Tables”
Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
Section 13.1.11, “CREATE EVENT Statement”
Section 13.1.15, “CREATE PROCEDURE and CREATE FUNCTION Statements”
Section 13.1.19, “CREATE TRIGGER Statement”
Section 12.21.3, “Expression Handling”
Section 11.2.7, “Fractional Seconds in Time Values”
Section 1.6, “How to Report Bugs or Problems”
Section 13.2.6, “LOAD DATA Statement”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 1.7, “MySQL Standards Compliance”
Section 4.4.3, “`mysql_install_db` — Initialize MySQL Data Directory”
Section B.3.4.2, “Problems Using DATE Columns”
Section 17.4.1.35, “Replication and Variables”
Section 5.1.10, “Server SQL Modes”
Section 5.1.7, “Server System Variables”
Section 13.7.5.14, “SHOW CREATE VIEW Statement”
Section 13.6.7.5, “SIGNAL Statement”
Section 5.4.4, “The Binary Log”
Section 21.29, “The INFORMATION_SCHEMA VIEWS Table”
Section 5.1.2.2, “Using a Sample Default Server Configuration File”
Section 4.2.2.2, “Using Option Files”
Section 5.1.8, “Using System Variables”

sql_notes

Section B.2, “Error Information Interfaces”
Section 5.1.7, “Server System Variables”
Section 13.7.5.41, “SHOW WARNINGS Statement”
Section 13.6.7.7, “The MySQL Diagnostics Area”

sql_quote_show_create

Section 5.1.7, “Server System Variables”
Section 13.7.5.8, “SHOW CREATE DATABASE Statement”
Section 13.7.5.12, “SHOW CREATE TABLE Statement”

sql_safe_updates

Section 4.5.1.6, “mysql Client Tips”
Section 5.1.7, “Server System Variables”

sql_select_limit

Section 4.5.1.6, “mysql Client Tips”
Section 5.1.7, “Server System Variables”

sql_slave_skip_counter

Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.1.3.4, “Restrictions on Replication with GTIDs”
Section 13.7.5.35, “SHOW SLAVE STATUS Statement”

sql_warnings

Section 5.1.7, “Server System Variables”

ssl_ca

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.3.3.1, “Creating SSL Certificates and Keys Using openssl”
Section 5.1.7, “Server System Variables”
Section 17.3.8, “Setting Up Replication to Use Encrypted Connections”
Section 6.3.4, “SSL Library-Dependent Capabilities”

ssl_capath

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 5.1.7, “Server System Variables”
Section 6.3.4, “SSL Library-Dependent Capabilities”

ssl_cert

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.3.3.1, “Creating SSL Certificates and Keys Using openssl”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 17.3.8, “Setting Up Replication to Use Encrypted Connections”

ssl_cipher

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”
Section 5.1.7, “Server System Variables”
Section 6.3.4, “SSL Library-Dependent Capabilities”

ssl_crl

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 5.1.7, “Server System Variables”
Section 6.3.4, “SSL Library-Dependent Capabilities”

ssl_crlpath

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 5.1.7, “Server System Variables”
Section 6.3.4, “SSL Library-Dependent Capabilities”

ssl_key

Section 4.2.3, “Command Options for Connecting to the Server”
Section 6.3.1, “Configuring MySQL to Use Encrypted Connections”
Section 6.3.3.1, “Creating SSL Certificates and Keys Using openssl”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 17.3.8, “Setting Up Replication to Use Encrypted Connections”

storage_engine

Section 13.1.14, “CREATE LOGFILE GROUP Statement”
Section 17.4.1.35, “Replication and Variables”
Section 5.1.7, “Server System Variables”

Section 17.3.3, “Using Replication with Different Source and Replica Storage Engines”

stored_program_cache

Section 8.10.4, “Caching of Prepared Statements and Stored Programs”
Section 5.1.7, “Server System Variables”

sync_binlog

Section 17.1.4.4, “Binary Log Options and Variables”
Section 17.3.2, “Handling an Unexpected Halt of a Replica Server”
Section 14.2, “InnoDB and the ACID Model”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 17.4.1.26, “Replication and Source or Replica Shutdowns”
Section 5.4.4, “The Binary Log”
Section 4.6.8.3, “Using mysqlbinlog to Back Up Binary Log Files”

sync_frm

Section 5.1.7, “Server System Variables”

sync_master_info

Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
Section 17.1.4.3, “Replica Server Options and Variables”

sync_relay_log

Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
Section 17.3.2, “Handling an Unexpected Halt of a Replica Server”
Section 17.1.4.3, “Replica Server Options and Variables”

sync_relay_log_info

Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 17.4.1.26, “Replication and Source or Replica Shutdowns”

system_time_zone

Section 5.1.13, “MySQL Server Time Zone Support”
Section 17.4.1.31, “Replication and Time Zones”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”

T

[index top]

table_definition_cache

Section 5.1.2.1, “Changes to Server Defaults”
Section 8.12.4.1, “How MySQL Uses Memory”
Section 5.1.7, “Server System Variables”
Section 1.3, “What Is New in MySQL 5.6”

table_open_cache

Section 2.11.3, “Changes in MySQL 5.6”
Section 5.1.2.1, “Changes to Server Defaults”
Section 5.1.1, “Configuring the Server”

Section B.3.2.17, “File Not Found and Similar Errors”
Section 8.14.3, “General Thread States”
Section 8.4.3.1, “How MySQL Opens and Closes Tables”
Section 8.12.4.1, “How MySQL Uses Memory”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”
Section 1.3, “What Is New in MySQL 5.6”

table_open_cache_instances

Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”

thread_cache_size

Section 5.1.2.1, “Changes to Server Defaults”
Section 5.1.11.1, “Connection Interfaces”
Section 5.8.1.4, “Debugging mysqld under gdb”
Section 5.1.9, “Server Status Variables”
Section 5.1.7, “Server System Variables”

thread_concurrency

Section 5.1.7, “Server System Variables”

thread_handling

Section 5.1.7, “Server System Variables”
Section 5.5.3.1, “Thread Pool Elements”

thread_pool_algorithm

Section 5.1.7, “Server System Variables”
Section 21.32.1, “The INFORMATION_SCHEMA TP_THREAD_GROUP_STATE Table”
Section 5.5.3.3, “Thread Pool Operation”

thread_pool_high_priority_connection

Section 5.1.7, “Server System Variables”
Section 5.5.3.3, “Thread Pool Operation”

thread_pool_max_unused_threads

Section 5.1.7, “Server System Variables”
Section 5.5.3.3, “Thread Pool Operation”

thread_pool_prio_kickup_timer

Section 5.1.7, “Server System Variables”
Section 21.32.1, “The INFORMATION_SCHEMA TP_THREAD_GROUP_STATE Table”
Section 21.32.2, “The INFORMATION_SCHEMA TP_THREAD_GROUP_STATS Table”
Section 5.5.3.3, “Thread Pool Operation”
Section 5.5.3.4, “Thread Pool Tuning”

thread_pool_size

Section 5.1.7, “Server System Variables”
Section 5.5.3.3, “Thread Pool Operation”
Section 5.5.3.4, “Thread Pool Tuning”

thread_pool_stall_limit

Section 5.1.7, “Server System Variables”
Section 21.32.1, “The INFORMATION_SCHEMA TP_THREAD_GROUP_STATE Table”

Section 21.32.2, “The INFORMATION_SCHEMA TP_THREAD_GROUP_STATS Table”
Section 5.5.3.3, “Thread Pool Operation”
Section 5.5.3.4, “Thread Pool Tuning”

thread_stack

Section 5.1.11.1, “Connection Interfaces”
Section 8.12.4.1, “How MySQL Uses Memory”
Section 5.1.7, “Server System Variables”
Section 20.2.1, “Stored Routine Syntax”

time_format

Section 5.1.7, “Server System Variables”
Section 1.3, “What Is New in MySQL 5.6”

time_zone

Section 13.1.11, “CREATE EVENT Statement”
Section 12.7, “Date and Time Functions”
Section 20.4.4, “Event Metadata”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 5.1.13, “MySQL Server Time Zone Support”
Section 17.4.1.31, “Replication and Time Zones”
Section 17.4.1.35, “Replication and Variables”
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 11.2.2, “The DATE, DATETIME, and TIMESTAMP Types”

timed_mutexes

Section 5.1.7, “Server System Variables”
Section 1.3, “What Is New in MySQL 5.6”

timestamp

Section 15.8.3, “FEDERATED Storage Engine Notes and Tips”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.4.1.35, “Replication and Variables”
Section 5.1.7, “Server System Variables”

tls_version

Section 6.3.2, “Encrypted Connection TLS Protocols and Ciphers”

tmp_table_size

Section 8.12.4.1, “How MySQL Uses Memory”
Section 8.4.4, “Internal Temporary Table Use in MySQL”
Section 13.6.6.5, “Restrictions on Server-Side Cursors”
Section 5.1.7, “Server System Variables”

tmpdir

Section 17.3.1.2, “Backing Up Raw Data from a Replica”
Section B.3.2.12, “Can't create/write to file”
Section 7.2, “Database Backup Methods”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 2.9.7, “MySQL Source-Configuration Options”
Section 14.13.5, “Online DDL Failure Conditions”
Section 14.13.3, “Online DDL Space Requirements”
Section 8.2.1.13, “ORDER BY Optimization”
Section 17.1.4.3, “Replica Server Options and Variables”
Section 5.1.7, “Server System Variables”

transaction_alloc_block_size
Section 5.1.7, “Server System Variables”

transaction_allow_batching
NDB Cluster System Variables

transaction_prealloc_size
Section 5.1.7, “Server System Variables”

tx_isolation
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 13.3.6, “SET TRANSACTION Statement”

tx_read_only
Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 13.3.6, “SET TRANSACTION Statement”

U

[index top]

unique_checks
Section 14.6.1.5, “Converting Tables from MyISAM to InnoDB”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 17.4.1.35, “Replication and Variables”
Section 5.1.7, “Server System Variables”
Section 5.4.4, “The Binary Log”

updatable_views_with_limit
Section 5.1.7, “Server System Variables”
Section 20.5.3, “Updatable and Insertable Views”

V

[index top]

validate_password_dictionary_file
Section 6.4.3.2, “Password Validation Plugin Options and Variables”
Section 6.4.3, “The Password Validation Plugin”

validate_password_length
Section 12.14, “Encryption and Compression Functions”
Section 6.4.3.2, “Password Validation Plugin Options and Variables”
Section 6.4.3, “The Password Validation Plugin”

validate_password_mixed_case_count
Section 6.4.3.2, “Password Validation Plugin Options and Variables”
Section 6.4.3, “The Password Validation Plugin”

validate_password_number_count
Section 6.4.3.2, “Password Validation Plugin Options and Variables”
Section 6.4.3, “The Password Validation Plugin”

validate_password_policy
Section 6.4.3.2, “Password Validation Plugin Options and Variables”
Section 6.4.3, “The Password Validation Plugin”

validate_password_special_char_count
Section 6.4.3.2, “Password Validation Plugin Options and Variables”
Section 6.4.3, “The Password Validation Plugin”

validate_user_plugins
Section 5.1.7, “Server System Variables”

version
Section 6.4.4.3, “Audit Log File Formats”
Section 12.16, “Information Functions”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 5.1.7, “Server System Variables”

version_comment
Section 5.1.7, “Server System Variables”
Section 13.7.5.40, “SHOW VARIABLES Statement”

version_compile_machine
Section 5.1.7, “Server System Variables”

version_compile_os
Section 5.1.7, “Server System Variables”

W

[index top]

wait_timeout
Section B.3.2.10, “Communication Errors and Aborted Connections”
Section B.3.2.8, “MySQL server has gone away”
Section 5.1.7, “Server System Variables”

warning_count
Section B.2, “Error Information Interfaces”
Section 5.1.7, “Server System Variables”
Section 13.7.5.18, “SHOW ERRORS Statement”
Section 13.7.5.41, “SHOW WARNINGS Statement”
Section 13.6.7.5, “SIGNAL Statement”
Section 13.6.7.7, “The MySQL Diagnostics Area”

Transaction Isolation Level Index

R | S

R

[index top]

READ COMMITTED
Section 14.7.2.3, “Consistent Nonlocking Reads”

Section 18.1.6.1, “Differences Between the NDB and InnoDB Storage Engines”
Section 14.7.5.3, “How to Minimize and Handle Deadlocks”
Section 14.7.1, “InnoDB Locking”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 18.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section A.1, “MySQL 5.6 FAQ: General”
Section A.10, “MySQL 5.6 FAQ: NDB Cluster”
Section 18.1.6.3, “NDB and InnoDB Feature Usage Summary”
Section 8.5.2, “Optimizing InnoDB Transaction Management”
Section 13.3.6, “SET TRANSACTION Statement”
Section 5.4.4.2, “Setting The Binary Log Format”
Section 14.7.2.1, “Transaction Isolation Levels”
Section 13.1.33, “TRUNCATE TABLE Statement”

READ UNCOMMITTED

Section 14.7.2.3, “Consistent Nonlocking Reads”
Including Delete-marked Records in Persistent Statistics Calculations
Section 14.20.2, “InnoDB memcached Architecture”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 18.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 14.20.5.6, “Performing DML and DDL Statements on the Underlying InnoDB Table”
Section 13.3.6, “SET TRANSACTION Statement”
Section 5.4.4.2, “Setting The Binary Log Format”
Section 14.7.2.1, “Transaction Isolation Levels”
Section 13.1.33, “TRUNCATE TABLE Statement”

READ-COMMITTED

Section 5.1.6, “Server Command Options”
Section 13.3.6, “SET TRANSACTION Statement”

READ-UNCOMMITTED

Section 5.1.6, “Server Command Options”
Section 13.3.6, “SET TRANSACTION Statement”

REPEATABLE READ

Section 14.7.2.3, “Consistent Nonlocking Reads”
Section 14.20.5.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”
Section 14.7.1, “InnoDB Locking”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 18.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 4.5.4, “`mysqldump` — A Database Backup Program”
Section 8.5.2, “Optimizing InnoDB Transaction Management”
Section 13.3.6, “SET TRANSACTION Statement”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”
Section 14.7.2.1, “Transaction Isolation Levels”
Section 13.3.7, “XA Transactions”

REPEATABLE-READ

Section 5.1.6, “Server Command Options”
Section 5.1.7, “Server System Variables”
Section 13.3.6, “SET TRANSACTION Statement”

S

[index top]

SERIALIZABLE

Section 8.10.3.1, “How the Query Cache Operates”
Section 14.14, “InnoDB Startup Options and System Variables”
Section 18.1.7.3, “Limits Relating to Transaction Handling in NDB Cluster”
Section 14.7.3, “Locks Set by Different SQL Statements in InnoDB”
Section 5.4.4.3, “Mixed Binary Logging Format”
Section 5.1.6, “Server Command Options”
Section 13.3.6, “SET TRANSACTION Statement”
Section 13.3.1, “START TRANSACTION, COMMIT, and ROLLBACK Statements”
Section 14.7.2.1, “Transaction Isolation Levels”
Section 13.3.7, “XA Transactions”

MySQL Glossary

These terms are commonly used in information about the MySQL database server. This glossary originated as a reference for terminology about the InnoDB storage engine, and the majority of definitions are InnoDB-related.

A

.ARM file
: Metadata for `ARCHIVE` tables. Contrast with **.ARZ file**. Files with this extension are always included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.
See Also .ARZ file, MySQL Enterprise Backup, mysqlbackup command.

.ARZ file
: Data for ARCHIVE tables. Contrast with **.ARM file**. Files with this extension are always included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.
See Also .ARM file, MySQL Enterprise Backup, mysqlbackup command.

ACID
: An acronym standing for atomicity, consistency, isolation, and durability. These properties are all desirable in a database system, and are all closely tied to the notion of a **transaction**. The transactional features of `InnoDB` adhere to the ACID principles.

  Transactions are **atomic** units of work that can be **committed** or **rolled back**. When a transaction makes multiple changes to the database, either all the changes succeed when the transaction is committed, or all the changes are undone when the transaction is rolled back.

  The database remains in a consistent state at all times — after each commit or rollback, and while transactions are in progress. If related data is being updated across multiple tables, queries see either all old values or all new values, not a mix of old and new values.

  Transactions are protected (isolated) from each other while they are in progress; they cannot interfere with each other or see each other's uncommitted data. This isolation is achieved through the **locking** mechanism. Experienced users can adjust the **isolation level**, trading off less protection in favor of increased performance and **concurrency**, when they can be sure that the transactions really do not interfere with each other.

  The results of transactions are durable: once a commit operation succeeds, the changes made by that transaction are safe from power failures, system crashes, race conditions, or other potential dangers that many non-database applications are vulnerable to. Durability typically involves writing to disk storage, with a certain amount of redundancy to protect against power failures or software crashes during write operations. (In `InnoDB`, the **doublewrite buffer** assists with durability.)
  See Also atomic, commit, concurrency, doublewrite buffer, isolation level, locking, rollback, transaction.

adaptive flushing
: An algorithm for **InnoDB** tables that smooths out the I/O overhead introduced by **checkpoints**. Instead of **flushing** all modified **pages** from the **buffer pool** to the **data files** at once, MySQL periodically flushes small sets of modified pages. The adaptive flushing algorithm extends this process by estimating the optimal rate to perform these periodic flushes, based on the rate of flushing and how fast **redo** information is generated.
See Also buffer pool, checkpoint, data files, flush, InnoDB, page, redo log.

adaptive hash index
: An optimization for `InnoDB` tables that can speed up lookups using `=` and `IN` operators, by constructing a **hash index** in memory. MySQL monitors index searches for `InnoDB` tables, and if queries could benefit from a hash index, it builds one automatically for index **pages** that are frequently accessed.
In a sense, the adaptive hash index configures MySQL at runtime to take advantage of ample main memory, coming closer to the architecture of main-memory databases. This feature is controlled by the `innodb_adaptive_hash_index` configuration option. Because this feature benefits some workloads and not others, and the memory used for the hash index is reserved in the **buffer pool**, typically you should benchmark with this feature both enabled and disabled.

The hash index is always built based on an existing **B-tree** index on the table. MySQL can build a hash index on a prefix of any length of the key defined for the B-tree, depending on the pattern of searches against the index. A hash index can be partial; the whole B-tree index does not need to be cached in the buffer pool.

In MySQL 5.6 and higher, another way to take advantage of fast single-value lookups with `InnoDB` tables is to use the `InnoDB` **memcached** plugin. See Section 14.20, "InnoDB memcached Plugin" for details.
See Also B-tree, buffer pool, hash index, memcached, page, secondary index.

ADO.NET
An object-relational mapping (ORM) framework for applications built using .NET technologies such as **ASP.NET**. Such applications can interface with MySQL through the **Connector/NET** component.
See Also .NET, ASP.net, Connector/NET, Mono, Visual Studio.

AIO
Acronym for **asynchronous I/O**. You might see this acronym in `InnoDB` messages or keywords.
See Also asynchronous I/O.

ANSI
In **ODBC**, an alternative method of supporting character sets and other internationalization aspects. Contrast with **Unicode**. **Connector/ODBC** 3.51 is an ANSI driver, while Connector/ODBC 5.1 is a Unicode driver.
See Also Connector/ODBC, ODBC, Unicode.

Antelope
The code name for the original `InnoDB` **file format**. It supports the **REDUNDANT** and **COMPACT** row formats, but not the newer **DYNAMIC** and **COMPRESSED** row formats available in the **Barracuda** file format.
See Also Barracuda, compact row format, compressed row format, dynamic row format, file format, innodb_file_format, redundant row format.

API
APIs provide low-level access to the MySQL protocol and MySQL resources from **client** programs. Contrast with the higher-level access provided by a **Connector**.
See Also C API, client, connector, native C API, Perl API, PHP API, Python API, Ruby API.

application programming interface (API)
A set of functions or procedures. An API provides a stable set of names and types for functions, procedures, parameters, and return values.

apply
When a backup produced by the **MySQL Enterprise Backup** product does not include the most recent changes that occurred while the backup was underway, the process of updating the backup files to include those changes is known as the **apply** step. It is specified by the `apply-log` option of the `mysqlbackup` command.

Before the changes are applied, we refer to the files as a **raw backup**. After the changes are applied, we refer to the files as a **prepared backup**. The changes are recorded in the **ibbackup_logfile** file; once the apply step is finished, this file is no longer necessary.
See Also hot backup, ibbackup_logfile, MySQL Enterprise Backup, prepared backup, raw backup.

ASP.net
A framework for developing web-based applications using **.NET** technologies and languages. Such applications can interface with MySQL through the **Connector/NET** component.

Another technology for writing server-side web pages with MySQL is **PHP**.
See Also .NET, ADO.NET, Connector/NET, Mono, PHP, Visual Studio.

assembly
A library of compiled code in a **.NET** system, accessed through **Connector/NET**. Stored in the **GAC** to allow versioning without naming conflicts.
See Also .NET, GAC.

asynchronous I/O

A type of I/O operation that allows other processing to proceed before the I/O is completed. Also known as **nonblocking I/O** and abbreviated as **AIO**. `InnoDB` uses this type of I/O for certain operations that can run in parallel without affecting the reliability of the database, such as reading pages into the **buffer pool** that have not actually been requested, but might be needed soon.

Historically, `InnoDB` used asynchronous I/O on Windows systems only. Starting with the InnoDB Plugin 1.1 and MySQL 5.5, `InnoDB` uses asynchronous I/O on Linux systems. This change introduces a dependency on `libaio`. Asynchronous I/O on Linux systems is configured using the `innodb_use_native_aio` option, which is enabled by default. On other Unix-like systems, InnoDB uses synchronous I/O only.
See Also buffer pool, nonblocking I/O.

atomic

In the SQL context, **transactions** are units of work that either succeed entirely (when **committed**) or have no effect at all (when **rolled back**). The indivisible ("atomic") property of transactions is the “A” in the acronym **ACID**.
See Also ACID, commit, rollback, transaction.

atomic DDL

An atomic *DDL* statement is one that combines the *data dictionary* updates, *storage engine* operations, and *binary log* writes associated with a DDL operation into a single, atomic transaction. The transaction is either fully committed or rolled back, even if the server halts during the operation. Atomic DDL support was added in MySQL 8.0. For more information, see Atomic Data Definition Statement Support.
See Also binary log, data dictionary, DDL, storage engine.

atomic instruction

Special instructions provided by the CPU, to ensure that critical low-level operations cannot be interrupted.

auto-increment

A property of a table column (specified by the `AUTO_INCREMENT` keyword) that automatically adds an ascending sequence of values in the column.

It saves work for the developer, not to have to produce new unique values when inserting new rows. It provides useful information for the query optimizer, because the column is known to be not null and with unique values. The values from such a column can be used as lookup keys in various contexts, and because they are auto-generated there is no reason to ever change them; for this reason, primary key columns are often specified as auto-incrementing.

Auto-increment columns can be problematic with statement-based replication, because replaying the statements on a replica might not produce the same set of column values as on the source, due to timing issues. When you have an auto-incrementing primary key, you can use statement-based replication only with the setting `innodb_autoinc_lock_mode=1`. If you have `innodb_autoinc_lock_mode=2`, which allows higher concurrency for insert operations, use **row-based replication** rather than **statement-based replication**. The setting `innodb_autoinc_lock_mode=0` should not be used except for compatibility purposes.

Consecutive lock mode (`innodb_autoinc_lock_mode=1`) is the default setting prior to MySQL 8.0.3. As of MySQL 8.0.3, interleaved lock mode (`innodb_autoinc_lock_mode=2`) is the default, which reflects the change from statement-based to row-based replication as the default replication type.
See Also auto-increment locking, innodb_autoinc_lock_mode, primary key, row-based replication, statement-based replication.

auto-increment locking

The convenience of an **auto-increment** primary key involves some tradeoff with concurrency. In the simplest case, if one transaction is inserting values into the table, any other transactions must wait to do their own inserts into that table, so that rows inserted by the first transaction receive consecutive primary key values. `InnoDB` includes optimizations and the `innodb_autoinc_lock_mode` option so that you can configure and optimal balance between predictable sequences of auto-increment values and maximum **concurrency** for insert operations.
See Also auto-increment, concurrency, innodb_autoinc_lock_mode.

autocommit
: A setting that causes a **commit** operation after each **SQL** statement. This mode is not recommended for working with `InnoDB` tables with **transactions** that span several statements. It can help performance for **read-only transactions** on `InnoDB` tables, where it minimizes overhead from **locking** and generation of **undo** data, especially in MySQL 5.6.4 and up. It is also appropriate for working with `MyISAM` tables, where transactions are not applicable.
See Also commit, locking, read-only transaction, SQL, transaction, undo.

availability
: The ability to cope with, and if necessary recover from, failures on the host, including failures of MySQL, the operating system, or the hardware and maintenance activity that may otherwise cause downtime. Often paired with **scalability** as critical aspects of a large-scale deployment.
See Also scalability.

B

B-tree
: A tree data structure that is popular for use in database indexes. The structure is kept sorted at all times, enabling fast lookup for exact matches (equals operator) and ranges (for example, greater than, less than, and `BETWEEN` operators). This type of index is available for most storage engines, such as `InnoDB` and `MyISAM`.

  Because B-tree nodes can have many children, a B-tree is not the same as a binary tree, which is limited to 2 children per node.

  Contrast with **hash index**, which is only available in the `MEMORY` storage engine. The `MEMORY` storage engine can also use B-tree indexes, and you should choose B-tree indexes for `MEMORY` tables if some queries use range operators.

  The use of the term B-tree is intended as a reference to the general class of index design. B-tree structures used by MySQL storage engines may be regarded as variants due to sophistications not present in a classic B-tree design. For related information, refer to the `InnoDB` Page Structure Fil Header section of the MySQL Internals Manual.
  See Also hash index.

backticks
: Identifiers within MySQL SQL statements must be quoted using the backtick character (`` ` ``) if they contain special characters or reserved words. For example, to refer to a table named `FOO#BAR` or a column named `SELECT`, you would specify the identifiers as `` `FOO#BAR` `` and `` `SELECT` ``. Since the backticks provide an extra level of safety, they are used extensively in program-generated SQL statements, where the identifier names might not be known in advance.

  Many other database systems use double quotation marks (`"`) around such special names. For portability, you can enable `ANSI_QUOTES` mode in MySQL and use double quotation marks instead of backticks to qualify identifier names.
  See Also SQL.

backup
: The process of copying some or all table data and metadata from a MySQL instance, for safekeeping. Can also refer to the set of copied files. This is a crucial task for DBAs. The reverse of this process is the **restore** operation.

  With MySQL, **physical backups** are performed by the **MySQL Enterprise Backup** product, and **logical backups** are performed by the `mysqldump` command. These techniques have different characteristics in terms of size and representation of the backup data, and speed (especially speed of the restore operation).

  Backups are further classified as **hot**, **warm**, or **cold** depending on how much they interfere with normal database operation. (Hot backups have the least interference, cold backups the most.)
  See Also cold backup, hot backup, logical backup, MySQL Enterprise Backup, mysqldump, physical backup, warm backup.

Barracuda

The code name for an `InnoDB` **file format** that supports the **COMPRESSED** row format that enables InnoDB table compression, and the **DYNAMIC** row format that improves the storage layout for long variable-length columns.

The **MySQL Enterprise Backup** product version 3.5 and above supports backing up tablespaces that use the Barracuda file format.
See Also Antelope, compact row format, compressed row format, dynamic row format, file format, file-per-table, general tablespace, innodb_file_format, MySQL Enterprise Backup, row format, system tablespace.

beta

An early stage in the life of a software product, when it is available only for evaluation, typically without a definite release number or a number less than 1. `InnoDB` does not use the beta designation, preferring an **early adopter** phase that can extend over several point releases, leading to a **GA** release.
See Also early adopter, GA.

binary log

A file containing a record of all statements or row changes that attempt to change table data. The contents of the binary log can be replayed to bring replicas up to date in a **replication** scenario, or to bring a database up to date after restoring table data from a backup. The binary logging feature can be turned on and off, although Oracle recommends always enabling it if you use replication or perform backups.

You can examine the contents of the binary log, or replay it during replication or recovery, by using the `mysqlbinlog` command. For full information about the binary log, see Section 5.4.4, “The Binary Log”. For MySQL configuration options related to the binary log, see Section 17.1.4.4, “Binary Log Options and Variables”.

For the **MySQL Enterprise Backup** product, the file name of the binary log and the current position within the file are important details. To record this information for the source when taking a backup in a replication context, you can specify the `--slave-info` option.

Prior to MySQL 5.0, a similar capability was available, known as the update log. In MySQL 5.0 and higher, the binary log replaces the update log.
See Also binlog, MySQL Enterprise Backup, replication.

binlog

An informal name for the **binary log** file. For example, you might see this abbreviation used in e-mail messages or forum discussions.
See Also binary log.

blind query expansion

A special mode of **full-text search** enabled by the `WITH QUERY EXPANSION` clause. It performs the search twice, where the search phrase for the second search is the original search phrase concatenated with the few most highly relevant documents from the first search. This technique is mainly applicable for short search phrases, perhaps only a single word. It can uncover relevant matches where the precise search term does not occur in the document.
See Also full-text search.

BLOB

An SQL data type (`TINYBLOB`, `BLOB`, `MEDIUMBLOB`, and `LONGBLOB`) for objects containing any kind of binary data, of arbitrary size. Used for storing documents, images, sound files, and other kinds of information that cannot easily be decomposed to rows and columns within a MySQL table. The techniques for handling BLOBs within a MySQL application vary with each **Connector** and **API**. MySQL `Connector/ODBC` defines `BLOB` values as `LONGVARBINARY`. For large, free-form collections of character data, the industry term is **CLOB**, represented by the MySQL `TEXT` data types.
See Also API, CLOB, connector, Connector/ODBC.

bottleneck

A portion of a system that is constrained in size or capacity, that has the effect of limiting overall throughput. For example, a memory area might be smaller than necessary; access to a single required resource might

prevent multiple CPU cores from running simultaneously; or waiting for disk I/O to complete might prevent the CPU from running at full capacity. Removing bottlenecks tends to improve **concurrency**. For example, the ability to have multiple `InnoDB` **buffer pool** instances reduces contention when multiple sessions read from and write to the buffer pool simultaneously.
See Also buffer pool, concurrency.

bounce
A **shutdown** operation immediately followed by a restart. Ideally with a relatively short **warmup** period so that performance and throughput quickly return to a high level.
See Also shutdown.

buddy allocator
A mechanism for managing different-sized **pages** in the InnoDB **buffer pool**.
See Also buffer pool, page, page size.

buffer
A memory or disk area used for temporary storage. Data is buffered in memory so that it can be written to disk efficiently, with a few large I/O operations rather than many small ones. Data is buffered on disk for greater reliability, so that it can be recovered even when a **crash** or other failure occurs at the worst possible time. The main types of buffers used by InnoDB are the **buffer pool**, the **doublewrite buffer**, and the **change buffer**.
See Also buffer pool, change buffer, crash, doublewrite buffer.

buffer pool
The memory area that holds cached `InnoDB` data for both tables and indexes. For efficiency of high-volume read operations, the buffer pool is divided into **pages** that can potentially hold multiple rows. For efficiency of cache management, the buffer pool is implemented as a linked list of pages; data that is rarely used is aged out of the cache, using a variation of the **LRU** algorithm. On systems with large memory, you can improve concurrency by dividing the buffer pool into multiple **buffer pool instances**.

Several `InnoDB` status variables, `INFORMATION_SCHEMA` tables, and `performance_schema` tables help to monitor the internal workings of the buffer pool. Starting in MySQL 5.6, you can avoid a lengthy warmup period after restarting the server, particularly for instances with large buffer pools, by saving the buffer pool state at server shutdown and restoring the buffer pool to the same state at server startup. See Section 14.8.3.5, “Saving and Restoring the Buffer Pool State”.
See Also buffer pool instance, LRU, page, warm up.

buffer pool instance
Any of the multiple regions into which the **buffer pool** can be divided, controlled by the `innodb_buffer_pool_instances` configuration option. The total memory size specified by `innodb_buffer_pool_size` is divided among all buffer pool instances. Typically, having multiple buffer pool instances is appropriate for systems that allocate multiple gigabytes to the `InnoDB` buffer pool, with each instance being one gigabyte or larger. On systems loading or looking up large amounts of data in the buffer pool from many concurrent sessions, having multiple buffer pool instances reduces contention for exclusive access to data structures that manage the buffer pool.
See Also buffer pool.

built-in
The built-in `InnoDB` storage engine within MySQL is the original form of distribution for the storage engine. Contrast with the **InnoDB Plugin**. Starting with MySQL 5.5, the InnoDB Plugin is merged back into the MySQL code base as the built-in `InnoDB` storage engine (known as InnoDB 1.1).

This distinction is important mainly in MySQL 5.1, where a feature or bug fix might apply to the InnoDB Plugin but not the built-in `InnoDB`, or vice versa.
See Also InnoDB.

business rules
The relationships and sequences of actions that form the basis of business software, used to run a commercial company. Sometimes these rules are dictated by law, other times by company policy. Careful

planning ensures that the relationships encoded and enforced by the database, and the actions performed through application logic, accurately reflect the real policies of the company and can handle real-life situations.

For example, an employee leaving a company might trigger a sequence of actions from the human resources department. The human resources database might also need the flexibility to represent data about a person who has been hired, but not yet started work. Closing an account at an online service might result in data being removed from a database, or the data might be moved or flagged so that it could be recovered if the account is re-opened. A company might establish policies regarding salary maximums, minimums, and adjustments, in addition to basic sanity checks such as the salary not being a negative number. A retail database might not allow a purchase with the same serial number to be returned more than once, or might not allow credit card purchases above a certain value, while a database used to detect fraud might allow these kinds of things.
See Also relational.

C

.cfg file
A metadata file used with the `InnoDB` **transportable tablespace** feature. It is produced by the command `FLUSH TABLES ... FOR EXPORT`, puts one or more tables in a consistent state that can be copied to another server. The `.cfg` file is copied along with the corresponding **.ibd file**, and used to adjust the internal values of the `.ibd` file, such as the **space ID**, during the `ALTER TABLE ... IMPORT TABLESPACE` step.
See Also .ibd file, space ID, transportable tablespace.

C
A programming language that combines portability with performance and access to low-level hardware features, making it a popular choice for writing operating systems, drivers, and other kinds of system software. Many complex applications, languages, and reusable modules feature pieces written in C, tied together with high-level components written in other languages. Its core syntax is familiar to **C++**, **Java**, and **C#** developers.
See Also C API, C++, C#, Java.

C API
The C **API** code is distributed with MySQL. It is included in the **libmysqlclient** library and enables **C** programs to access a database.
See Also API, C, libmysqlclient.

C#
A programming language combining strong typing and object-oriented features, running within the Microsoft **.NET** framework or its open-source counterpart **Mono**. Often used for creating applications with the **ASP.net** framework. Its syntax is familiar to **C**, **C++** and **Java** developers.
See Also .NET, ASP.net, C, Connector/NET, C++, Java, Mono.

C++
A programming language with core syntax familiar to **C** developers. Provides access to low-level operations for performance, combined with higher-level data types, object-oriented features, and garbage collection. To write C++ applications for MySQL, you use the **Connector/C++** component.
See Also C, Connector/C++.

cache
The general term for any memory area that stores copies of data for frequent or high-speed retrieval. In `InnoDB`, the primary kind of cache structure is the **buffer pool**.
See Also buffer, buffer pool.

cardinality
The number of different values in a table **column**. When queries refer to columns that have an associated **index**, the cardinality of each column influences which access method is most efficient. For example, for a column with a **unique constraint**, the number of different values is equal to the number of rows in the table. If a table has a million rows but only 10 different values for a particular column, each value occurs (on average) 100,000 times. A query such as `SELECT c1 FROM t1 WHERE c1 = 50;` thus might return 1 row or a huge

number of rows, and the database server might process the query differently depending on the cardinality of `c1`.

If the values in a column have a very uneven distribution, the cardinality might not be a good way to determine the best query plan. For example, `SELECT c1 FROM t1 WHERE c1 = x;` might return 1 row when `x=50` and a million rows when `x=30`. In such a case, you might need to use **index hints** to pass along advice about which lookup method is more efficient for a particular query.

Cardinality can also apply to the number of distinct values present in multiple columns, as in a **composite index**.
See Also column, composite index, index, index hint, persistent statistics, random dive, selectivity, unique constraint.

change buffer
A special data structure that records changes to **pages** in **secondary indexes**. These values could result from SQL `INSERT`, `UPDATE`, or `DELETE` statements (**DML**). The set of features involving the change buffer is known collectively as **change buffering**, consisting of **insert buffering**, **delete buffering**, and **purge buffering**.

Changes are only recorded in the change buffer when the relevant page from the secondary index is not in the **buffer pool**. When the relevant index page is brought into the buffer pool while associated changes are still in the change buffer, the changes for that page are applied in the buffer pool (**merged**) using the data from the change buffer. Periodically, the **purge** operation that runs during times when the system is mostly idle, or during a slow shutdown, writes the new index pages to disk. The purge operation can write the disk blocks for a series of index values more efficiently than if each value were written to disk immediately.

Physically, the change buffer is part of the **system tablespace**, so that the index changes remain buffered across database restarts. The changes are only applied (**merged**) when the pages are brought into the buffer pool due to some other read operation.

The kinds and amount of data stored in the change buffer are governed by the `innodb_change_buffering` and `innodb_change_buffer_max_size` configuration options. To see information about the current data in the change buffer, issue the `SHOW ENGINE INNODB STATUS` command.

Formerly known as the **insert buffer**.
See Also buffer pool, change buffering, delete buffering, DML, insert buffer, insert buffering, merge, page, purge, purge buffering, secondary index, system tablespace.

change buffering
The general term for the features involving the **change buffer**, consisting of **insert buffering**, **delete buffering**, and **purge buffering**. Index changes resulting from SQL statements, which could normally involve random I/O operations, are held back and performed periodically by a background **thread**. This sequence of operations can write the disk blocks for a series of index values more efficiently than if each value were written to disk immediately. Controlled by the `innodb_change_buffering` and `innodb_change_buffer_max_size` configuration options.
See Also change buffer, delete buffering, insert buffering, purge buffering.

checkpoint
As changes are made to data pages that are cached in the **buffer pool**, those changes are written to the **data files** sometime later, a process known as **flushing**. The checkpoint is a record of the latest changes (represented by an **LSN** value) that have been successfully written to the data files.
See Also buffer pool, data files, flush, fuzzy checkpointing, LSN.

checksum
In `InnoDB`, a validation mechanism to detect corruption when a **page** in a **tablespace** is read from disk into the `InnoDB` **buffer pool**. This feature is controlled by the `innodb_checksums` configuration option in MySQL 5.5. `innodb_checksums` is deprecated in MySQL 5.6.3, replaced by `innodb_checksum_algorithm`.

The `innochecksum` command helps diagnose corruption problems by testing the checksum values for a specified **tablespace** file while the MySQL server is shut down.

MySQL also uses checksums for replication purposes. For details, see the configuration options `binlog_checksum`, `master_verify_checksum`, and `slave_sql_verify_checksum`.
See Also buffer pool, page, tablespace.

child table
In a **foreign key** relationship, a child table is one whose rows refer (or point) to rows in another table with an identical value for a specific column. This is the table that contains the `FOREIGN KEY ... REFERENCES` clause and optionally `ON UPDATE` and `ON DELETE` clauses. The corresponding row in the **parent table** must exist before the row can be created in the child table. The values in the child table can prevent delete or update operations on the parent table, or can cause automatic deletion or updates in the child table, based on the `ON CASCADE` option used when creating the foreign key.
See Also foreign key, parent table.

clean page
A **page** in the `InnoDB` **buffer pool** where all changes made in memory have also been written (**flushed**) to the data files. The opposite of a **dirty page**.
See Also buffer pool, data files, dirty page, flush, page.

clean shutdown
A **shutdown** that completes without errors and applies all changes to `InnoDB` tables before finishing, as opposed to a **crash** or a **fast shutdown**. Synonym for **slow shutdown**.
See Also crash, fast shutdown, shutdown, slow shutdown.

client
A program that runs outside the database server, communicating with the database by sending requests through a **Connector**, or an **API** made available through **client libraries**. It can run on the same physical machine as the database server, or on a remote machine connected over a network. It can be a special-purpose database application, or a general-purpose program like the `mysql` command-line processor.
See Also API, client libraries, connector, mysql, server.

client libraries
Files containing collections of functions for working with databases. By compiling your program with these libraries, or installing them on the same system as your application, you can run a database application (known as a **client**) on a machine that does not have the MySQL server installed; the application accesses the database over a network. With MySQL, you can use the **libmysqlclient** library from the MySQL server itself.
See Also client, libmysqlclient.

client-side prepared statement
A type of **prepared statement** where the caching and reuse are managed locally, emulating the functionality of **server-side prepared statements**. Historically, used by some **Connector/J**, **Connector/ODBC**, and **Connector/PHP** developers to work around issues with server-side stored procedures. With modern MySQL server versions, server-side prepared statements are recommended for performance, scalability, and memory efficiency.
See Also Connector/J, Connector/ODBC, Connector/PHP, prepared statement.

CLOB
An SQL data type (`TINYTEXT`, `TEXT`, `MEDIUMTEXT`, or `LONGTEXT`) for objects containing any kind of character data, of arbitrary size. Used for storing text-based documents, with associated character set and collation order. The techniques for handling CLOBs within a MySQL application vary with each **Connector** and **API**. MySQL Connector/ODBC defines `TEXT` values as `LONGVARCHAR`. For storing binary data, the equivalent is the **BLOB** type.
See Also API, BLOB, connector, Connector/ODBC.

clustered index
The `InnoDB` term for a **primary key** index. `InnoDB` table storage is organized based on the values of the primary key columns, to speed up queries and sorts involving the primary key columns. For best performance, choose the primary key columns carefully based on the most performance-critical queries. Because modifying the columns of the clustered index is an expensive operation, choose primary columns that are rarely or never updated.

In the Oracle Database product, this type of table is known as an **index-organized table**.
See Also index, primary key, secondary index.

cold backup
A **backup** taken while the database is shut down. For busy applications and websites, this might not be practical, and you might prefer a **warm backup** or a **hot backup**.
See Also backup, hot backup, warm backup.

column
A data item within a **row**, whose storage and semantics are defined by a data type. Each **table** and **index** is largely defined by the set of columns it contains.

Each column has a **cardinality** value. A column can be the **primary key** for its table, or part of the primary key. A column can be subject to a **unique constraint**, a **NOT NULL constraint**, or both. Values in different columns, even across different tables, can be linked by a **foreign key** relationship.

In discussions of MySQL internal operations, sometimes **field** is used as a synonym.
See Also cardinality, foreign key, index, NOT NULL constraint, primary key, row, table, unique constraint.

column index
An **index** on a single column.
See Also composite index, index.

column prefix
When an **index** is created with a length specification, such as `CREATE INDEX idx ON t1 (c1(N))`, only the first N characters of the column value are stored in the index. Keeping the index prefix small makes the index compact, and the memory and disk I/O savings help performance. (Although making the index prefix too small can hinder query optimization by making rows with different values appear to the query optimizer to be duplicates.)

For columns containing binary values or long text strings, where sorting is not a major consideration and storing the entire value in the index would waste space, the index automatically uses the first N (typically 768) characters of the value to do lookups and sorts.
See Also index.

command interceptor
Synonym for **statement interceptor**. One aspect of the **interceptor** design pattern available for both **Connector/NET** and **Connector/J**. What Connector/NET calls a command, Connector/J refers to as a statement. Contrast with **exception interceptor**.
See Also Connector/J, Connector/NET, exception interceptor, interceptor, statement interceptor.

commit
A **SQL** statement that ends a **transaction**, making permanent any changes made by the transaction. It is the opposite of **rollback**, which undoes any changes made in the transaction.

`InnoDB` uses an **optimistic** mechanism for commits, so that changes can be written to the data files before the commit actually occurs. This technique makes the commit itself faster, with the tradeoff that more work is required in case of a rollback.

By default, MySQL uses the **autocommit** setting, which automatically issues a commit following each SQL statement.
See Also autocommit, optimistic, rollback, SQL, transaction.

compact row format
The default `InnoDB` **row format** for `InnoDB` tables since MySQL 5.0.3. The **COMPACT** row format provides a more compact representation for nulls and variable-length columns than the prior default (**REDUNDANT** row format).

For additional information about `InnoDB COMPACT` row format, see Section 14.11, “InnoDB Row Formats”.

See Also Antelope, dynamic row format, file format, redundant row format, row format.

composite index
An **index** that includes multiple columns.
See Also index.

compressed backup
The compression feature of the **MySQL Enterprise Backup** product makes a compressed copy of each tablespace, changing the extension from `.ibd` to `.ibz`. Compressing backup data allows you to keep more backups on hand, and reduces the time to transfer backups to a different server. The data is uncompressed during the restore operation. When a compressed backup operation processes a table that is already compressed, it skips the compression step for that table, because compressing again would result in little or no space savings.

A set of files produced by the **MySQL Enterprise Backup** product, where each **tablespace** is compressed. The compressed files are renamed with a `.ibz` file extension.

Applying **compression** at the start of the backup process helps to avoid storage overhead during the compression process, and to avoid network overhead when transferring the backup files to another server. The process of **applying** the **binary log** takes longer, and requires uncompressing the backup files.
See Also apply, binary log, compression, hot backup, MySQL Enterprise Backup, tablespace.

compressed row format
A **row format** that enables data and index **compression** for `InnoDB` tables. It was introduced in the `InnoDB` Plugin, available as part of the **Barracuda** file format. Large fields are stored away from the page that holds the rest of the row data, as in **dynamic row format**. Both index pages and large fields are compressed, yielding memory and disk savings. Depending on the structure of the data, the decrease in memory and disk usage might or might not outweigh the performance overhead of uncompressing the data as it is used. See Section 14.9, “InnoDB Table Compression” for usage details.

For additional information about `InnoDB COMPRESSED` row format, see DYNAMIC Row Format.
See Also Barracuda, compression, dynamic row format, row format.

compressed table
A table for which the data is stored in compressed form. For `InnoDB`, it is a table created with `ROW_FORMAT=COMPRESSED`. See Section 14.9, “InnoDB Table Compression” for more information.
See Also compressed row format, compression.

compression
A feature with wide-ranging benefits from using less disk space, performing less I/O, and using less memory for caching. `InnoDB` table and index data can be kept in a compressed format during database operation.

The data is uncompressed when needed for queries, and re-compressed when changes are made by **DML** operations. After you enable compression for a table, this processing is transparent to users and application developers. DBAs can consult `InnoDB INFORMATION_SCHEMA` tables to monitor how efficiently the compression parameters work for the MySQL instance and for particular compressed tables.

When `InnoDB` table data is compressed, the compression applies to the table itself, any associated index data, and the pages loaded into the **buffer pool**. Compression does not apply to pages in the **undo buffer**.

The table compression feature requires using MySQL 5.5 or higher, or the InnoDB Plugin in MySQL 5.1 or earlier, and creating the table using the **Barracuda** file format and **compressed row format**, with the **innodb_file_per_table** setting enabled.

Compression for each table is influenced by the `KEY_BLOCK_SIZE` clause of the `CREATE TABLE` and `ALTER TABLE` statements. In MySQL 5.6 and higher, compression is also affected by the server-wide configuration options `innodb_compression_failure_threshold_pct`, `innodb_compression_level`, and `innodb_compression_pad_pct_max`. See Section 14.9, “InnoDB Table Compression” for usage details.

Another type of compression is the **compressed backup** feature of the **MySQL Enterprise Backup** product.

See Also Barracuda, buffer pool, compressed backup, compressed row format, DML, transparent page compression.

compression failure
Not actually an error, rather an expensive operation that can occur when using **compression** in combination with **DML** operations. It occurs when: updates to a compressed **page** overflow the area on the page reserved for recording modifications; the page is compressed again, with all changes applied to the table data; the re-compressed data does not fit on the original page, requiring MySQL to split the data into two new pages and compress each one separately. To check the frequency of this condition, query the `INFORMATION_SCHEMA.INNODB_CMP` table and check how much the value of the `COMPRESS_OPS` column exceeds the value of the `COMPRESS_OPS_OK` column. Ideally, compression failures do not occur often; when they do, you can adjust the `innodb_compression_level`, `innodb_compression_failure_threshold_pct`, and `innodb_compression_pad_pct_max` configuration options.
See Also compression, DML, page.

concatenated index
See composite index.

concurrency
The ability of multiple operations (in database terminology, **transactions**) to run simultaneously, without interfering with each other. Concurrency is also involved with performance, because ideally the protection for multiple simultaneous transactions works with a minimum of performance overhead, using efficient mechanisms for **locking**.
See Also ACID, locking, transaction.

configuration file
The file that holds the **option** values used by MySQL at startup. Traditionally, on Linux and Unix this file is named `my.cnf`, and on Windows it is named `my.ini`. You can set a number of options related to InnoDB under the `[mysqld]` section of the file.

See Section 4.2.2.2, “Using Option Files” for information about where MySQL searches for configuration files.

When you use the **MySQL Enterprise Backup** product, you typically use two configuration files: one that specifies where the data comes from and how it is structured (which could be the original configuration file for your server), and a stripped-down one containing only a small set of options that specify where the backup data goes and how it is structured. The configuration files used with the **MySQL Enterprise Backup** product must contain certain options that are typically left out of regular configuration files, so you might need to add options to your existing configuration file for use with **MySQL Enterprise Backup**.
See Also my.cnf, MySQL Enterprise Backup, option, option file.

connection
The communication channel between an application and a MySQL server. The performance and scalability of a database applications is influenced by on how quickly a database connection can be established, how many can be made simultaneously, and how long they persist. The parameters such as **host**, **port**, and so on are represented as a **connection string** in **Connector/NET**, and as a **DSN** in **Connector/ODBC**. High-traffic systems make use of an optimization known as the **connection pool**.
See Also connection pool, connection string, Connector/NET, Connector/ODBC, DSN, host, port.

connection pool
A cache area that allows database **connections** to be reused within the same application or across different applications, rather than setting up and tearing down a new connection for every database operation. This technique is common with **J2EE** application servers. **Java** applications using **Connector/J** can use the connection pool features of **Tomcat** and other application servers. The reuse is transparent to applications; the application still opens and closes the connection as usual.
See Also connection, Connector/J, J2EE, Tomcat.

connection string
A representation of the parameters for a database **connection**, encoded as a string literal so that it can be used in program code. The parts of the string represent connection parameters such as **host** and **port**. A

connection string contains several key-value pairs, separated by semicolons. Each key-value pair is joined with an equal sign. Frequently used with **Connector/NET** applications; see Creating a Connector/NET Connection String for details.
See Also connection, Connector/NET, host, port.

connector
MySQL Connectors provide connectivity to the MySQL server for **client** programs. Several programming languages and frameworks each have their own associated Connector. Contrast with the lower-level access provided by an **API**.
See Also API, client, Connector/C++, Connector/J, Connector/NET, Connector/ODBC.

Connector/C++
Connector/C++ 8.0 can be used to access MySQL servers that implement a document store, or in a traditional way using SQL queries. It enables development of C++ applications using X DevAPI, or plain C applications using X DevAPI for C. It also enables development of C++ applications that use the legacy JDBC-based API from Connector/C++ 1.1. For more information, see MySQL Connector/C++ 8.0 Developer Guide.
See Also client, connector, JDBC.

Connector/J
A **JDBC** driver that provides connectivity for **client** applications developed in the **Java** programming language. Different versions are available that are compatible with the JDBC 3.0 and JDBC 4.0 specifications. MySQL Connector/J is a JDBC Type 4 driver: a pure-Java implementation of the MySQL protocol that does not rely on the MySQL **client libraries**. For full details, see MySQL Connector/J 5.1 Developer Guide.
See Also client, client libraries, connector, Java, JDBC.

Connector/NET
A MySQL **connector** for developers writing applications using languages, technologies, and frameworks such as **C#**, **.NET**, **Mono**, **Visual Studio**, **ASP.net**, and **ADO.net**.
See Also ADO.NET, ASP.net, connector, C#, Mono, Visual Studio.

Connector/ODBC
The family of MySQL ODBC drivers that provide access to a MySQL database using the industry standard Open Database Connectivity (**ODBC**) API. Formerly called MyODBC drivers. For full details, see MySQL Connector/ODBC Developer Guide.
See Also connector, ODBC.

Connector/PHP
A version of the `mysql` and `mysqli` **APIs** for **PHP** optimized for the Windows operating system.
See Also connector, PHP, PHP API.

consistent read
A read operation that uses **snapshot** information to present query results based on a point in time, regardless of changes performed by other transactions running at the same time. If queried data has been changed by another transaction, the original data is reconstructed based on the contents of the **undo log**. This technique avoids some of the **locking** issues that can reduce **concurrency** by forcing transactions to wait for other transactions to finish.

With **REPEATABLE READ isolation level**, the snapshot is based on the time when the first read operation is performed. With **READ COMMITTED** isolation level, the snapshot is reset to the time of each consistent read operation.

Consistent read is the default mode in which `InnoDB` processes `SELECT` statements in **READ COMMITTED** and **REPEATABLE READ** isolation levels. Because a consistent read does not set any locks on the tables it accesses, other sessions are free to modify those tables while a consistent read is being performed on the table.

For technical details about the applicable isolation levels, see Section 14.7.2.3, “Consistent Nonlocking Reads”.

See Also concurrency, isolation level, locking, READ COMMITTED, REPEATABLE READ, snapshot, transaction, undo log.

constraint
An automatic test that can block database changes to prevent data from becoming inconsistent. (In computer science terms, a kind of assertion related to an invariant condition.) Constraints are a crucial component of the **ACID** philosophy, to maintain data consistency. Constraints supported by MySQL include **FOREIGN KEY constraints** and **unique constraints**.
See Also ACID, foreign key, unique constraint.

counter
A value that is incremented by a particular kind of `InnoDB` operation. Useful for measuring how busy a server is, troubleshooting the sources of performance issues, and testing whether changes (for example, to configuration settings or indexes used by queries) have the desired low-level effects. Different kinds of counters are available through **Performance Schema** tables and **INFORMATION_SCHEMA** tables, particularly `INFORMATION_SCHEMA.INNODB_METRICS`.
See Also INFORMATION_SCHEMA, metrics counter, Performance Schema.

covering index
An **index** that includes all the columns retrieved by a query. Instead of using the index values as pointers to find the full table rows, the query returns values from the index structure, saving disk I/O. `InnoDB` can apply this optimization technique to more indexes than MyISAM can, because `InnoDB` **secondary indexes** also include the **primary key** columns. `InnoDB` cannot apply this technique for queries against tables modified by a transaction, until that transaction ends.

Any **column index** or **composite index** could act as a covering index, given the right query. Design your indexes and queries to take advantage of this optimization technique wherever possible.
See Also column index, composite index, index, primary key, secondary index.

CPU-bound
A type of **workload** where the primary **bottleneck** is CPU operations in memory. Typically involves read-intensive operations where the results can all be cached in the **buffer pool**.
See Also bottleneck, buffer pool, workload.

crash
MySQL uses the term “crash” to refer generally to any unexpected **shutdown** operation where the server cannot do its normal cleanup. For example, a crash could happen due to a hardware fault on the database server machine or storage device; a power failure; a potential data mismatch that causes the MySQL server to halt; a **fast shutdown** initiated by the DBA; or many other reasons. The robust, automatic **crash recovery** for **InnoDB** tables ensures that data is made consistent when the server is restarted, without any extra work for the DBA.
See Also crash recovery, fast shutdown, InnoDB, shutdown.

crash recovery
The cleanup activities that occur when MySQL is started again after a **crash**. For **InnoDB** tables, changes from incomplete transactions are replayed using data from the **redo log**. Changes that were **committed** before the crash, but not yet written into the **data files**, are reconstructed from the **doublewrite buffer**. When the database is shut down normally, this type of activity is performed during shutdown by the **purge** operation.

During normal operation, committed data can be stored in the **change buffer** for a period of time before being written to the data files. There is always a tradeoff between keeping the data files up-to-date, which introduces performance overhead during normal operation, and buffering the data, which can make shutdown and crash recovery take longer.
See Also change buffer, commit, crash, data files, doublewrite buffer, InnoDB, purge, redo log.

CRUD
Acronym for “create, read, update, delete”, a common sequence of operations in database applications. Often denotes a class of applications with relatively simple database usage (basic **DDL**, **DML** and **query** statements in **SQL**) that can be implemented quickly in any language.

See Also DDL, DML, query, SQL.

cursor
An internal MySQL data structure that represents the result set of an SQL statement. Often used with **prepared statements** and **dynamic SQL**. It works like an iterator in other high-level languages, producing each value from the result set as requested.

Although SQL usually handles the processing of cursors for you, you might delve into the inner workings when dealing with performance-critical code.
See Also dynamic SQL, prepared statement, query.

D

data definition language
See DDL.

data dictionary
Metadata that keeps track of InnoDB-related objects such as **tables**, **indexes**, and table **columns**. This metadata is physically located in the `InnoDB` **system tablespace**. For historical reasons, it overlaps to some degree with information stored in the **.frm files**.

Because the **MySQL Enterprise Backup** product always backs up the system tablespace, all backups include the contents of the data dictionary.
See Also column, file-per-table, .frm file, index, MySQL Enterprise Backup, system tablespace, table.

data directory
The directory under which each MySQL **instance** keeps the **data files** for `InnoDB` and the directories representing individual databases. Controlled by the `datadir` configuration option.
See Also data files, instance.

data files
The files that physically contain **table** and **index** data.

The `InnoDB` **system tablespace**, which holds the `InnoDB` **data dictionary** and is capable of holding data for multiple `InnoDB` tables, is represented by one or more `.ibdata` data files.

File-per-table tablespaces, which hold data for a single `InnoDB` table, are represented by a `.ibd` data file.

General tablespaces (introduced in MySQL 5.7.6), which can hold data for multiple `InnoDB` tables, are also represented by a `.ibd` data file.
See Also data dictionary, file-per-table, general tablespace, .ibd file, ibdata file, index, system tablespace, table, tablespace.

data manipulation language
See DML.

data warehouse
A database system or application that primarily runs large **queries**. The read-only or read-mostly data might be organized in **denormalized** form for query efficiency. Can benefit from the optimizations for **read-only transactions** in MySQL 5.6 and higher. Contrast with **OLTP**.
See Also denormalized, OLTP, query, read-only transaction.

database
Within the MySQL **data directory**, each database is represented by a separate directory. The InnoDB **system tablespace**, which can hold table data from multiple databases within a MySQL **instance**, is kept in **data files** that reside outside of individual database directories. When **file-per-table** mode is enabled, the **.ibd files** representing individual InnoDB tables are stored inside the database directories unless created elsewhere using the `DATA DIRECTORY` clause. General tablespaces, introduced in MySQL 5.7.6, also hold table data in **.ibd files**. Unlike file-per-table **.ibd files**, general tablespace **.ibd files** can hold table data from

multiple databases within a MySQL **instance**, and can be assigned to directories relative to or independent of the MySQL data directory.

For long-time MySQL users, a database is a familiar notion. Users coming from an Oracle Database background may find that the MySQL meaning of a database is closer to what Oracle Database calls a **schema**.
See Also data files, file-per-table, .ibd file, instance, schema, system tablespace.

DCL
Data control language, a set of **SQL** statements for managing privileges. In MySQL, consists of the `GRANT` and `REVOKE` statements. Contrast with **DDL** and **DML**.
See Also DDL, DML, SQL.

DDEX provider
A feature that lets you use the data design tools within **Visual Studio** to manipulate the schema and objects within a MySQL database. For MySQL applications using **Connector/NET**, the MySQL Visual Studio Plugin acts as a DDEX provider with MySQL 5.0 and later.
See Also Visual Studio.

DDL
Data definition language, a set of **SQL** statements for manipulating the database itself rather than individual table rows. Includes all forms of the `CREATE`, `ALTER`, and `DROP` statements. Also includes the `TRUNCATE` statement, because it works differently than a `DELETE FROM` *`table_name`* statement, even though the ultimate effect is similar.

DDL statements automatically **commit** the current **transaction**; they cannot be **rolled back**.

The `InnoDB` online DDL feature enhances performance for `CREATE INDEX`, `DROP INDEX`, and many types of `ALTER TABLE` operations. See Section 14.13, “InnoDB and Online DDL” for more information. Also, the `InnoDB` file-per-table setting can affect the behavior of `DROP TABLE` and `TRUNCATE TABLE` operations.

Contrast with **DML** and **DCL**.
See Also commit, DCL, DML, file-per-table, rollback, SQL, transaction.

deadlock
A situation where different **transactions** are unable to proceed, because each holds a **lock** that the other needs. Because both transactions are waiting for a resource to become available, neither one ever releases the locks it holds.

A deadlock can occur when the transactions lock rows in multiple tables (through statements such as `UPDATE` or `SELECT ... FOR UPDATE`), but in the opposite order. A deadlock can also occur when such statements lock ranges of index records and **gaps**, with each transaction acquiring some locks but not others due to a timing issue.

For background information on how deadlocks are automatically detected and handled, see Section 14.7.5.2, “Deadlock Detection”. For tips on avoiding and recovering from deadlock conditions, see Section 14.7.5.3, “How to Minimize and Handle Deadlocks”.
See Also gap, lock, transaction.

deadlock detection
A mechanism that automatically detects when a **deadlock** occurs, and automatically **rolls back** one of the **transactions** involved (the **victim**).
See Also deadlock, rollback, transaction, victim.

delete
When `InnoDB` processes a `DELETE` statement, the rows are immediately marked for deletion and no longer are returned by queries. The storage is reclaimed sometime later, during the periodic garbage collection known as the **purge** operation. For removing large quantities of data, related operations with their own performance characteristics are **TRUNCATE** and **DROP**.

See Also drop, purge, truncate.

delete buffering
: The technique of storing changes to secondary index pages, resulting from `DELETE` operations, in the **change buffer** rather than writing the changes immediately, so that the physical writes can be performed to minimize random I/O. (Because delete operations are a two-step process, this operation buffers the write that normally marks an index record for deletion.) It is one of the types of **change buffering**; the others are **insert buffering** and **purge buffering**.
See Also change buffer, change buffering, insert buffer, insert buffering, purge buffering.

denormalized
: A data storage strategy that duplicates data across different tables, rather than linking the tables with **foreign keys** and **join** queries. Typically used in **data warehouse** applications, where the data is not updated after loading. In such applications, query performance is more important than making it simple to maintain consistent data during updates. Contrast with **normalized**.
See Also data warehouse, foreign key, join, normalized.

descending index
: A type of **index** available with some database systems, where index storage is optimized to process `ORDER BY column DESC` clauses. Currently, although MySQL allows the `DESC` keyword in the `CREATE TABLE` statement, it does not use any special storage layout for the resulting index.
See Also index.

dirty page
: A **page** in the `InnoDB` **buffer pool** that has been updated in memory, where the changes are not yet written (**flushed**) to the **data files**. The opposite of a **clean page**.
See Also buffer pool, clean page, data files, flush, page.

dirty read
: An operation that retrieves unreliable data, data that was updated by another transaction but not yet **committed**. It is only possible with the **isolation level** known as **read uncommitted**.

  This kind of operation does not adhere to the **ACID** principle of database design. It is considered very risky, because the data could be **rolled back**, or updated further before being committed; then, the transaction doing the dirty read would be using data that was never confirmed as accurate.

  Its opposite is **consistent read**, where `InnoDB` ensures that a transaction does not read information updated by another transaction, even if the other transaction commits in the meantime.
  See Also ACID, commit, consistent read, isolation level, READ UNCOMMITTED, rollback.

disk-based
: A kind of database that primarily organizes data on disk storage (hard drives or equivalent). Data is brought back and forth between disk and memory to be operated upon. It is the opposite of an **in-memory database**. Although `InnoDB` is disk-based, it also contains features such as he **buffer pool**, multiple buffer pool instances, and the **adaptive hash index** that allow certain kinds of workloads to work primarily from memory.
See Also adaptive hash index, buffer pool, in-memory database.

disk-bound
: A type of **workload** where the primary **bottleneck** is disk I/O. (Also known as **I/O-bound**.) Typically involves frequent writes to disk, or random reads of more data than can fit into the **buffer pool**.
See Also bottleneck, buffer pool, workload.

DML
: Data manipulation language, a set of **SQL** statements for performing `INSERT`, `UPDATE`, and `DELETE` operations. The `SELECT` statement is sometimes considered as a DML statement, because the `SELECT ... FOR UPDATE` form is subject to the same considerations for **locking** as `INSERT`, `UPDATE`, and `DELETE`.

  DML statements for an `InnoDB` table operate in the context of a **transaction**, so their effects can be **committed** or **rolled back** as a single unit.

Contrast with **DDL** and **DCL**.
See Also commit, DCL, DDL, locking, rollback, SQL, transaction.

document id
In the `InnoDB` **full-text search** feature, a special column in the table containing the **FULLTEXT index**, to uniquely identify the document associated with each **ilist** value. Its name is `FTS_DOC_ID` (uppercase required). The column itself must be of `BIGINT UNSIGNED NOT NULL` type, with a unique index named `FTS_DOC_ID_INDEX`. Preferably, you define this column when creating the table. If `InnoDB` must add the column to the table while creating a `FULLTEXT` index, the indexing operation is considerably more expensive.
See Also full-text search, FULLTEXT index, ilist.

doublewrite buffer
`InnoDB` uses a file flush technique called doublewrite. Before writing **pages** to the **data files**, `InnoDB` first writes them to a storage area called the doublewrite buffer. Only after the write and the flush to the doublewrite buffer have completed, does `InnoDB` write the pages to their proper positions in the data file. If there is an operating system, storage subsystem, or `mysqld` process crash in the middle of a page write, `InnoDB` can later find a good copy of the page from the doublewrite buffer during **crash recovery**.

Although data is always written twice, the doublewrite buffer does not require twice as much I/O overhead or twice as many I/O operations. Data is written to the buffer itself as a large sequential chunk, with a single `fsync()` call to the operating system.

To turn off the doublewrite buffer, specify the option `innodb_doublewrite=0`.
See Also crash recovery, data files, page, purge.

drop
A kind of **DDL** operation that removes a schema object, through a statement such as `DROP TABLE` or `DROP INDEX`. It maps internally to an `ALTER TABLE` statement. From an `InnoDB` perspective, the performance considerations of such operations involve the time that the **data dictionary** is locked to ensure that interrelated objects are all updated, and the time to update memory structures such as the **buffer pool**. For a **table**, the drop operation has somewhat different characteristics than a **truncate** operation (`TRUNCATE TABLE` statement).
See Also buffer pool, data dictionary, DDL, table, truncate.

DSN
Acronym for “Database Source Name”. It is the encoding for **connection** information within **Connector/ODBC**. See Configuring a Connector/ODBC DSN on Windows for full details. It is the equivalent of the **connection string** used by **Connector/NET**.
See Also connection, connection string, Connector/NET, Connector/ODBC.

dynamic cursor
A type of **cursor** supported by **ODBC** that can pick up new and changed results when the rows are read again. Whether and how quickly the changes are visible to the cursor depends on the type of table involved (transactional or non-transactional) and the isolation level for transactional tables. Support for dynamic cursors must be explicitly enabled.
See Also cursor, ODBC.

dynamic row format
A row format introduced in the `InnoDB` Plugin, available as part of the **Barracuda file format**. Because long variable-length column values are stored outside of the page that holds the row data, it is very efficient for rows that include large objects. Since the large fields are typically not accessed to evaluate query conditions, they are not brought into the **buffer pool** as often, resulting in fewer I/O operations and better utilization of cache memory.

As of MySQL 5.7.9, the default row format is defined by `innodb_default_row_format`, which has a default value of `DYNAMIC`.

For additional information about `InnoDB` `DYNAMIC` row format, see DYNAMIC Row Format.
See Also Barracuda, buffer pool, file format, row format.

dynamic SQL
: A feature that lets you create and execute **prepared statements** using more robust, secure, and efficient methods to substitute parameter values than the naive technique of concatenating the parts of the statement into a string variable.
See Also prepared statement.

dynamic statement
: A **prepared statement** created and executed through **dynamic SQL**.
See Also dynamic SQL, prepared statement.

E

early adopter
: A stage similar to **beta**, when a software product is typically evaluated for performance, functionality, and compatibility in a non-mission-critical setting.
See Also beta.

Eiffel
: A programming language including many object-oriented features. Some of its concepts are familiar to **Java** and **C#** developers. For the open-source Eiffel **API** for MySQL, see Section 23.13, “MySQL Eiffel Wrapper”.
See Also API, C#, Java.

embedded
: The embedded MySQL server library (**libmysqld**) makes it possible to run a full-featured MySQL server inside a **client** application. The main benefits are increased speed and more simple management for embedded applications.
See Also client, libmysqld.

error log
: A type of **log** showing information about MySQL startup and critical runtime errors and **crash** information. For details, see Section 5.4.2, “The Error Log”.
See Also crash, log.

eviction
: The process of removing an item from a cache or other temporary storage area, such as the `InnoDB` **buffer pool**. Often, but not always, uses the **LRU** algorithm to determine which item to remove. When a **dirty page** is evicted, its contents are **flushed** to disk, and any dirty **neighbor pages** might be flushed also.
See Also buffer pool, dirty page, flush, LRU, neighbor page.

exception interceptor
: A type of **interceptor** for tracing, debugging, or augmenting SQL errors encountered by a database application. For example, the interceptor code could issue a `SHOW WARNINGS` statement to retrieve additional information, and add descriptive text or even change the type of the exception returned to the application. Because the interceptor code is only called when SQL statements return errors, it does not impose any performance penalty on the application during normal (error-free) operation.

  In **Java** applications using **Connector/J**, setting up this type of interceptor involves implementing the `com.mysql.jdbc.ExceptionInterceptor` interface, and adding a `exceptionInterceptors` property to the **connection string**.

  In **Visual Studio** applications using **Connector/NET**, setting up this type of interceptor involves defining a class that inherits from the `BaseExceptionInterceptor` class and specifying that class name as part of the connection string.
  See Also Connector/J, Connector/NET, interceptor, Java, Visual Studio.

exclusive lock
: A kind of **lock** that prevents any other **transaction** from locking the same row. Depending on the transaction **isolation level**, this kind of lock might block other transactions from writing to the same row, or might also

block other transactions from reading the same row. The default InnoDB isolation level, **REPEATABLE READ**, enables higher **concurrency** by allowing transactions to read rows that have exclusive locks, a technique known as **consistent read**.
See Also concurrency, consistent read, isolation level, lock, REPEATABLE READ, shared lock, transaction.

extent
A group of **pages** within a **tablespace**. For the default **page size** of 16KB, an extent contains 64 pages. In MySQL 5.6, the page size for an InnoDB instance can be 4KB, 8KB, or 16KB, controlled by the innodb_page_size configuration option. For 4KB, 8KB, and 16KB pages sizes, the extent size is always 1MB (or 1048576 bytes).

Support for 32KB and 64KB InnoDB page sizes was added in MySQL 5.7.6. For a 32KB page size, the extent size is 2MB. For a 64KB page size, the extent size is 4MB.

InnoDB features such as **segments**, **read-ahead** requests and the **doublewrite buffer** use I/O operations that read, write, allocate, or free data one extent at a time.
See Also doublewrite buffer, page, page size, read-ahead, segment, tablespace.

F

.frm file
A file containing the metadata, such as the table definition, of a MySQL table.

For backups, you must always keep the full set of .frm files along with the backup data to be able to restore tables that are altered or dropped after the backup.

Although each InnoDB table has a .frm file, InnoDB maintains its own table metadata in the **system tablespace**.

.frm files are backed up by the **MySQL Enterprise Backup** product. These files must not be modified by an ALTER TABLE operation while the backup is taking place, which is why backups that include non-InnoDB tables perform a FLUSH TABLES WITH READ LOCK operation to freeze such activity while backing up .frm files. Restoring a backup can result in .frm files being created, changed, or removed to match the state of the database at the time of the backup.
See Also data dictionary, MySQL Enterprise Backup, system tablespace.

failover
The ability to automatically switch to a standby server in the event of a failure. In the MySQL context, failover involves a standby database server. Often supported within **J2EE** environments by the application server or framework.
See Also Connector/J, J2EE.

Fast Index Creation
A capability first introduced in the InnoDB Plugin, now part of MySQL in 5.5 and higher, that speeds up creation of InnoDB **secondary indexes** by avoiding the need to completely rewrite the associated table. The speedup applies to dropping secondary indexes also.

Because index maintenance can add performance overhead to many data transfer operations, consider doing operations such as ALTER TABLE ... ENGINE=INNODB or INSERT INTO ... SELECT * FROM ... without any secondary indexes in place, and creating the indexes afterward.

In MySQL 5.6, this feature becomes more general. You can read and write to tables while an index is being created, and many more kinds of ALTER TABLE operations can be performed without copying the table, without blocking **DML** operations, or both. Thus in MySQL 5.6 and higher, this set of features is referred to as **online DDL** rather than Fast Index Creation.

For related information, see Section 14.13, “InnoDB and Online DDL”.
See Also DML, index, online DDL, secondary index.

fast shutdown

The default **shutdown** procedure for `InnoDB`, based on the configuration setting `innodb_fast_shutdown=1`. To save time, certain **flush** operations are skipped. This type of shutdown is safe during normal usage, because the flush operations are performed during the next startup, using the same mechanism as in **crash recovery**. In cases where the database is being shut down for an upgrade or downgrade, do a **slow shutdown** instead to ensure that all relevant changes are applied to the **data files** during the shutdown.
See Also crash recovery, data files, flush, shutdown, slow shutdown.

file format

The file format for `InnoDB` tables, enabled using the `innodb_file_format` configuration option. Supported file formats are **Antelope** and **Barracuda**. Antelope is the original `InnoDB` file format and supports the **REDUNDANT** and **COMPACT** row formats. Barracuda is the newer `InnoDB` file format and supports the **COMPRESSED** and **DYNAMIC** row formats.
See Also Antelope, Barracuda, file-per-table, .ibd file, ibdata file, row format.

file-per-table

A general name for the setting controlled by the `innodb_file_per_table` option, which is an important configuration option that affects aspects of `InnoDB` file storage, availability of features, and I/O characteristics. As of MySQL 5.6.7, `innodb_file_per_table` is enabled by default.

With the `innodb_file_per_table` option enabled, you can create a table in its own **.ibd file** rather than in the shared **ibdata files** of the **system tablespace**. When table data is stored in an individual **.ibd file**, you have more flexibility to choose **row formats** required for features such as data **compression**. The `TRUNCATE TABLE` operation is also faster, and reclaimed space can be used by the operating system rather than remaining reserved for `InnoDB`.

The **MySQL Enterprise Backup** product is more flexible for tables that are in their own files. For example, tables can be excluded from a backup, but only if they are in separate files. Thus, this setting is suitable for tables that are backed up less frequently or on a different schedule.
See Also compressed row format, compression, file format, .ibd file, ibdata file, innodb_file_per_table, MySQL Enterprise Backup, row format, system tablespace.

fill factor

In an `InnoDB` **index**, the proportion of a **page** that is taken up by index data before the page is split. The unused space when index data is first divided between pages allows for rows to be updated with longer string values without requiring expensive index maintenance operations. If the fill factor is too low, the index consumes more space than needed, causing extra I/O overhead when reading the index. If the fill factor is too high, any update that increases the length of column values can cause extra I/O overhead for index maintenance. See Section 14.6.2.2, “The Physical Structure of an InnoDB Index” for more information.
See Also index, page.

fixed row format

This row format is used by the `MyISAM` storage engine, not by `InnoDB`. If you create an `InnoDB` table with the option `ROW_FORMAT=FIXED` in MySQL 5.7.6 or earlier, `InnoDB` uses the **compact row format** instead, although the `FIXED` value might still show up in output such as `SHOW TABLE STATUS` reports. As of MySQL 5.7.7, `InnoDB` returns an error if `ROW_FORMAT=FIXED` is specified.
See Also compact row format, row format.

flush

To write changes to the database files, that had been buffered in a memory area or a temporary disk storage area. The `InnoDB` storage structures that are periodically flushed include the **redo log**, the **undo log**, and the **buffer pool**.

Flushing can happen because a memory area becomes full and the system needs to free some space, because a **commit** operation means the changes from a transaction can be finalized, or because a **slow shutdown** operation means that all outstanding work should be finalized. When it is not critical to flush all the buffered data at once, `InnoDB` can use a technique called **fuzzy checkpointing** to flush small batches of pages to spread out the I/O overhead.

See Also buffer pool, commit, fuzzy checkpointing, redo log, slow shutdown, undo log.

flush list
An internal `InnoDB` data structure that tracks **dirty pages** in the **buffer pool**: that is, **pages** that have been changed and need to be written back out to disk. This data structure is updated frequently by `InnoDB` internal **mini-transactions**, and so is protected by its own **mutex** to allow concurrent access to the buffer pool.
See Also buffer pool, dirty page, LRU, mini-transaction, mutex, page, page cleaner.

foreign key
A type of pointer relationship, between rows in separate `InnoDB` tables. The foreign key relationship is defined on one column in both the **parent table** and the **child table**.

In addition to enabling fast lookup of related information, foreign keys help to enforce **referential integrity**, by preventing any of these pointers from becoming invalid as data is inserted, updated, and deleted. This enforcement mechanism is a type of **constraint**. A row that points to another table cannot be inserted if the associated foreign key value does not exist in the other table. If a row is deleted or its foreign key value changed, and rows in another table point to that foreign key value, the foreign key can be set up to prevent the deletion, cause the corresponding column values in the other table to become **null**, or automatically delete the corresponding rows in the other table.

One of the stages in designing a **normalized** database is to identify data that is duplicated, separate that data into a new table, and set up a foreign key relationship so that the multiple tables can be queried like a single table, using a **join** operation.
See Also child table, FOREIGN KEY constraint, join, normalized, NULL, parent table, referential integrity, relational.

FOREIGN KEY constraint
The type of **constraint** that maintains database consistency through a **foreign key** relationship. Like other kinds of constraints, it can prevent data from being inserted or updated if data would become inconsistent; in this case, the inconsistency being prevented is between data in multiple tables. Alternatively, when a **DML** operation is performed, `FOREIGN KEY` constraints can cause data in **child rows** to be deleted, changed to different values, or set to **null**, based on the `ON CASCADE` option specified when creating the foreign key.
See Also child table, constraint, DML, foreign key, NULL.

FTS
In most contexts, an acronym for **full-text search**. Sometimes in performance discussions, an acronym for **full table scan**.
See Also full table scan, full-text search.

full backup
A **backup** that includes all the **tables** in each MySQL **database**, and all the databases in a MySQL **instance**. Contrast with **partial backup**.
See Also backup, database, instance, partial backup, table.

full table scan
An operation that requires reading the entire contents of a table, rather than just selected portions using an **index**. Typically performed either with small lookup tables, or in data warehousing situations with large tables where all available data is aggregated and analyzed. How frequently these operations occur, and the sizes of the tables relative to available memory, have implications for the algorithms used in query optimization and managing the **buffer pool**.

The purpose of indexes is to allow lookups for specific values or ranges of values within a large table, thus avoiding full table scans when practical.
See Also buffer pool, index.

full-text search
The MySQL feature for finding words, phrases, Boolean combinations of words, and so on within table data, in a faster, more convenient, and more flexible way than using the SQL `LIKE` operator or writing your own application-level search algorithm. It uses the SQL function `MATCH()` and **FULLTEXT indexes**.

See Also FULLTEXT index.

FULLTEXT index
: The special kind of **index** that holds the **search index** in the MySQL **full-text search** mechanism. Represents the words from values of a column, omitting any that are specified as **stopwords**. Originally, only available for `MyISAM` tables. Starting in MySQL 5.6.4, it is also available for **InnoDB** tables.
See Also full-text search, index, InnoDB, search index, stopword.

fuzzy checkpointing
: A technique that **flushes** small batches of **dirty pages** from the **buffer pool**, rather than flushing all dirty pages at once which would disrupt database processing.
See Also buffer pool, dirty page, flush.

G

GA
: "Generally available", the stage when a software product leaves **beta** and is available for sale, official support, and production use.
See Also beta.

GAC
: Acronym for "Global Assembly Cache". A central area for storing libraries (**assemblies**) on a **.NET** system. Physically consists of nested folders, treated as a single virtual folder by the **.NET** CLR.
See Also .NET, assembly.

gap
: A place in an `InnoDB` **index** data structure where new values could be inserted. When you lock a set of rows with a statement such as `SELECT ... FOR UPDATE`, `InnoDB` can create locks that apply to the gaps as well as the actual values in the index. For example, if you select all values greater than 10 for update, a gap lock prevents another transaction from inserting a new value that is greater than 10. The **supremum record** and **infimum record** represent the gaps containing all values greater than or less than all the current index values.
See Also concurrency, gap lock, index, infimum record, isolation level, supremum record.

gap lock
: A **lock** on a **gap** between index records, or a lock on the gap before the first or after the last index record. For example, `SELECT c1 FROM t WHERE c1 BETWEEN 10 and 20 FOR UPDATE;` prevents other transactions from inserting a value of 15 into the column `t.c1`, whether or not there was already any such value in the column, because the gaps between all existing values in the range are locked. Contrast with **record lock** and **next-key lock**.

  Gap locks are part of the tradeoff between performance and **concurrency**, and are used in some transaction **isolation levels** and not others.
  See Also gap, infimum record, lock, next-key lock, record lock, supremum record.

general log
: See general query log.

general query log
: A type of **log** used for diagnosis and troubleshooting of SQL statements processed by the MySQL server. Can be stored in a file or in a database table. You must enable this feature through the `general_log` configuration option to use it. You can disable it for a specific connection through the `sql_log_off` configuration option.

  Records a broader range of queries than the **slow query log**. Unlike the **binary log**, which is used for replication, the general query log contains `SELECT` statements and does not maintain strict ordering. For more information, see Section 5.4.3, "The General Query Log".
  See Also binary log, log, slow query log.

general tablespace
A shared `InnoDB` **tablespace** created using `CREATE TABLESPACE` syntax. General tablespaces can be created outside of the MySQL data directory, are capable of holding multiple **tables**, and support tables of all row formats. General tablespaces were introduced in MySQL 5.7.6.

Tables are added to a general tablespace using `CREATE TABLE tbl_name ... TABLESPACE [=] tablespace_name` or `ALTER TABLE tbl_name TABLESPACE [=] tablespace_name` syntax.

Contrast with **system tablespace** and **file-per-table** tablespace.

For more information, see General Tablespaces.
See Also file-per-table, system tablespace, table, tablespace.

Glassfish

See Also J2EE.

global transaction
A type of **transaction** involved in **XA** operations. It consists of several actions that are transactional in themselves, but that all must either complete successfully as a group, or all be rolled back as a group. In essence, this extends **ACID** properties "up a level" so that multiple ACID transactions can be executed in concert as components of a global operation that also has ACID properties.
See Also ACID, transaction, XA.

group commit
An `InnoDB` optimization that performs some low-level I/O operations (log write) once for a set of **commit** operations, rather than flushing and syncing separately for each commit.
See Also binary log, commit.

GUID
Acronym for "globally unique identifier", an ID value that can be used to associate data across different databases, languages, operating systems, and so on. (As an alternative to using sequential integers, where the same values could appear in different tables, databases, and so on referring to different data.) Older MySQL versions represented it as `BINARY(16)`. Currently, it is represented as `CHAR(36)`. MySQL has a `UUID()` function that returns GUID values in character format, and a `UUID_SHORT()` function that returns GUID values in integer format. Because successive GUID values are not necessarily in ascending sort order, it is not an efficient value to use as a primary key for large InnoDB tables.

H

hash index
A type of **index** intended for queries that use equality operators, rather than range operators such as greater-than or `BETWEEN`. It is available for `MEMORY` tables. Although hash indexes are the default for `MEMORY` tables for historic reasons, that storage engine also supports **B-tree** indexes, which are often a better choice for general-purpose queries.

MySQL includes a variant of this index type, the **adaptive hash index**, that is constructed automatically for `InnoDB` tables if needed based on runtime conditions.
See Also adaptive hash index, B-tree, index, InnoDB.

HDD
Acronym for "hard disk drive". Refers to storage media using spinning platters, usually when comparing and contrasting with **SSD**. Its performance characteristics can influence the throughput of a **disk-based** workload.
See Also disk-based, SSD.

heartbeat
A periodic message that is sent to indicate that a system is functioning properly. In a **replication** context, if the **source** stops sending such messages, one of the **replicas** can take its place. Similar techniques can be used between the servers in a cluster environment, to confirm that all of them are operating properly.

See Also replication, source.

high-water mark
A value representing an upper limit, either a hard limit that should not be exceeded at runtime, or a record of the maximum value that was actually reached. Contrast with **low-water mark**.
See Also low-water mark.

history list
A list of **transactions** with delete-marked records scheduled to be processed by the `InnoDB` **purge** operation. Recorded in the **undo log**. The length of the history list is reported by the command `SHOW ENGINE INNODB STATUS`. If the history list grows longer than the value of the `innodb_max_purge_lag` configuration option, each **DML** operation is delayed slightly to allow the purge operation to finish **flushing** the deleted records.

Also known as **purge lag**.
See Also DML, flush, purge, purge lag, rollback segment, transaction, undo log.

hole punching
Releasing empty blocks from a page. The `InnoDB` **transparent page compression** feature relies on hole punching support. For more information, see InnoDB Page Compression.
See Also sparse file, transparent page compression.

host
The network name of a database server, used to establish a **connection**. Often specified in conjunction with a **port**. In some contexts, the IP address `127.0.0.1` works better than the special name `localhost` for accessing a database on the same server as the application.
See Also connection, localhost, port.

hot
A condition where a row, table, or internal data structure is accessed so frequently, requiring some form of locking or mutual exclusion, that it results in a performance or scalability issue.

Although “hot” typically indicates an undesirable condition, a **hot backup** is the preferred type of backup.
See Also hot backup.

hot backup
A backup taken while the database is running and applications are reading and writing to it. The backup involves more than simply copying data files: it must include any data that was inserted or updated while the backup was in process; it must exclude any data that was deleted while the backup was in process; and it must ignore any changes that were not committed.

The Oracle product that performs hot backups, of `InnoDB` tables especially but also tables from `MyISAM` and other storage engines, is known as **MySQL Enterprise Backup**.

The hot backup process consists of two stages. The initial copying of the data files produces a **raw backup**. The **apply** step incorporates any changes to the database that happened while the backup was running. Applying the changes produces a **prepared** backup; these files are ready to be restored whenever necessary.
See Also apply, MySQL Enterprise Backup, prepared backup, raw backup.

I

.ibd file
The data file for **file-per-table** tablespaces and general tablespaces. File-per-table tablespace `.ibd` files contain a single table and associated index data. **General tablespace** `.ibd` files may contain table and index data for multiple tables. General tablespaces were introduced in MySQL 5.7.6.

The `.ibd` file extension does not apply to the **system tablespace**, which consists of one or more **ibdata files**.

If a file-per-table tablespace or general tablespace is created with the `DATA DIRECTORY =` clause, the `.ibd` file is located at the specified path, outside the normal data directory, and is pointed to by a **.isl file**.

When a `.ibd` file is included in a compressed backup by the **MySQL Enterprise Backup** product, the compressed equivalent is a `.ibz` file.
See Also database, file-per-table, general tablespace, ibdata file, .ibz file, innodb_file_per_table, .isl file, MySQL Enterprise Backup, system tablespace.

.ibz file
When the **MySQL Enterprise Backup** product performs a **compressed backup**, it transforms each **tablespace** file that is created using the **file-per-table** setting from a `.ibd` extension to a `.ibz` extension.

The compression applied during backup is distinct from the **compressed row format** that keeps table data compressed during normal operation. A compressed backup operation skips the compression step for a tablespace that is already in compressed row format, as compressing a second time would slow down the backup but produce little or no space savings.
See Also compressed backup, compressed row format, file-per-table, .ibd file, MySQL Enterprise Backup, tablespace.

.isl file
A file that specifies the location of an **.ibd file** for an `InnoDB` table created with the `DATA DIRECTORY =` clause in MySQL 5.6 and higher, or with the `CREATE TABLESPACE ... ADD DATAFILE` clause in MySQL 5.7 and higher. It functions like a symbolic link, without the platform restrictions of the actual symbolic link mechanism. You can store `InnoDB` **tablespaces** outside the **database** directory, for example, on an especially large or fast storage device depending on the usage of the table. For details, see Section 14.6.1.2, “Creating Tables Externally”, and General Tablespaces.
See Also database, .ibd file, table, tablespace.

I/O-bound
See disk-bound.

ib-file set
The set of files managed by `InnoDB` within a MySQL database: the **system tablespace**, **file-per-table** tablespace files, and **redo log** files. Depending on MySQL version and `InnoDB` configuration, may also include **general tablespace**, **temporary tablespace**, and **undo tablespace** files. This term is sometimes used in detailed discussions of `InnoDB` file structures and formats to refer to the set of files managed by `InnoDB` within a MySQL database.
See Also database, file-per-table, general tablespace, redo log, system tablespace, temporary tablespace, undo tablespace.

ibbackup_logfile
A supplemental backup file created by the **MySQL Enterprise Backup** product during a **hot backup** operation. It contains information about any data changes that occurred while the backup was running. The initial backup files, including `ibbackup_logfile`, are known as a **raw backup**, because the changes that occurred during the backup operation are not yet incorporated. After you perform the **apply** step to the raw backup files, the resulting files do include those final data changes, and are known as a **prepared backup**. At this stage, the `ibbackup_logfile` file is no longer necessary.
See Also apply, hot backup, MySQL Enterprise Backup, prepared backup, raw backup.

ibdata file
A set of files with names such as `ibdata1`, `ibdata2`, and so on, that make up the `InnoDB` **system tablespace**. For information about the structures and data that reside in the system tablespace `ibdata` files, see Section 14.6.3.1, “The System Tablespace”.

Growth of the `ibdata` files is influenced by the `innodb_autoextend_increment` configuration option.
See Also change buffer, data dictionary, doublewrite buffer, file-per-table, .ibd file, innodb_file_per_table, system tablespace, undo log.

ibtmp file
The `InnoDB` **temporary tablespace data file** for non-compressed `InnoDB` **temporary tables** and related objects. The configuration file option, `innodb_temp_data_file_path`, allows users to define a relative

path for the temporary tablespace data file. If `innodb_temp_data_file_path` is not specified, the default behavior is to create a single auto-extending 12MB data file named `ibtmp1` in the data directory, alongside `ibdata1`.
See Also data files, temporary table, temporary tablespace.

ib_logfile
A set of files, typically named `ib_logfile0` and `ib_logfile1`, that form the **redo log**. Also sometimes referred to as the **log group**. These files record statements that attempt to change data in `InnoDB` tables. These statements are replayed automatically to correct data written by incomplete transactions, on startup following a crash.

This data cannot be used for manual recovery; for that type of operation, use the **binary log**.
See Also binary log, log group, redo log.

ilist
Within an `InnoDB` **FULLTEXT index**, the data structure consisting of a document ID and positional information for a token (that is, a particular word).
See Also FULLTEXT index.

implicit row lock
A row lock that `InnoDB` acquires to ensure consistency, without you specifically requesting it.
See Also row lock.

in-memory database
A type of database system that maintains data in memory, to avoid overhead due to disk I/O and translation between disk blocks and memory areas. Some in-memory databases sacrifice durability (the “D” in the **ACID** design philosophy) and are vulnerable to hardware, power, and other types of failures, making them more suitable for read-only operations. Other in-memory databases do use durability mechanisms such as logging changes to disk or using non-volatile memory.

MySQL features that address the same kinds of memory-intensive processing include the `InnoDB` **buffer pool**, **adaptive hash index**, and **read-only transaction** optimization, the `MEMORY` storage engine, the `MyISAM` key cache, and the MySQL query cache.
See Also ACID, adaptive hash index, buffer pool, disk-based, read-only transaction.

incremental backup
A type of **hot backup**, performed by the **MySQL Enterprise Backup** product, that only saves data changed since some point in time. Having a full backup and a succession of incremental backups lets you reconstruct backup data over a long period, without the storage overhead of keeping several full backups on hand. You can restore the full backup and then apply each of the incremental backups in succession, or you can keep the full backup up-to-date by applying each incremental backup to it, then perform a single restore operation.

The granularity of changed data is at the **page** level. A page might actually cover more than one row. Each changed page is included in the backup.
See Also hot backup, MySQL Enterprise Backup, page.

index
A data structure that provides a fast lookup capability for **rows** of a **table**, typically by forming a tree structure (**B-tree)** representing all the values of a particular **column** or set of columns.

`InnoDB` tables always have a **clustered index** representing the **primary key**. They can also have one or more **secondary indexes** defined on one or more columns. Depending on their structure, secondary indexes can be classified as **partial**, **column**, or **composite** indexes.

Indexes are a crucial aspect of **query** performance. Database architects design tables, queries, and indexes to allow fast lookups for data needed by applications. The ideal database design uses a **covering index** where practical; the query results are computed entirely from the index, without reading the actual table data. Each **foreign key** constraint also requires an index, to efficiently check whether values exist in both the **parent** and **child** tables.

Although a B-tree index is the most common, a different kind of data structure is used for **hash indexes**, as in the `MEMORY` storage engine and the `InnoDB` **adaptive hash index**. **R-tree** indexes are used for spatial indexing of multi-dimensional information.
See Also adaptive hash index, B-tree, child table, clustered index, column index, composite index, covering index, foreign key, hash index, parent table, partial index, primary key, query, R-tree, row, secondary index, table.

index cache
A memory area that holds the token data for `InnoDB` **full-text search**. It buffers the data to minimize disk I/O when data is inserted or updated in columns that are part of a **FULLTEXT index**. The token data is written to disk when the index cache becomes full. Each `InnoDB` `FULLTEXT` index has its own separate index cache, whose size is controlled by the configuration option `innodb_ft_cache_size`.
See Also full-text search, FULLTEXT index.

index condition pushdown
Index condition pushdown (ICP) is an optimization that pushes part of a `WHERE` condition down to the storage engine if parts of the condition can be evaluated using fields from the **index**. ICP can reduce the number of times the **storage engine** must access the base table and the number of times the MySQL server must access the storage engine. For more information, see Section 8.2.1.5, “Index Condition Pushdown Optimization”.
See Also index, storage engine.

index hint
Extended SQL syntax for overriding the **indexes** recommended by the optimizer. For example, the `FORCE INDEX`, `USE INDEX`, and `IGNORE INDEX` clauses. Typically used when indexed columns have unevenly distributed values, resulting in inaccurate **cardinality** estimates.
See Also cardinality, index.

index prefix
In an **index** that applies to multiple columns (known as a **composite index**), the initial or leading columns of the index. A query that references the first 1, 2, 3, and so on columns of a composite index can use the index, even if the query does not reference all the columns in the index.
See Also composite index, index.

index statistics
See statistics.

infimum record
A **pseudo-record** in an **index**, representing the **gap** below the smallest value in that index. If a transaction has a statement such as `SELECT ... FROM ... WHERE col < 10 FOR UPDATE;`, and the smallest value in the column is 5, it is a lock on the infimum record that prevents other transactions from inserting even smaller values such as 0, -10, and so on.
See Also gap, index, pseudo-record, supremum record.

INFORMATION_SCHEMA
The name of the **database** that provides a query interface to the MySQL **data dictionary**. (This name is defined by the ANSI SQL standard.) To examine information (metadata) about the database, you can query tables such as `INFORMATION_SCHEMA.TABLES` and `INFORMATION_SCHEMA.COLUMNS`, rather than using `SHOW` commands that produce unstructured output.

The `INFORMATION_SCHEMA` database also contains tables specific to **InnoDB** that provide a query interface to the `InnoDB` data dictionary. You use these tables not to see how the database is structured, but to get real-time information about the workings of `InnoDB` tables to help with performance monitoring, tuning, and troubleshooting.
See Also data dictionary, database, InnoDB.

InnoDB
A MySQL component that combines high performance with **transactional** capability for reliability, robustness, and concurrent access. It embodies the **ACID** design philosophy. Represented as a **storage engine**; it handles tables created or altered with the `ENGINE=INNODB` clause. See Chapter 14, *The InnoDB Storage*

Engine for architectural details and administration procedures, and Section 8.5, “Optimizing for InnoDB Tables” for performance advice.

In MySQL 5.5 and higher, `InnoDB` is the default storage engine for new tables and the `ENGINE=INNODB` clause is not required.

`InnoDB` tables are ideally suited for **hot backups**. See Section 24.2, “MySQL Enterprise Backup Overview” for information about the **MySQL Enterprise Backup** product for backing up MySQL servers without interrupting normal processing.
See Also ACID, hot backup, MySQL Enterprise Backup, storage engine, transaction.

innodb_autoinc_lock_mode
The `innodb_autoinc_lock_mode` option controls the algorithm used for **auto-increment locking**. When you have an auto-incrementing **primary key**, you can use statement-based replication only with the setting `innodb_autoinc_lock_mode=1`. This setting is known as *consecutive* lock mode, because multi-row inserts within a transaction receive consecutive auto-increment values. If you have `innodb_autoinc_lock_mode=2`, which allows higher concurrency for insert operations, use row-based replication rather than statement-based replication. This setting is known as *interleaved* lock mode, because multiple multi-row insert statements running at the same time can receive **auto-increment** values that are interleaved. The setting `innodb_autoinc_lock_mode=0` should not be used except for compatibility purposes.

Consecutive lock mode (`innodb_autoinc_lock_mode=1`) is the default setting prior to MySQL 8.0.3. As of MySQL 8.0.3, interleaved lock mode (`innodb_autoinc_lock_mode=2`) is the default, which reflects the change from statement-based to row-based replication as the default replication type.
See Also auto-increment, auto-increment locking, mixed-mode insert, primary key.

innodb_file_format
The `innodb_file_format` option defines the **file format** to use for new `InnoDB` file-per-table **tablespaces**. Currently, you can specify the **Antelope** and **Barracuda** file formats.
See Also Antelope, Barracuda, file format, file-per-table, general tablespace, innodb_file_per_table, system tablespace, tablespace.

innodb_file_per_table
An important configuration option that affects many aspects of `InnoDB` file storage, availability of features, and I/O characteristics. In MySQL 5.6.7 and higher, it is enabled by default. The `innodb_file_per_table` option turns on **file-per-table** mode. With this mode enabled, a newly created `InnoDB` table and associated indexes can be stored in a file-per-table **.ibd file**, outside the **system tablespace**.

This option affects the performance and storage considerations for a number of SQL statements, such as `DROP TABLE` and `TRUNCATE TABLE`.

Enabling the `innodb_file_per_table` option allows you to take advantage of features such as table **compression** and named-table backups in **MySQL Enterprise Backup**.

For more information, see `innodb_file_per_table`, and Section 14.6.3.2, “File-Per-Table Tablespaces”.
See Also compression, file-per-table, .ibd file, MySQL Enterprise Backup, system tablespace.

innodb_lock_wait_timeout
The `innodb_lock_wait_timeout` option sets the balance between **waiting** for shared resources to become available, or giving up and handling the error, retrying, or doing alternative processing in your application. Rolls back any `InnoDB` transaction that waits more than a specified time to acquire a **lock**. Especially useful if **deadlocks** are caused by updates to multiple tables controlled by different storage engines; such deadlocks are not **detected** automatically.
See Also deadlock, deadlock detection, lock, wait.

innodb_strict_mode
The `innodb_strict_mode` option controls whether `InnoDB` operates in **strict mode**, where conditions that are normally treated as warnings, cause errors instead (and the underlying statements fail).
See Also strict mode.

insert
One of the primary **DML** operations in **SQL**. The performance of inserts is a key factor in **data warehouse** systems that load millions of rows into tables, and **OLTP** systems where many concurrent connections might insert rows into the same table, in arbitrary order. If insert performance is important to you, you should learn about **InnoDB** features such as the **insert buffer** used in **change buffering**, and **auto-increment** columns.
See Also auto-increment, change buffering, data warehouse, DML, InnoDB, insert buffer, OLTP, SQL.

insert buffer
The former name of the **change buffer**. In MySQL 5.5, support was added for buffering changes to secondary index pages for `DELETE` and `UPDATE` operations. Previously, only changes resulting from `INSERT` operations were buffered. The preferred term is now *change buffer*.
See Also change buffer, change buffering.

insert buffering
The technique of storing changes to secondary index pages, resulting from `INSERT` operations, in the **change buffer** rather than writing the changes immediately, so that the physical writes can be performed to minimize random I/O. It is one of the types of **change buffering**; the others are **delete buffering** and **purge buffering**.

Insert buffering is not used if the secondary index is **unique**, because the uniqueness of new values cannot be verified before the new entries are written out. Other kinds of change buffering do work for unique indexes.
See Also change buffer, change buffering, delete buffering, insert buffer, purge buffering, unique index.

insert intention lock
A type of **gap lock** that is set by `INSERT` operations prior to row insertion. This type of **lock** signals the intent to insert in such a way that multiple transactions inserting into the same index gap need not wait for each other if they are not inserting at the same position within the gap. For more information, see Section 14.7.1, “InnoDB Locking”.
See Also gap lock, lock, next-key lock.

instance
A single **mysqld** daemon managing a **data directory** representing one or more **databases** with a set of **tables**. It is common in development, testing, and some **replication** scenarios to have multiple instances on the same **server** machine, each managing its own data directory and listening on its own port or socket. With one instance running a **disk-bound** workload, the server might still have extra CPU and memory capacity to run additional instances.
See Also data directory, database, disk-bound, mysqld, replication, server, table.

instrumentation
Modifications at the source code level to collect performance data for tuning and debugging. In MySQL, data collected by instrumentation is exposed through an SQL interface using the `INFORMATION_SCHEMA` and `PERFORMANCE_SCHEMA` databases.
See Also INFORMATION_SCHEMA, Performance Schema.

intention exclusive lock
See intention lock.

intention lock
A kind of **lock** that applies to the table, used to indicate the kind of lock the **transaction** intends to acquire on rows in the table. Different transactions can acquire different kinds of intention locks on the same table, but the first transaction to acquire an *intention exclusive* (IX) lock on a table prevents other transactions from acquiring any S or X locks on the table. Conversely, the first transaction to acquire an *intention shared* (IS) lock on a table prevents other transactions from acquiring any X locks on the table. The two-phase process allows the lock requests to be resolved in order, without blocking locks and corresponding operations that are compatible. For more information about this locking mechanism, see Section 14.7.1, “InnoDB Locking”.
See Also lock, lock mode, locking, transaction.

intention shared lock
See intention lock.

interceptor
: Code for instrumenting or debugging some aspect of an application, which can be enabled without recompiling or changing the source of the application itself.
See Also command interceptor, Connector/J, Connector/NET, exception interceptor.

inverted index
: A data structure optimized for document retrieval systems, used in the implementation of `InnoDB` **full-text search**. The `InnoDB` **FULLTEXT index**, implemented as an inverted index, records the position of each word within a document, rather than the location of a table row. A single column value (a document stored as a text string) is represented by many entries in the inverted index.
See Also full-text search, FULLTEXT index, ilist.

IOPS
: Acronym for **I/O operations per second**. A common measurement for busy systems, particularly **OLTP** applications. If this value is near the maximum that the storage devices can handle, the application can become **disk-bound**, limiting **scalability**.
See Also disk-bound, OLTP, scalability.

isolation level
: One of the foundations of database processing. Isolation is the **I** in the acronym **ACID**; the isolation level is the setting that fine-tunes the balance between performance and reliability, consistency, and reproducibility of results when multiple **transactions** are making changes and performing queries at the same time.

  From highest amount of consistency and protection to the least, the isolation levels supported by InnoDB are: **SERIALIZABLE**, **REPEATABLE READ**, **READ COMMITTED**, and **READ UNCOMMITTED**.

  With `InnoDB` tables, many users can keep the default isolation level (*REPEATABLE READ*) for all operations. Expert users might choose the **READ COMMITTED** level as they push the boundaries of scalability with **OLTP** processing, or during data warehousing operations where minor inconsistencies do not affect the aggregate results of large amounts of data. The levels on the edges (**SERIALIZABLE** and **READ UNCOMMITTED**) change the processing behavior to such an extent that they are rarely used.
  See Also ACID, OLTP, READ COMMITTED, READ UNCOMMITTED, REPEATABLE READ, SERIALIZABLE, transaction.

J

J2EE
: Java Platform, Enterprise Edition: Oracle's enterprise Java platform. It consists of an API and a runtime environment for enterprise-class Java applications. For full details, see http://www.oracle.com/technetwork/java/javaee/overview/index.html. With MySQL applications, you typically use **Connector/J** for database access, and an application server such as **Tomcat** or **JBoss** to handle the middle-tier work, and optionally a framework such as **Spring**. Database-related features often offered within a J2EE stack include a **connection pool** and **failover** support.
See Also connection pool, Connector/J, failover, Java, JBoss, Spring, Tomcat.

Java
: A programming language combining high performance, rich built-in features and data types, object-oriented mechanisms, extensive standard library, and wide range of reusable third-party modules. Enterprise development is supported by many frameworks, application servers, and other technologies. Much of its syntax is familiar to **C** and **C++** developers. To write Java applications with MySQL, you use the **JDBC** driver known as **Connector/J**.
See Also C, Connector/J, C++, JDBC.

JBoss
: See Also J2EE.

JDBC
: Abbreviation for “Java Database Connectivity”, an **API** for database access from **Java** applications. Java developers writing MySQL applications use the **Connector/J** component as their JDBC driver.

See Also API, Connector/J, J2EE, Java.

JNDI

See Also Java.

join
A **query** that retrieves data from more than one table, by referencing columns in the tables that hold identical values. Ideally, these columns are part of an `InnoDB` **foreign key** relationship, which ensures **referential integrity** and that the join columns are **indexed**. Often used to save space and improve query performance by replacing repeated strings with numeric IDs, in a **normalized** data design.
See Also foreign key, index, normalized, query, referential integrity.

K

keystore

See Also SSL.

KEY_BLOCK_SIZE
An option to specify the size of data pages within an `InnoDB` table that uses **compressed row format**. The default is 8 kilobytes. Lower values risk hitting internal limits that depend on the combination of row size and compression percentage.

For `MyISAM` tables, `KEY_BLOCK_SIZE` optionally specifies the size in bytes to use for index key blocks. The value is treated as a hint; a different size could be used if necessary. A `KEY_BLOCK_SIZE` value specified for an individual index definition overrides a table-level `KEY_BLOCK_SIZE` value.
See Also compressed row format.

L

latch
A lightweight structure used by `InnoDB` to implement a **lock** for its own internal memory structures, typically held for a brief time measured in milliseconds or microseconds. A general term that includes both **mutexes** (for exclusive access) and **rw-locks** (for shared access). Certain latches are the focus of `InnoDB` performance tuning. Statistics about latch use and contention are available through the **Performance Schema** interface.
See Also lock, locking, mutex, Performance Schema, rw-lock.

libmysql
Informal name for the **libmysqlclient** library.
See Also libmysqlclient.

libmysqlclient
The library file, named `libmysqlclient.a` or `libmysqlclient.so`, that is typically linked into **client** programs written in **C**. Sometimes known informally as **libmysql** or the **mysqlclient** library.
See Also client, libmysql, mysqlclient.

libmysqld
This **embedded** MySQL server library makes it possible to run a full-featured MySQL server inside a **client** application. The main benefits are increased speed and more simple management for embedded applications. You link with the `libmysqld` library rather than **libmysqlclient**. The API is identical between all three of these libraries.
See Also client, embedded, libmysql, libmysqlclient.

lifecycle interceptor
A type of **interceptor** supported by **Connector/J**. It involves implementing the interface `com.mysql.jdbc.ConnectionLifecycleInterceptor`.

See Also Connector/J, interceptor.

list
The `InnoDB` **buffer pool** is represented as a list of memory **pages**. The list is reordered as new pages are accessed and enter the buffer pool, as pages within the buffer pool are accessed again and are considered newer, and as pages that are not accessed for a long time are **evicted** from the buffer pool. The buffer pool is divided into **sublists**, and the replacement policy is a variation of the familiar **LRU** technique.
See Also buffer pool, eviction, LRU, page, sublist.

load balancing
A technique for scaling read-only connections by sending query requests to different slave servers in a replication or Cluster configuration. With **Connector/J**, load balancing is enabled through the `com.mysql.jdbc.ReplicationDriver` class and controlled by the configuration property `loadBalanceStrategy`.
See Also Connector/J, J2EE.

localhost

See Also connection.

lock
The high-level notion of an object that controls access to a resource, such as a table, row, or internal data structure, as part of a **locking** strategy. For intensive performance tuning, you might delve into the actual structures that implement locks, such as **mutexes** and **latches**.
See Also latch, lock mode, locking, mutex.

lock escalation
An operation used in some database systems that converts many **row locks** into a single **table lock**, saving memory space but reducing concurrent access to the table. `InnoDB` uses a space-efficient representation for row locks, so that **lock** escalation is not needed.
See Also locking, row lock, table lock.

lock mode
A shared (S) **lock** allows a **transaction** to read a row. Multiple transactions can acquire an S lock on that same row at the same time.

An exclusive (X) lock allows a transaction to update or delete a row. No other transaction can acquire any kind of lock on that same row at the same time.

Intention locks apply to the table, and are used to indicate what kind of lock the transaction intends to acquire on rows in the table. Different transactions can acquire different kinds of intention locks on the same table, but the first transaction to acquire an intention exclusive (IX) lock on a table prevents other transactions from acquiring any S or X locks on the table. Conversely, the first transaction to acquire an intention shared (IS) lock on a table prevents other transactions from acquiring any X locks on the table. The two-phase process allows the lock requests to be resolved in order, without blocking locks and corresponding operations that are compatible.
See Also intention lock, lock, locking, transaction.

locking
The system of protecting a **transaction** from seeing or changing data that is being queried or changed by other transactions. The **locking** strategy must balance reliability and consistency of database operations (the principles of the **ACID** philosophy) against the performance needed for good **concurrency**. Fine-tuning the locking strategy often involves choosing an **isolation level** and ensuring all your database operations are safe and reliable for that isolation level.
See Also ACID, concurrency, isolation level, locking, transaction.

locking read
A `SELECT` statement that also performs a **locking** operation on an `InnoDB` table. Either `SELECT ... FOR UPDATE` or `SELECT ... LOCK IN SHARE MODE`. It has the potential to produce a **deadlock**, depending on

the **isolation level** of the transaction. The opposite of a **non-locking read**. Not allowed for global tables in a **read-only transaction**.

`SELECT ... FOR SHARE` replaces `SELECT ... LOCK IN SHARE MODE` in MySQL 8.0.1, but `LOCK IN SHARE MODE` remains available for backward compatibility.

See Section 14.7.2.4, “Locking Reads”.
See Also deadlock, isolation level, locking, non-locking read, read-only transaction.

log
In the `InnoDB` context, “log” or “log files” typically refers to the **redo log** represented by the **ib_logfile*N*** files. Another type of `InnoDB` log is the **undo log**, which is a storage area that holds copies of data modified by active transactions.

Other kinds of logs that are important in MySQL are the **error log** (for diagnosing startup and runtime problems), **binary log** (for working with replication and performing point-in-time restores), the **general query log** (for diagnosing application problems), and the **slow query log** (for diagnosing performance problems).
See Also binary log, error log, general query log, ib_logfile, redo log, slow query log, undo log.

log buffer
The memory area that holds data to be written to the **log files** that make up the **redo log**. It is controlled by the `innodb_log_buffer_size` configuration option.
See Also log file, redo log.

log file
One of the **ib_logfile*N*** files that make up the **redo log**. Data is written to these files from the **log buffer** memory area.
See Also ib_logfile, log buffer, redo log.

log group
The set of files that make up the **redo log**, typically named `ib_logfile0` and `ib_logfile1`. (For that reason, sometimes referred to collectively as **ib_logfile**.)
See Also ib_logfile, redo log.

logical
A type of operation that involves high-level, abstract aspects such as tables, queries, indexes, and other SQL concepts. Typically, logical aspects are important to make database administration and application development convenient and usable. Contrast with **physical**.
See Also logical backup, physical.

logical backup
A **backup** that reproduces table structure and data, without copying the actual data files. For example, the **`mysqldump`** command produces a logical backup, because its output contains statements such as `CREATE TABLE` and `INSERT` that can re-create the data. Contrast with **physical backup**. A logical backup offers flexibility (for example, you could edit table definitions or insert statements before restoring), but can take substantially longer to **restore** than a physical backup.
See Also backup, mysqldump, physical backup, restore.

loose_
A prefix added to `InnoDB` configuration options after server **startup**, so any new configuration options not recognized by the current level of MySQL do not cause a startup failure. MySQL processes configuration options that start with this prefix, but gives a warning rather than a failure if the part after the prefix is not a recognized option.
See Also startup.

low-water mark
A value representing a lower limit, typically a threshold value at which some corrective action begins or becomes more aggressive. Contrast with **high-water mark**.
See Also high-water mark.

LRU
An acronym for “least recently used”, a common method for managing storage areas. The items that have not been used recently are **evicted** when space is needed to cache newer items. `InnoDB` uses the LRU mechanism by default to manage the **pages** within the **buffer pool**, but makes exceptions in cases where a page might be read only a single time, such as during a **full table scan**. This variation of the LRU algorithm is called the **midpoint insertion strategy**. For more information, see Section 14.5.1, “Buffer Pool”.
See Also buffer pool, eviction, full table scan, midpoint insertion strategy, page.

LSN
Acronym for “log sequence number”. This arbitrary, ever-increasing value represents a point in time corresponding to operations recorded in the **redo log**. (This point in time is regardless of **transaction** boundaries; it can fall in the middle of one or more transactions.) It is used internally by `InnoDB` during **crash recovery** and for managing the **buffer pool**.

Prior to MySQL 5.6.3, the LSN was a 4-byte unsigned integer. The LSN became an 8-byte unsigned integer in MySQL 5.6.3 when the redo log file size limit increased from 4GB to 512GB, as additional bytes were required to store extra size information. Applications built on MySQL 5.6.3 or later that use LSN values should use 64-bit rather than 32-bit variables to store and compare LSN values.

In the **MySQL Enterprise Backup** product, you can specify an LSN to represent the point in time from which to take an **incremental backup**. The relevant LSN is displayed by the output of the `mysqlbackup` command. Once you have the LSN corresponding to the time of a full backup, you can specify that value to take a subsequent incremental backup, whose output contains another LSN for the next incremental backup.
See Also buffer pool, crash recovery, incremental backup, MySQL Enterprise Backup, redo log, transaction.

M

.MRG file
A file containing references to other tables, used by the `MERGE` storage engine. Files with this extension are always included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.
See Also MySQL Enterprise Backup, mysqlbackup command.

.MYD file
A file that MySQL uses to store data for a `MyISAM` table.
See Also .MYI file, MySQL Enterprise Backup, mysqlbackup command.

.MYI file
A file that MySQL uses to store indexes for a `MyISAM` table.
See Also .MYD file, MySQL Enterprise Backup, mysqlbackup command.

master
See source.

master thread
An `InnoDB` **thread** that performs various tasks in the background. Most of these tasks are I/O related, such as writing changes from the **change buffer** to the appropriate secondary indexes.

To improve **concurrency**, sometimes actions are moved from the master thread to separate background threads. For example, in MySQL 5.6 and higher, **dirty pages** are **flushed** from the **buffer pool** by the **page cleaner** thread rather than the master thread.
See Also buffer pool, change buffer, concurrency, dirty page, flush, page cleaner, thread.

MDL
Acronym for “metadata lock”.
See Also metadata lock.

medium trust
Synonym for **partial trust**. Because the range of trust settings is so broad, “partial trust” is preferred, to avoid the implication that there are only three levels (low, medium, and full).

See Also Connector/NET, partial trust.

memcached
: A popular component of many MySQL and **NoSQL** software stacks, allowing fast reads and writes for single values and caching the results entirely in memory. Traditionally, applications required extra logic to write the same data to a MySQL database for permanent storage, or to read data from a MySQL database when it was not cached yet in memory. Now, applications can use the simple `memcached` protocol, supported by client libraries for many languages, to communicate directly with MySQL servers using `InnoDB` or `NDB` tables. These NoSQL interfaces to MySQL tables allow applications to achieve higher read and write performance than by issuing SQL statements directly, and can simplify application logic and deployment configurations for systems that already incorporate `memcached` for in-memory caching.

  The `memcached` interface to `InnoDB` tables is available in MySQL 5.6 and higher; see Section 14.20, “InnoDB memcached Plugin” for details. The `memcached` interface to `NDB` tables is available in NDB Cluster 7.2 and later; see http://dev.mysql.com/doc/ndbapi/en/ndbmemcache.html for details.
  See Also InnoDB, NoSQL.

merge
: To apply changes to data cached in memory, such as when a page is brought into the **buffer pool**, and any applicable changes recorded in the **change buffer** are incorporated into the page in the buffer pool. The updated data is eventually written to the **tablespace** by the **flush** mechanism.
  See Also buffer pool, change buffer, flush, tablespace.

metadata lock
: A type of **lock** that prevents **DDL** operations on a table that is being used at the same time by another **transaction**. For details, see Section 8.11.4, “Metadata Locking”.

  Enhancements to **online** operations, particularly in MySQL 5.6 and higher, are focused on reducing the amount of metadata locking. The objective is for DDL operations that do not change the table structure (such as `CREATE INDEX` and `DROP INDEX` for `InnoDB` tables) to proceed while the table is being queried, updated, and so on by other transactions.
  See Also DDL, lock, online, transaction.

metrics counter
: A feature implemented by the `INNODB_METRICS` table in the **INFORMATION_SCHEMA**, in MySQL 5.6 and higher. You can query **counts** and totals for low-level `InnoDB` operations, and use the results for performance tuning in combination with data from the **Performance Schema**.
  See Also counter, INFORMATION_SCHEMA, Performance Schema.

midpoint insertion strategy
: The technique of initially bringing **pages** into the `InnoDB` **buffer pool** not at the “newest” end of the list, but instead somewhere in the middle. The exact location of this point can vary, based on the setting of the `innodb_old_blocks_pct` option. The intent is that pages that are only read once, such as during a **full table scan**, can be aged out of the buffer pool sooner than with a strict **LRU** algorithm. For more information, see Section 14.5.1, “Buffer Pool”.
  See Also buffer pool, full table scan, LRU, page.

mini-transaction
: An internal phase of `InnoDB` processing, when making changes at the **physical** level to internal data structures during **DML** operations. A mini-transaction (mtr) has no notion of **rollback**; multiple mini-transactions can occur within a single **transaction**. Mini-transactions write information to the **redo log** that is used during **crash recovery**. A mini-transaction can also happen outside the context of a regular transaction, for example during **purge** processing by background threads.
  See Also commit, crash recovery, DML, physical, purge, redo log, rollback, transaction.

mixed-mode insert
: An `INSERT` statement where **auto-increment** values are specified for some but not all of the new rows. For example, a multi-value `INSERT` could specify a value for the auto-increment column in some cases and `NULL` in other cases. `InnoDB` generates auto-increment values for the rows where the column value was specified as `NULL`. Another example is an `INSERT ... ON DUPLICATE KEY UPDATE` statement, where

auto-increment values might be generated but not used, for any duplicate rows that are processed as `UPDATE` rather than `INSERT` statements.

Can cause consistency issues between **source** and **replica** servers in a **replication** configuration. Can require adjusting the value of the **innodb_autoinc_lock_mode** configuration option.
See Also auto-increment, innodb_autoinc_lock_mode, replica, replication, source.

MM.MySQL
An older JDBC driver for MySQL that evolved into **Connector/J** when it was integrated with the MySQL product.
See Also Connector/J.

Mono
An Open Source framework developed by Novell, that works with **Connector/NET** and **C#** applications on Linux platforms.
See Also Connector/NET, C#.

mtr
See mini-transaction.

multi-core
A type of processor that can take advantage of multithreaded programs, such as the MySQL server.

multiversion concurrency control
See MVCC.

mutex
Informal abbreviation for “mutex variable”. (Mutex itself is short for “mutual exclusion”.) The low-level object that `InnoDB` uses to represent and enforce exclusive-access **locks** to internal in-memory data structures. Once the lock is acquired, any other process, thread, and so on is prevented from acquiring the same lock. Contrast with **rw-locks**, which `InnoDB` uses to represent and enforce shared-access **locks** to internal in-memory data structures. Mutexes and rw-locks are known collectively as **latches**.
See Also latch, lock, Performance Schema, Pthreads, rw-lock.

MVCC
Acronym for “multiversion concurrency control”. This technique lets `InnoDB` **transactions** with certain **isolation levels** perform **consistent read** operations; that is, to query rows that are being updated by other transactions, and see the values from before those updates occurred. This is a powerful technique to increase **concurrency**, by allowing queries to proceed without waiting due to **locks** held by the other transactions.

This technique is not universal in the database world. Some other database products, and some other MySQL storage engines, do not support it.
See Also ACID, concurrency, consistent read, isolation level, lock, transaction.

my.cnf
The name, on Unix or Linux systems, of the MySQL **option file**.
See Also my.ini, option file.

my.ini
The name, on Windows systems, of the MySQL **option file**.
See Also my.cnf, option file.

MyODBC drivers
Obsolete name for **Connector/ODBC**.
See Also Connector/ODBC.

mysql
The `mysql` program is the command-line interpreter for the MySQL database. It processes **SQL** statements, and also MySQL-specific commands such as `SHOW TABLES`, by passing requests to the `mysqld` daemon.
See Also mysqld, SQL.

MySQL Enterprise Backup
: A licensed product that performs **hot backups** of MySQL databases. It offers the most efficiency and flexibility when backing up `InnoDB` tables, but can also back up `MyISAM` and other kinds of tables.
See Also hot backup, InnoDB.

mysqlbackup command
: A command-line tool of the **MySQL Enterprise Backup** product. It performs a **hot backup** operation for `InnoDB` tables, and a warm backup for `MyISAM` and other kinds of tables. See Section 24.2, "MySQL Enterprise Backup Overview" for more information about this command.
See Also hot backup, MySQL Enterprise Backup, warm backup.

mysqlclient
: The informal name for the library that is implemented by the file **libmysqlclient**, with extension `.a` or `.so`.
See Also libmysqlclient.

mysqld
: `mysqld`, also known as MySQL Server, is a single multithreaded program that does most of the work in a MySQL installation. It does not spawn additional processes. MySQL Server manages access to the MySQL data directory that contains databases, tables, and other information such as log files and status files.

  `mysqld` runs as a Unix daemon or Windows service, constantly waiting for requests and performing maintenance work in the background.
  See Also instance, mysql.

MySQLdb
: The name of the open-source **Python** module that forms the basis of the MySQL **Python API**.
See Also Python, Python API.

mysqldump
: A command that performs a **logical backup** of some combination of databases, tables, and table data. The results are SQL statements that reproduce the original schema objects, data, or both. For substantial amounts of data, a **physical backup** solution such as **MySQL Enterprise Backup** is faster, particularly for the **restore** operation.
See Also logical backup, MySQL Enterprise Backup, physical backup, restore.

N

.NET
: See Also ADO.NET, ASP.net, Connector/NET, Mono, Visual Studio.

native C API
: Synonym for **libmysqlclient**.
See Also libmysql.

natural key
: An indexed column, typically a **primary key**, where the values have some real-world significance. Usually advised against because:

  - If the value should ever change, there is potentially a lot of index maintenance to re-sort the **clustered index** and update the copies of the primary key value that are repeated in each **secondary index**.
  - Even seemingly stable values can change in unpredictable ways that are difficult to represent correctly in the database. For example, one country can change into two or several, making the original country code obsolete. Or, rules about unique values might have exceptions. For example, even if taxpayer IDs are intended to be unique to a single person, a database might have to handle records that violate that rule, such as in cases of identity theft. Taxpayer IDs and other sensitive ID numbers also make poor primary keys, because they may need to be secured, encrypted, and otherwise treated differently than other columns.

Thus, it is typically better to use arbitrary numeric values to form a **synthetic key**, for example using an **auto-increment** column.
See Also auto-increment, clustered index, primary key, secondary index, synthetic key.

neighbor page
Any **page** in the same **extent** as a particular page. When a page is selected to be **flushed**, any neighbor pages that are **dirty** are typically flushed as well, as an I/O optimization for traditional hard disks. In MySQL 5.6 and up, this behavior can be controlled by the configuration variable `innodb_flush_neighbors`; you might turn that setting off for SSD drives, which do not have the same overhead for writing smaller batches of data at random locations.
See Also dirty page, extent, flush, page.

next-key lock
A combination of a **record lock** on the index record and a gap lock on the gap before the index record.
See Also gap lock, locking, record lock.

non-locking read
A **query** that does not use the `SELECT ... FOR UPDATE` or `SELECT ... LOCK IN SHARE MODE` clauses. The only kind of query allowed for global tables in a **read-only transaction**. The opposite of a **locking read**. See Section 14.7.2.3, "Consistent Nonlocking Reads".

`SELECT ... FOR SHARE` replaces `SELECT ... LOCK IN SHARE MODE` in MySQL 8.0.1, but `LOCK IN SHARE MODE` remains available for backward compatibility.
See Also locking read, query, read-only transaction.

non-repeatable read
The situation when a query retrieves data, and a later query within the same **transaction** retrieves what should be the same data, but the queries return different results (changed by another transaction committing in the meantime).

This kind of operation goes against the **ACID** principle of database design. Within a transaction, data should be consistent, with predictable and stable relationships.

Among different **isolation levels**, non-repeatable reads are prevented by the **serializable read** and **repeatable read** levels, and allowed by the **consistent read**, and **read uncommitted** levels.
See Also ACID, consistent read, isolation level, READ UNCOMMITTED, REPEATABLE READ, SERIALIZABLE, transaction.

nonblocking I/O
An industry term that means the same as **asynchronous I/O**.
See Also asynchronous I/O.

normalized
A database design strategy where data is split into multiple tables, and duplicate values condensed into single rows represented by an ID, to avoid storing, querying, and updating redundant or lengthy values. It is typically used in **OLTP** applications.

For example, an address might be given a unique ID, so that a census database could represent the relationship **lives at this address** by associating that ID with each member of a family, rather than storing multiple copies of a complex value such as **123 Main Street, Anytown, USA**.

For another example, although a simple address book application might store each phone number in the same table as a person's name and address, a phone company database might give each phone number a special ID, and store the numbers and IDs in a separate table. This normalized representation could simplify large-scale updates when area codes split apart.

Normalization is not always recommended. Data that is primarily queried, and only updated by deleting entirely and reloading, is often kept in fewer, larger tables with redundant copies of duplicate values. This data representation is referred to as **denormalized**, and is frequently found in data warehousing applications.
See Also denormalized, foreign key, OLTP, relational.

NoSQL
: A broad term for a set of data access technologies that do not use the **SQL** language as their primary mechanism for reading and writing data. Some NoSQL technologies act as key-value stores, only accepting single-value reads and writes; some relax the restrictions of the **ACID** methodology; still others do not require a pre-planned **schema**. MySQL users can combine NoSQL-style processing for speed and simplicity with SQL operations for flexibility and convenience, by using the **memcached** API to directly access some kinds of MySQL tables. The `memcached` interface to `InnoDB` tables is available in MySQL 5.6 and higher; see Section 14.20, "InnoDB memcached Plugin" for details. The `memcached` interface to `NDB` tables is available in NDB Cluster 7.2 and later; see ndbmemcache—Memcache API for NDB Cluster (NO LONGER SUPPORTED).
  See Also ACID, InnoDB, memcached, schema, SQL.

NOT NULL constraint
: A type of **constraint** that specifies that a **column** cannot contain any **NULL** values. It helps to preserve **referential integrity**, as the database server can identify data with erroneous missing values. It also helps in the arithmetic involved in query optimization, allowing the optimizer to predict the number of entries in an index on that column.
  See Also column, constraint, NULL, primary key, referential integrity.

NULL
: A special value in **SQL**, indicating the absence of data. Any arithmetic operation or equality test involving a `NULL` value, in turn produces a `NULL` result. (Thus it is similar to the IEEE floating-point concept of NaN, "not a number".) Any aggregate calculation such as `AVG()` ignores rows with `NULL` values, when determining how many rows to divide by. The only test that works with `NULL` values uses the SQL idioms `IS NULL` or `IS NOT NULL`.

  `NULL` values play a part in **index** operations, because for performance a database must minimize the overhead of keeping track of missing data values. Typically, `NULL` values are not stored in an index, because a query that tests an indexed column using a standard comparison operator could never match a row with a `NULL` value for that column. For the same reason, unique indexes do not prevent `NULL` values; those values simply are not represented in the index. Declaring a `NOT NULL` constraint on a column provides reassurance that there are no rows left out of the index, allowing for better query optimization (accurate counting of rows and estimation of whether to use the index).

  Because the **primary key** must be able to uniquely identify every row in the table, a single-column primary key cannot contain any `NULL` values, and a multi-column primary key cannot contain any rows with `NULL` values in all columns.

  Although the Oracle database allows a `NULL` value to be concatenated with a string, `InnoDB` treats the result of such an operation as `NULL`.
  See Also index, primary key, SQL.

O

.OPT file
: A file containing database configuration information. Files with this extension are included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.
  See Also MySQL Enterprise Backup, mysqlbackup command.

ODBC
: Acronym for Open Database Connectivity, an industry-standard API. Typically used with Windows-based servers, or applications that require ODBC to communicate with MySQL. The MySQL ODBC driver is called **Connector/ODBC**.
  See Also Connector/ODBC.

off-page column
: A column containing variable-length data (such as `BLOB` and `VARCHAR`) that is too long to fit on a **B-tree** page. The data is stored in **overflow pages**. The **DYNAMIC** row format is more efficient for such storage than the older **COMPACT** row format.

See Also B-tree, compact row format, dynamic row format, overflow page.

OLTP

Acronym for “Online Transaction Processing”. A database system, or a database application, that runs a workload with many **transactions**, with frequent writes as well as reads, typically affecting small amounts of data at a time. For example, an airline reservation system or an application that processes bank deposits. The data might be organized in **normalized** form for a balance between **DML** (insert/update/delete) efficiency and **query** efficiency. Contrast with **data warehouse**.

With its **row-level locking** and **transactional** capability, **InnoDB** is the ideal storage engine for MySQL tables used in OLTP applications.
See Also data warehouse, DML, InnoDB, query, row lock, transaction.

online

A type of operation that involves no downtime, blocking, or restricted operation for the database. Typically applied to **DDL**. Operations that shorten the periods of restricted operation, such as **fast index creation**, have evolved into a wider set of **online DDL** operations in MySQL 5.6.

In the context of backups, a **hot backup** is an online operation and a **warm backup** is partially an online operation.
See Also DDL, Fast Index Creation, hot backup, online DDL, warm backup.

online DDL

A feature that improves the performance, concurrency, and availability of `InnoDB` tables during **DDL** (primarily `ALTER TABLE`) operations. See Section 14.13, “InnoDB and Online DDL” for details.

The details vary according to the type of operation. In some cases, the table can be modified concurrently while the `ALTER TABLE` is in progress. The operation might be able to be performed without a table copy, or using a specially optimized type of table copy. DML log space usage for in-place operations is controlled by the `innodb_online_alter_log_max_size` configuration option.

This feature is an enhancement of the **Fast Index Creation** feature in MySQL 5.5.
See Also DDL, Fast Index Creation, online.

optimistic

A methodology that guides low-level implementation decisions for a relational database system. The requirements of performance and **concurrency** in a relational database mean that operations must be started or dispatched quickly. The requirements of consistency and **referential integrity** mean that any operation could fail: a transaction might be rolled back, a **DML** operation could violate a constraint, a request for a lock could cause a deadlock, a network error could cause a timeout. An optimistic strategy is one that assumes most requests or attempts succeed, so that relatively little work is done to prepare for the failure case. When this assumption is true, the database does little unnecessary work; when requests do fail, extra work must be done to clean up and undo changes.

`InnoDB` uses optimistic strategies for operations such as **locking** and **commits**. For example, data changed by a transaction can be written to the data files before the commit occurs, making the commit itself very fast, but requiring more work to undo the changes if the transaction is rolled back.

The opposite of an optimistic strategy is a **pessimistic** one, where a system is optimized to deal with operations that are unreliable and frequently unsuccessful. This methodology is rare in a database system, because so much care goes into choosing reliable hardware, networks, and algorithms.
See Also commit, concurrency, DML, locking, pessimistic, referential integrity.

optimizer

The MySQL component that determines the best **indexes** and **join** order to use for a **query**, based on characteristics and data distribution of the relevant **tables**.
See Also index, join, query, table.

option

A configuration parameter for MySQL, either stored in the **option file** or passed on the command line.

For the **options** that apply to **InnoDB** tables, each option name starts with the prefix `innodb_`.
See Also InnoDB, option, option file.

option file
The file that holds the configuration **options** for the MySQL instance. Traditionally, on Linux and Unix this file is named `my.cnf`, and on Windows it is named `my.ini`.
See Also configuration file, my.cnf, my.ini, option.

overflow page
Separately allocated disk **pages** that hold variable-length columns (such as `BLOB` and `VARCHAR`) that are too long to fit on a **B-tree** page. The associated columns are known as **off-page columns**.
See Also B-tree, off-page column, page.

P

.par file
A file containing partition definitions. Files with this extension are included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.

With the introduction of native partitioning support for `InnoDB` tables in MySQL 5.7.6, `.par` files are no longer created for partitioned `InnoDB` tables. Partitioned `MyISAM` tables continue to use `.par` files in MySQL 5.7. In MySQL 8.0, partitioning support is only provided by the `InnoDB` storage engine. As such, `.par` files are no longer used as of MySQL 8.0.
See Also MySQL Enterprise Backup, mysqlbackup command.

page
A unit representing how much data `InnoDB` transfers at any one time between disk (the **data files**) and memory (the **buffer pool**). A page can contain one or more **rows**, depending on how much data is in each row. If a row does not fit entirely into a single page, `InnoDB` sets up additional pointer-style data structures so that the information about the row can be stored in one page.

One way to fit more data in each page is to use **compressed row format**. For tables that use BLOBs or large text fields, **compact row format** allows those large columns to be stored separately from the rest of the row, reducing I/O overhead and memory usage for queries that do not reference those columns.

When `InnoDB` reads or writes sets of pages as a batch to increase I/O throughput, it reads or writes an **extent** at a time.

All the `InnoDB` disk data structures within a MySQL instance share the same **page size**.
See Also buffer pool, compact row format, compressed row format, data files, extent, page size, row.

page cleaner
An `InnoDB` background **thread** that **flushes dirty pages** from the **buffer pool**. Prior to MySQL 5.6, this activity was performed by the **master thread**. The number of page cleaner threads is controlled by the `innodb_page_cleaners` configuration option, introduced in MySQL 5.7.4.
See Also buffer pool, dirty page, flush, master thread, thread.

page size
For releases up to and including MySQL 5.5, the size of each `InnoDB` **page** is fixed at 16 kilobytes. This value represents a balance: large enough to hold the data for most rows, yet small enough to minimize the performance overhead of transferring unneeded data to memory. Other values are not tested or supported.

Starting in MySQL 5.6, the page size for an `InnoDB` **instance** can be either 4KB, 8KB, or 16KB, controlled by the `innodb_page_size` configuration option. As of MySQL 5.7.6, `InnoDB` also supports 32KB and 64KB page sizes. For 32KB and 64KB page sizes, `ROW_FORMAT=COMPRESSED` is not supported and the maximum record size is 16KB.

Page size is set when creating the MySQL instance, and it remains constant afterward. The same page size applies to all `InnoDB` **tablespaces**, including the **system tablespace**, **file-per-table** tablespaces, and **general tablespaces**.

Smaller page sizes can help performance with storage devices that use small block sizes, particularly for **SSD** devices in **disk-bound** workloads, such as for **OLTP** applications. As individual rows are updated, less data is copied into memory, written to disk, reorganized, locked, and so on.
See Also disk-bound, file-per-table, general tablespace, instance, OLTP, page, SSD, system tablespace, tablespace.

parent table
The table in a **foreign key** relationship that holds the initial column values pointed to from the **child table**. The consequences of deleting, or updating rows in the parent table depend on the `ON UPDATE` and `ON DELETE` clauses in the foreign key definition. Rows with corresponding values in the child table could be automatically deleted or updated in turn, or those columns could be set to `NULL`, or the operation could be prevented.
See Also child table, foreign key.

partial backup
A **backup** that contains some of the **tables** in a MySQL database, or some of the databases in a MySQL instance. Contrast with **full backup**.
See Also backup, full backup, table.

partial index
An **index** that represents only part of a column value, typically the first N characters (the **prefix**) of a long `VARCHAR` value.
See Also index, index prefix.

partial trust
An execution environment typically used by hosting providers, where applications have some permissions but not others. For example, applications might be able to access a database server over a network, but be "sandboxed" with regard to reading and writing local files.
See Also Connector/NET.

Performance Schema
The `performance_schema` schema, in MySQL 5.5 and up, presents a set of tables that you can query to get detailed information about the performance characteristics of many internal parts of the MySQL server.
See Chapter 22, *MySQL Performance Schema*.
See Also INFORMATION_SCHEMA, latch, mutex, rw-lock.

Perl
A programming language with roots in Unix scripting and report generation. Incorporates high-performance regular expressions and file I/O. Large collection of reusable modules available through repositories such as CPAN.
See Also Perl API.

Perl API
An open-source **API** for MySQL applications written in the **Perl** language. Implemented through the `DBI` and `DBD::mysql` modules. For details, see Section 23.9, "MySQL Perl API".
See Also API, Perl.

persistent statistics
A feature that stores **index** statistics for `InnoDB` **tables** on disk, providing better **plan stability** for **queries**. For more information, see Section 14.8.11.1, "Configuring Persistent Optimizer Statistics Parameters".
See Also index, optimizer, plan stability, query, table.

pessimistic
A methodology that sacrifices performance or concurrency in favor of safety. It is appropriate if a high proportion of requests or attempts might fail, or if the consequences of a failed request are severe. `InnoDB` uses what is known as a pessimistic **locking** strategy, to minimize the chance of **deadlocks**. At the application level, you might avoid deadlocks by using a pessimistic strategy of acquiring all locks needed by a transaction at the very beginning.

Many built-in database mechanisms use the opposite **optimistic** methodology.

See Also deadlock, locking, optimistic.

phantom
A row that appears in the result set of a query, but not in the result set of an earlier query. For example, if a query is run twice within a **transaction**, and in the meantime, another transaction commits after inserting a new row or updating a row so that it matches the `WHERE` clause of the query.

This occurrence is known as a phantom read. It is harder to guard against than a **non-repeatable read**, because locking all the rows from the first query result set does not prevent the changes that cause the phantom to appear.

Among different **isolation levels**, phantom reads are prevented by the **serializable read** level, and allowed by the **repeatable read**, **consistent read**, and **read uncommitted** levels.
See Also consistent read, isolation level, non-repeatable read, READ UNCOMMITTED, REPEATABLE READ, SERIALIZABLE, transaction.

PHP
A programming language originating with web applications. The code is typically embedded as blocks within the source of a web page, with the output substituted into the page as it is transmitted by the web server. This is in contrast to applications such as CGI scripts that print output in the form of an entire web page. The PHP style of coding is used for highly interactive and dynamic web pages. Modern PHP programs can also be run as command-line or GUI applications.

MySQL applications are written using one of the **PHP APIs**. Reusable modules can be written in **C** and called from PHP.

Another technology for writing server-side web pages with MySQL is **ASP.net**.
See Also ASP.net, C, PHP API.

PHP API
Several **APIs** are available for writing MySQL applications in the **PHP** language: the original MySQL API (`Mysql`) the MySQL Improved Extension (`Mysqli`) the MySQL Native Driver (`Mysqlnd`) the MySQL functions (`PDO_MYSQL`), and Connector/PHP. For details, see MySQL and PHP.
See Also API, PHP.

physical
A type of operation that involves hardware-related aspects such as disk blocks, memory pages, files, bits, disk reads, and so on. Typically, physical aspects are important during expert-level performance tuning and problem diagnosis. Contrast with **logical**.
See Also logical, physical backup.

physical backup
A **backup** that copies the actual data files. For example, the **`mysqlbackup`** command of the **MySQL Enterprise Backup** product produces a physical backup, because its output contains data files that can be used directly by the `mysqld` server, resulting in a faster **restore** operation. Contrast with **logical backup**.
See Also backup, logical backup, MySQL Enterprise Backup, restore.

PITR
Acronym for **point-in-time recovery**.
See Also point-in-time recovery.

plan stability
A property of a **query execution plan**, where the optimizer makes the same choices each time for a given **query**, so that performance is consistent and predictable.
See Also query, query execution plan.

point-in-time recovery
The process of restoring a **backup** to recreate the state of the database at a specific date and time. Commonly abbreviated “PITR”. Because it is unlikely that the specified time corresponds exactly to the time

of a backup, this technique usually requires a combination of a **physical backup** and a **logical backup**. For example, with the **MySQL Enterprise Backup** product, you restore the last backup that you took before the specified point in time, then replay changes from the **binary log** between the time of the backup and the PITR time.
See Also backup, binary log, logical backup, MySQL Enterprise Backup, physical backup.

port
The number of the TCP/IP socket the database server listens on, used to establish a **connection**. Often specified in conjunction with a **host**. Depending on your use of network encryption, there might be one port for unencrypted traffic and another port for **SSL** connections.
See Also connection, host, SSL.

prefix
See index prefix.

prepared backup
A set of backup files, produced by the **MySQL Enterprise Backup** product, after all the stages of applying **binary logs** and **incremental backups** are finished. The resulting files are ready to be **restored**. Prior to the apply steps, the files are known as a **raw backup**.
See Also binary log, hot backup, incremental backup, MySQL Enterprise Backup, raw backup, restore.

prepared statement
An SQL statement that is analyzed in advance to determine an efficient execution plan. It can be executed multiple times, without the overhead for parsing and analysis each time. Different values can be substituted for literals in the `WHERE` clause each time, through the use of placeholders. This substitution technique improves security, protecting against some kinds of SQL injection attacks. You can also reduce the overhead for converting and copying return values to program variables.

Although you can use prepared statements directly through SQL syntax, the various **Connectors** have programming interfaces for manipulating prepared statements, and these APIs are more efficient than going through SQL.
See Also client-side prepared statement, connector, server-side prepared statement.

primary key
A set of columns—and by implication, the index based on this set of columns—that can uniquely identify every row in a table. As such, it must be a unique index that does not contain any `NULL` values.

`InnoDB` requires that every table has such an index (also called the **clustered index** or **cluster index**), and organizes the table storage based on the column values of the primary key.

When choosing primary key values, consider using arbitrary values (a **synthetic key**) rather than relying on values derived from some other source (a **natural key**).
See Also clustered index, index, natural key, synthetic key.

process
An instance of an executing program. The operating system switches between multiple running processes, allowing for a certain degree of **concurrency**. On most operating systems, processes can contain multiple **threads** of execution that share resources. Context-switching between threads is faster than the equivalent switching between processes.
See Also concurrency, thread.

pseudo-record
An artificial record in an index, used for **locking** key values or ranges that do not currently exist.
See Also infimum record, locking, supremum record.

Pthreads
The POSIX threads standard, which defines an API for threading and locking operations on Unix and Linux systems. On Unix and Linux systems, `InnoDB` uses this implementation for **mutexes**.
See Also mutex.

purge
A type of garbage collection performed by one or more separate background threads (controlled by `innodb_purge_threads`) that runs on a periodic schedule. Purge parses and processes **undo log** pages from the **history list** for the purpose of removing clustered and secondary index records that were marked for deletion (by previous `DELETE` statements) and are no longer required for **MVCC** or **rollback**. Purge frees undo log pages from the history list after processing them.
See Also history list, MVCC, rollback, undo log.

purge buffering
The technique of storing changes to secondary index pages, resulting from `DELETE` operations, in the **change buffer** rather than writing the changes immediately, so that the physical writes can be performed to minimize random I/O. (Because delete operations are a two-step process, this operation buffers the write that normally purges an index record that was previously marked for deletion.) It is one of the types of **change buffering**; the others are **insert buffering** and **delete buffering**.
See Also change buffer, change buffering, delete buffering, insert buffer, insert buffering.

purge lag
Another name for the `InnoDB` **history list**. Related to the `innodb_max_purge_lag` configuration option.
See Also history list, purge.

purge thread
A **thread** within the `InnoDB` process that is dedicated to performing the periodic **purge** operation. In MySQL 5.6 and higher, multiple purge threads are enabled by the `innodb_purge_threads` configuration option.
See Also purge, thread.

Python
A programming language used in a broad range of fields, from Unix scripting to large-scale applications. Includes runtime typing, built-in high-level data types, object-oriented features, and an extensive standard library. Often used as a “glue” language between components written in other languages. The MySQL **Python API** is the open-source **MySQLdb** module.
See Also MySQLdb, Python API.

Python API

See Also API, Python.

Q

query
In **SQL**, an operation that reads information from one or more **tables**. Depending on the organization of data and the parameters of the query, the lookup might be optimized by consulting an **index**. If multiple tables are involved, the query is known as a **join**.

For historical reasons, sometimes discussions of internal processing for statements use “query” in a broader sense, including other types of MySQL statements such as **DDL** and **DML** statements.
See Also DDL, DML, index, join, SQL, table.

query execution plan
The set of decisions made by the optimizer about how to perform a **query** most efficiently, including which **index** or indexes to use, and the order in which to **join** tables. **Plan stability** involves the same choices being made consistently for a given query.
See Also index, join, plan stability, query.

query log
See general query log.

quiesce
To reduce the amount of database activity, often in preparation for an operation such as an `ALTER TABLE`, a **backup**, or a **shutdown**. Might or might not involve doing as much **flushing** as possible, so that **InnoDB** does not continue doing background I/O.

In MySQL 5.6 and higher, the syntax `FLUSH TABLES ... FOR EXPORT` writes some data to disk for `InnoDB` tables that make it simpler to back up those tables by copying the data files.
See Also backup, flush, InnoDB, shutdown.

R

R-tree
A tree data structure used for spatial indexing of multi-dimensional data such as geographical coordinates, rectangles or polygons.
See Also B-tree.

RAID
Acronym for “Redundant Array of Inexpensive Drives”. Spreading I/O operations across multiple drives enables greater **concurrency** at the hardware level, and improves the efficiency of low-level write operations that otherwise would be performed in sequence.
See Also concurrency.

random dive
A technique for quickly estimating the number of different values in a column (the column's **cardinality**). `InnoDB` samples pages at random from the index and uses that data to estimate the number of different values.
See Also cardinality.

raw backup
The initial set of backup files produced by the **MySQL Enterprise Backup** product, before the changes reflected in the **binary log** and any **incremental backups** are applied. At this stage, the files are not ready to **restore**. After these changes are applied, the files are known as a **prepared backup**.
See Also binary log, hot backup, ibbackup_logfile, incremental backup, MySQL Enterprise Backup, prepared backup, restore.

READ COMMITTED
An **isolation level** that uses a **locking** strategy that relaxes some of the protection between **transactions**, in the interest of performance. Transactions cannot see uncommitted data from other transactions, but they can see data that is committed by another transaction after the current transaction started. Thus, a transaction never sees any bad data, but the data that it does see may depend to some extent on the timing of other transactions.

When a transaction with this isolation level performs `UPDATE ... WHERE` or `DELETE ... WHERE` operations, other transactions might have to wait. The transaction can perform `SELECT ... FOR UPDATE`, and `LOCK IN SHARE MODE` operations without making other transactions wait.

`SELECT ... FOR SHARE` replaces `SELECT ... LOCK IN SHARE MODE` in MySQL 8.0.1, but `LOCK IN SHARE MODE` remains available for backward compatibility.
See Also ACID, isolation level, locking, REPEATABLE READ, SERIALIZABLE, transaction.

read phenomena
Phenomena such as **dirty reads**, **non-repeatable reads**, and **phantom** reads which can occur when a transaction reads data that another transaction has modified.
See Also dirty read, non-repeatable read, phantom.

READ UNCOMMITTED
The **isolation level** that provides the least amount of protection between transactions. Queries employ a **locking** strategy that allows them to proceed in situations where they would normally wait for another transaction. However, this extra performance comes at the cost of less reliable results, including data that has been changed by other transactions and not committed yet (known as **dirty read**). Use this isolation level with great caution, and be aware that the results might not be consistent or reproducible, depending on what other transactions are doing at the same time. Typically, transactions with this isolation level only do queries, not insert, update, or delete operations.

See Also ACID, dirty read, isolation level, locking, transaction.

read view
An internal snapshot used by the **MVCC** mechanism of `InnoDB`. Certain **transactions**, depending on their **isolation level**, see the data values as they were at the time the transaction (or in some cases, the statement) started. Isolation levels that use a read view are **REPEATABLE READ**, **READ COMMITTED**, and **READ UNCOMMITTED**.
See Also isolation level, MVCC, READ COMMITTED, READ UNCOMMITTED, REPEATABLE READ, transaction.

read-ahead
A type of I/O request that prefetches a group of **pages** (an entire **extent**) into the **buffer pool** asynchronously, in case these pages are needed soon. The linear read-ahead technique prefetches all the pages of one extent based on access patterns for pages in the preceding extent. The random read-ahead technique prefetches all the pages for an extent once a certain number of pages from the same extent are in the buffer pool. Random read-ahead is not part of MySQL 5.5, but is re-introduced in MySQL 5.6 under the control of the `innodb_random_read_ahead` configuration option.
See Also buffer pool, extent, page.

read-only transaction
A type of **transaction** that can be optimized for `InnoDB` tables by eliminating some of the bookkeeping involved with creating a **read view** for each transaction. Can only perform **non-locking read** queries. It can be started explicitly with the syntax `START TRANSACTION READ ONLY`, or automatically under certain conditions. See Section 8.5.3, "Optimizing InnoDB Read-Only Transactions" for details.
See Also non-locking read, read view, transaction.

record lock
A lock on an index record. For example, `SELECT c1 FROM t WHERE c1 = 10 FOR UPDATE;` prevents any other transaction from inserting, updating, or deleting rows where the value of `t.c1` is 10. Contrast with **gap lock** and **next-key lock**.
See Also gap lock, lock, next-key lock.

redo
The data, in units of records, recorded in the **redo log** when DML statements make changes to `InnoDB` tables. It is used during **crash recovery** to correct data written by incomplete **transactions**. The ever-increasing **LSN** value represents the cumulative amount of redo data that has passed through the redo log.
See Also crash recovery, DML, LSN, redo log, transaction.

redo log
A disk-based data structure used during **crash recovery**, to correct data written by incomplete **transactions**. During normal operation, it encodes requests to change `InnoDB` table data, which result from SQL statements or low-level API calls through NoSQL interfaces. Modifications that did not finish updating the **data files** before an unexpected **shutdown** are replayed automatically.

The redo log is physically represented as a set of files, typically named `ib_logfile0` and `ib_logfile1`. The data in the redo log is encoded in terms of records affected; this data is collectively referred to as **redo**. The passage of data through the redo logs is represented by the ever-increasing **LSN** value. The original 4GB limit on maximum size for the redo log is raised to 512GB in MySQL 5.6.3.

The disk layout of the redo log is influenced by the configuration options `innodb_log_file_size`, `innodb_log_group_home_dir`, and (rarely) `innodb_log_files_in_group`. The performance of redo log operations is also affected by the **log buffer**, which is controlled by the `innodb_log_buffer_size` configuration option.
See Also crash recovery, data files, ib_logfile, log buffer, LSN, redo, shutdown, transaction.

redo log archiving
An `InnoDB` feature that, when enabled, sequentially writes redo log records to an archive file to avoid potential loss of data than can occur when a backup utility fails to keep pace with redo log generation while a backup operation is in progress. For more information, see Redo Log Archiving.

See Also redo log.

redundant row format
The oldest InnoDB **row format**. Prior to MySQL 5.0.3, it was the only row format available in InnoDB. From MySQL 5.0.3 to MySQL 5.7.8, the default row format is **COMPACT**. As of MySQL 5.7.9, the default row format is defined by the innodb_default_row_format configuration option, which has a default setting of **DYNAMIC**. You can still specify the **REDUNDANT** row format for compatibility with older InnoDB tables.

For more information, see Section 14.11, “InnoDB Row Formats”.
See Also compact row format, dynamic row format, row format.

referential integrity
The technique of maintaining data always in a consistent format, part of the **ACID** philosophy. In particular, data in different tables is kept consistent through the use of **foreign key constraints**, which can prevent changes from happening or automatically propagate those changes to all related tables. Related mechanisms include the **unique constraint**, which prevents duplicate values from being inserted by mistake, and the **NOT NULL constraint**, which prevents blank values from being inserted by mistake.
See Also ACID, FOREIGN KEY constraint, NOT NULL constraint, unique constraint.

relational
An important aspect of modern database systems. The database server encodes and enforces relationships such as one-to-one, one-to-many, many-to-one, and uniqueness. For example, a person might have zero, one, or many phone numbers in an address database; a single phone number might be associated with several family members. In a financial database, a person might be required to have exactly one taxpayer ID, and any taxpayer ID could only be associated with one person.

The database server can use these relationships to prevent bad data from being inserted, and to find efficient ways to look up information. For example, if a value is declared to be unique, the server can stop searching as soon as the first match is found, and it can reject attempts to insert a second copy of the same value.

At the database level, these relationships are expressed through SQL features such as **columns** within a table, unique and NOT NULL **constraints**, **foreign keys**, and different kinds of join operations. Complex relationships typically involve data split between more than one table. Often, the data is **normalized**, so that duplicate values in one-to-many relationships are stored only once.

In a mathematical context, the relations within a database are derived from set theory. For example, the OR and AND operators of a WHERE clause represent the notions of union and intersection.
See Also ACID, column, constraint, foreign key, normalized.

relevance
In the **full-text search** feature, a number signifying the similarity between the search string and the data in the **FULLTEXT index**. For example, when you search for a single word, that word is typically more relevant for a row where it occurs several times in the text than a row where it appears only once.
See Also full-text search, FULLTEXT index.

REPEATABLE READ
The default **isolation level** for InnoDB. It prevents any rows that are queried from being changed by other **transactions**, thus blocking **non-repeatable reads** but not **phantom** reads. It uses a moderately strict **locking** strategy so that all queries within a transaction see data from the same snapshot, that is, the data as it was at the time the transaction started.

When a transaction with this isolation level performs UPDATE ... WHERE, DELETE ... WHERE, SELECT ... FOR UPDATE, and LOCK IN SHARE MODE operations, other transactions might have to wait.

SELECT ... FOR SHARE replaces SELECT ... LOCK IN SHARE MODE in MySQL 8.0.1, but LOCK IN SHARE MODE remains available for backward compatibility.
See Also ACID, consistent read, isolation level, locking, phantom, transaction.

repertoire
Repertoire is a term applied to character sets. A character set repertoire is the collection of characters in the set. See Section 10.2.1, “Character Set Repertoire”.

replica
A database **server** machine in a **replication** topology that receives changes from another server (the **source**) and applies those same changes. Thus it maintains the same contents as the source, although it might lag somewhat behind.

In MySQL, replicas are commonly used in disaster recovery, to take the place of a source that fails. They are also commonly used for testing software upgrades and new settings, to ensure that database configuration changes do not cause problems with performance or reliability.

Replicas typically have high workloads, because they process all the **DML** (write) operations relayed from the source, as well as user queries. To ensure that replicas can apply changes from the source fast enough, they frequently have fast I/O devices and sufficient CPU and memory to run multiple database instances on the same server. For example, the source might use hard drive storage while the replicas use **SSD**s.
See Also DML, replication, server, source, SSD.

replication
The practice of sending changes from a **source**, to one or more **replicas**, so that all databases have the same data. This technique has a wide range of uses, such as load-balancing for better scalability, disaster recovery, and testing software upgrades and configuration changes. The changes can be sent between the databases by methods called **row-based replication** and **statement-based replication**.
See Also replica, row-based replication, source, statement-based replication.

restore
The process of putting a set of backup files from the **MySQL Enterprise Backup** product in place for use by MySQL. This operation can be performed to fix a corrupted database, to return to some earlier point in time, or (in a **replication** context) to set up a new **replica**. In the **MySQL Enterprise Backup** product, this operation is performed by the `copy-back` option of the `mysqlbackup` command.
See Also hot backup, MySQL Enterprise Backup, mysqlbackup command, prepared backup, replica, replication.

rollback
A **SQL** statement that ends a **transaction**, undoing any changes made by the transaction. It is the opposite of **commit**, which makes permanent any changes made in the transaction.

By default, MySQL uses the **autocommit** setting, which automatically issues a commit following each SQL statement. You must change this setting before you can use the rollback technique.
See Also ACID, autocommit, commit, SQL, transaction.

rollback segment
The storage area containing the **undo logs**. Rollback segments have traditionally resided in the **system tablespace**. As of MySQL 5.6, rollback segments can reside in **undo tablespaces**. As of MySQL 5.7, rollback segments are also allocated to the *global temporary tablespace*.
See Also system tablespace, undo log, undo tablespace.

row
The logical data structure defined by a set of **columns**. A set of rows makes up a **table**. Within `InnoDB` **data files**, each **page** can contain one or more rows.

Although `InnoDB` uses the term **row format** for consistency with MySQL syntax, the row format is a property of each table and applies to all rows in that table.
See Also column, data files, page, row format, table.

row format
The disk storage format for **rows** of an `InnoDB` **table**. As `InnoDB` gains new capabilities such as **compression**, new row formats are introduced to support the resulting improvements in storage efficiency and performance.

The row format of an `InnoDB` table is specified by the `ROW_FORMAT` option or by the `innodb_default_row_format` configuration option (introduced in MySQL 5.7.9). Row formats include

`REDUNDANT`, `COMPACT`, `COMPRESSED`, and `DYNAMIC`. To view the row format of an `InnoDB` table, issue the `SHOW TABLE STATUS` statement or query `InnoDB` table metadata in the `INFORMATION_SCHEMA`.
See Also compact row format, compressed row format, compression, dynamic row format, redundant row format, row, table.

row lock
A **lock** that prevents a row from being accessed in an incompatible way by another **transaction**. Other rows in the same table can be freely written to by other transactions. This is the type of **locking** done by **DML** operations on **InnoDB** tables.

Contrast with **table locks** used by `MyISAM`, or during **DDL** operations on `InnoDB` tables that cannot be done with **online DDL**; those locks block concurrent access to the table.
See Also DDL, DML, InnoDB, lock, locking, online DDL, table lock, transaction.

row-based replication
A form of **replication** where events are propagated from the **source** specifying how to change individual rows on the **replica**. It is safe to use for all settings of the `innodb_autoinc_lock_mode` option.
See Also auto-increment locking, innodb_autoinc_lock_mode, replica, replication, source, statement-based replication.

row-level locking
The **locking** mechanism used for **InnoDB** tables, relying on **row locks** rather than **table locks**. Multiple **transactions** can modify the same table concurrently. Only if two transactions try to modify the same row does one of the transactions wait for the other to complete (and release its row locks).
See Also InnoDB, locking, row lock, table lock, transaction.

Ruby
A programming language that emphasizes dynamic typing and object-oriented programming. Some syntax is familiar to **Perl** developers. There are two popular **Ruby APIs** for developing MySQL applications. (This manual does not cover higher-level Ruby frameworks.)
See Also API, Perl, Ruby API.

Ruby API
Two APIs are available for Ruby programmers developing MySQL applications. The MySQL/Ruby API is based on the **libmysqlclient** API library. The Ruby/MySQL API is written to use the native MySQL network protocol (a native driver). For full details, see Section 23.11, “MySQL Ruby APIs”.
See Also libmysql, Ruby.

rw-lock
The low-level object that `InnoDB` uses to represent and enforce shared-access **locks** to internal in-memory data structures following certain rules. Contrast with **mutexes**, which `InnoDB` uses to represent and enforce exclusive access to internal in-memory data structures. Mutexes and rw-locks are known collectively as **latches**.

`rw-lock` types include `s-locks` (shared locks), `x-locks` (exclusive locks), and `sx-locks` (shared-exclusive locks).

- An `s-lock` provides read access to a common resource.
- An `x-lock` provides write access to a common resource while not permitting inconsistent reads by other threads.
- An `sx-lock` provides write access to a common resource while permitting inconsistent reads by other threads. `sx-locks` were introduced in MySQL 5.7 to optimize concurrency and improve scalability for read-write workloads.

The following matrix summarizes rw-lock type compatibility.

	S	***SX***	***X***
S	Compatible	Compatible	Conflict

	S	*SX*	*X*
SX	Compatible	Conflict	Conflict
X	Conflict	Conflict	Conflict

See Also latch, lock, mutex, Performance Schema.

S

savepoint
: Savepoints help to implement nested **transactions**. They can be used to provide scope to operations on tables that are part of a larger transaction. For example, scheduling a trip in a reservation system might involve booking several different flights; if a desired flight is unavailable, you might **roll back** the changes involved in booking that one leg, without rolling back the earlier flights that were successfully booked.
See Also rollback, transaction.

scalability
: The ability to add more work and issue more simultaneous requests to a system, without a sudden drop in performance due to exceeding the limits of system capacity. Software architecture, hardware configuration, application coding, and type of workload all play a part in scalability. When the system reaches its maximum capacity, popular techniques for increasing scalability are **scale up** (increasing the capacity of existing hardware or software) and **scale out** (adding new servers and more instances of MySQL). Often paired with **availability** as critical aspects of a large-scale deployment.
See Also availability, scale out, scale up.

scale out
: A technique for increasing **scalability** by adding new servers and more instances of MySQL. For example, setting up replication, NDB Cluster, connection pooling, or other features that spread work across a group of servers. Contrast with **scale up**.
See Also scalability, scale up.

scale up
: A technique for increasing **scalability** by increasing the capacity of existing hardware or software. For example, increasing the memory on a server and adjusting memory-related parameters such as `innodb_buffer_pool_size` and `innodb_buffer_pool_instances`. Contrast with **scale out**.
See Also scalability, scale out.

schema
: Conceptually, a schema is a set of interrelated database objects, such as tables, table columns, data types of the columns, indexes, foreign keys, and so on. These objects are connected through SQL syntax, because the columns make up the tables, the foreign keys refer to tables and columns, and so on. Ideally, they are also connected logically, working together as part of a unified application or flexible framework. For example, the **INFORMATION_SCHEMA** and **performance_schema** databases use “schema” in their names to emphasize the close relationships between the tables and columns they contain.

  In MySQL, physically, a **schema** is synonymous with a **database**. You can substitute the keyword `SCHEMA` instead of `DATABASE` in MySQL SQL syntax, for example using `CREATE SCHEMA` instead of `CREATE DATABASE`.

  Some other database products draw a distinction. For example, in the Oracle Database product, a **schema** represents only a part of a database: the tables and other objects owned by a single user.
  See Also database, INFORMATION_SCHEMA, Performance Schema.

search index
: In MySQL, **full-text search** queries use a special kind of index, the **FULLTEXT index**. In MySQL 5.6.4 and up, `InnoDB` and `MyISAM` tables both support `FULLTEXT` indexes; formerly, these indexes were only available for `MyISAM` tables.
See Also full-text search, FULLTEXT index.

secondary index

A type of `InnoDB` **index** that represents a subset of table columns. An `InnoDB` table can have zero, one, or many secondary indexes. (Contrast with the **clustered index**, which is required for each `InnoDB` table, and stores the data for all the table columns.)

A secondary index can be used to satisfy queries that only require values from the indexed columns. For more complex queries, it can be used to identify the relevant rows in the table, which are then retrieved through lookups using the clustered index.

Creating and dropping secondary indexes has traditionally involved significant overhead from copying all the data in the `InnoDB` table. The **fast index creation** feature makes both `CREATE INDEX` and `DROP INDEX` statements much faster for `InnoDB` secondary indexes.
See Also clustered index, Fast Index Creation, index.

segment

A division within an `InnoDB` **tablespace**. If a tablespace is analogous to a directory, the segments are analogous to files within that directory. A segment can grow. New segments can be created.

For example, within a **file-per-table** tablespace, table data is in one segment and each associated index is in its own segment. The **system tablespace** contains many different segments, because it can hold many tables and their associated indexes. Prior to MySQL 8.0, the system tablespace also includes one or more **rollback segments** used for **undo logs**.

Segments grow and shrink as data is inserted and deleted. When a segment needs more room, it is extended by one **extent** (1 megabyte) at a time. Similarly, a segment releases one extent's worth of space when all the data in that extent is no longer needed.
See Also extent, file-per-table, rollback segment, system tablespace, tablespace, undo log.

selectivity

A property of data distribution, the number of distinct values in a column (its **cardinality**) divided by the number of records in the table. High selectivity means that the column values are relatively unique, and can retrieved efficiently through an index. If you (or the query optimizer) can predict that a test in a `WHERE` clause only matches a small number (or proportion) of rows in a table, the overall **query** tends to be efficient if it evaluates that test first, using an index.
See Also cardinality, query.

semi-consistent read

A type of read operation used for `UPDATE` statements, that is a combination of **READ COMMITTED** and **consistent read**. When an `UPDATE` statement examines a row that is already locked, `InnoDB` returns the latest committed version to MySQL so that MySQL can determine whether the row matches the `WHERE` condition of the `UPDATE`. If the row matches (must be updated), MySQL reads the row again, and this time `InnoDB` either locks it or waits for a lock on it. This type of read operation can only happen when the transaction has the READ COMMITTED **isolation level**, or when the `innodb_locks_unsafe_for_binlog` option is enabled. `innodb_locks_unsafe_for_binlog` was removed in MySQL 8.0.
See Also consistent read, isolation level, READ COMMITTED.

SERIALIZABLE

The **isolation level** that uses the most conservative locking strategy, to prevent any other **transactions** from inserting or changing data that was read by this transaction, until it is finished. This way, the same query can be run over and over within a transaction, and be certain to retrieve the same set of results each time. Any attempt to change data that was committed by another transaction since the start of the current transaction, cause the current transaction to wait.

This is the default isolation level specified by the SQL standard. In practice, this degree of strictness is rarely needed, so the default isolation level for `InnoDB` is the next most strict, **REPEATABLE READ**.
See Also ACID, consistent read, isolation level, locking, REPEATABLE READ, transaction.

server

A type of program that runs continuously, waiting to receive and act upon requests from another program (the **client**). Because often an entire computer is dedicated to running one or more server programs (such as

a database server, a web server, an application server, or some combination of these), the term **server** can also refer to the computer that runs the server software.
See Also client, mysqld.

server-side prepared statement
A **prepared statement** managed by the MySQL server. Historically, issues with server-side prepared statements led **Connector/J** and **Connector/PHP** developers to sometimes use **client-side prepared statements** instead. With modern MySQL server versions, server-side prepared statements are recommended for performance, scalability, and memory efficiency.
See Also client-side prepared statement, Connector/J, Connector/PHP, prepared statement.

servlet

See Also Connector/J.

shared lock
A kind of **lock** that allows other **transactions** to read the locked object, and to also acquire other shared locks on it, but not to write to it. The opposite of **exclusive lock**.
See Also exclusive lock, lock, transaction.

shared tablespace
Another way of referring to the **system tablespace** or a **general tablespace**. General tablespaces were introduced in MySQL 5.7. More than one table can reside in a shared tablespace. Only a single table can reside in a *file-per-table* tablespace.
See Also general tablespace, system tablespace.

sharp checkpoint
The process of **flushing** to disk all **dirty** buffer pool pages whose redo entries are contained in certain portion of the **redo log**. Occurs before `InnoDB` reuses a portion of a log file; the log files are used in a circular fashion. Typically occurs with write-intensive **workloads**.
See Also dirty page, flush, redo log, workload.

shutdown
The process of stopping the MySQL server. By default, this process cleans up operations for **InnoDB** tables, so `InnoDB` can be **slow** to shut down, but fast to start up later. If you skip the cleanup operations, it is **fast** to shut down but the cleanup must be performed during the next restart.

The shutdown mode for `InnoDB` is controlled by the `innodb_fast_shutdown` option.
See Also fast shutdown, InnoDB, slow shutdown, startup.

slave
See replica.

slow query log
A type of **log** used for performance tuning of SQL statements processed by the MySQL server. The log information is stored in a file. You must enable this feature to use it. You control which categories of “slow” SQL statements are logged. For more information, see Section 5.4.5, “The Slow Query Log”.
See Also general query log, log.

slow shutdown
A type of **shutdown** that does additional `InnoDB` flushing operations before completing. Also known as a **clean shutdown**. Specified by the configuration parameter `innodb_fast_shutdown=0` or the command `SET GLOBAL innodb_fast_shutdown=0;`. Although the shutdown itself can take longer, that time should be saved on the subsequent startup.
See Also clean shutdown, fast shutdown, shutdown.

snapshot
A representation of data at a particular time, which remains the same even as changes are **committed** by other **transactions**. Used by certain **isolation levels** to allow **consistent reads**.
See Also commit, consistent read, isolation level, transaction.

sort buffer
The buffer used for sorting data during creation of an `InnoDB` index. Sort buffer size is configured using the `innodb_sort_buffer_size` configuration option.

source
A database server machine in a **replication** scenario that processes the initial insert, update, and delete requests for data. These changes are propagated to, and repeated on, other servers known as **replicas**.
See Also replica, replication.

space ID
An identifier used to uniquely identify an `InnoDB` **tablespace** within a MySQL instance. The space ID for the **system tablespace** is always zero; this same ID applies to all tables within the system tablespace or within a general tablespace. Each **file-per-table** tablespace and **general tablespace** has its own space ID.

Prior to MySQL 5.6, this hardcoded value presented difficulties in moving `InnoDB` tablespace files between MySQL instances. Starting in MySQL 5.6, you can copy tablespace files between instances by using the **transportable tablespace** feature involving the statements `FLUSH TABLES ... FOR EXPORT`, `ALTER TABLE ... DISCARD TABLESPACE`, and `ALTER TABLE ... IMPORT TABLESPACE`. The information needed to adjust the space ID is conveyed in the **.cfg file** which you copy along with the tablespace. See Section 14.6.1.3, “Importing InnoDB Tables” for details.
See Also .cfg file, file-per-table, general tablespace, .ibd file, system tablespace, tablespace, transportable tablespace.

sparse file
A type of file that uses file system space more efficiently by writing metadata representing empty blocks to disk instead of writing the actual empty space. The `InnoDB` **transparent page compression** feature relies on sparse file support. For more information, see InnoDB Page Compression.
See Also hole punching, transparent page compression.

spin
A type of **wait** operation that continuously tests whether a resource becomes available. This technique is used for resources that are typically held only for brief periods, where it is more efficient to wait in a “busy loop” than to put the thread to sleep and perform a context switch. If the resource does not become available within a short time, the spin loop ceases and another wait technique is used.
See Also latch, lock, mutex, wait.

Spring
A Java-based application framework designed for assisting in application design by providing a way to configure components.
See Also J2EE.

SQL
The Structured Query Language that is standard for performing database operations. Often divided into the categories **DDL**, **DML**, and **queries**. MySQL includes some additional statement categories such as **replication**. See Chapter 9, *Language Structure* for the building blocks of SQL syntax, Chapter 11, *Data Types* for the data types to use for MySQL table columns, Chapter 13, *SQL Statements* for details about SQL statements and their associated categories, and Chapter 12, *Functions and Operators* for standard and MySQL-specific functions to use in queries.
See Also DDL, DML, query, replication.

SQLState
An error code defined by the **JDBC** standard, for exception handling by applications using **Connector/J**.
See Also Connector/J, JDBC.

SSD
Acronym for “solid-state drive”. A type of storage device with different performance characteristics than a traditional hard disk drive (**HDD**): smaller storage capacity, faster for random reads, no moving parts, and with a number of considerations affecting write performance. Its performance characteristics can influence the throughput of a **disk-bound** workload.
See Also disk-bound, HDD.

SSL
: Acronym for “secure sockets layer”. Provides the encryption layer for network communication between an application and a MySQL database server.
See Also keystore, truststore.

startup
: The process of starting the MySQL server. Typically done by one of the programs listed in Section 4.3, “Server and Server-Startup Programs”. The opposite of **shutdown**.
See Also shutdown.

statement interceptor
: A type of **interceptor** for tracing, debugging, or augmenting SQL statements issued by a database application. Sometimes also known as a **command interceptor**.

  In **Java** applications using **Connector/J**, setting up this type of interceptor involves implementing the `com.mysql.jdbc.StatementInterceptorV2` interface, and adding a `statementInterceptors` property to the **connection string**.

  In **Visual Studio** applications using **Connector/NET**, setting up this type of interceptor involves defining a class that inherits from the `BaseCommandInterceptor` class and specifying that class name as part of the connection string.
  See Also command interceptor, connection string, Connector/J, Connector/NET, interceptor, Java, Visual Studio.

statement-based replication
: A form of **replication** where SQL statements are sent from the **source** and replayed on the **replica**. It requires some care with the setting for the `innodb_autoinc_lock_mode` option, to avoid potential timing problems with **auto-increment locking**.
  See Also auto-increment locking, innodb_autoinc_lock_mode, replica, replication, row-based replication, source.

statistics
: Estimated values relating to each `InnoDB` **table** and **index**, used to construct an efficient **query execution plan**. The main values are the **cardinality** (number of distinct values) and the total number of table rows or index entries. The statistics for the table represent the data in its **primary key** index. The statistics for a **secondary index** represent the rows covered by that index.

  The values are estimated rather than counted precisely because at any moment, different **transactions** can be inserting and deleting rows from the same table. To keep the values from being recalculated frequently, you can enable **persistent statistics**, where the values are stored in `InnoDB` system tables, and refreshed only when you issue an `ANALYZE TABLE` statement.

  You can control how **NULL** values are treated when calculating statistics through the `innodb_stats_method` configuration option.

  Other types of statistics are available for database objects and database activity through the **INFORMATION_SCHEMA** and **PERFORMANCE_SCHEMA** tables.
  See Also cardinality, index, INFORMATION_SCHEMA, NULL, Performance Schema, persistent statistics, primary key, query execution plan, secondary index, table, transaction.

stemming
: The ability to search for different variations of a word based on a common root word, such as singular and plural, or past, present, and future verb tense. This feature is currently supported in `MyISAM` **full-text search** feature but not in **FULLTEXT indexes** for `InnoDB` tables.
  See Also full-text search, FULLTEXT index.

stopword
: In a **FULLTEXT index**, a word that is considered common or trivial enough that it is omitted from the **search index** and ignored in search queries. Different configuration settings control stopword processing for `InnoDB` and `MyISAM` tables. See Section 12.10.4, “Full-Text Stopwords” for details.

See Also FULLTEXT index, search index.

storage engine
A component of the MySQL database that performs the low-level work of storing, updating, and querying data. In MySQL 5.5 and higher, **InnoDB** is the default storage engine for new tables, superceding `MyISAM`. Different storage engines are designed with different tradeoffs between factors such as memory usage versus disk usage, read speed versus write speed, and speed versus robustness. Each storage engine manages specific tables, so we refer to `InnoDB` tables, `MyISAM` tables, and so on.

The **MySQL Enterprise Backup** product is optimized for backing up `InnoDB` tables. It can also back up tables handled by `MyISAM` and other storage engines.
See Also InnoDB, MySQL Enterprise Backup, table type.

stored object
A stored program or view.

stored program
A stored routine (procedure or function), trigger, or Event Scheduler event.

stored routine
A stored procedure or function.

strict mode
The general name for the setting controlled by the `innodb_strict_mode` option. Turning on this setting causes certain conditions that are normally treated as warnings, to be considered errors. For example, certain invalid combinations of options related to **file format** and **row format**, that normally produce a warning and continue with default values, now cause the `CREATE TABLE` operation to fail. `innodb_strict_mode` is enabled by default in MySQL 5.7.

MySQL also has something called strict mode. See Section 5.1.10, “Server SQL Modes”.
See Also file format, innodb_strict_mode, row format.

sublist
Within the list structure that represents the **buffer pool**, pages that are relatively old and relatively new are represented by different portions of the **list**. A set of parameters control the size of these portions and the dividing point between the new and old pages.
See Also buffer pool, eviction, list, LRU.

supremum record
A **pseudo-record** in an index, representing the **gap** above the largest value in that index. If a transaction has a statement such as `SELECT ... FROM ... WHERE col > 10 FOR UPDATE;`, and the largest value in the column is 20, it is a lock on the supremum record that prevents other transactions from inserting even larger values such as 50, 100, and so on.
See Also gap, infimum record, pseudo-record.

surrogate key
Synonym name for **synthetic key**.
See Also synthetic key.

synthetic key
An indexed column, typically a **primary key**, where the values are assigned arbitrarily. Often done using an **auto-increment** column. By treating the value as completely arbitrary, you can avoid overly restrictive rules and faulty application assumptions. For example, a numeric sequence representing employee numbers might have a gap if an employee was approved for hiring but never actually joined. Or employee number 100 might have a later hiring date than employee number 500, if they left the company and later rejoined. Numeric values also produce shorter values of predictable length. For example, storing numeric codes meaning “Road”, “Boulevard”, “Expressway”, and so on is more space-efficient than repeating those strings over and over.

Also known as a **surrogate key**. Contrast with **natural key**.
See Also auto-increment, natural key, primary key, surrogate key.

system tablespace
One or more data files (**ibdata files**) containing metadata for `InnoDB`-related objects (the `InnoDB` **data dictionary**), and the storage areas for the **change buffer**, the **doublewrite buffer**, and possibly **undo logs**. Depending on the `innodb_file_per_table` setting, it might also contain table and index data for `InnoDB` tables. The data and metadata in the system tablespace apply to all **databases** in a MySQL **instance**.

Prior to MySQL 5.6.7, the default was to keep all `InnoDB` tables and indexes inside the system tablespace, often causing this file to become very large. Because the system tablespace never shrinks, storage problems could arise if large amounts of temporary data were loaded and then deleted. In MySQL 5.6.7 and higher, the default is **file-per-table** mode, where each table and its associated indexes are stored in a separate **.ibd file**. This default makes it easier to use `InnoDB` features that rely on the **Barracuda** file format, such as table **compression**, efficient storage of **off-page columns**, and large index key prefixes (`innodb_large_prefix`).

Keeping all table data in the system tablespace or in separate `.ibd` files has implications for storage management in general. The **MySQL Enterprise Backup** product might back up a small set of large files, or many smaller files. On systems with thousands of tables, the file system operations to process thousands of `.ibd` files can cause bottlenecks.

In MySQL 5.6 and higher, the `innodb_undo_tablespaces` defines the number of undo tablespaces for undo logs.
See Also Barracuda, change buffer, compression, data dictionary, database, doublewrite buffer, dynamic row format, file-per-table, general tablespace, .ibd file, ibdata file, innodb_file_per_table, instance, MySQL Enterprise Backup, off-page column, tablespace, undo log.

T

.TRG file
A file containing trigger parameters. Files with this extension are always included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.
See Also MySQL Enterprise Backup, mysqlbackup command, .TRN file.

.TRN file
A file containing trigger namespace information. Files with this extension are always included in backups produced by the `mysqlbackup` command of the **MySQL Enterprise Backup** product.
See Also MySQL Enterprise Backup, mysqlbackup command, .TRG file.

table
Each MySQL table is associated with a particular **storage engine**. **InnoDB** tables have particular **physical** and **logical** characteristics that affect performance, **scalability**, **backup**, administration, and application development.

In terms of file storage, an `InnoDB` table belongs to one of the following tablespace types:

- The shared `InnoDB` **system tablespace**, which is comprised of one or more **ibdata files**.
- A **file-per-table** tablespace, comprised of an individual **.ibd file**.
- A shared **general tablespace**, comprised of an individual `.ibd` file. General tablespaces were introduced in MySQL 5.7.6.

`.ibd` data files contain both table and **index** data.

`InnoDB` tables created in file-per-table tablespaces can use the **Barracuda** file format, and Barracuda tables can use **DYNAMIC** or **COMPRESSED** row format. These row formats enable `InnoDB` features such as **compression**, efficient storage of **off-page columns**, and large index key prefixes (see `innodb_large_prefix`). General tablespaces support all row formats regardless of the `innodb_file_format` setting.

Up to MySQL 5.7.5, `InnoDB` tables inside the system tablespace had to use the **Antelope** file format for backward compatibility with MySQL 5.1 and earlier. The **Antelope** file format supports **COMPACT** and

REDUNDANT row format. The system tablespace supports tables that use **DYNAMIC** row format as of MySQL 5.7.6.

The **rows** of an `InnoDB` table are organized into an index structure known as the **clustered index**, with entries sorted based on the **primary key** columns of the table. Data access is optimized for queries that filter and sort on the primary key columns, and each index contains a copy of the associated primary key columns for each entry. Modifying values for any of the primary key columns is an expensive operation. Thus an important aspect of `InnoDB` table design is choosing a primary key with columns that are used in the most important queries, and keeping the primary key short, with rarely changing values.
See Also Antelope, backup, Barracuda, clustered index, compact row format, compressed row format, compression, dynamic row format, Fast Index Creation, file-per-table, .ibd file, index, off-page column, primary key, redundant row format, row, system tablespace, tablespace.

table lock
A lock that prevents any other **transaction** from accessing a table. `InnoDB` makes considerable effort to make such locks unnecessary, by using techniques such as **online DDL**, **row locks** and **consistent reads** for processing **DML** statements and **queries**. You can create such a lock through SQL using the `LOCK TABLE` statement; one of the steps in migrating from other database systems or MySQL storage engines is to remove such statements wherever practical.
See Also consistent read, DML, lock, locking, online DDL, query, row lock, table, transaction.

table scan
See full table scan.

table statistics
See statistics.

table type
Obsolete synonym for **storage engine**. We refer to `InnoDB` tables, `MyISAM` tables, and so on.
See Also InnoDB, storage engine.

tablespace
A data file that can hold data for one or more `InnoDB` **tables** and associated **indexes**.

The **system tablespace** contains the `InnoDB` **data dictionary**, and prior to MySQL 5.6 holds all other `InnoDB` tables by default.

The `innodb_file_per_table` option, enabled by default in MySQL 5.6 and higher, allows tables to be created in their own tablespaces. File-per-table tablespaces support features such as efficient storage of **off-page columns**, table compression, and transportable tablespaces. See Section 14.6.3.2, “File-Per-Table Tablespaces” for details.

`InnoDB` introduced general tablespaces in MySQL 5.7.6. General tablespaces are shared tablespaces created using `CREATE TABLESPACE` syntax. They can be created outside of the MySQL data directory, are capable of holding multiple tables, and support tables of all row formats.

MySQL NDB Cluster also groups its tables into tablespaces. See Section 18.5.10.1, “NDB Cluster Disk Data Objects” for details.
See Also compressed row format, data dictionary, data files, file-per-table, general tablespace, index, innodb_file_per_table, system tablespace, table.

Tcl
A programming language originating in the Unix scripting world. Sometimes extended by code written in **C**, **C++**, or **Java**. For the open-source Tcl **API** for MySQL, see Section 23.12, “MySQL Tcl API”.
See Also API.

temporary table
A **table** whose data does not need to be truly permanent. For example, temporary tables might be used as storage areas for intermediate results in complicated calculations or transformations; this intermediate data would not need to be recovered after a crash. Database products can take various shortcuts to improve the

performance of operations on temporary tables, by being less scrupulous about writing data to disk and other measures to protect the data across restarts.

Sometimes, the data itself is removed automatically at a set time, such as when the transaction ends or when the session ends. With some database products, the table itself is removed automatically too.
See Also table.

temporary tablespace
The tablespace for non-compressed `InnoDB` **temporary tables** and related objects, introduced in MySQL 5.7. The `innodb_temp_data_file_path` configuration file option defines the relative path, name, size, and attributes for temporary tablespace data files. If `innodb_temp_data_file_path` is not specified, the default behavior is to create a single auto-extending 12MB data file named `ibtmp1` in the data directory. The temporary tablespace is recreated on each server start and receives a dynamically generated space ID. The temporary tablespace cannot reside on a raw device. Startup is refused if the temporary tablespace cannot be created.

The temporary tablespace is removed on normal shutdown or on an aborted initialization. The temporary tablespace is not removed when a crash occurs. In this case, the database administrator may remove the temporary tablespace manually or restart the server with the same configuration, which removes and recreates the temporary tablespace.
See Also temporary table.

text collection
The set of columns included in a **FULLTEXT index**.
See Also FULLTEXT index.

thread
A unit of processing that is typically more lightweight than a **process**, allowing for greater **concurrency**.
See Also concurrency, master thread, process, Pthreads.

Tomcat
An open source **J2EE** application server, implementing the Java Servlet and JavaServer Pages programming technologies. Consists of a web server and Java servlet container. With MySQL, typically used in conjunction with **Connector/J**.
See Also J2EE.

torn page
An error condition that can occur due to a combination of I/O device configuration and hardware failure. If data is written out in chunks smaller than the `InnoDB` **page size** (by default, 16KB), a hardware failure while writing could result in only part of a page being stored to disk. The `InnoDB` **doublewrite buffer** guards against this possibility.
See Also doublewrite buffer.

TPS
Acronym for “**transactions** per second”, a unit of measurement sometimes used in benchmarks. Its value depends on the **workload** represented by a particular benchmark test, combined with factors that you control such as the hardware capacity and database configuration.
See Also transaction, workload.

transaction
Transactions are atomic units of work that can be **committed** or **rolled back**. When a transaction makes multiple changes to the database, either all the changes succeed when the transaction is committed, or all the changes are undone when the transaction is rolled back.

Database transactions, as implemented by `InnoDB`, have properties that are collectively known by the acronym **ACID**, for atomicity, consistency, isolation, and durability.
See Also ACID, commit, isolation level, lock, rollback.

transaction ID
An internal field associated with each **row**. This field is physically changed by `INSERT`, `UPDATE`, and `DELETE` operations to record which **transaction** has locked the row.

See Also implicit row lock, row, transaction.

transparent page compression
A feature added in MySQL 5.7.8 that permits page-level compression for `InnoDB` tables that reside in **file-per-table** tablespaces. Page compression is enabled by specifying the `COMPRESSION` attribute with `CREATE TABLE` or `ALTER TABLE`. For more information, see InnoDB Page Compression.
See Also file-per-table, hole punching, sparse file.

transportable tablespace
A feature that allows a **tablespace** to be moved from one instance to another. Traditionally, this has not been possible for `InnoDB` tablespaces because all table data was part of the **system tablespace**. In MySQL 5.6 and higher, the `FLUSH TABLES ... FOR EXPORT` syntax prepares an `InnoDB` table for copying to another server; running `ALTER TABLE ... DISCARD TABLESPACE` and `ALTER TABLE ... IMPORT TABLESPACE` on the other server brings the copied data file into the other instance. A separate **.cfg file**, copied along with the **.ibd file**, is used to update the table metadata (for example the **space ID**) as the tablespace is imported. See Section 14.6.1.3, “Importing InnoDB Tables” for usage information.
See Also .cfg file, .ibd file, space ID, system tablespace, tablespace.

troubleshooting
The process of determining the source of a problem. Some of the resources for troubleshooting MySQL problems include:

- Section 2.10.2.1, “Troubleshooting Problems Starting the MySQL Server”
- Section 6.2.14, “Troubleshooting Problems Connecting to MySQL”
- Section B.3.3.2, “How to Reset the Root Password”
- Section B.3.2, “Common Errors When Using MySQL Programs”
- Section 14.21, “InnoDB Troubleshooting”.

truncate
A **DDL** operation that removes the entire contents of a table, while leaving the table and related indexes intact. Contrast with **drop**. Although conceptually it has the same result as a `DELETE` statement with no `WHERE` clause, it operates differently behind the scenes: `InnoDB` creates a new empty table, drops the old table, then renames the new table to take the place of the old one. Because this is a DDL operation, it cannot be **rolled back**.

If the table being truncated contains **foreign keys** that reference another table, the truncation operation uses a slower method of operation, deleting one row at a time so that corresponding rows in the referenced table can be deleted as needed by any `ON DELETE CASCADE` clause. (MySQL 5.5 and higher do not allow this slower form of truncate, and return an error instead if foreign keys are involved. In this case, use a `DELETE` statement instead.
See Also DDL, drop, foreign key, rollback.

truststore

See Also SSL.

tuple
A technical term designating an ordered set of elements. It is an abstract notion, used in formal discussions of database theory. In the database field, tuples are usually represented by the columns of a table row. They could also be represented by the result sets of queries, for example, queries that retrieved only some columns of a table, or columns from joined tables.
See Also cursor.

two-phase commit
An operation that is part of a distributed **transaction**, under the **XA** specification. (Sometimes abbreviated as 2PC.) When multiple databases participate in the transaction, either all databases **commit** the changes, or all databases **roll back** the changes.

See Also commit, rollback, transaction, XA.

U

undo
: Data that is maintained throughout the life of a **transaction**, recording all changes so that they can be undone in case of a **rollback** operation. It is stored in **undo logs** either within the **system tablespace** (in MySQL 5.7 or earlier) or in separate **undo tablespaces**. As of MySQL 8.0, undo logs reside in undo tablespaces by default.
See Also rollback, rollback segment, system tablespace, transaction, undo log, undo tablespace.

undo buffer
: See undo log.

undo log
: A storage area that holds copies of data modified by active **transactions**. If another transaction needs to see the original data (as part of a **consistent read** operation), the unmodified data is retrieved from this storage area.

In MySQL 5.6 and MySQL 5.7, you can use the `innodb_undo_tablespaces` variable have undo logs reside in **undo tablespaces**, which can be placed on another storage device such as an **SSD**. In MySQL 8.0, undo logs reside in two default undo tablespaces that are created when MySQL is initialized, and additional undo tablespaces can be created using `CREATE UNDO TABLESPACE` syntax.

The undo log is split into separate portions, the **insert undo buffer** and the **update undo buffer**.
See Also consistent read, rollback segment, SSD, system tablespace, transaction, undo tablespace.

undo log segment
: A collection of **undo logs**. Undo log segments exists within **rollback segments**. An undo log segment might contain undo logs from multiple transactions. An undo log segment can only be used by one transaction at a time but can be reused after it is released at transaction **commit** or **rollback**. May also be referred to as an "undo segment".
See Also commit, rollback, rollback segment, undo log.

undo tablespace
: An undo tablespace contains **undo logs**. Undo logs exist within **undo log segments**, which are contained within **rollback segments**. Rollback segments have traditionally resided in the system tablespace. As of MySQL 5.6, rollback segments can reside in undo tablespaces. In MySQL 5.6 and MySQL 5.7, the number of undo tablespaces is controlled by the `innodb_undo_tablespaces` configuration option. In MySQL 8.0, two default undo tablespaces are created when the MySQL instance is initialized, and additional undo tablespaces can be created using `CREATE UNDO TABLESPACE` syntax.

For more information, see Section 14.6.3.3, "Undo Tablespaces".
See Also rollback segment, system tablespace, undo log, undo log segment.

Unicode
: A system for supporting national characters, character sets, code pages, and other internationalization aspects in a flexible and standardized way.

Unicode support is an important aspect of the **ODBC** standard. **Connector/ODBC** 5.1 is a Unicode driver, as opposed to Connector/ODBC 3.51, which is an **ANSI** driver.
See Also ANSI, Connector/ODBC, ODBC.

unique constraint
: A kind of **constraint** that asserts that a column cannot contain any duplicate values. In terms of **relational** algebra, it is used to specify 1-to-1 relationships. For efficiency in checking whether a value can be inserted (that is, the value does not already exist in the column), a unique constraint is supported by an underlying **unique index**.
See Also constraint, relational, unique index.

unique index
An index on a column or set of columns that have a **unique constraint**. Because the index is known not to contain any duplicate values, certain kinds of lookups and count operations are more efficient than in the normal kind of index. Most of the lookups against this type of index are simply to determine if a certain value exists or not. The number of values in the index is the same as the number of rows in the table, or at least the number of rows with non-null values for the associated columns.

Change buffering optimization does not apply to unique indexes. As a workaround, you can temporarily set `unique_checks=0` while doing a bulk data load into an `InnoDB` table.
See Also cardinality, change buffering, unique constraint, unique key.

unique key
The set of columns (one or more) comprising a **unique index**. When you can define a `WHERE` condition that matches exactly one row, and the query can use an associated unique index, the lookup and error handling can be performed very efficiently.
See Also cardinality, unique constraint, unique index.

V

variable-length type
A data type of variable length. `VARCHAR`, `VARBINARY`, and `BLOB` and `TEXT` types are variable-length types.

`InnoDB` treats fixed-length fields greater than or equal to 768 bytes in length as variable-length fields, which can be stored **off-page**. For example, a `CHAR(255)` column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with `utf8mb4`.
See Also off-page column, overflow page.

victim
The **transaction** that is automatically chosen to be **rolled back** when a **deadlock** is detected. `InnoDB` rolls back the transaction that has updated the fewest rows.

Deadlock detection can be disabled using the `innodb_deadlock_detect` configuration option.
See Also deadlock, deadlock detection, innodb_lock_wait_timeout, transaction.

view
A stored query that when invoked produces a result set. A view acts as a virtual table.

Visual Studio
For supported versions of Visual Studio, see the following references:

- Connector/NET: Connector/NET Versions
- Connector/C++ 8.0: Platform Support and Prerequisites

See Also Connector/C++, Connector/NET.

W

wait
When an operation, such as acquiring a **lock**, **mutex**, or **latch**, cannot be completed immediately, `InnoDB` pauses and tries again. The mechanism for pausing is elaborate enough that this operation has its own name, the **wait**. Individual threads are paused using a combination of internal `InnoDB` scheduling, operating system `wait()` calls, and short-duration **spin** loops.

On systems with heavy load and many transactions, you might use the output from the `SHOW INNODB STATUS` command or **Performance Schema** to determine whether threads are spending too much time waiting, and if so, how you can improve **concurrency**.
See Also concurrency, latch, lock, mutex, Performance Schema, spin.

warm backup
: A **backup** taken while the database is running, but that restricts some database operations during the backup process. For example, tables might become read-only. For busy applications and websites, you might prefer a **hot backup**.
See Also backup, cold backup, hot backup.

warm up
: To run a system under a typical **workload** for some time after startup, so that the **buffer pool** and other memory regions are filled as they would be under normal conditions. This process happens naturally over time when a MySQL server is restarted or subjected to a new workload.

Typically, you run a workload for some time to warm up the buffer pool before running performance tests, to ensure consistent results across multiple runs; otherwise, performance might be artificially low during the first run.

In MySQL 5.6, you can speed up the warmup process by enabling the `innodb_buffer_pool_dump_at_shutdown` and `innodb_buffer_pool_load_at_startup` configuration options, to bring the contents of the buffer pool back into memory after a restart. These options are enabled by default in MySQL 5.7. See Section 14.8.3.5, "Saving and Restoring the Buffer Pool State".
See Also buffer pool, workload.

workload
: The combination and volume of **SQL** and other database operations, performed by a database application during typical or peak usage. You can subject the database to a particular workload during performance testing to identify **bottlenecks**, or during capacity planning.
See Also bottleneck, CPU-bound, disk-bound, SQL.

write combining
: An optimization technique that reduces write operations when **dirty pages** are **flushed** from the `InnoDB` **buffer pool**. If a row in a page is updated multiple times, or multiple rows on the same page are updated, all of those changes are stored to the data files in a single write operation rather than one write for each change.
See Also buffer pool, dirty page, flush.

X

XA
: A standard interface for coordinating distributed **transactions**, allowing multiple databases to participate in a transaction while maintaining **ACID** compliance. For full details, see Section 13.3.7, "XA Transactions".

XA Distributed Transaction support is enabled by default. If you are not using this feature, you can disable the `innodb_support_xa` configuration option, avoiding the performance overhead of an extra fsync for each transaction.

As of MySQL 5.7.10, disabling `innodb_support_xa` is not permitted as it makes replication unsafe and prevents performance gains associated with **binary log** group commit. The `innodb_support_xa` configuration option is removed in MySQL 8.0.
See Also ACID, binary log, commit, transaction, two-phase commit.

Y

young
: A characteristic of a **page** in the `InnoDB` **buffer pool** meaning that it has been accessed recently, and so is moved within the buffer pool data structure, so that it is not **flushed** too soon by the **LRU** algorithm. This term is used in some **INFORMATION_SCHEMA** column names of tables related to the buffer pool.
See Also buffer pool, flush, INFORMATION_SCHEMA, LRU, page.